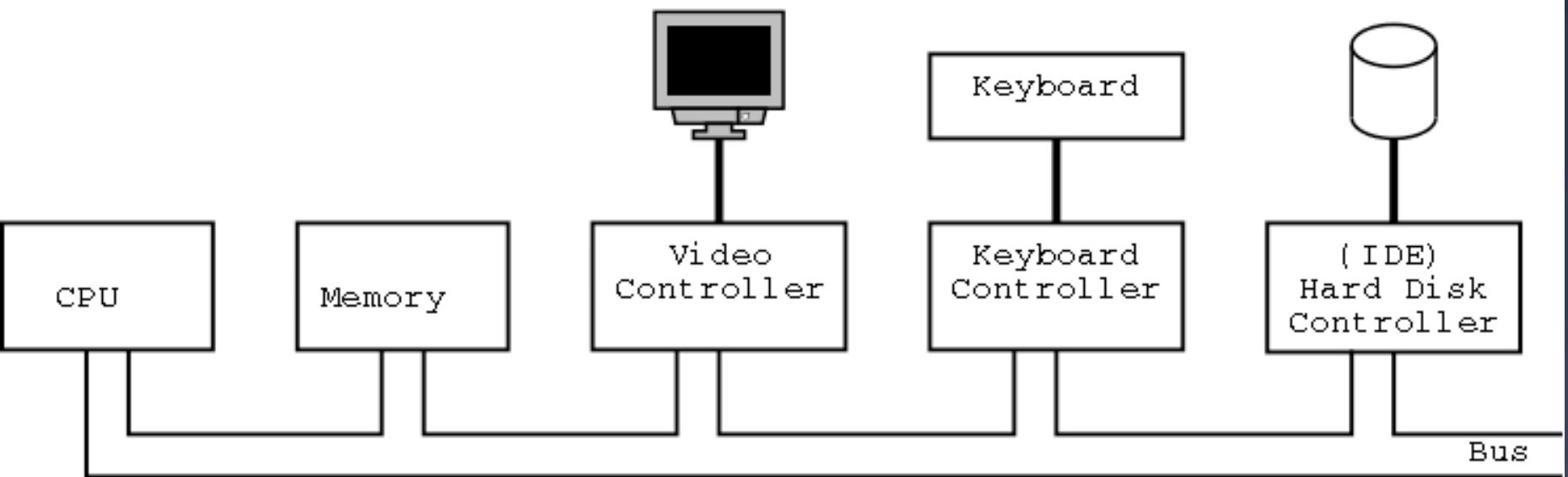


Sistemas Operacionais

Prof. Alan Durham

Um pouco de história

Arquitetura de um Computador (simplificada)



* Obs: atualmente barramentos separados para memória e vídeo

Alguns conceitos importantes

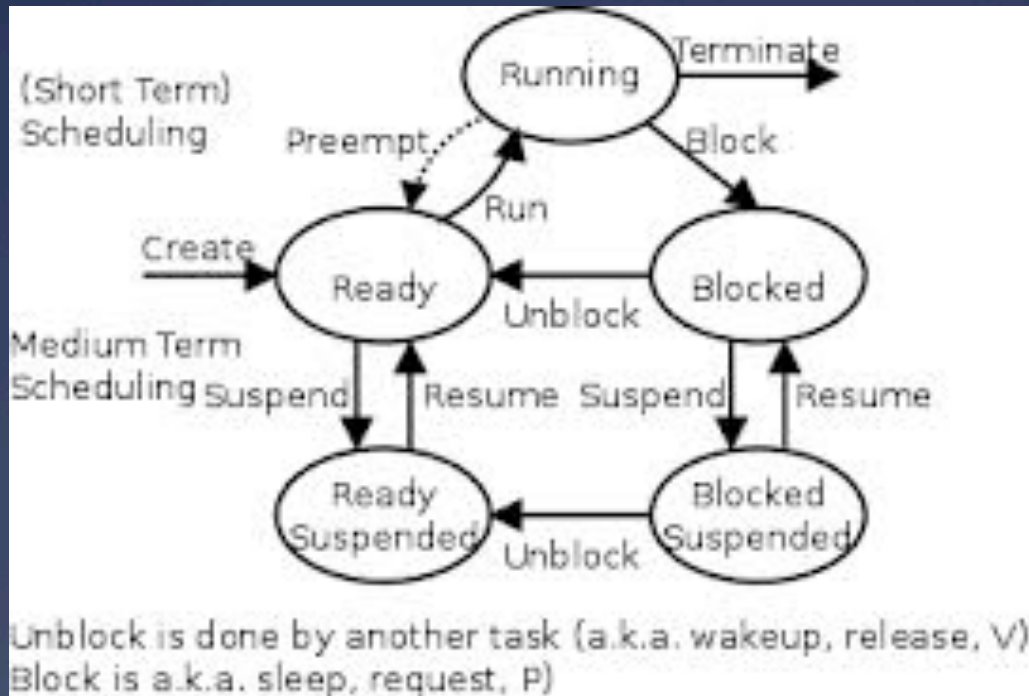
- * Barramento
 - * Trilhas comunicando um ou mais dispositivos, apenas uma mensagem de cada vez, todos recebem mensagem
- * Processador
 - * Program counter, pilha, PSW (prog. Status word), modo usuario e privilegiado, multicore, singlecore.
 - * Trap (interrupção) – sinal eletrônico que faz processador entrar em modo privilegiado e muda programa para endereço pré-especificado
- * Memória
 - * Proteção e troca de contexto: memória virtual, código relocavel

Alguns conceitos importantes

- * Discos
 - * Prato, superfície, cilindros, trilhas, setores, cabeças, tempo de busca, latencia de rotação, taxa de transferência
- * Outros dispositivos de E/S (I/O):
 - * Discos, vídeo, teclado, mouse, interfaces de rede, tec.
 - * CPU não se comunica com dispositivo (em geral analógico), mas com circuito digital de controle (controlador)
 - * Controlador conectado ao barramento, alguns registradores mapeados na CPU
 - * Cada controlador tem associado um “device driver”
 - * Quando CPU sabe que acabou? -> busy waiting ou interrupção.

Processos

- * Um programa sendo executado
- * Linha de processamento, registradores (pilha, etc.) estado da memória
- * 5 estados possíveis: rodando, pronto, bloqueado, suspenso (bloqueado e pronto)



fonte:<http://nyc>

Multiprogramação

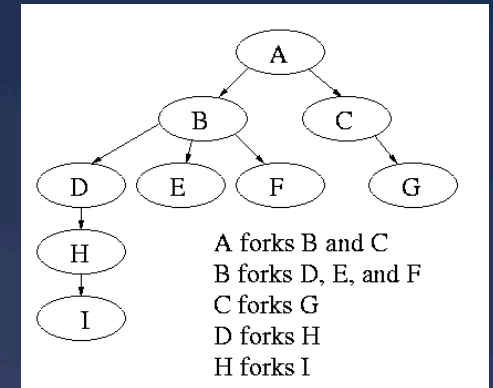
- * Multiprogramação:
 - * CPU deve ser otimizada em relação a execução
 - * pseudo paralelismo na CPU
 - * Paralelismo real nos periféricos
 - * NÃO é Tempo Real
 - * Threads (processos leves): linha de processamento distinta, contexto de registradores), mesmo estado de memória
- * Tabela de processos
- * Processos não podem assumir nada em relação a tempo real

Minix?

- * Anos 80: AT&T fecha código do sistema Unix (S7) e proíbe seu ensino em universidades
- * Dificuldade de construção de SO leva muitos cursos a ser tornarem apenas teóricos ou a propor apenas pequenas simulações
 - * Apenas algoritmos, sem compreensão da complexidade da construção de um SO.
- * A Tanenbaum se propõe a construir Minix:
 - * Compatível com versão 7 do Unix (mesmas chamadas de sistema)
 - * Escrito para ser lido, fácil modificação:
 - * Código C
 - * farto uso de comentários (original tinha 11.000 linhas de código e 3000 comentários.
 - * Fácil modificação
 - * Fontes disponíveis
 - * SO “de verdade”

Processos (Minix)

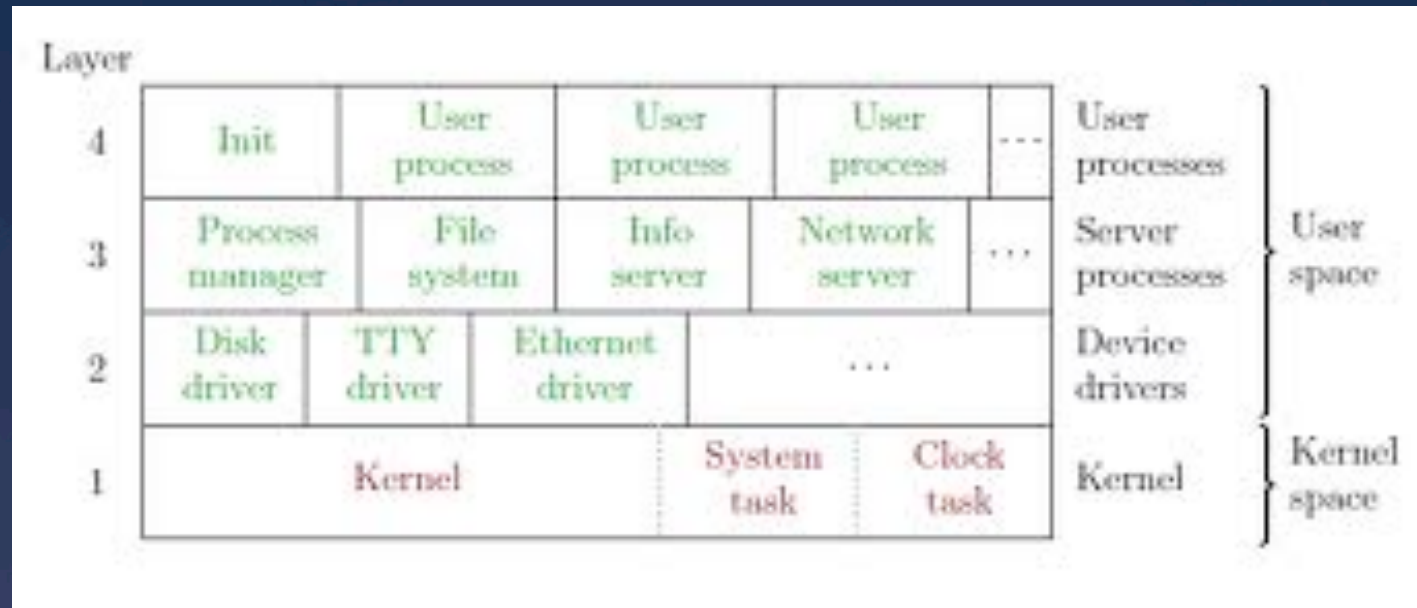
- * Processos podem se comunicar
 - * Mensagens
 - * Arquivos
 - * Pipes: `ls -l | more`
 - * Sinais (signal) – espécie de sinal assíncrono que faz com que receptor tenha código desviado para endereço pré-determinado
- * Organização hierárquica (Minix/Unix/Linux)
 - * Uid
 - * Obs: instrução `fork` cria novo processo, retorno é uid para pai, zero para filho



Arquivos Minix

- * Organização hierárquica
 - * Diretório *root*
 - * Caminhos absolutos e relativos
- * Proteção 9 bits (rwx)
- * Arquivo especial:
 - * Acesso transparente a I/O
 - * Bloco (disco) /caracter (teclado, etc.)
 - * Standard input/output: descritores de arquivo 0 e 1.
 - * Montar/desmontar: Utilizado para acesso a periféricos de maneira modular
 - * Major device number (tipo) minor device number
 - * /dev/hda0, /dev/hda1
- * Pipes
 - * Pseudo arquivo (implementado na memória) conectando dois processos (um escreve outro lê)
- * `Cat file1 file2 | sort >/dev/lp0`

Estrutura do Minix



- * 4 camadas: kernel (escalonamento de processos, tempo, tabelas, sistema de mensagens), drivers (abstração dos dispositivos), estratégias de escalonamento, sistemas de arquivo e comunicação (apresentam “máquina virtual), processos de usuário
- * Apenas 1 camada em “espaço de kernel”
- * Outras camadas possuem processos sem compartilhamento de memória: comunicação por mensagens
- * Comunicação apenas entre camadas adjacentes

Chamadas de Sistema no Minix

- * Biblioteca de chamada de procedimentos em C
- * Implementação como mensagem
 - * Código da função constrói mensagem e chama primitiva de envio de mensagem
 - * Processo fica bloqueado aguardando mensagem de retorno
 - * Código de biblioteca recupera mensagem de volta e retorna resultado como retorno de função
- * Controle de envio de mensagens impede mensagens entre camadas não adjacentes.

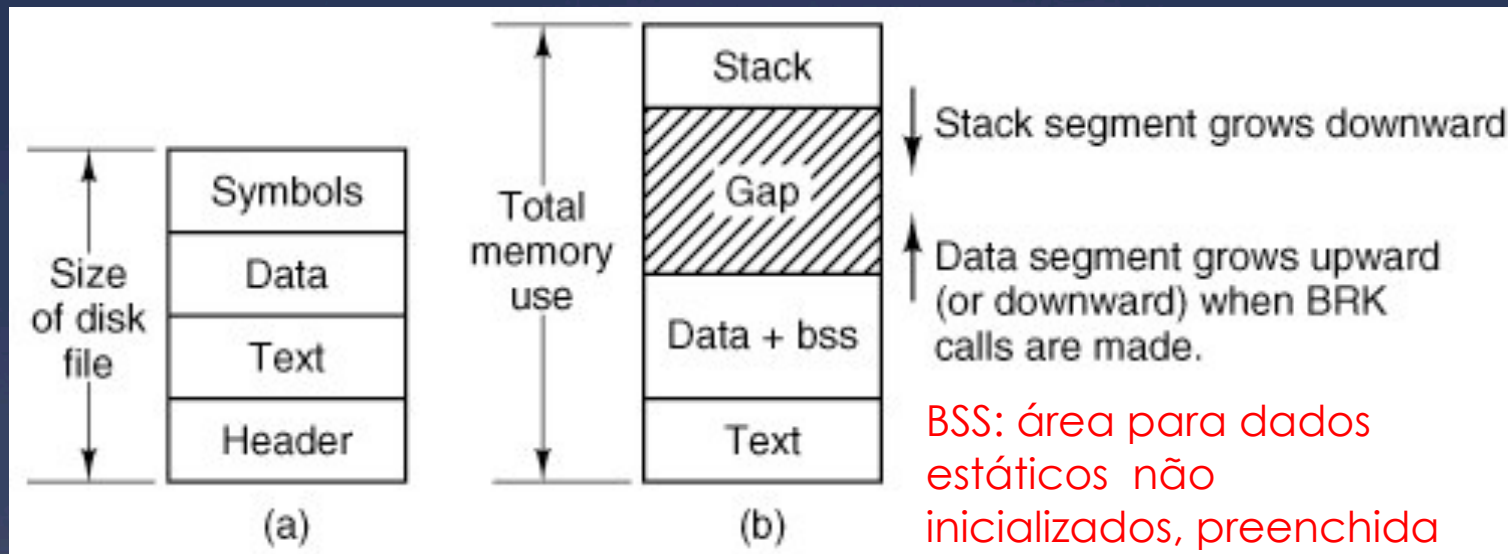
As chamadas de sistema do Minix: administração de processos

- * **Fork()** (única maneira de criar novos processos)
 - * Duas cópias começam idênticas mas memória separada
 - * Chamada volta zero para processo filho e pid para processo pai
- * **Wait()**
 - * Processo pai aguarda final de execução de um dos filhos
- * **Execve(name, argv, envp)**
 - * A imagem do processo é substituída pelo conteúdo de um arquivo, passados argumentos e ambiente (que descreve terminal, etc.)
- * **Exit()**
 - * Termina processo e retorna parâmetro como código de saída (zero ok)

As chamadas de sistema do Minix: administração de processos

* **BRK()**:

- * Altera tamanho da área de dados pelo parâmetro



BSS: área para dados estáticos não inicializados, preenchida com zeros.

Fonte: <http://flylib.com/books/en/3.275.1.44/>

* **GETPID()**

- * Retorna o PID do processo

Fonte figura: <http://www.os-forum.com/minix/net/general-comment-display.php?commentid=46>

As chamadas de sistema do Minix: Sinais

- * *SIGNAL(sig, func)*
 - * Determina captura de um sinal pela função func
- * *KILL(pid, signal)*
 - * Manda um sinal a um processo
- * *ALARM(seconds)*
 - * Agenda um o envio do sinal SIGALARM
- * *PAUSE()*
 - * Suspende o processo até que este receba o próximo sinal

As chamadas de sistema do Minix: Administração de Arquivos

- * *CREAT(name, mode)*
 - * Cria um novo arquivo e abre, ou abre e seta tamanho para zero, retorna descritor
- * *Mknod(nome, modo, addr)*
 - * Cria um i-node regular, especial ou de diretório, retorna descritor, addr é major e minor device number
- * *Open(file, how)*
 - * Abre um arquivo, retorna descritor
- * *Close(fd)*
 - * Fecha arquivo aberto

As chamadas de sistema do Minix: Administração de Arquivos

- * *Read(fd, buffer, nbytes)*
 - * Lê dados em um buffer, retorna bytes lidos
- * *Write(fd, buffer, nbytes)*
 - * Grava dados de um buffer, retorna numero de bytes gravados
- * *Lseek(fd, offset, whence)*
 - * Muda o ponteiro do arquivo, retorna nova posição, *whence* indica se offset relativo a pos. atual, inicio ou fim.
- * *stat(name, &buf), fstat(fd, &buf)*
 - * Lê e retorna estado de um arquivo de seu i-node (p. 29)

As chamadas de sistema do Minix: Administração de Arquivos

- * *Dup(df1)*
 - * Aloca um novo descritor de arquivos para um arquivo aberto.
- * *Pipe(&df[0])*
 - * Cria um pipe, retorna fd de leitura e escrita no vetor
- * *ioctl(fd, request, argp)*
 - * Executa operações em arquivos especiais (em geral terminais)
 - * Cooked mode, raw mode, cbreak mode.x (p. 30)

Uma nota em dup

- * Usado para redirecionar stdin, stdout
- * Criação de descritores (fds) é sempre sequencial a partir do primeiro livre

```
fd = dup(1);  
close(1);  
open(name);#a partir de agora arquivo recebe saída  
...  
close(1);  
dup(fd); #novamente stdout vai para terminal.
```

Criando um pipe

```
#define STD_IN 0
#define STD_OUT 1
Pipeline(process1, process2)
Char *process1, * process2; //nomes dos progs
{
    int fd[2];
    pipe(&fd[0]);
    if (fork() != 0){
        /* processo pai, escreve no pipe*/
        close(fd[0]); // um processo nao precisa ler do pipe
        close(STD_OUT);
        dup([fd[1]]); // agora aloquei pipe para STD_OUT
        close(fd[1]); /nao preciso mais do pipe
        execl(process1, process1, 0);
    }
    else{
        /* processo filho, le do pipe*/
        close(fd[1]); // um processo nao precisa escrever no pipe
        close(STD_IN);
        dup([fd[0]]); // agora aloquei pipe para STD_IN
        close(fd[0]); /nao preciso mais do pipe
        execl(process2, process2, 0);
    }
}
```

As chamadas de sistema do Minix: Administração de Diretórios e Sistema de arquivos

- * *LINK(name1, name2)*
 - * Faz a entrada name2 ser o mesmo inode que name1
- * *UNLINK(name)*
 - * Remove uma entrada do diretório corrente
- * *MOUNT(special, name, rwflag)*
 - * Monta um sistema de arquivos
- * *UNMOUNT(special)*
 - * Desmonta um sistema de arquivos

As chamadas de sistema do Minix: Administração de Diretórios e Sistema de arquivos

- * *SYNC()*

- * Sincroniza todos os blocos de disco da cache.

- * *CHDIR(name)*

- * Muda o diretório de trabalho

- * *CHROOT(dirname)*

- * Muda o diretório raiz (CUIDADO)

As chamadas de sistema do Minix: Proteção

- * CHMOD(nome, modo)
 - * Muda os bits de proteção do arquivo
- * UID = GETUID()
 - * Retorna o uid do processo chamador
- * GID = GETGID()
 - * Retorna o gid do processo chamador
- * SETUID(uid)
 - * Muda o uid do processo chamador
- * SETGID(gid)
 - * Muda o gid do processo chamador

As chamadas de sistema do Minix: Proteção

- * CHOWN(nome, owner, group)
 - * Muda o dono e o grupo de um arquivo
- * UMASK(complmode)
 - * Set a mascara usada para os bits de proteção

As chamadas de sistema do Minix: Gerenciamento de Tempo

- * TIME(&seconds)
 - * Retorna o tempo em segundos desde 1/1/1970
- * STIME(tp)
 - * Reinicializa o tempo desde 1/1/1970
- * UTIME(arquivo, timep)
 - * Seta o valor do ultimo acesso do arquivo
- * TIMES(bufer)
 - * Retorna o tempo de usuario e de sistema

Tabela de Processos

- * Armazena dados sobre cada processo necessários à sua execução
- * Essencial para troca de contexto
- * Informações sempre deve estar presentes
- * No minix existem cópias parciais da tabela em nos processos servidores
 - * Atualização por mensagens

Tabela de Processos

- * Armazena dados sobre cada processo necessários à sua execução
- * Essencial para troca de contexto
- * Informações sempre deve estar presentes
- * No minix existem cópias parciais da tabela em nos processos servidores
 - * Atualização por mensagens

Tabelas de processos no Minix

Kernel	Process management	File management
Registers	Ptr to text segment	UMASK mask
Program counter	Ptr to data segment	Root directory
Program status word	Ptr to bss segment	Working directory
Stack pointer	Exit status	File descriptors
Process state	Signal status	Real UID
Current scheduling priority	Process ID	Effective UID
Maximum scheduling priority	Parent process	Real GID
Scheduling ticks left	Process group	Effective GID
Quantum size	Children's CPU time	Controlling TTY
CPU time used	Real UID	Save area for read/write
Message queue ptrs	Effective UID	System call parameters
Pending signal bits	Real GID	Various flag bits
Various flag bits	Effective GID	
Process name	File info for sharing text	
	Bitmaps for signals	
	Various flag bits	
	Process name	

Processos: como funciona esquema de interrupção: funcionamento do hardware

- * Associado a cada classe de dispositivo de E/S existe uma entrada em um vetor chamado *vetor de interrupção*
 - * *Vetor contém endereços do procedimento de serviço da interrupção*
- * *Quando ocorre uma interrupção o hardware da máquina armazena algumas informações na pilha*
 - * *apontador de instruções (program counter)*
 - * *PSW*
 - * *talvez alguns registradores*
- * *CPU então vai para a entrada relativa do vetor de interrupção.*
- * *A partir daí tudo por software*

Processos: como funciona esquema de interrupção: funcionamento do software

- * Código inicial em Assembler
 - * Procedimento de serviço de interrupção armazena conteúdo dos registradores na tabela de processos (assembler)
 - * Numero do processo atual e apontador na tabela de processos mantidos em variáveis globais para acesso rápido
 - * Informação depositada pelo hardware removida da pilha e apontador de pilha redirecionado para pilha do administrador de processos (assembler)
 - * Rotina em C chamada para realizar o resto do trabalho

Processos: como funciona esquema de interrupção: funcionamento do software

- * Código em C
 - * Construção de uma mensagem para ser enviada ao processo associado ao driver (que deve estar bloqueado esperando)
 - * Mensagem indica que ocorreu interrupção para distingui-la de mensagens do servidor de arquivos
 - * Estado do *driver* alterado para *pronto*
 - * Escalonador de processos é chamado
 - * Como drivers são os processos de maior prioridade o driver respectivo deve ser chamado
 - * Se processo interrompido com mesma prioridade, driver espera
 - * Procedimento em C chamado pelo tratador de interrupção retorna
- * Código em assembler
 - * Registradores armazenados na entrada da tabela de processos relativa ao próximo a ser executada são carregados de volta
 - * Controle é retornado ao processo a ser escalonado

Exclusão mútua

Condições de Concorrência (condições de corrida)

- * Processos rotineiramente precisam se comunicar
 - * Se processo quer ler arquivo, precisa enviar requisição ao sistema de arquivos
 - * Sistema de arquivos precisa se comunicar com driver
 - * Num esquema de pipes, a saída de um processo precisa ser comunicada a outro processo
- * Em muitos sistemas operacionais, quando processos trabalham juntos, eles compartilham algum armazenamento comum onde cada um pode ler e escrever
 - * Memória ou arquivo
- * Quando processos funcionam de maneira independente, isso pode criar uma *condição de concorrência* (ou “*condição de corrida*”)
 - * Processos diferentes dependem de informação compartilhada e resultado depende do escalonamento de processos.

Exemplo: duas *threads* atualizam variável global

O que
queremos

Instrução	Efeito
Declara-se a variável local <i>i</i>	Inteiro <i>i</i> = 0
T1 lê o valor de <i>i</i> da memória e armazena no registrador1	registrador1 ← <i>i</i> :: registrador1 = 0
T1 incrementa o valor de <i>i</i> no registrador1	registrador1 ← registrador1 + 1 :: registrador1 = 1
T1 armazena o valor do registrador1 na memória	<i>i</i> ← registrador1 :: <i>i</i> = 1
T2 lê o valor de <i>i</i> na memória no registrador2	registrador2 ← <i>i</i> :: registrador2 = 1
T2 incrementa o valor de <i>i</i> no registrador2	registrador2 ← registrador2 + 1 :: registrador2 = 2
T2 armazena o valor do registrador2 na memória	<i>i</i> ← registrador2 :: <i>i</i> = 2

O que pode
acontecer

Instrução	Efeito
Declara-se a variável local <i>i</i>	Inteiro <i>i</i> = 0
T1 lê o valor de <i>i</i> da memória e armazena no registrador1	registrador1 ← <i>i</i> :: registrador1 = 0
T2 lê o valor de <i>i</i> na memória no registrador2	registrador2 ← <i>i</i> :: registrador2 = 0
T1 incrementa o valor de <i>i</i> no registrador1	registrador1 ← registrador1 + 1 :: registrador1 = 1
T2 incrementa o valor de <i>i</i> no registrador2	registrador2 ← registrador2 + 1 :: registrador2 = 1
T1 armazena o valor do registrador1 na memória	<i>i</i> ← registrador1 :: <i>i</i> = 1
T2 armazena o valor do registrador2 na memória	<i>i</i> ← registrador2 :: <i>i</i> = 1

Fonte \:wikipedia (http://pt.wikipedia.org/wiki/Condi%C3%A7%C3%A3o_de_corrida)

Outro Exemplo: spooling

- * Dois processos querem imprimir
- * Diretório de impressão e processo de impressão
- * Variáveis *out* e *in* indicam início/fim da fila
 - * Processo 1 lê valor de *in*
 - * Processo 2 lê valor de *in*
 - * Processo 1 coloca endereço do arquivo na fila
 - * Processo 2 sobrescreve endereço do arquivo na fila

Como resolver

- * Região crítica:
 - * regiões dos processos que podem gerar condições de concorrência são chamadas regiões críticas
- * 4 condições para uma boa solução
 - * Só um processo deve entrar na região crítica de cada vez
 - * Não deve ser feita nenhuma hipótese sobre a velocidade relativa dos processos
 - * Nenhum processo executando fora de sua região crítica deve bloquear outro processo
 - * Nenhum processo deve esperar um tempo arbitrariamente longo para entrar na sua região crítica (adiamento indefinido)

Solução 1: inibir interrupções

- * Hardware possui instrução específica para inibir interrupções
 - * SO usa quando código do kernel está processando
- * Solução ruim para outros processos
 - * Loops infinitos
 - * Operações inválidas
 - * Etc.

Exclusão mútua por software: tentativa 1

```
Int A=0; /*var  
global*/
```

Processo 1

```
While (A == 1){};
```

```
A = 1;
```

```
.....<região crítica>
```

```
A = 0;
```

Processo 2

```
While (A == 1){};
```

```
A = 1;
```

```
.....<região crítica>
```

```
A = 0;
```

Exclusão mútua por software: tentativa 1

```
Int A=0; /*var  
global*/
```

Processo 1

```
While (TRUE){  
  
    While (A == 1){};  
  
    A = 1;  
  
    .....<região crítica>  
  
    A = 0;  
  
}
```

Processo 2

```
While (TRUE){  
  
    while (A == 1){};  
  
    A = 1;  
  
    .....<região crítica>  
  
    A = 0;  
  
}
```

Problema? Sincronização no teste

Exclusão mútua por software: tentativa 2 (revezamento)

```
Int vez = 1; /*var global*/
```

Processo 1

```
While (TRUE){  
  
    while (vez != 1){}; /*espera*/  
  
    ....< região crítica>.....  
  
    vez = 2;  
  
}
```

Processo 2

```
While (TRUE){  
  
    while (vez != 2){}; /*espera*/  
  
    ....< região crítica>.....  
  
    vez = 1;  
  
}
```

Exclusão mútua por software: tentativa 2 (revezamento)

```
Int vez = 1; /*var global*/
```

Processo 1

```
While (TRUE){  
  
    while (vez != 1){}; /*espera*/  
  
    ....< região crítica>.....  
  
    vez = 2;  
  
}
```

Processo 2

```
While (TRUE){  
  
    while (vez != 2){}; /*espera*/  
  
    ....< região crítica>.....  
  
    vez = 2;  
  
}
```

Problema? Processo que não usa a vez bloqueia o outro

Exclusão mútua por software: tentativa 3 (revezamento)

```
Int p1dentro = 0;  
Int p2dentro = 0;
```

Processo 1

```
While (TRUE){  
  
    while (p2dentro){}; /*espera*/  
  
    p1dentro = TRUE;  
  
    ....< região crítica>.....  
  
    p1dentro = 0;  
  
    ....<resto do código>...  
  
}
```

Processo 2

```
While (TRUE){  
  
    while (p1dentro){}; /*espera*/  
  
    p2dentro = TRUE;  
  
    ....< região crítica>.....  
  
    p2dentro = 0;  
  
    ....<resto do código>...  
  
}
```

Exclusão mútua por software: tentativa 3 (revezamento)

```
Int p1dentro = 0;  
Int p2dentro = 0;
```

Processo 1

```
While (TRUE){  
  
    while (p2dentro){}; /*espera*/  
  
    p1dentro = TRUE;  
  
    .....< região crítica>.....  
  
    p1dentro = 0;  
  
    ....<resto do código>...  
  
}
```

Processo 2

```
While (TRUE){  
  
    while (p1dentro){}; /*espera*/  
  
    p2dentro = TRUE;  
  
    .....< região crítica>.....  
  
    p2dentro = 0;  
  
    ....<resto do código>...  
  
}
```

Problema? Dois testam ao mesmo tempo e entram

Exclusão mútua por software: tentativa 4 (gentileza)

```
Int p1querEntrar= 0;  
Int p2querEntrar= 0;
```

Processo 1

```
While (TRUE){  
  
    p1querEntar = TRUE;  
  
    while (p2querEntrar){}; /*espera*/  
  
    ....< região crítica>.....  
  
    p1querEntrar= FALSE;  
  
    ....<resto do código>...  
  
}
```

Processo 2

```
While (TRUE){  
  
    p2querEntrar = TRUE;  
  
    while (p1querEntrar){}; /*espera*/  
  
    ....< região crítica>.....  
  
    p2querEntrar = FALSE;  
  
    ....<resto do código>....  
  
}
```

Exclusão mútua por software: tentativa 4 (gentileza)

```
Int p1querEntrar= 0;  
Int p2querEntrar= 0;
```

Processo 1

```
While (TRUE){  
  
    p1querEntar = TRUE;  
  
    while (p2querEntrar){}; /*espera*/  
  
    ....< região crítica>.....  
  
    p1querEntrar= FALSE;  
  
    ....<resto do código>...  
  
}
```

Processo 2

```
While (TRUE){  
  
    p2querEntrar = TRUE;  
  
    while (p1querEntrar){}; /*espera*/  
  
    ....< região crítica>.....  
  
    p2querEntrar = FALSE;  
  
    ....<resto do código>....  
  
}
```

Problema? Sincronização em atualizar a variável para TRUE – todos bloqueados

Exclusão mútua por software: tentativa 5

```
Int p1querEntrar= 0;  
Int p2querEntrar= 0;
```

Processo 1

```
While (TRUE){  
    p1querEntar = TRUE;  
    while (p2querEntrar){  
        p1querEntrar = FALSE;  
        wait(random());  
        p1querEntrar = TRUE;  
    }; /*espera*/  
    .....< região crítica>.....  
    p1querEntrar= FALSE;  
    ....<resto do código>...  
}
```

Processo 2

```
While (TRUE){  
    p2querEntar = TRUE;  
    while (p1querEntrar){  
        p2querEntrar = FALSE;  
        wait(random());  
        p2querEntrar = TRUE;  
    }; /*espera*/  
    .....< região crítica>.....  
    p2querEntrar= FALSE;  
    ....<resto do código>...  
}
```

Exclusão mútua por software: tentativa 5

```
Int p1querEntrar= 0;  
Int p2querEntrar= 0;
```

Processo 1

```
While (TRUE){  
    p1querEntar = TRUE;  
    while (p2querEntrar){  
        p1querEntrar = FALSE;  
        wait(random());  
        p1querEntrar = TRUE;  
    }; /*espera*/  
    .....< região crítica>.....  
    p1querEntrar= FALSE;  
    ....<resto do código>...  
}
```

Processo 2

```
While (TRUE){  
    p2querEntar = TRUE;  
    while (p1querEntrar){  
        p2querEntrar = FALSE;  
        wait(random());  
        p2querEntrar = TRUE;  
    }; /*espera*/  
    .....< região crítica>.....  
    p2querEntrar= FALSE;  
    ....<resto do código>...  
}
```

Problema? Sincronização ruim ainda permite adiamento indefinido
(com menor probabilidade)

Exclusão mútua por software: tentativa 6: Algoritmo de Dekker

```
Int p1querEntrar= 0;  
Int p2querEntrar= 0;  
Int favorito = 1;
```

Processo 1

```
While (TRUE){  
    p1querEntrar = TRUE;  
    while (p2querEntrar){  
        if (favorito == 2){  
            p1querEntrar = FALSE;  
            while (favorito == 2) {};  
            p1querEntrar = TRUE;  
        }; /*espera*/  
        .....< região crítica>.....  
        favorito = 2;  
        p1querEntrar= FALSE;  
        ....<resto do código>...  
    }  
}
```

Processo 2

```
While (TRUE){  
    p2querEntrar = TRUE;  
    while (p1querEntrar){  
        if (favorito == 1){  
            p2querEntrar = FALSE;  
            while (favorito == 1) {};  
            p2querEntrar = TRUE;  
        }; /*espera*/  
        .....< região crítica>.....  
        favorito = 1;  
        p2querEntrar= FALSE;  
        ....<resto do código>...  
    }  
}
```

Indicação de favorito garante sincronização

Exclusão mútua por software: tentativa 7: Algoritmo de Peterson (81)

```
Int p1querEntrar= 0;  
Int p2querEntrar= 0;  
Int favorito = 1;
```

Processo 1

```
While (TRUE){  
    p1querEntar = TRUE;  
    favorito = 2;  
    while (p2querEntrar && favorito == 2){};  
    ....< região crítica>.....  
    p1querEntrar= FALSE;  
    ....<resto do código>...  
}
```

Processo 2

```
While (TRUE){  
    p2querEntar = TRUE;  
    favorito = 1;  
    while (p1querEntrar && favorito == 1)  
    {};  
    ....< região crítica>.....  
    p2querEntrar= FALSE;  
    ....<resto do código>...  
}
```

Solução mais sintética

• Exclusão mútua por Hardware: instrução Test_and_set

• Instrução especial do hardware, atômica

• Test_and_set(a , b) => {a = b; b = TRUE; }

• Int ativo = FALSE; /

PROCESSO 1

PROCESSO 2

```
While (TRUE){  
    int p1_nao_pode_entrar = TRUE;  
    while (p1_nao_pode_entrar){  
        test_and_set(p1_nao_pode_entrar,  
                    ativo);  
    };  
    ....< região crítica>.....  
    ativo = FALSE;  
    ....<resto do código>...  
}
```

```
While (TRUE){  
    int p2_nao_pode_entrar = TRUE;  
    while (p2_nao_pode_entrar){  
        test_and_set(p2_nao_pode_entrar,  
                    ativo);  
    };  
    ....< região crítica>.....  
    ativo = FALSE;  
    ....<resto do código>...  
}
```

Problema: adiamento indefinido (pouco provável)

Exclusão mútua: Semáforos (SO/compiladores)

- Novo tipo de variável
- 2 ações atômicas:
 - P(semáforo), down(semáforo)
 - Se valor > 0 subtrai 1 e continua
 - Senão trava e espera mudar em fila
 - V(semáforo), up(semáforo)
 - Se semáforo tem fila, libera o primeiro
 - Senão incrementa valor em 1
- Variação – semáforo binário
 - Valor nunca excede 1

Exclusão mútua: Semáforos (SO/compiladores)

- Facilita implementação de exclusão mútua em vários processos
- Pode ser oferecido no SO ou pelo compilador (e.g. Java)
 - Em compiladores implementado usando outro esquema disponível de exclusão mútua
- Código

```
Semáforo mutex = 1;  
While (TRUE){  
    P(mutex);  
    ...<região crítica>.....  
    V(mutex);  
    ...<região não crítica>....
```

Outro problemas com Semáforos produtores e consumidores

```
Semaforo_binario mutex; /*exclusão mútua*/
```

```
Semaforo_contador vazio= TAMBUFFER; /*controle buffer*/
```

```
Semaforo_contador cheio = 0; /*controle buffer */
```

PRODUTOR

CONSUMIDOR

```
While (TRUE){
```

```
registro item_produzido;
```

```
produz(&item_produzido);
```

```
P(vazio);
```

```
P(mutex);
```

```
coloca_item(item_produzido);
```

```
V(mutex);
```

```
V(cheio);
```

```
}
```

```
While (TRUE){
```

```
registro item_consumido;
```

```
P(cheio);
```

```
P(mutex);
```

```
pega_item(&item_consumido);
```

```
V(mutex);
```

```
V(vazio);
```

```
consome(item_consumido);
```

```
}
```

Outro problemas com Semáforos produtores e consumidores

```
Semaforo_binario mutex;
```

```
Semaforo_contador vazio= TAMBUFFER;
```

```
Semaforo_contador cheio = 0;
```

PRODUTOR

CONSUMIDOR

```
While (TRUE){
```

```
registro item_produzido;
```

```
produz(&item_produzido);
```

```
P(vazio);
```

```
P(mutex);
```

```
coloca_item(item_produzido);
```

```
V(mutex);
```

```
V(cheio);
```

```
}
```

```
While (TRUE){
```

```
registro item_consumido;
```

```
P(cheio);
```

```
P(mutex);
```

```
pega_item(&item_consumido);
```

```
V(mutex);
```

```
V(vazio);
```

```
consume(item_consumido);
```

```
}
```

CUIDADO: ordem das operações é importante (tente trocar P()'s ou V()'s)

Solução 11: Monitores (compilador)

- * Implementado pelo compilador
- * Somente 1 processo “entra” no monitor de cada vez
 - * Controle de entrada por técnicas de exclusão mútua
- * Processo pode emitir um WAIT
 - * “sai” do monitor entre em uma fila associada à variável da operação WAIT
 - * Reativado pela operação SIGNAL
- * Processo pode emitir SIGNAL logo antes de sair do monitor
 - * Quem estiver esperando na fila entra em seguida (se houver).

Solução 11: Monitores ("estilo C", originalmente em ADA)

```
Monitor ProdutorConsumidor {
```

```
    condition full, empty;
```

```
    int count;
```

```
    procedure coloca_item(item) {
```

```
        if (count == TAMBUFFER) {WAIT(full);} /* espera buffer ter espaço*/
```

```
        entra_item(item); /*altera buffer comum*/
```

```
        count + + ;
```

```
        if (count == 1) {SIGNAL(empty)}; /*avisa que tem dado*/
```

```
    }
```

```
    procedure pega_item(&item) {
```

```
        if (count == 0) { WAIT(empty);} /*espera dado*/
```

```
        remove_item(&item); /*retira do buffer*/
```

```
        count - - ;
```

```
        if (count == TAMBUFFER - 1) { SIGNAL(full)} /*buffer não está mais cheio*/
```

```
    }
```

Solução 11: Monitores ("estilo C", originalmente em ADA)

```
Monitor ProdutorConsumidor {
```

```
    condition full, empty;
```

```
    int count;
```

```
    procedure coloca_item (item) {
```

```
        if (count == TAMBUFFER) {WAIT(full);} /* espera buffer ter espaço*/
```

```
        entra_item(item); /*altera buffer comum*/
```

```
        count + + ;
```

```
        if (count == 1) {SIGNAL(empty)}; /*avisa que tem dado*/
```

```
    }
```

```
    procedure pega_item(&item) {
```

```
        if (count == 0) { WAIT(empty);} /*espera dado*/
```

```
        remove_item(&item); /*retira do buffer*/
```

```
        count - - ;
```

```
        if (count == TAMBUFFER - 1) { SIGNAL(full)} /*buffer não está mais cheio*/
```

```
    }
```


Solução 11: Monitores

PRODUTOR

```
While (TRUE){  
    registro item_produzido;  
    produz(&item_produzido);  
    coloca_item(item_produzido);  
}
```

CONSUMIDOR

```
While (TRUE){  
    registro item_consumido;  
    pega_item(&item_consumido);  
    consome(item_consumido);  
}
```

Solução 11: Monitores

- * Vantagem: exclusão mútua implementada automaticamente -> menos sujeito a erros
- * Desvantagem: necessário apoio do compilador
 - * Linguagem precisa ser estendida
- * Comunicação: da mesma maneira que semáforos, serve para memória compartilhada, não para comunicação entre processos em sistemas distribuídos

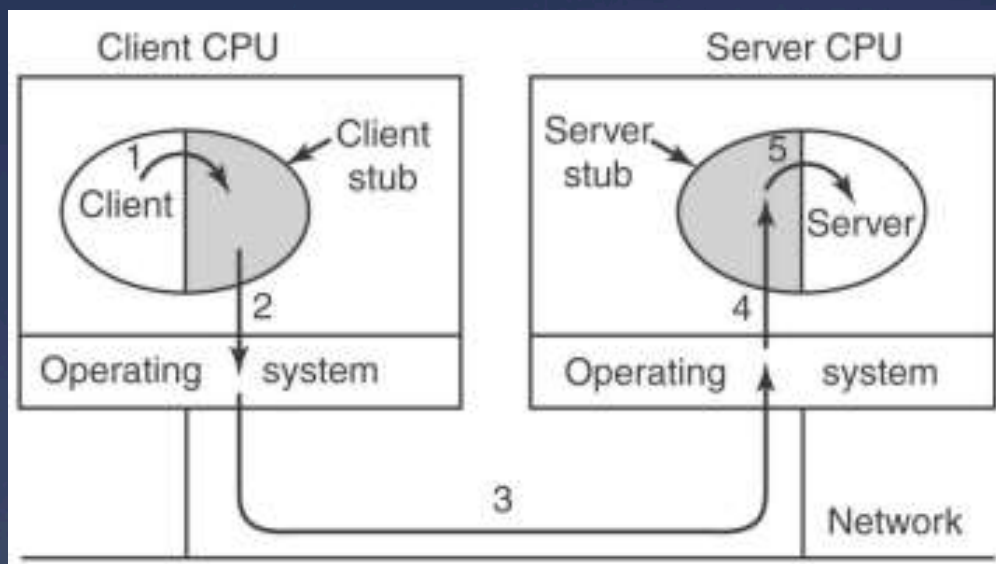
CUIDADO: ordem das operações é importante (tente trocar P()'s ou V()'s)

Comunicação entre processos: envio de mensagens

- * Semáforos, Monitores
 - * Apenas para memória compartilhada
- * Envio de mensagens
 - * `Send(destino, mensagem);`
 - * `Receive(fonte, &mensagem) /* fonte pode ser Pid ou ANY */`
- * Questões
 - * Sincrono/Assincrono
 - * Confirmação
 - * Mensagem de confirmação para certificação de entrega
 - * Re-transmissão se certificação muito demorada
 - * Mensagens duplicadas devem ser ignoradas (contar mensagens)
 - * Endereçamento
 - * `processo@maquina`, [processo@maquina.dominio](#)
 - * Autenticação: criptografia
 - * Eficiência quando em mesma máquina
 - * Mailbox (pode compartilhar) vs. bloqueio total (MINIX)

Comunicação entre processos: chamada de procedimento remota

- * Para cliente envio de mensagem parece chada de procedimento
- * SEMPRE síncrono



Fonte figura: <http://www.cs.ru.nl/~ths/a3/html/h6/h6.html>

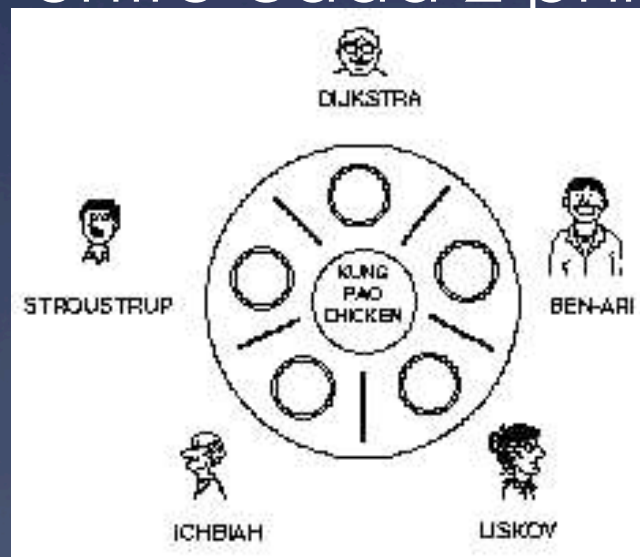
- * Envio de mensagens é transparente
 - * Processos não sabe que residem em máquinas distintas
 - * Localização de processos pode ser dinâmica

Chamada de procedimento remota problemas

- * Passagem de parâmetros por referência
- * Representação diferente de informações
 - * ponto flutuante, inteiros, ASCII
- * Comportamento em caso de falha
 - * Tentar de novo? Matar processo? Quando?
 - * At least once, at most once, maybe

Problemas clássicos de comunicação entre processos: Filósofos Comilões (dining philosophers)

- * Vários filósofos em uma mesa, pratos de macarrão chinês
- * 1 pauzinho entre cada 2 filósofos (NÃO GARFO!!!!)



Fonte: <http://www.seas.gwu.edu>

- * Precisa pegar 2 pauzinhos para comer

Filósofos Comilões

Uma não solução

```
#define N 5
Philosopher(i){
    int l;
    think();
    take_chopstick(i);
    take_chopstick((i+1) % N);
    eat();
    put_chopstick(i);
    put_chopstick(i+1);
}
```

- Problemas?

Filósofos Comilões

Uma não solução

```
#define N 5
Philosopher(i){
    int l;
    think();
    take_chopstick(i);
    take_chopstick((i+1) % N);
    eat();
    put_chopstick(i);
    put_chopstick(i+1);
}
```

- Problemas?
 - Todos pegam esquerdo
 - Todos pegam direito
 - Ninguém mais faz nada: Deadlock(!!!)

Filósofos Comilões

Uma não solução

```
#define N 5
Philosopher(i){
    int l;
    think();
    take_chopstick(i);
    while(!take_chopstick((i+1) % N){
        put_chopstick(i);
        wait(DELAY);
        take_chopstick(i);
    }
    eat();
    put_chopstick(i);
    put_chopstick(i+1);
}
```

- E se coloca de volta pauzinho?

Filósofos Comilões

Uma não solução

```
#define N 5
Philosopher(i){
    int l;
    think();
    take_chopstick(i);
    while(!take_chopstick((i+1) % N){
        put_chopstick(i);
        wait(DELAY);
        take_chopstick(i);
    }
    eat();
    put_chopstick(i);
    put_chopstick(i+1);
}
```

- E se coloca de volta pauzinho?
- Adiantamento indefinido

Filósofos Comilões

Uma não solução

```
#define N 5
Philosopher(i){
    int l;
    think();
    take_chopstick(i);
    while(!take_chopstick((i+1) % N){
        put_chopstick(i);
        wait(random);
        take_chopstick(i);
    }
    eat();
    put_chopstick(i);
    put_chopstick(i+1);
}
```

- Espera randomica?

Filósofos Comilões

Uma não solução

```
#define N 5
Philosopher(i){
    int l;
    think();
    take_chopstick(i);
    while(!take_chopstick((i+1) % N){
        put_chopstick(i);
        wait(random);
        take_chopstick(i);
    }
    eat();
    put_chopstick(i);
    put_chopstick(i+1);
}
```

- Espera randomica?
- Adiamiento menos provável mas possível

Filósofos Comilões

Uma solução

```
#define N 5
#define LEFT (i-1)%N
#define RIGHT (i+1)%N;
#define THINKING 0;
#define HUNGRY 1
#define EATING 2;
Semaphore mutex = 1;
Semaphore s[n];
Philosopher( int i){
    while (TRUE) {
        think();
        take_chopsticks((i);
        eat();
        put_chopsticks(i);
    }
}
```

Filósofos Comilões

Uma solução

```
#define N 5
#define LEFT (i-1)%N
#define RIGHT (i+1)%N;
#define THINKING 0;
#define HUNGRY 1
#define EATING 2;
Semaphore mutex = 1;
Semaphore s[n];
Philosopher( int i){
    while (TRUE) {
        think();
        take_chopsticks((i);
        eat();
        put_chopsticks(i);
    }
}
```

```
Take_chopsticks(int i){
    p(mutex);
    state[i] = HUNGRY;
    test(i);
    v(mutex);
    p(s[i]);
}
Put_chopsticks(int i){
    p(mutex);
    state[i] = THINKING;
    test(LEFT); // vizinho come?
    test(RIGHT); //vizinho come?
    v(mutex);
}
```


Filósofos Comilões

Uma solução

```
#define N 5
#define LEFT (i-1)%N
#define RIGHT (i+1)%N;
#define THINKING 0;
#define HUNGRY 1
#define EATING 2;
Semaphore mutex = 1;
Semaphore s[n];
Philosopher( int i){
    while (TRUE) {
        think();
        take_chopsticks((i);
        eat();
        put_chopsticks(i);
    }
}
```

```
Take_chopsticks(int i){
    p(mutex);
    state[i] = HUNGRY;
    test(i);
    v(mutex);
    p(s[i]);
}
Put_chopsticks(int i){
    p(mutex);
    state[i] = THINKING;
    test(LEFT); // vizinho come?
    test(RIGHT); //vizinho come?
    v(mutex);
}
```

```
Test(int i){
    if( state[i] == HUNGRY &&
        state[LEFT] != EATING &&
        state[RIGHT] != EATING &&
        ){
        state[i] = EATING;
        v(s[i];
    }
}
```

Problemas Clássicos de Comunicação: leitores e escritores

- * Modela acesso a banco de dados (ex. Sistema de reservas de aerolinha)
- * Muitos processo querem ler e escrever simultaneamente
- * Muitos podem ler ao mesmo tempo
- * Se um processo está escrevendo niguem pode ter acesso, nem leitores

Leitores e escritores

Uma solução

```
Semaphore mutex = 1;  
Semaphore db = 1;  
Int leitores = 0;
```

```
Leitor(){  
    while (TRUE){  
        p(mutex);  
        leitores ++;  
        if (leitores == 1) p(db);  
        v(mutex);  
        read_data();  
        p(mutex);  
        leitores --;  
        if (leitores == 0) v(db); //libera  
banco  
        v(mutex); //libera rc  
        use_data();  
    }  
}
```

```
Escritor() {  
    create_data();  
    p(db); //reserva banco  
    write_database();  
    v(db); //libera banco  
}
```

Equivalência dos mecanismos de exclusão mútua

- * Semáforos -> Monitores
 - * mutex para cada monitor
 - * Semáforo binário para cada variável de condição (começam com valor zero)
- * Mensagens-> semáforos
 - * 1 processo por semáforo
 - * Mensagens síncronas
 - * P => sendrec para processo controlador
 - * V=> send para processo controlador
 - * Processo controlador mantém fila de espera e responde mensagem p apenas quando fila está vazia, ou quando outro processo dá v.

Equivalência dos mecanismos de exclusão mútua

- * Monitores => semáforos
 - * 1 variável de condição por semáforo
- * Semáforos -> mensagens
 - * 1 processo por semáforo
 - * Mensagens síncronas
 - * P => sendrec para processo controlador
 - * V=> send para processo controlador
 - * Processo controlador mantém fila de espera e responde mensagem p apenas quando fila está vazia, ou quando outro processo dá v.

Escalonamento de Processos

- * Quando se tem vários processos na memória é necessária uma política para escolher próximo processo a ter o controle da CPU
- * Objetivos
 - * Justiça – todo mundo roda
 - * Eficiência – CPU utilizada 100% do tempo pelos processos
 - * Tempo de resposta – usuários interativos não devem notar latência
 - * Fluxo – maximizar o número de processos que completam (BATCH)
 - * Minimizar tempo para processos batch completarem
- * Alguns dos objetivos acima contraditórios (Quais?)
- * Problema: processos imprevisíveis.

Escalonamento de Processos

- * Praticamente todos os SOs têm interrupção periódica por relógio
- * Evita que um usuário tenha controle do sistema o tempo todo
- * NOTA: MS Windows por muito tempo não utilizou este mecanismo (multiprocessamento cooperativo)

Escalonamento de Processos:

- * Alto nível
 - * *Job scheduling* – quais tarefas concorrem pelos recursos do sistema (BATCH)
- * Nível intermediário
 - * Quais processos podem concorrer pela CPU
 - * suspende | ativa processos para controlar carga do sistema
- * Baixo nível
 - * Qual dos processos que estão prontos deve ter o controle da CPU

Escalonamento de Processos

* Preemptivo | Não preemptivo

- * CPU pode | não pode ser retirada de um processo

- * Preempção ocorre ao custo de *overhead*

 - * No mínimo estado do processo deve ser salvo

- * Preempção é necessária em sistemas onde tempo de resposta é importante (tempo real | sistemas interativos multi-usuários....)

* Interrupção de relógio

- * Instruções de hardware determinam tempo até que próxima interrupção seja gerada pelo relógio do sistema

Escalonamento de Processos: algoritmos

* Data limite

- * Tempo real
- * Processo ao iniciar determina quando sua tarefa precisa estar completa
- * Planejamento complexo: envolve cuidadosa estimativa de uso dos recursos do sistema (disco, CPU, etc.)
- * Sobrecarga de planejamento

* Fifo

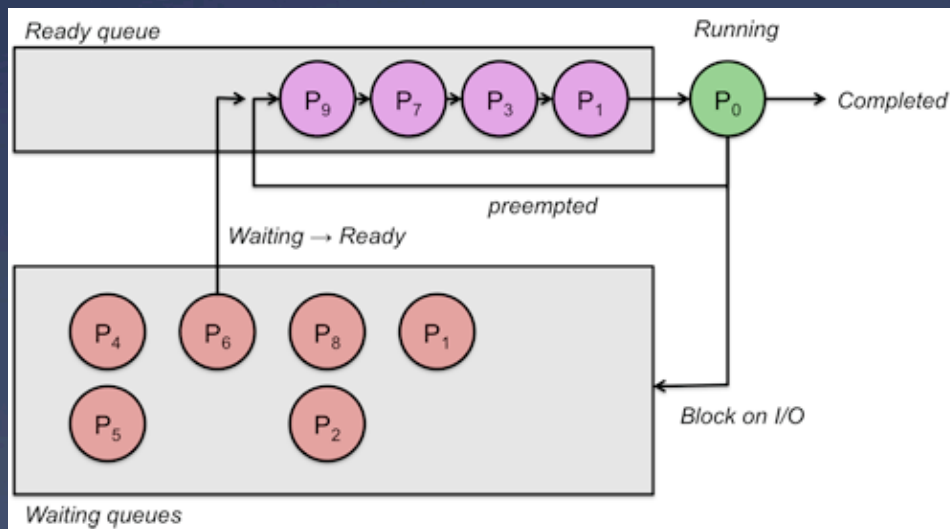
- * Mais básico
- * Razoável apenas em sistemas Batch

Escalonamento de Processos: algoritmos

- * Shortest Job First (Batch)
 - * Estimativa de tempo por processo
 - * Objetivo é aumentar fluxo de processos
- * Shortest Remaining Time First (Batch)
 - * Especialização do anterior, mas melhora justiça
- * Prioridade
 - * Número estabelece ordenação nos processos
 - * Estática vs. Dinâmica
 - * Comprada
 - * Highest response ratio
 - * $\text{Prioridade} = (\text{tempo espera} + \text{tempo serviço}) / \text{tempo serviço}$

Escalonamento de Processos: algoritmos

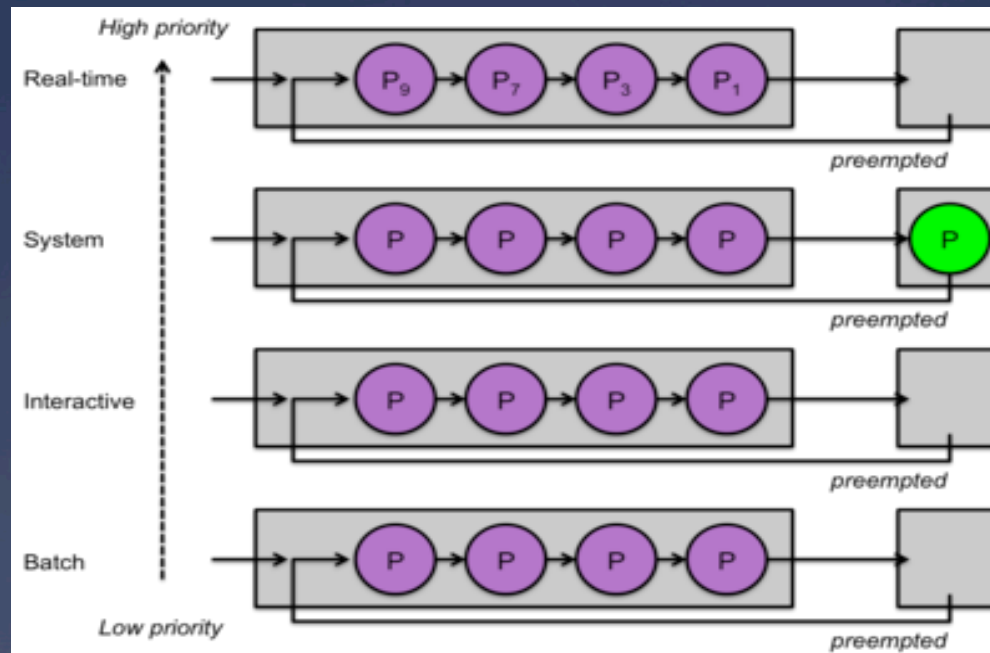
- * Round-robin
- * Fila circular
- * Revezamento estrito



Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html>

Escalonamento de Processos: algoritmos

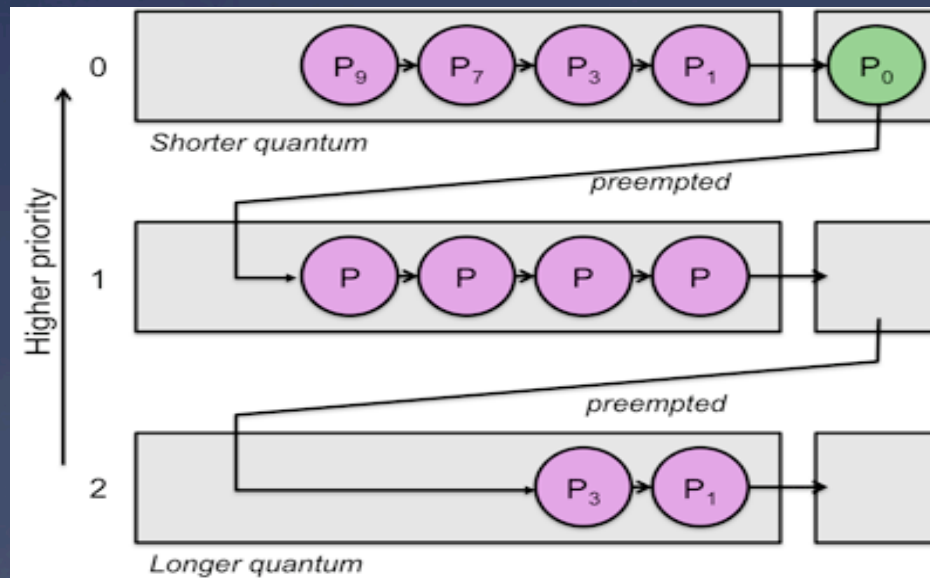
- * Multi-level queues
 - * 1 fila por classe de prioridade
 - * Round-robin em cada fila



Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html>

Escalonamento de Processos: algoritmos

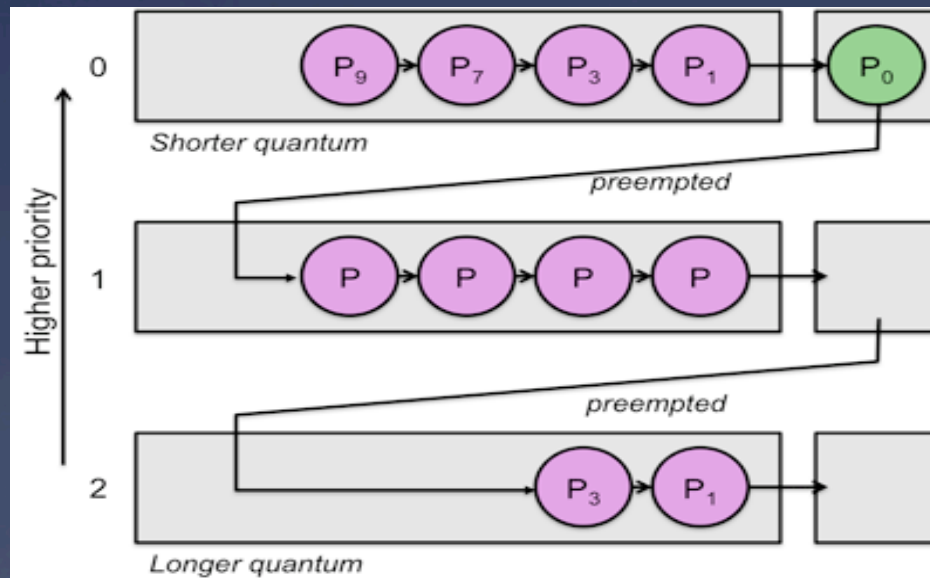
- * Multi-level FEEDBACK queues
- * Interrupção de tempo reduz prioridade
- * Promoção: tempo sem rodar, interrupção I/O



Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html>

Escalonamento de Processos: algoritmos

- * Multi-level FEEDBACK queues
- * Interrupção de tempo reduz prioridade
- * Promoção: tempo sem rodar, interrupção I/O
- * **Menor prioridade maior fatia de tempo**



Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html>

Escalonamento de Processos: algoritmos

- * Escalonamento garantido
 - * Tenta dar a cada USUÁRIO fatia igual de tempo
 - * Tempo do usuário real:
 - * Tempo desde o login / n
 - * Escolhe processo de usuário com fatia real mais distante do real
- * Pode ser usado para GRUPO de usuários
 - * VM da IBM usava para sistemas operacionais
- * Vantagens
 - * Justo com usuários/grupos de usuários
 - * Favorece usuários com poucos processos
- * Desvantagens
 - * Injusto com processos
 - * Favorece usuários com poucos processos
 - * Pode dar fatia a usuário que não vai precisar
 - * overhead

Escalonamento de Processos: algoritmos

- * Sistemas multi-processados
 - * Até agora um processador
 - * Temos também: múltiplas cpus, múltiplos cores, processadores com *hiper-threading*
 - * Mesmos algoritmos se aplicam
 - * Assume-se SMP (*simetrical multi processing*)
 - * Todos tem acesso à memória e aos mesmos recursos
 - * Cada processador com um timer, escalonador roda neste processador
 - * Comum uma fila por processador: pode reutilizar cache e TLB (veremos mais tarde)

Escalonamento de Processos: algoritmos

* Sistemas multi-processados

* Afinidade de processos

- * *Hard affinity* – sistema garante que processo roda no mesmo processador
- * *Soft Affinity* - sistema tenta na mesma CPU mas pode mudar processo para outra CPU
 - * Melhor que cpu sem nada a fazer
 - * Sistema tenta balanceamento de carga nas cpus de maneira que cada um tenha processos suficientes
- * *Push-migration* – sistema operacional verifica periodicamente carga em cada processador (numero de processos na fila) – muda se houver desbalanceamento.
- * *Pull-migration* – escalonador verifica sua fila, se estiver vazia, procura na fila de outros processadores e “rouba” processos.
- * Linux combina *pull* e *push*

Escalonamento de Processos: algoritmos

* Tempo Real

* Afinidade de processos

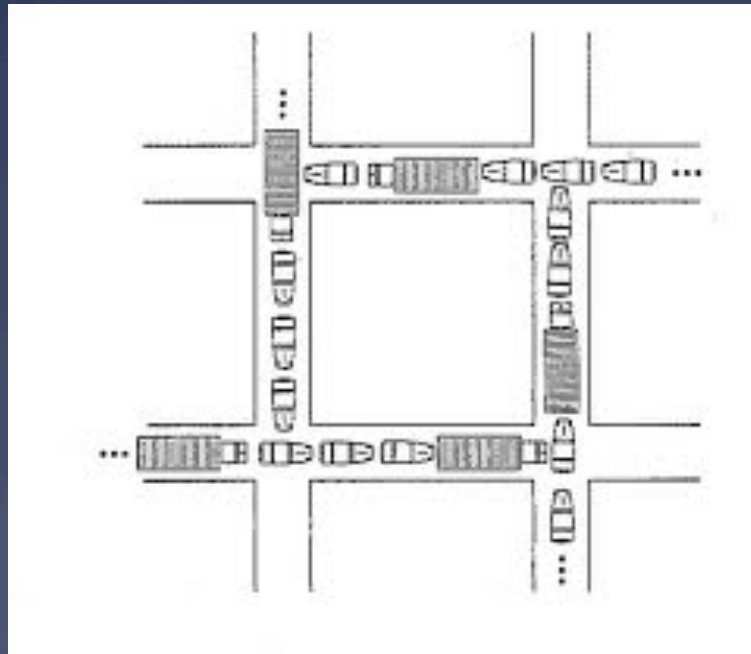
- * *Hard affinity* – sistema garante que processo roda no mesmo processador
- * *Soft Affinity* - sistema tenta na mesma CPU mas pode mudar processo para outra CPU
 - * Melhor que cpu sem nada a fazer
 - * Sistema tenta balanceamento de carga nas cpus de maneira que cada um tenha processos suficientes
- * *Push-migration* – sistema operacional verifica periodicamente carga em cada processador (numero de processos na fila) – muda se houver desbalanceamento
- * *Pull-migration* – escalonador verifica sua fila, se estiver vazia, procura na fila de outros processadores e “rouba” processos.
- * Linux combina *pull* e *push*

Deadlocks (impasse)

- * Competição por recursos (e.g. dispositivos, arquivos, plotters, canais de comunicação....)
- * Recursos muitas vezes tem uso exclusivo
- * Procedimento
 - * Pedir recurso (**espera**)
 - * Usa
 - * Libera recurso
- * Um processo em impasse está esperando por um evento que nunca irá ocorrer
- * NUNCA – diferente de adiamento indefinido (starvation)

Deadlocks (impasse): modelagem

- * Um conjunto de processos está em impasse se cada processo no conjunto está esperando por um evento que apenas outro processo pode causar



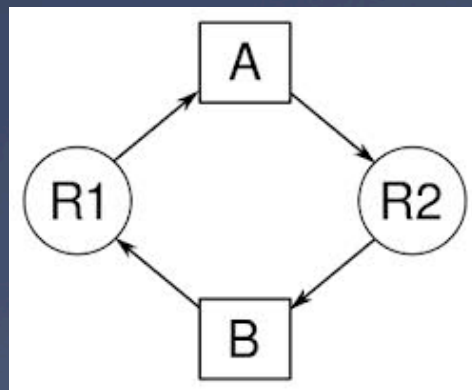
Fonte: <http://diuf.unifr.ch/courses/00-01/os/9900/ex05.html>

Deadlocks (impasse): modelagem

- * 4 condições para que impasses ocorram (Coffmann *et al*, 1971)
 1. **Exclusão mútua**: cada recurso pode apenas ser designado a um processo
 2. **Espera e segura**: processos que requisitaram recursos previamente podem requisitar novos
 3. **Não preempção**: recursos previamente designados a processos não podem ser retirados. Os processos precisam liberá-los explicitamente
 4. **Espera circular**: deve existir um conjunto de 2 ou mais processos que podem ser organizados em uma lista circular onde cada processo está esperando um recurso do processo anterior da lista.

Deadlocks (impasse): modelagem

- * Estas quatro condições podem ser modeladas utilizando grafos dirigidos
 - * 2 tipos de nós: processos (circulos) e recursos (quadrados)
 - * Aresta de recurso a processo indica alocação
 - * Aresta de processo a recurso indica requisição.
 - * Ciclo no grafo indica que há impasse



Fonte: <http://pt.wikipedia.org/wiki/Deadlock>

Deadlocks (impasse): Estratégias

- * Ignorar o problema: Algoritmo do avestruz
 - * Avalia-se que o custo de tratar o problema é muito alto
 - * Se usuário tiver problema, apenas mata o processo
 - * Abordagem Unix
- * Detecção e recuperação
- * Prevenção
- * “evitamento” dinâmico (avoidance)

Deadlocks (impasse): detecção e recuperação

- * Manter gráfico de alocação
- * Quando detectar ocorrência de ciclo, terminar um dos processos

Deadlocks (impasse): prevenção – atacar as condições necessárias (Havender)

- * Espera e segura (wait for) – programa requisita todos os recursos de uma vez
 - * Desperdício de recursos
 - * Variação – dividir programa em vários passos, com alocação de recursos por passos.
 - * Espera indefinida – recursos não disponíveis ao mesmo tempo
- * Não preempção – se não consegue um recurso libera todos
 - * Adiamento indefinido
 - * Mais difícil de detectar
- * Espera circular – numerar os recursos e alocar sempre em ordem
 - * Numeração significa que se novos recursos são adicionados programas podem ter que ser reescritos
 - * Se numeração não reflete ordem de uso dos recursos, temos desperdício
 - * Aplicativos têm que se preocupar com restrição arbitrária e dependente da instalação.

Deadlocks (impasse): evitando impasses (avoidance)



- * Algoritmo do banqueiro (Dijkstra)
 - * Metáfora de “empréstimos” e “pagamentos”
 - * Usado em recursos do mesmo tipo (extensível a vários recursos)
 - * Cada processo deve declarar uso máximo de recursos
 - * 3 números associados a cada processo: empréstimo, máximo, diferença
 - * Pelo algoritmo, recurso é concedido se o estado resultante é “seguro” ou seja todos os processo eventualmente ter suas necessidades máximas de recurso atendida
 - * Máximo de cada processo \leq máximo do sistema
 - * Supõe-se que todos os processos conseguem terminar se conseguirem o máximo de recursos.
 - * Deve existir recursos disponíveis suficientes para algum dos processos conseguir sua quantidade máxima.
 - * Liberação dos recursos deste último processo deve liberar recursos para outro processo ter sua alocação máxima, e assim sucessivamente até que todos terminem

Deadlocks (impasse): Algoritmo do Banqueiro

- * Ao conceder novos recursos (“empréstimo”), banqueiro verifica se ainda tem recursos para conceder ao processo mais próximo do seu limite máximo
- * Se tem verifica se o uso dos recursos que este processo tem possibilitaria que o segundo processo mais próximo de suas necessidades terminaria.
- * Teste é repetido, se todos os processos conseguirem terminar, estado é seguro.

Deadlocks (impasse): Algoritmo do Banqueiro

- * Estado abaixo é seguro
 - * B precisa de apenas 2 recursos e consegue terminar
 - * Em seguida, C precisa de 5 recursos (com os 4 liberados por B)
 - * Com 7 disponíveis, C consegue terminar.

	Has	Max		Has	Max		Has	Max		Has	Max		Has	Max
A	3	9	A	3	9	A	3	9	A	3	9	A	3	9
B	2	4	B	4	4	B	0	—	B	0	—	B	0	—
C	2	7	C	2	7	C	2	7	C	7	7	C	0	—
Free: 3		Free: 1		Free: 5		Free: 0		Free: 7						
(a)		(b)		(c)		(d)		(e)						

Fonte: <http://www.mfatih.com/operating-systems-informations/deadlocksdetectionsrecovery-and-avoidance.html>

Deadlocks (impasse): Algoritmo do Banqueiro para vários recursos

- * Substituímos uma matriz por 2:
 - * 1 linha por processo
 - * 1 coluna por recurso
 - * Recursos máximos (N)
 - * Recursos Alocados ao processo (A)
- * E um vetor: recursos disponíveis (D)
- * Formalmente
 1. Existe i , onde $(N_i - A_{ij}) \leq D$? (um vetor \leq que outro se todas as posições \leq)
 2. Se sim,
 - * $D = D + A_i$
 - * Retire processo i da lista
 - * volte ao passo 1.
 3. Se lista de processos está vazia, estado é seguro.

Deadlocks (impasse): Algoritmo do Banqueiro para vários recursos

Estado seguro

Banker's Algorithm for Deadlock Avoidance with multiple resource types:

	Allocation				Max				Need			
	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	3	0	1	1	5	1	1	1	2	1	0	0
P ₁	0	1	0	0	0	2	1	2	0	1	1	2
P ₂	1	1	1	0	4	2	1	0	3	1	0	0
P ₃	1	1	0	1	1	1	1	1	0	0	1	0
P ₄	0	0	0	0	2	1	1	0	2	1	1	0

Available:			
A	B	C	D
1	0	2	0

(Work)

Is the state safe????

+			
A	B	C	D
2	1	2	1

We can run P₃ to completion, then Available:

A	B	C	D
2	1	2	1

We can run P₄ to completion, then Available:

A	B	C	D
5	1	3	2

We can run P₀ to completion, then Available:

A	B	C	D
5	2	3	2

We can run P₁ to completion, then Available:

A	B	C	D
6	3	4	2

We can run P₂ to completion, then Available:

Estado inseguro

	Allocation				Request			
	A	B	C	D	A	B	C	D
P ₀	2	1	2	2	2	1	4	4
P ₁	4	0	2	1	2	1	2	2
P ₂	1	3	2	1	1	2	2	2
P ₃	1	1	1	0	2	0	0	1
P ₄	2	0	2	1	1	1	1	1

Available:			
A	B	C	D
3	0	1	1

(Work)

Is there deadlock???

+			
A	B	C	D
4	1	2	1

We can run P₃ to completion, then Available:

A	B	C	D
6	1	4	2

We can run P₄ to completion, then Available:

A	B	C	D
10	1	6	3

We can run P₁ to completion, then Available:

We cannot run P₀ (not enough D) or P₂ (not enough B), so deadlock!!!

Fonte: <http://www.cs.uni.edu/~fienu/courses/copy-of-operating-systems/lecture-notes/notes98f-12.lwp/notes98f-12.htm>

Deadlocks (impasse): Algoritmo do Banqueiro para vários recursos

- * Substituímos uma matriz por 2:
 - * Recursos máximos
 - * Recursos Alocados ao processo
- * E um vetor: recursos disponíveis
- * Estado abaixo é seguro?

	Allocation			Need			Available		
	<i>q</i> ₁	<i>q</i> ₂	<i>q</i> ₃	<i>q</i> ₁	<i>q</i> ₂	<i>q</i> ₃	<i>q</i> ₁	<i>q</i> ₂	<i>q</i> ₃
<i>p</i> ₁	0	3	1	7	2	2	3	1	3
<i>p</i> ₂	2	0	0	1	2	2			
<i>p</i> ₃	1	0	0	8	0	2			
<i>p</i> ₄	2	1	1	0	1	1			
<i>p</i> ₅	2	0	2	2	3	1			

Fonte: <http://www.mfatih.com/operating-systems-informations/deadlocksdetectionsrecovery-and-avoidance.html>

Entrada e Saída

- * Dispositivos

- * De bloco:

- * informação em blocos de tamanho fixo (128b-> 4K)
 - * Cada bloco com número sequencial
 - * Blocos podem ser acessados independentemente
 - * Discos, USB, disketes, CD, DVD

- * De caracter

- * Informação na forma de sequência de caracteres
 - * Leitura/escrita sequencial, sem retorno (streams)
 - * Terminais antigos, teclado, mouse, interface de rede, sensores...

Entrada e Saída

- * Organização
 - * Dispositivos
 - * Controladoras
 - * Software dependente de dispositivo
 - * CompC
- * Sistema Operacional lida com controlador



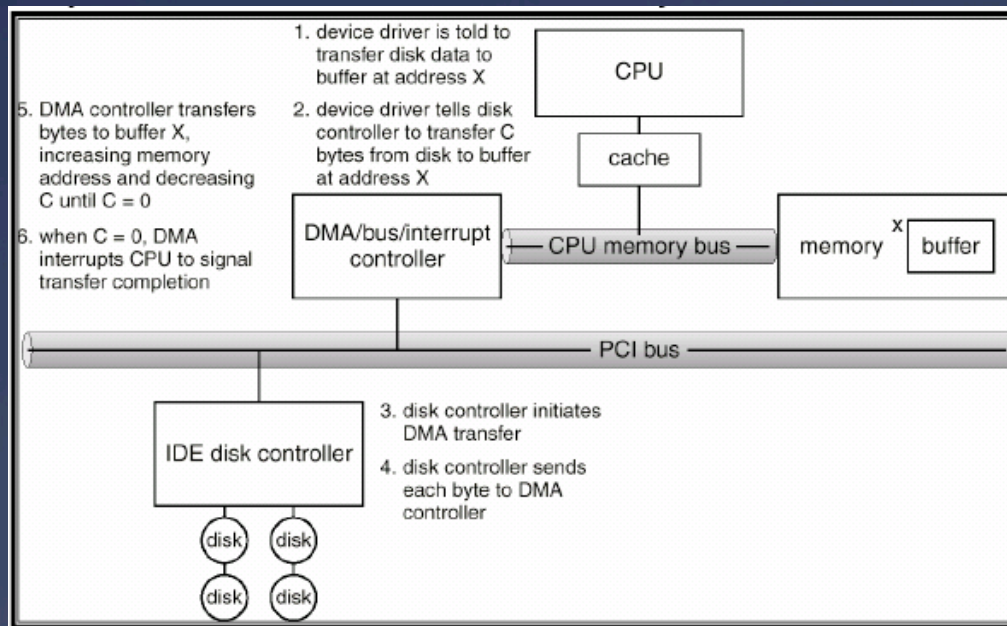
Entrada e Saída

- * Cada controlador tem alguns registradores que são endereçados pela CPU e acessados diretamente
- * Uma possibilidade é usar espaço de endereçamento especial para E/S, sendo que cada controlador é designado para um intervalo
- * SO executa entrada e saída “escrevendo” comandos nos registradores dos controladores.
- * Quando comando é aceito, CPU é usada para outro processo
- * Quando requisição é realizada, controlador gera um sinal eletrônico que causa uma interrupção
- * Sistema consulta vetor de interrupção para localizar rotina de tratamento
- * CPU faz a transferência da informação do buffer do controlador para a memória

Entrada e Saída: DMA

* Direct Memory Access

- * Informação é transferida do dispositivo para a memória sem intervenção da CPU
- * Interrupção gerada após final da transferência
- * Durante transferência, CPU utilizada para outros processos



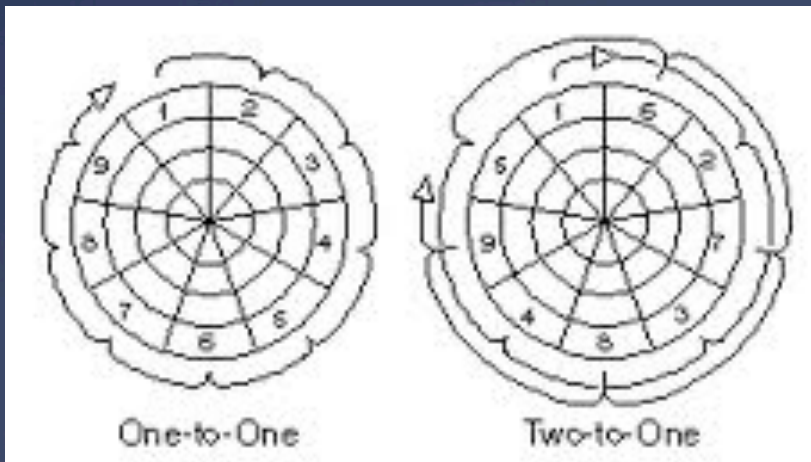
Fonte: <http://www.pling.org.uk/cs/ops.html>

* CPU fornece local do buffer e tamanho da transferência além do número do setor do disco

* Controlador tem buffer local, pois transferência do dispositivo não pode ser interrompida

Entrada e Saída: Interleaving

- * Objetivo é maximizar performance de pico
- * Setores na mesma trilha são colocados sequencialmente, mas de maneira alternada
- * Desta maneira não é necessário esperar revolução completa do disco para ler próximo setor
- * Problema poderia ser resolvido pelo SO, remapeando arquivos, mas como fixo para hardware, melhor no controlador
- * Grau ideal varia conforme velocidade relativa disco/barramento



Fonte: <http://www.webopedia.com/TERM/I/interleaving.html>

Software para Entrada e Saída

- * Objetivos
 - * Independência do tipo de dispositivo
 - * “sort < input > output” deveria funcionar sempre
 - * Uniformidade da escolha de nomes
 - * Independente do dispositivo
 - * E.g. em Unix podemos montar dispositivos de bloco em qualquer diretório
 - * Detecção de recuperação de erros
 - * Deve ser feita o mais próximo possível do hardware possível
 - * Maior parte dos erros pode ser corrigido por repetição
 - * Bloqueio vs. Interrupção
 - * No nível de usuário geralmente bloqueio – mais fácil de codificar
 - * No nível do SO geralmente interrupção , pois permite utilização dos recursos do sistema enquanto E/S é realizada

Software para Entrada e Saída

* Quatro camadas

1. Processamento de interrupção
2. Drivers de dispositivo
3. Software de E/S independente de dispositivo
4. Software do usuário

Processamento de Interrupções

- * Interrupções eliminam a ilusão de multiprocessamento
- * Objetivo: esconder interrupções o mais internamente possível ao sistema
- * Em Minix, processos são bloqueados quando comando de E/S emitido e interrupção é esperada (mesmo drivers)
- * Quando interrupção ocorre o processador de interrupção faz o necessário para desbloquear processo
 - * V() em semáforo, envio de mensagem (Minix), etc.

Drivers de dispositivos

- * Um para cada tipo de dispositivo (ou classe de dispositivos semelhantes, ex. SATA)
- * Encarregado de traduzir comandos emitidos pelo software independente de dispositivo para particularidades do controlador
 - * Início e fim de buffer, tamanho da transferência, etc.
- * Encapsula conhecimento sobre
 - * Número e função dos registradores de um controlador
 - * Particularidades na organização do dispositivo
 - * Linear, cilindros+trilhas+setores,
 - * cabeças,
 - * movimento do braço do disco,
 - * fatores de interleaving,
 - * atrasos mecânicos

Drivers de dispositivos

* Exemplo: disquete

1. Bloco N é requisitado
2. Se driver está livre, pedido é aceito, senão requisição é colocada em fila de espera
3. Requisição é traduzida em termos concretos:
 1. verificar onde estão setores correspondentes ao bloco pedido,
 2. onde está braço do disco,
 3. se motor do braço está funcionando,
 4. se disquete está girando
4. Decide comandos a serem emitidos (iniciar rotação, mover braço, etc.)
5. Emite comandos, escrevendo nos registradores do controlador (um de cada vez ou lista ligada, dependendo do controlador)
6. Driver bloqueia esperando resultado
 - * Quando operação termina sem demora (ex. Memória gráfica), driver não precisa bloquear
7. Após operação concluída, verifica erros, e retorna resultado ou trata erro

Software independente de dispositivo

- * A maior parte do software de E/S é independente de dispositivo
- * Cuidado: ainda temos CLASSES de E/S: bloco e caracter
- * Divisão exata de limites entre software independente de dispositivo e drivers depende do sistema
 - * em alguns sistemas, driver faz funções poderiam ser independentes, oferecendo visão mais abstrata

Software independente de dispositivo

* Funções básicas

1. Interface uniforme para uso dos drivers
2. Administração de nomes de dispositivos
 - * Mapeia nomes simbólicos para dispositivos reais
 - * Em Minix l-node para arquivo especial com “major device number usado para localizar driver e “minor device number” para passado para driver como parâmetro
3. Proteção dos dispositivos
 - * Como prevenir acesso indevido?
 - * Minix/UNIX usa proteção do sistema de arquivos
4. Possibilitar blocos de tamanho padrão
 - * Dispositivos diferentes podem ter tamanhos de setores distintos
 - * Camadas mais altas vêem apenas “dispositivo virtual” padrão.

Software independente de dispositivo

* Funções básicas (cont.)

1. “Buffering”

- * Bloco: Hardware lê/grava blocos grandes, usuário pode ler/gravar até um byte
- * Char: usuários podem escrever mais rápido que dispositivo pode aceitar/ teclado pode chegar antes do uso.

2. Cuidar da administração do espaço alocado em dispositivos de bloco

3. Reservar e liberar dispositivos dedicados

- * Fitas magnéticas, gravador de backup, dvd, ...

4. Relato de erros

1. Maior parte feita pelos drivers, já que erros em sua maioria são específicos para cada dispositivo
2. Porém, ex: quando bloco não podem mais ser lido?

Software no nível de usuário

- * Rotinas de E/S em Bibliotecas
 - * Printf, scanf, fprintf, etc
- * Comandos de leitura/escrita em linguagens
- * Colocam parâmetros nos locais corretos para chamadas de sistema
- * Rotinas que constroem E/S a partir de especificações
 - * Atoi, printf
- * Outros programas
 - * Spooling
 - * Spooling directory + daemon; daemon é o único com acesso à impressora
 - * Email, acesso à redes, etc.

E/S em MINIX

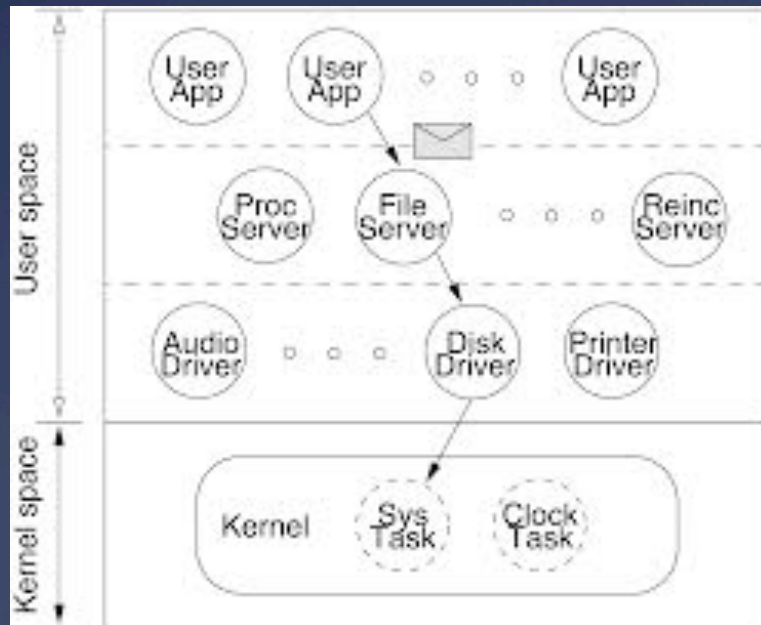
- * Interrupções
 - * Já tratado, feito no kernel
- * Drivers dos dispositivos
 - * 1 para cada classe de dispositivo de E/S
 - * Processo completos, com seu estado, reistradores, mapas de memória, etc.
 - * Comunicação inter-processos e com sistema através de mensagens
 - * Cada driver escrito em arquivo fonte separado
 - * Escrita em áreas de outros processos através de mensagens ao kernel
 - * Processos normais, apenas com privilégios diferentes de mensagens

E/S em MINIX

* Implementação diferente do Unix

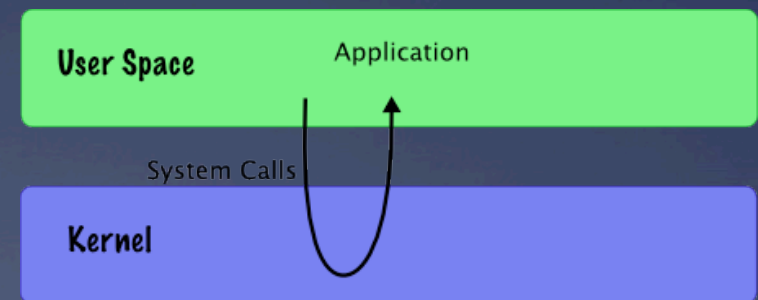
MINIX:

- * processo independente, comunicação por mensagens
- * Mais modular



UNIX

- * Duas áreas no processo user space e kernel space
- * Chamada ao sistema como chamada de procedimento
- * Causa interrupção, kernel verifica se ok, muda para kernel space
- * Mais rápido



<http://ph7spot.com/musings/introduction-to-unix-signals-and-system-calls>

Principais Drivers do Minix: RAM DISK

- * Permite tratar a memória como dispositivo de block qualquer, mas com acesso instantâneo
- * Várias áreas da memória podem ser usadas como RAM disk, cada uma com minor device number diferente
- * RAM disks podem ter intersecção

Principais Drivers do Minix: RAM DISK

- * 4 Ramdisks
 - * /dev/ram
 - * disco RAM propriamente dito, usado para salvar os arquivos mais frequentemente usados com o objetivo de acelerar seu acesso.
 - * Utilizado para verões cd/dvd.
 - * Em minix1 era usado para rodar Minix em computadores com apenas 1 floppy, colocava-se o "root" em um RAMdisk
 - * /dev/mem
 - * Usado para tornar acesso a memória padronizado como qualquer acesso a dispositivo de bloco
 - * Só pode ser usado por super-usuário
 - * /dev/kmem
 - * Semelhante ao anterior, mas tem como endereço zero o primeiro byte da memória do kernel
 - * Localização intersecta a do ramdisk anterior
 - * /dev/null
 - * Arquivo especial que aceita dados e os descarta
 - * Utilizado por usuários da shell para descartar saídas
 - * Código para manusear ramdisks é idêntico, só sua localização, fornecida pelos vetores "ram_origin" e "ram_limit" difere

Principais Drivers do Minix: Discos

* Hardware

- * Discos são organizados em cilindros, cada um contendo uma ou mais trilhas (dependendo do número de cabeças)
- * Cada trilha é dividida em setores, todos com mesmo tamanho em bytes (512 a 4096)
- * Controladores podem executar buscas (seek) em mais de 1 disco ao mesmo tempo. Alguns podem executar leitura OU escrita em um disco e busca em vários. Leitura e/ou escritas simultâneas não são feitas por limitações de tempo real
- * Demoras para início/parada do motor, movimento do braço do disco e rotação do disco são grandes em relação aos tempos de acesso a memória (nanossegundos vs. Milisegundos), e devem ser levados em conta para maximizar a performance do sistema.

Principais Drivers do Minix: Discos

- * Software: algoritmos para escalonamento de disco
 - * Otimizando-se o movimento do braço do disco pode-se melhorar significativamente a performance de sistemas com muitos processos
 - * Requisições são geralmente mantidas em uma tabela por cilindro, cada entrada em uma lista ligada.
 - * Suporemos requisições para cilindros: 3, 36, 16, 34, 9, 13, e braço no cilindro 14.

Algoritmos de escalonamento

- * FIFO
 - * Dispensa explicações
 - * No exemplo: 3, 36, 16, 34, 9, 13 (111 trilhas percorridas)
- * Menor tempo de busca Primeiro (SSTF – shortest seek time first)
 - * Próxima requisição é aquela que envolve menor movimento do braço
 - * No exemplo,: 13, 16, 9, 3, 34, 36 (53 trilhas percorridas)
 - * Melhor fluxo (throughput)
 - * Espera indefinidamente longa é provável, conjunto pequeno de processos pode monopolizar discos

Algoritmos de escalonamento (cont.)

- * Algoritmo do elevador
 - * Próxima requisição a ser tratada é aquela que envolve menor movimento do braço, na mesma direção do último movimento, senão apenas o mais próximo
 - * No exemplo: 13, 9, 3, 16, 34, 36 (44)
 - * Favorece setores internos do disco
- * Elevador unidirecional:
 - * usa apenas uma direção para atender os pedidos
 - * Supondo de fora para dentro: 13, 9, 3, 36, 34, 16 (64)
 - * Elimina favorecimento de trilhas
 - * Aceleração do braço tem demora, voltar direto ao início mais rápido que percorrer soma de trilhas
- * Variação: após início do movimento, novas requisições entram em lista separada
 - * Elimina possibilidade de adiamento indefinido

Algoritmos de escalonamento (cont.)

- * Escalonamento de setores dentro de trilha : ordenação otimiza atrasos de rotação desnecessários
- * Buffer de trilha: tempo de busca para nova trilha maior que ler trilha completa
 - * Alguns drivers guardam trilha inteira para otimizar acessos consecutivos na mesma trilha.
 - * As vezes feita pelo controlador

Erros

- * Erro de programação (# de setor inválido) – abortar ou avisar
- * Checksum error” transiente (poeira?): repetir
- * Checksum error” permanente (bloco com dano físico) – marcar bloco
 - * Problema: backup pode ser por bloco
 - * Arquivos com blocos inválidos (sistemas de arquivo)
 - * Trilhas reserva, blocos inválidos substituídos por outros reserva automaticamente
- * Erro de busca: (braço foi para setor errado): alguns controladores resolvem, senão driver
- * Erro de controlador: “botão de reset”, invocado pelo driver

Minix Floppy: um exemplo

- * Floppy:
 - * dispositivo removível barato comum até os anos 90
 - * Disco de material flexível
 - * Alto desgaste, baixa velocidade
- * Pouca probabilidade de concorrência no floppy
 - * Blocos de 1k, setores de 512b: Blocos grandes minimizam acesso ao disco, mas são aposta
 - * FIFO: pouca probabilidade de concorrência no acesso ao floppy

Minix Floppy: um exemplo

- * Do_rdwt (rotina do driver para leitura e gravação)
 - * Dma-setup: inicializa dados do chip DMA para que esta transfira diretamente dados à memória
 - * Start_motor: verifica se motor está funcionando, senão emite comando para ativá-lo
 - * Seek: verifica se braço está no lugar certo, senão instrui controlador e dá um “receive”
 - * Transfer: emite comando de leitura/escrita e em seguida dá um receive, após o qual verifica registradores do controlador . Se erro, retorna código de erro.
 - * Clock_mess: manda mensagem ao clock driver para acionar timer que chama procedimento para desligar o motor em 3 segundos (caso haja novo acesso em breve, motor já está ligado, senão, desligamento evita desgaste excessivo
 - * Stop_motor: procedimento para parar motor.

Drivers: Relógio

- * Hardware
 - * Dois tipos:
 - * Ligado à rede elétrica (gera interrupção em ciclos de 1/60 de segundo)
 - * Cristal (oscilador de cristal + contador + registrador com constante): contador é inicializado e decrementado a cada sinal periódico do cristal (de 1 a mais de 20Mhz), ao zerar controlador gera interrupção.
 - * Relógios programáveis:
 - * Dois modos: deve ser reiniciado a cada interrupção (one-shot mode) e reinicialização automática (square-wave mode)
 - * Pode-se escolher periodicidade da interrupção variando-se constante armazenada no registrador
- * Hardware somente gera interrupções a intervalos predefinidos

Drivers de Relógio: funções

- * Manter hora correta
 - * 64 bits
 - * Tempo em segundos +tics
 - * Hora do boot em segundos + tics (32 bits)
- * Não deixar processo rodarem por mais que sua fatia de tempo
 - * Quando processo começa a rodar, relógio é iniciado pelo valor do quantum, quando interrupção é gerada, escalonador é chamado
- * Contabilizar uso de CPU
 - * Campo na tabela de processos contabilizado por tics
 - * Atualização feita a cada interrupção (não a cada tic- muito caro)

Drivers de Relógio: funções

- * Implementar ALARMES (chamada ao sistema por processo de usuário)
 - * Alarme geralmente é um sinal
 - * Pode-se usar lista ligada com tempos dos alarmes, com diferença para próximo alarme e contador de tics até próximo alarme; quando hora do dia atualizada, driver verifica se algum timer venceu
- * Fornecer alarmes especiais para o sistema
 - * Watchdog timers; semelhante aos sinais para usuário, mas driver chama procedimento estabelecido quando se programou alarme (e.g. Stop_motor no floppy Minx)
- * Coleta de estatísticas em geral
 - * Profiling: a cada tic, driver verifica se processo está sendo “profiled” e atualiza contador de tics do “bin” correspondente ao PC

Drivers de Relógio: MINIX

- * 3 tipos de mensagem
 - * SET_ALARM
 - * Numero do processo, procedimento a ser chamado, espera
 - * Para usuário chamada de sistema através de processo servidor
 - * Cada processo pode ter apenas um alarme, novo alarme cancela o anterior.
 - * GET_TIME
 - * Retorna segundos desde 1/1/1970
 - * SET_TIME
 - * Apenas pelo super-usuário (Porque?)
 - * CLOCK_TICK
 - * Mensagem enviada ao driver quando interrupção de relógio ocorre
 - * Driver atualiza contadores, verifica alarmes e vê se fatia de tempo venceu

Drivers de Terminal: Hardware

RS 232

- * terminais com tela e teclado que se comunicam por interface serial, 1 bit de cada vez, em geral 10 bits de 25 pinos: 1 para transmitir, 1 para receber e 23 para funções de controle (300-9600bps)
- * 10 bits para transmitir 1 bit de informação: codificação mais comum usa 1 bit de início, 7-8 bits de dados com 1 bit de paridade, 1 ou 2 bits de parada.
- * Gradualmente substituídos por USB nos micros pessoais mas ainda muito usados em caixas registradoras, leitoras de cod. de barra, e dispositivos de controle remoto industrial
- * Hardcopy (impressoras), glass TTY (terminais burros), Terminal inteligente (memória+CPU, entende comandos após sequência de escape), blit (terminal inteligente gráfico serial)
- * Barramento/UART (Universal Assynchronous Receiver/Transmitter)
 - * Driver escreve um caracter por vez que UART transmite.

Drivers de Terminal: Hardware Mapeados na memória

- * memória especial (vídeo RAM), que é parte do espaço de endereçamento – controlador de vídeo
- * Parte integral do computador
- * Controlador do vídeo, no mesmo cartão que a VRAM, lê VRAM e manda sinal de vídeo para monitor
- * “Character mapped display” (IBM-PC original)
 - * cada caracter ocupava 2 bytes, um com dado outro com atributos (cor, reverso, piscar, etc).
 - * Tela com 25x80 caracteres
 - * Caracter aparece na tela no próximo ciclo do monitor (50hz, etc.)
 - * Muito mais rápido que RS232: 12mseg vs 2083mseg
- * Bit Mapped –
 - * Cada pixel no terminal controlado por um ou mais bits na memória de vídeo
 - * Extremamente rápido
 - * Possibilita gráficos complexos
 - * Padrão para terminais gráficos atuais
- * Teclado é separado do vídeo – para estes interface muitas vezes é paralelea, mas existem terminais RS232.
 - * Teclado fornece # da tecla e se foi pressionado ou liberado (permite compor teclas)

Drivers de Terminal: tarefas de teclado

- * Coletar caracteres
- * Converter # da tecla para código (ASCII, etc.)
- * Bufferização (usuário ainda não está esperando entrada)
 - * “pool” de buffers : alocados dinamicamente e conforme necessidade
 - * Buffer direto no registro do terminal – driver mais simples
- * Echo (imprimir na tela o que foi digitado)
 - * Echo em hardware é ruim (modo senha.....)
 - * Pode ser complicado se usuário digita quando programa está “imprimindo”
 - * Determinar ação quando mais de 80 caracteres (wrapping?)
 - * TABS (cálculo envolve saber onde está cursor em terminais mapeados)
 - * Conversão de “return” em “line feed”
 - * “cooked mode” tem caracteres com significado especial (^d, etc.) -> deve traduzir para série de comandos para echo correto
 - * Caracter de escape
 - * DEL, BREAK, RUBOUT
 - * OBS: editores sofisticados usam modo “Raw” e decodificam sequências de caracteres no nível de usuário.

Drivers de Terminal: tarefas de tela

- * Mais simples que teclado
- * RS232 diferente de Mapeado em Memória
- * RS232 –
 - * buffers de saída diferentes para cada terminal
 - * Após toda a saída ser copiada para buffer, character é enviado e driver bloqueia até completar transmissão
- * Mapeado em Memória - Character
 - * Caracteres extraídos 1 de cada vez do “espaço de usuário” e copiados para a VRAM
 - * Exige processamento de caracteres tipo backspace, cr, etc.
 - * CR no fim da tela envolve “scrolling” – pode envolver cópia ou ajuda do terminal (que pode ter apontador para “início”)
 - * Posicionamento do cursor

O System Task do Minix

- * Processo rodando no espaço de endereçamento do kernel (Juntamente com Clock Task)
- * (Até o Minix 2, todos os drivers eram *tasks*)
- * Recebe todas as chamadas ao Kernel
 - * Divisão em camadas do Minix impede comunicação dos servidores com Kernel, porém existem tarefas que devem ser relegadas a este (ex. Criação de processo com chamada `fork`)
 - * Recebe também chamadas dos Drivers

O System Task do Minix

- * Mensagens tratadas
 - * SYS_FORK – ProcessManager informa sistema que novo processo deve ser criado.. Mensagem contém índices onde pai e filho devem estar na tabela (da qual PM e FS tem sua própria cópia) .
 - * Procedimento *do_fork* copia entrada do processo pai no processo filho e zera contabilidade do filho
 - * SYS_NEWMAP – após criar nova entrada na tabela, PM aloca memória para processo filho. Com esta mensagem, MM informa o Kernel sobre novo mapa de memória para processo filho. Passado um poteiro para o mapa. Informação também usada para preencher *p_regs* contendo os “segment registers”
 - * SYS_EXEC – Chamada pelo PM quando o processo invoca *exec()*. Cria nova pilha par ao processo e aloca argumentos e ambiente nela. SYS_EXEC recebe apontador para nova pilha e informa kernel, além de zerar alarmes
 - * SYS_XIT – um processo termina quando emite a chamada EXIT ou quanto recebe um sinal não tratado. Nets caos, SYS_XIT é chamado. *do-xit* elimina processo de suas filas e envia dados de contabilização.

O System Task do Minix

* Mensagens tratadas

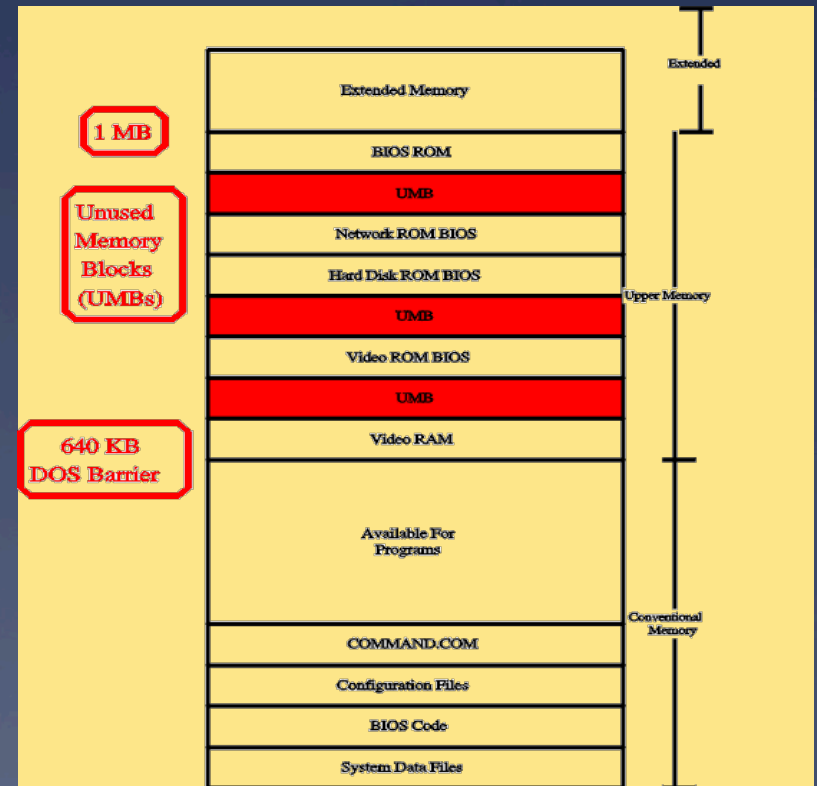
- * SYS_GETSP – usado pelo PM quando chamadas BRK e SBRK são emitidas, para ver se segmento de dados e de pilha se cruzam.
- * SYS_TIMES – usado por PM para implentar a chamada times.
- * SYS_ABORT – chamada por processos servidores (PM, etc.) quando erro crítico é encontrado. do_abort chama “panic”
- * SYS_SIG – cuida da maior parte das tarefas ligadas ao processamento de sinais, que apenas Kernel pode executar (empilhar PSW, CS register, # do sinal na pilha do processo sinalizado). do-sig chama build_sig e empilha resultado
- * SYS-COPY - usado por drivers e PM para copiar mensagens de e para processos

Administração de Memória

- * Quando?
 - * Sempre que existe um Sistema Operacional
- * Hierarquia
 - * CACHE \Leftrightarrow MEM. PRINCIPAL \Leftrightarrow MEM. SECUNDÁRIA
 - * Princípio da localidade central na idéia de Cache
- * Tipos de administração de memória:
 - * Escolha (fetch) – quando escolher próximo pedaço a ser transferido para a memória
 - * “demand”- só quando trecho de memória é requisitado,
 - * Antecipatória – sistemas de alta performance
 - * Colocação – onde colocar um programa/trecho na memória
 - * Reposição (“replacement”) – o que retirar da memória para livrar espaço

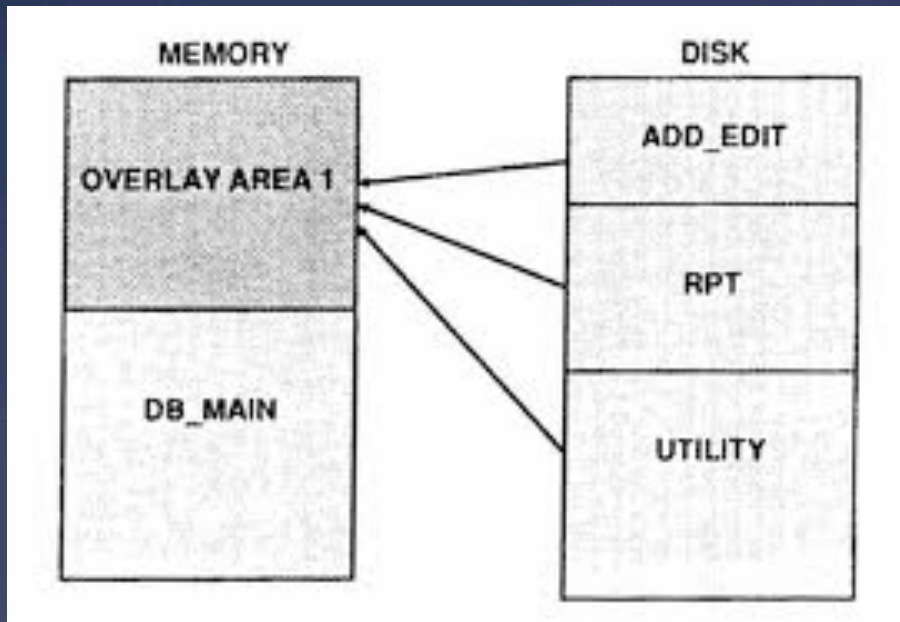
Memória Real

- * Monoprogramação sem “swapping”
- * Só 1 programa + SO na memória de cada vez.
- * Quando processo termina (e só então) - novo processo carregado em seu lugar



Memória Real

- * Programas muito grandes: OVERLAY
 - * programador define que parte programa é carregado de cada vez
 - * Area fixa e área de troca de memória
 - * Comandos para substituir partes da memória



Fonte: http://homepage.smc.edu/morgan_david/cs40/cs40.htm

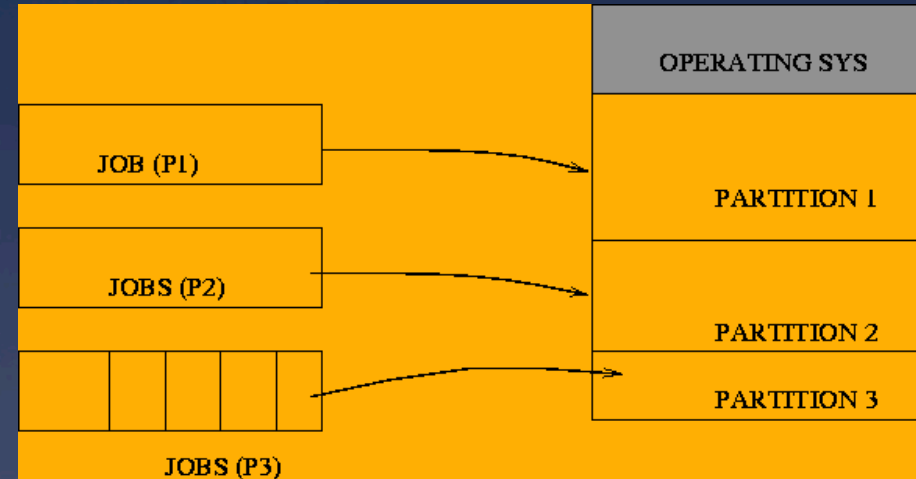
Memória Real

- * Multiprogramação
 - * Computadores usavam em média apenas 20% da CPU durante execução dos programas
 - * Computadores extremamente caros: necessário maximizar uso
 - * Mais de um programa na memória de cada vez permite usar CPU quando outro programa está aguardando ação de E/S
 - * Como fazer com memória?

Fonte: http://homepage.smc.edu/morgan_david/cs40/cs40.htm

Memória Real

- * Particionamento de memória
 - * Memória dividida em várias partes
 - * Programa colocado em uma das partições
- * Partições fixas
 - * Partições a memória pré-definidas na instalação
 - * Proteção por “boundary registers” (no IBM 360 código de acesso na PSW)
 - * 1 fila de “jobs” para cada partição
 - * Problemas: **Fragmentação Interna** (uso parcial da memória da partição) , partições podem ficar ociosas muito tempo.



Fonte: <http://users.cs.cf.ac.uk/O.F.Rana/os/lectureos8/node3.html>

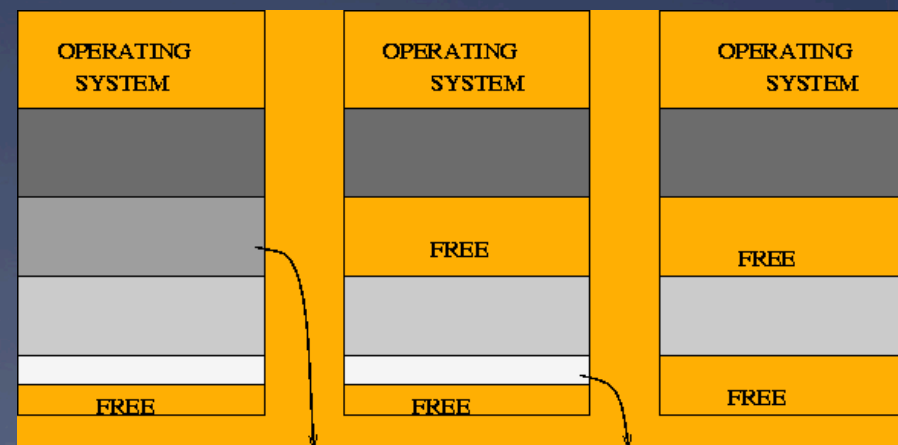
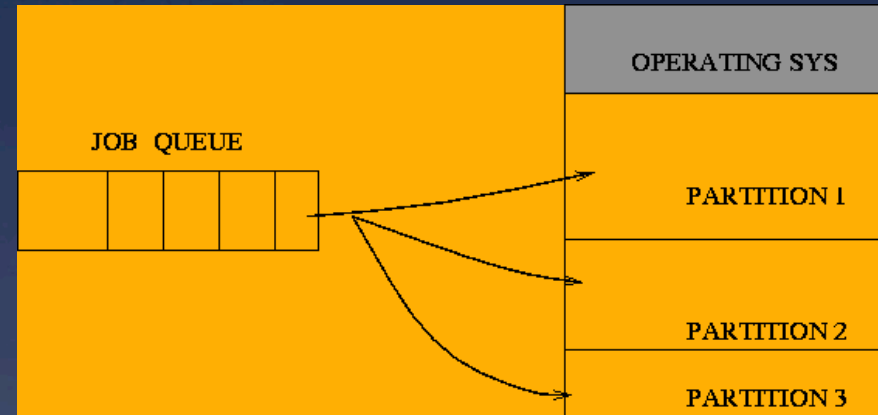
Fonte: http://homepage.smc.edu/morgan_david/cs40/cs40.htm

Memória Real: partições fixas

- * Tradução e “loading” absolutos:
 - * Compilador informado sobre qual será partição utilizada e gera diretamente endereços absolutos dentro da partição.
 - * Fila de jobs por partição
- * Tradução e “loading” relocáveis
 - * Compilador compila programa de maneira que ele possa rodar em qualquer partição
 - * Mantêm lista de endereços absolutos e os atualiza carregar programa
 - * Endereço relativo a um registrador-base (possível mover programa na memória)
 - * Uma fila de jobs, processo roda onde couber
 - * Nota: difícil em C.

Memória Real: Partições variáveis

- * Sem limites fixos
- * Programa ocupam apenas espaço previsto
- * Necessária “lista livre” para pedaços disponíveis
- * Problema: **Fragmentação Externa** – buracos pequenos demais para serem aproveitados



Fonte: <http://users.cs.cf.ac.uk/O.F.Rana/os/lectureos8/node4.html>

Memória Real: Partições variáveis

- * Sem limites fixos
- * Programa ocupam apenas espaço previsto
- * Necessária “lista livre” para pedaços disponíveis
- * Problema: **Fragmentação Externa** – buracos pequenos demais para serem aproveitados. Estratégias
 - * “coalescing holes” – buracos adjacentes se juntam
 - * Compactação – mantem espaço livre contíguo

QUAIS?

Fonte:<http://users.cs.cf.ac.uk/O.F.Rana/os/lectureos8/node4.html>

Memória Real: Partições variáveis

- * Sem limites fixos
- * Programa ocupam apenas espaço previsto
- * Necessária “lista livre” para pedaços disponíveis
- * Problema: **Fragmentação Externa** – buracos pequenos demais para serem aproveitados. Estratégias
 - * “coalescing holes” – buracos adjacentes se juntam
 - * Compactação – mantem espaço livre contíguo
 - * Tempo de compactação é improdutivo (overhead)
 - * Sistema precisa parar (ruim para tempo real e interativo)
 - * Como envolve relocar programas, pode ser necessário manter tabelas com info de relocação.

Fonte:<http://users.cs.cf.ac.uk/O.F.Rana/os/lectureos8/node4.html>

Memória Real: Partições variáveis

* Estratégias de colocação

1. First-fit , next fit – rápido e simples
2. Best-fit – quanto menos desperdício de memória melhor
3. Worst-fit – melhor deixar buracos maiores, estes tem maior chance de serem usados.

Memória Real: Partições variáveis

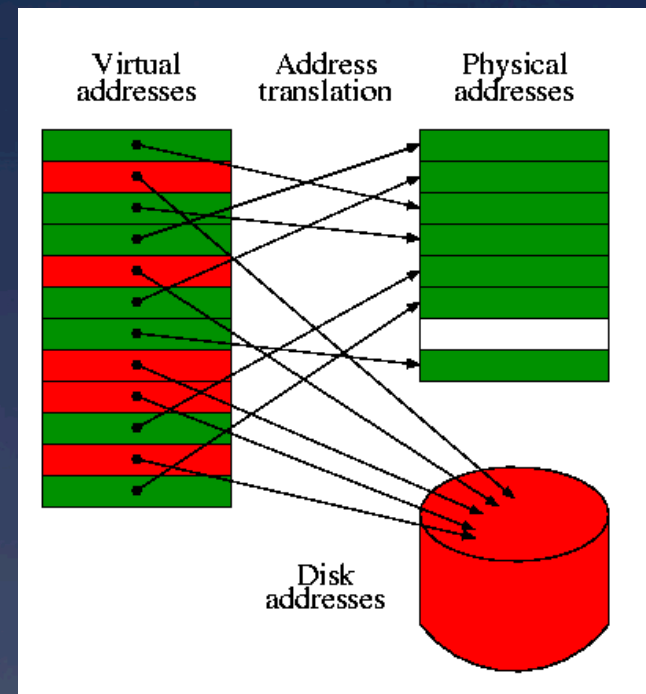
- * Implementação
 - * Bitmap –
 - * 1 bit para cada região de memória:
 - * encontra “zero runs” ao alocar
 - * Operação muito lenta
 - * Lista ligada
 - * Mais rápido que anterior
 - * Uma ou duas listas (partições usadas e livres)
 - * “Buddy system”
 - * Tamanhos pré definidos como potências de 2 (2, 4, 8, etc.)
 - * 1 lista para cada tamanho (32 listas em memória de 4GB, MUITO menos na época)
 - * Quando espaço é liberado, vê se tem contíguo livre, daí junta, e repete o processo: busca em apenas uma lista
 - * RÁPIDO
 - * Fragmentação interna (média 25%, potencial quase 50%)
- * Sem “lanche de graça”

Memória Real: Swapping

- * Possível com ambos os esquemas anteriores (partição fixa e variável)
- * Programa pode ser retirado da memória para otimização de recursos
- * Programas retirados quando:
 - * Inativo e quero rodar outro (quando é inativo?)
 - * Programa precisa de mais memória – programa precisa trocar de partição
- * Precisa de “swap space” na memória secundária

Memória virtual

- * Endereços “virtuais” que podem não ser o mesmo do endereço real
- * Necessário mecanismo de tradução eficiente sob preção de perder vantagem de reuso da CPU: apoio de hardware
- * Habilidade de endereçar espaço maior que memória
- * Como memória virtual pode ser maior que real, necessário armazenar trechos não utilizados
- * Primeira implementação: Sistema Atlas (Univ. Manchester, 1960)
- * Endereços contínuos na memória virtual não são necessariamente contínuos na memória real

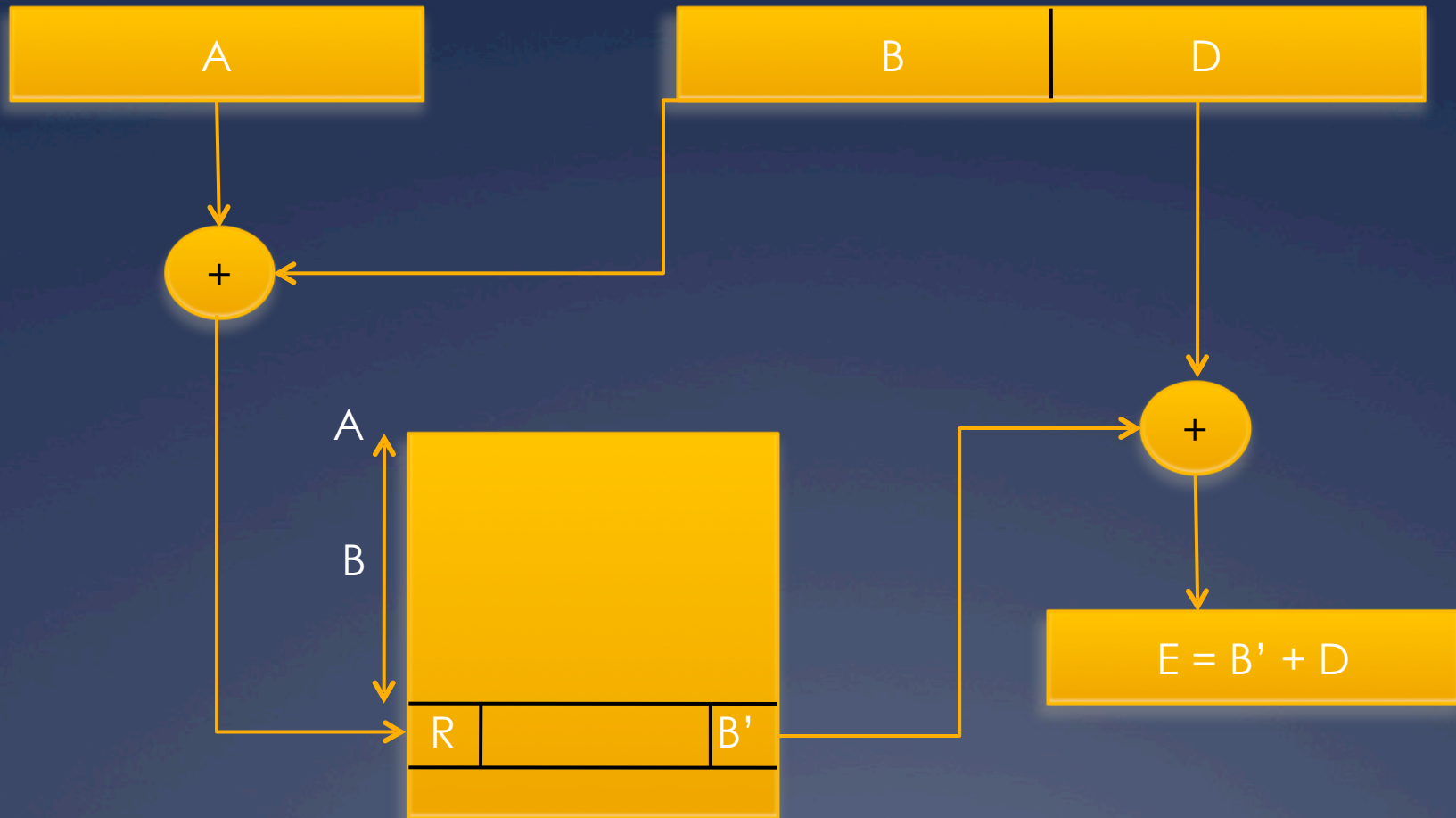


Fonte: <http://www.broken Thorn.com/Resources/OSDev18.html>

Memória virtual

- * Simplifica codificação de programas e compilação do programa
- * Memória organizada sempre em blocos
 - * Overhead e manipulação de 1 palavra de cada vez é excessivo
 - * Blocos pequenos envolvem maior overhead
 - * Blocos grandes, demoram mais para transferir, cabem em menor número na memória real.
 - * PÁGINAS: blocos com tamanho fixo
 - * SEGMENTOS: blocos com tamanho variável
- * Funcionamento
 - * Cada endereço tem 2 partes: base e deslocamento
 - * Tabela de mapeamento de endereços (por processo? Por memória?)
 - * Registrador especial indica início da tabela de tradução.

Memória virtual: endereçamento



Memória virtual: funcionamento

- * Funcionamento básico
 - * Hardware do computador verifica tabela de indexação
 - * Se página está na memória, acessa o endereço real
 - * Se página não está na memória, lê seu conteúdo da memória secundária (e.g. disco)
 - * Caso memória esteja cheia escolhe bloco para ser removido temporariamente
 - * Eventualmente remoção temporária pode envolver escrita no disco
- * Campos da tabela de indexação
 - * Endereço na memória real do início do bloco
 - * Endereço na memória secundária do bloco
 - * Usado para buscar páginas ausentes e para gravar páginas que são retiradas da memória
 - * Dirty bit
 - * Define quando página deve ser escrita na memória secundária
 - * Residence bit
 - * Define se precisamos buscar a página na memória secundária
 - * Reference bit
 - * Informa se a página foi acessada após a primeira carga
 - * Proteção
 - * Usado para verificar se acesso é válido

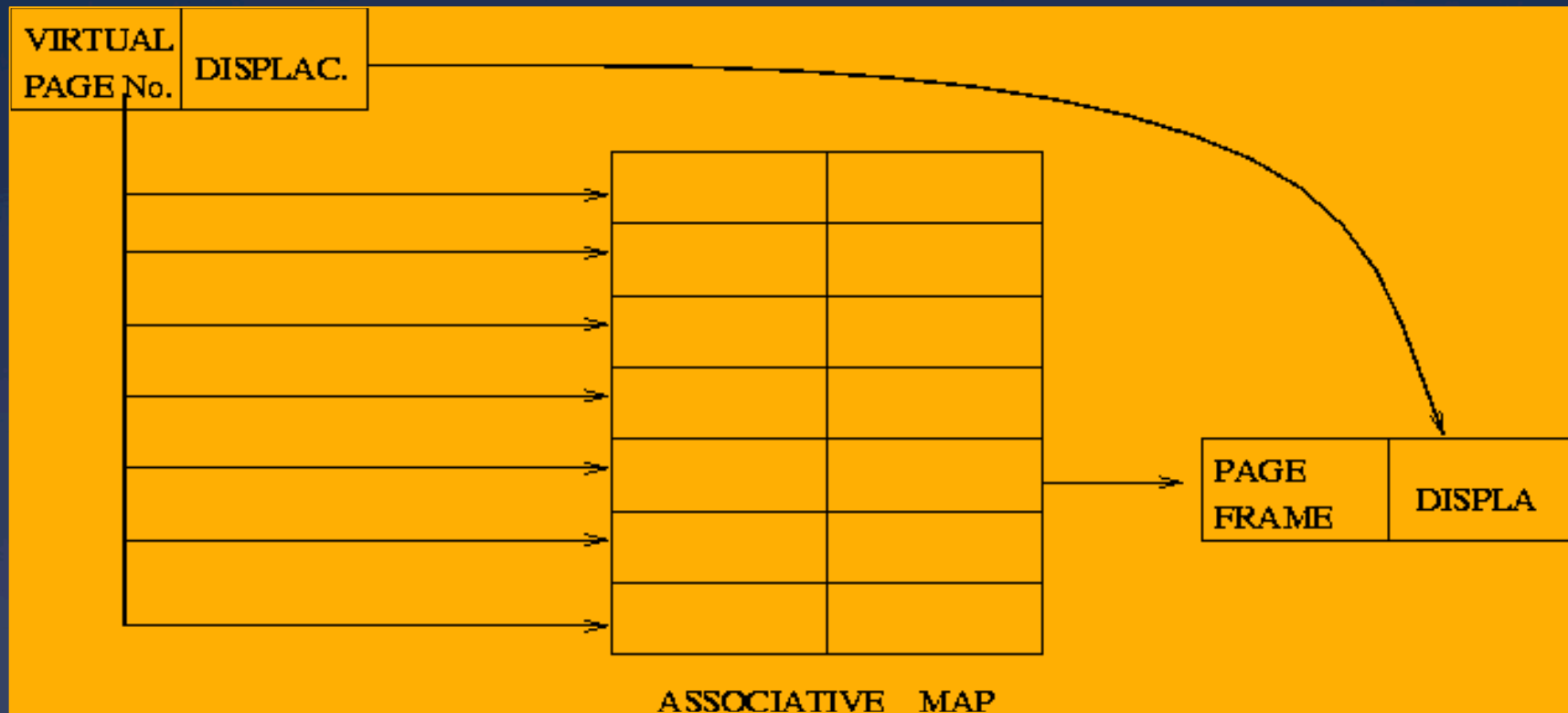
Paginação

- * Blocos de tamanho fixo chamados “páginas”
- * Páginas da memória alinhadas em endereços múltiplos do tamanho ds páginas
 - * Para obter endereço real basta concatenar número da pagina real com deslocamento.
 - * Entrada na tabela de mapeamento:
 - * Residence bit
 - * Endereço da página na memória secundária
 - * Número da página na memória real
- * Problema: fragmentação interna
 - * Media $\frac{1}{2}$ página de desperdício de memória (não tanto)

Paginação: técnicas

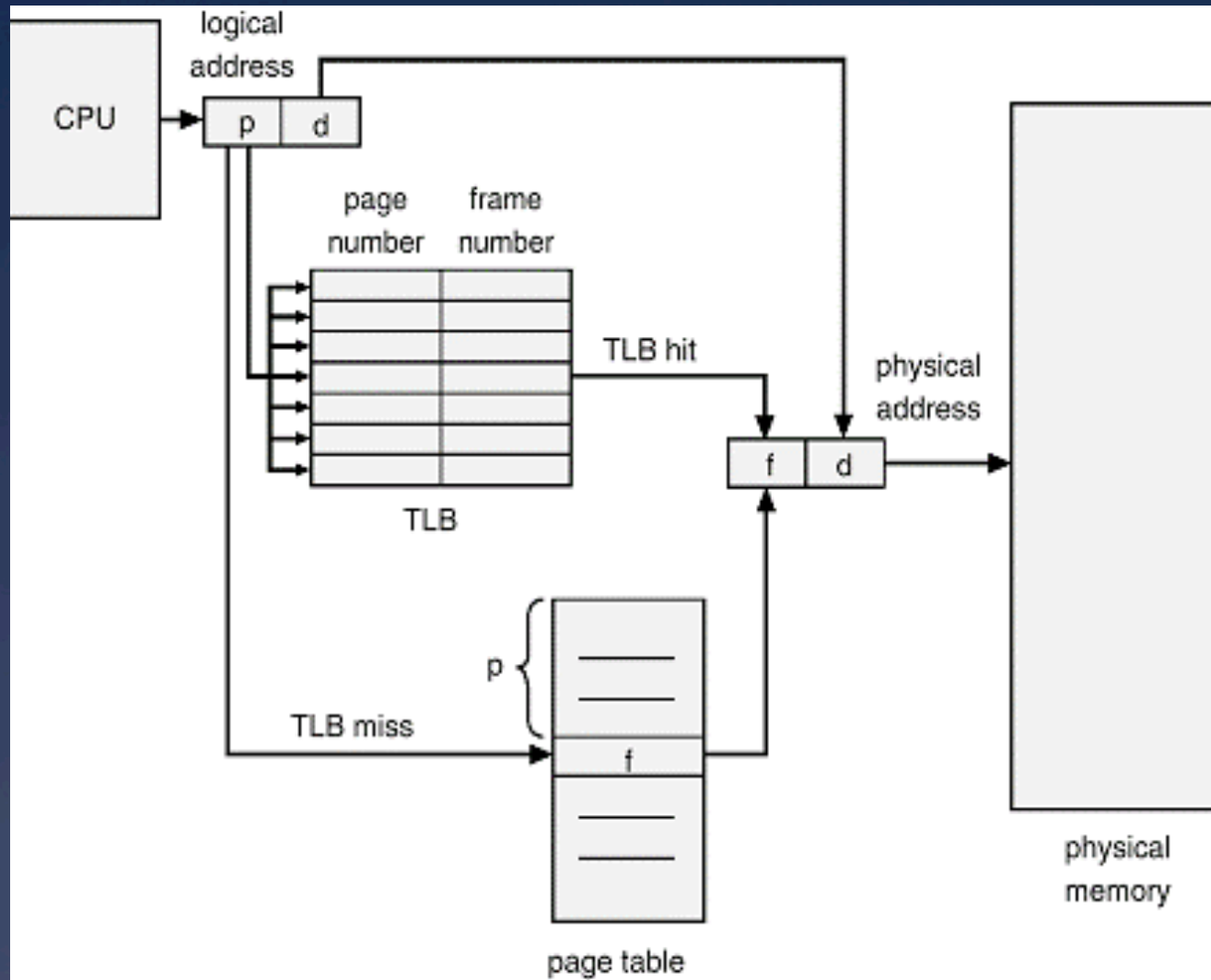
- * Mapeamento direto
 - * Antes de um processo começar a rodar, SO carrega endereço da tabela de mapeamento no registrador especial
 - * Ciclo de mapeamento direto domina tempo de execução da instrução:
 - * Mapeamento direto duplica tempo de acesso a memória (dois acessos para obter um endereço: acesso à tabela e acesso final)
 - * Pode-se tentar tabela em memória alta performance (cache)
- * Mapeamento associativo
 - * Memória associativa é endereçada por conteúdo
 - * Verificação feita em paralelo, tempo de acesso uma ordem de magnitude mais rápido que memória ordinária
 - * Memória associativa guarda tabela com últimos acessos
 - * Endereçamento da MA feito pelo número da página virtual, conteúdo é número da página real
 - * Princípio da localidade garante funcionamento

Paginação: memória associativa



Fonte: <http://users.cs.cf.ac.uk/O.F.Rana/os/lectureos9/node2.html>

Tabela de páginas + memória associativa



[http://www.gitam.edu/eresource/comp/gvr\(os\)/8.4.htm](http://www.gitam.edu/eresource/comp/gvr(os)/8.4.htm)

TLBs

- * Memória de acesso rápido para aumentar velocidade da tradução do endereço virtual para endereço real
- * Em geral de 4 a 64 entradas
- * Várias maneiras de organização
 - * Mapeamento direto:
 - * primeiros bits do endereço indicam entrada da tlb, que então verifica se outros bits da página virtual armazenada são iguais, caso seja usado número da página real.
 - * Cada página pode estar em apenas uma posição da TLB
 - * Podemos ter que remover entrada mesmo quando há entradas vazias.
 - * Mapeamento totalmente associativo:
 - * busca simultânea de todas as chaves
 - * Mapeamento n-associativo:
 - * bancos associativos, cada um n entradas
 - * primeiros bits indicam banco, busca no banco é simultânea
 - * Cada página pode estar em n posições da TLB
- * Custo aumenta com associatividade, velocidade também.
- * Controladas por hardware ou por software (falha de TLB e SO repõe entrada com campo da tabela de páginas)

Segmentação

- * Um programa pode ocupar vários “segmentos” de memória, cada um de tamanho diferente
- * Elimina fragmentação interna
- * Divisò em segmentos geralmente é “lógica”
- * Proteção
 - * Implementação mais complexa para proteção de acesso inválido
 - * Porém implementação de códigos de acesso por segmento simplificada (divisão lógica não física)
 - * Códigos de acesso: leitura, escrita, execução, “append”

Segmentação

- * Campos típicos de uma tabela de segmentos
 - * Residence bit
 - * Tamanho
 - * Bits de acesso (r,w,e,a)
 - * Endereço do início do segmento na memória
 - * Endereço do na memória secundária
- * Endereçamento direto ou associativo
 - * Semelhante à paginação mas SOMA base e deslocamento ao invés de concatenar (mais lento)
- * Fragmentação externa: semelhante à partições variáveis

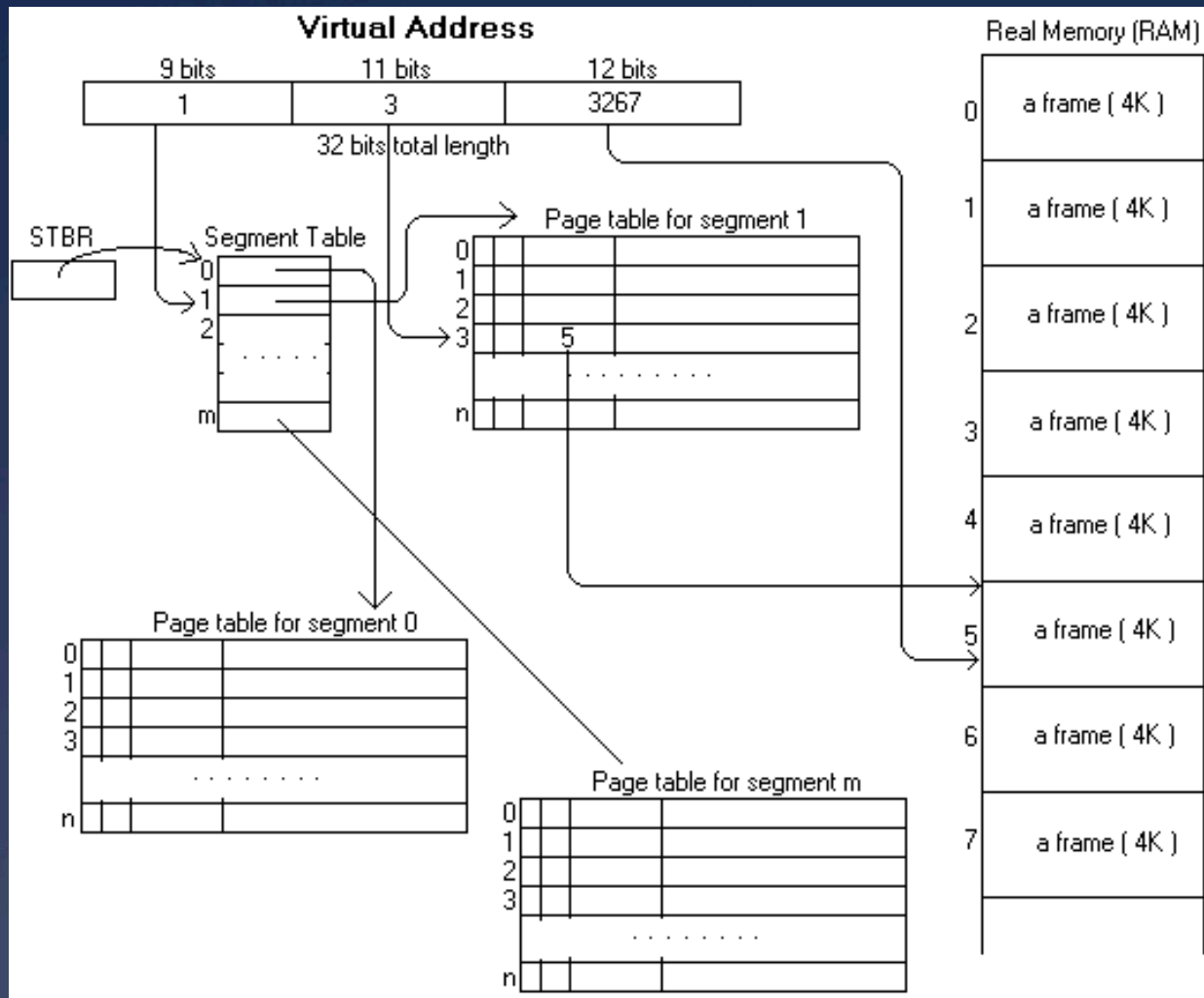
Compartilhamento de memória

- * 2 entradas em tabelas de blocos diferentes podem apontar para mesmo segmento/página da memória
- * Compartilhamento pode reduzir utilização de memória
 - * Ex. Código reentrante: dados separados, mesmo segmento de código
- * Compartilhamento pode aumentar velocidade (menos falhas de página)
- * Compartilhamento mais fácil em sistemas segmentados pois divisão de blocos de memória é lógica.
 - * Em sistemas paginados administração de “páginas parciais” pode ser complexa, solução é aumentar fragmentação interna, alocando páginas
 - * Mais complexa com crescimento dinâmico pois inversão de nova página envolve mudar indicações de compartilhamento desta
 - * Outros processos que compartilham área precisam ser notificados

Segmentação + Paginação

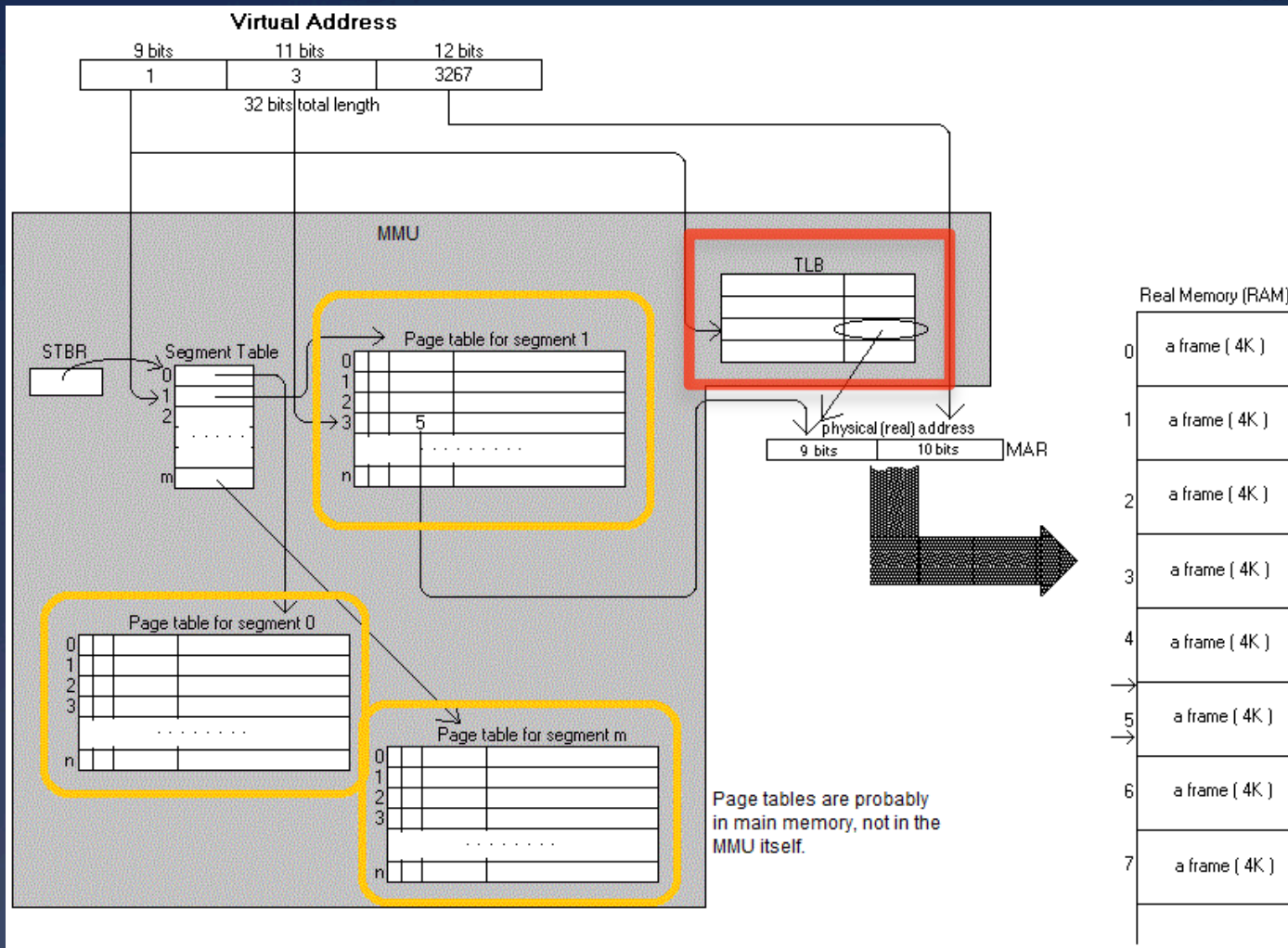
- * Memória dividida em páginas
- * Páginas agrupadas em segmentos
- * Uma tabela de segmentos
- * Cada segmento tem uma tabela de páginas
- * Endereço maior que memória
- * Vantagens de ambos os sistemas
 - * Sem fragmentação externa
 - * Unidades lógicas para compartilhamento
- * Maior overhead de processamento (minimizado pela TLB)
- * Endereço dividido em 3 partes
 - * Segmento
 - * Página
 - * deslocamento

Segmentação + Paginação



fonte: <http://ptolemy.canisius.edu/~meyer/253/BOOK/ch12/FULLPAGE/ch12-10.html>

Segmentação + Paginação com TLB



fonte: <http://ptolemy.canisius.edu/~meyer/253/BOOK/ch12/FULLPAGE/ch12-10.html>

Administração de memória virtual

- * Estratégias de carregamento (*Fetch strategies*)
 - * Paginação por demanda é de longe o mais usado
 - * Carregamento preventivo ou antecipatório: difícil de ser previsto, porém útil quando “page frame” menor que bloco de disco
- * Estratégias de colocação (*placement strategies*)
 - * Em qual dos lugares vagos colocar?
 - * Sistemas segmentados: mesmas estratégias que partições variáveis (first fit, best fit, worst fit)
 - * Sistemas com paginação (paginação pura ou segmentação com paginação) => INDIFERENTE

Administração de memória virtual: estratégias de reposição de páginas

- * É possível que em sistemas multiprocessados com muitos usuários todos os blocos de memória estejam ocupados
- * Quando novo bloco precisa ser carregado é necessário política para escolher qual bloco alocado deve ser substituído (trataremos só de páginas para simplificar)
- * Idealmente deveria sempre repor o bloco (página) que vai demorar mais tempo para ser utilizado novamente
- * Como na maioria dos casos é impossível prever o futuro, várias políticas de reposição usam heurísticas diferentes para aproximar este princípio
- * Princípio da localidade é importante neste caso.

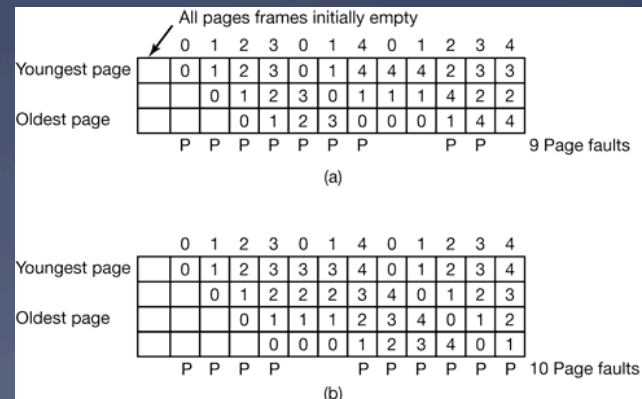
Administração de memória virtual: estratégias de reposição de páginas

- * Randômica
 - * Supõe poucos processos com uso grande de memória
 - * Como maioria das páginas não devem estar sendo utilizadas (princípio da localidade), a chance de se obter uma com muita utilização é baixa.
 - * Simples e barato
 - * Raramente usado

Administração de memória virtual: estratégias de reposição de páginas

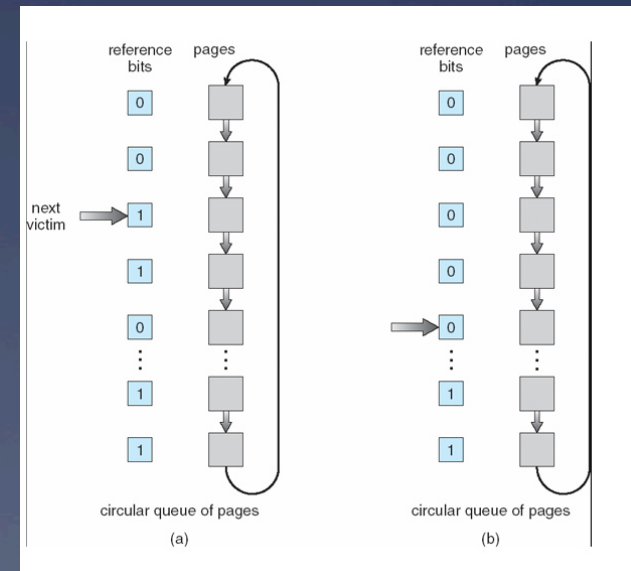
* FIFO

- * Princípio: se página está faz mais tempo na memória, já deve ter sido bastante usada e portanto localidade já deve ter mudado
- * Problema: muitas vezes algumas páginas ficam muito tempo na memória porque são muito usadas (ex. Código de editor de textos, rotinas de chamada do SO)
- * Anomalia FIFO (Belady et al., 1969) : em alguns casos, mais memória na FIFO pode causar mais falhas de página
 - * Mem. Real Com 3 ou 4 páginas
 - * Páginas virtuais acessadas
 - * 0,1,2,3,0,1,4,0,1,2,3,4



Administração de memória virtual: estratégias de reposição de páginas

- * Variação de FIFO: fifo segunda chance
- * Só são repostas as páginas com o “reference bit” desligado
- * Quando é necessário repor a página, a fila vai sendo examinada
- * Se a página da frente tem o “reference bit” desligado, página é reutilizada
- * Caso contrário, o bit é desligado e a página movida para o final da fila



Administração de memória virtual: estratégias de reposição de páginas

- * LRU- Least recently used
 - * Repõe a página menos usada recentemente, ou, em outras palavras, que foi usada há mais tempo
 - * Se página não é usada a muito tempo, não deve ser usada no futuro próximo
 - * Considerada boa política: problema é implementação
 - * Possíveis implementações:
 - * Contadores para cada página, atualizado a cada acesso, resposta página com menor contador
 - * Pilha: quando página é acessada, tira-se do meio da pilha e coloca no topo. Reposta página embaixo da pilha
- * Alto overhead (cada acesso)
- * Falhas
 - * Loops longos
 - * “deeply nested calls”

Administração de memória virtual: estratégias de reposição de páginas

- * NUR- not used recently
 - * Aproximação de LRU com menor *overhead* (sobrecarga)
 - * Mantém marcadas páginas usadas “recentemente”
 - * Levam em conta também que é melhor repor página que não foi alterada na memória (1 vs 2 acesso a disco)
 - * Usados 2 bits por página: *access bit*, *dirty bit*
 - * Procura de páginas a serem repostas
 1. Página não referenciada
 2. Página referenciada mas não alterada
 3. Páginas referenciadas e alteradas
 - * Problema: eventualmente todas as páginas serão referenciadas
 - * Periódicamente zerado o “access bit” – páginas muito usadas ficarão vulneráveis por pouco tempo.

Administração de memória virtual: estratégias de reposição de páginas

- * NUR- not used recently
 - * Implementação
 - * Hardware com atualização automática de dirty bit e access bit
 - * Uso de residence bit e bit de proteção de escrita
 - * Residence bit substitui access bit - Páginas após serem carregadas não são marcadas como residentes. Tabela adicional indica que página está presente. No próximo acesso, após falha de página, algoritmo de paginação apenas seta o “residence bit”
 - * Protection bit substitui dirty bit – página ao ser carregada é marcada como protegida para escrita. Primeira escrita gera falha de proteção. Algoritmo de paginação seta bit em tabela adicional indicando que página foi modificada e retira proteção.

Administração de memória virtual: estratégias de reposição de páginas

- * Working set
 - * Peter Denning (1968) criou teoria de que cada programa tem seu “conjunto de trabalho” (ex, página com o loop e páginas com matrizes sendo multiplicadas)
 - * Para um programa rodar eficientemente, seu conjunto de trabalho precisa estar na memória real
 - * Política de working set tenta manter este conjunto sempre na memória
 - * Um processo deve ser mantido em memória apenas se seu conjunto de trabalho está inteiro na memória, senão deve ser retirado (swapped out)
 - * Idéia é prevenir “thrashing” (quantidade excessiva de falhas de página que causa velocidade inviável de processamento) e, ao mesmo tempo, manter o maior número de processos em memória possível

Administração de memória virtual: estratégias de reposição de páginas

- * Working set: implementação
 - * Conjunto de trabalho $CE(t,j)$ calculado no tempo T é o conjunto de páginas utilizado nos últimos j acessos
 - * j é a “janela” do CE.
 - * Problema: como manter o working set:
 - * Como calcular T ?
 - * janela é móvel, ao acessarmos página nova, página mais antiga referenciada pode sair do CE – muito caro calcular t
- * Para evitar sobrecarga de contar todos os acessos, mantém-se alternativamente, aproximação do tempo do último acesso e usar janela de tempo.
 - * Podemos usar contabilidade a cada TIC e resetar “reference bit
 - * se resetarmos reference bit periodicamente temos NRU

Administração de memória virtual: estratégias de reposição de páginas

- * Working set: WSClock
 - * Conceitos de WS e de fifo segunda chance
 - * Funciona bem e é utilizado
 - * Porém um conjunto não muito claro de idéias diferentes, típico de sistemas reais
- * Funcionamento
 - * Para cada processo temos residence bit (R) e dirty bit(M) mantidos por hardware. A cada k clicks R é resetado
 - * A cada tic todas as páginas são examinadas e o tempo é guardado para aquelas onde R vale 1. Campo de tempo é aproximação do último acesso, pela resolução do relógio (na verdade podem ser k tics)
 - * Páginas que tem tempo anterior à $t - j$ guardado são candidatas a reposição, se não houver, usamos página mais “antiga”
 - * Preferência dada a páginas com M limpo
 - * Da mesma maneira que no algoritmo clock, mantida lista circular.

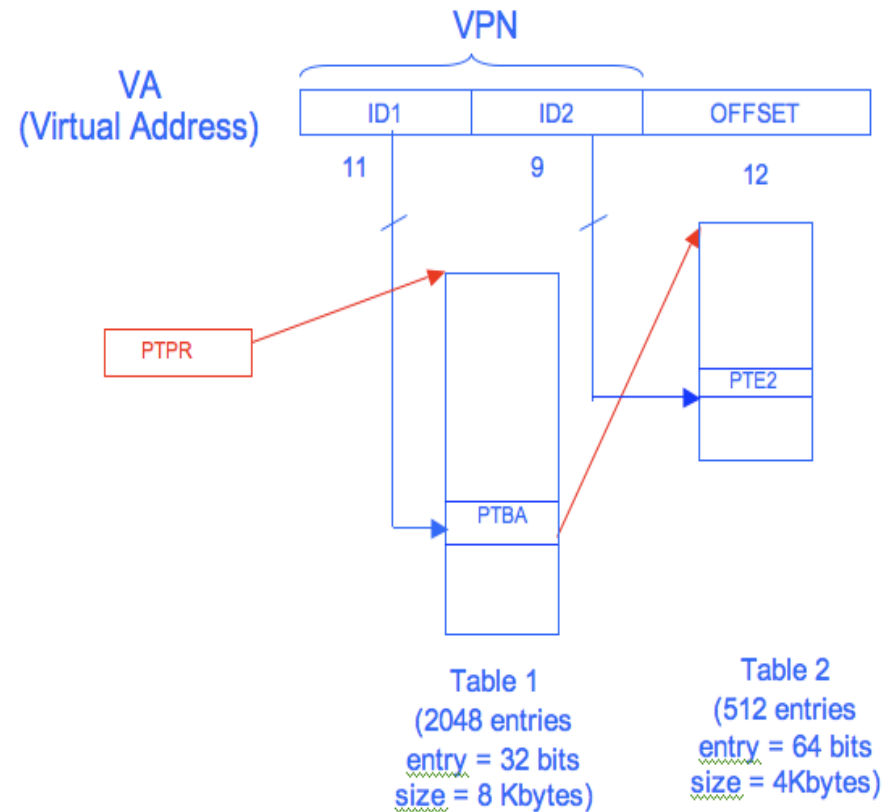
Considerações sobre paginação

- * Tamanho da página

Outros mecanismos de paginação

TSAR MMU

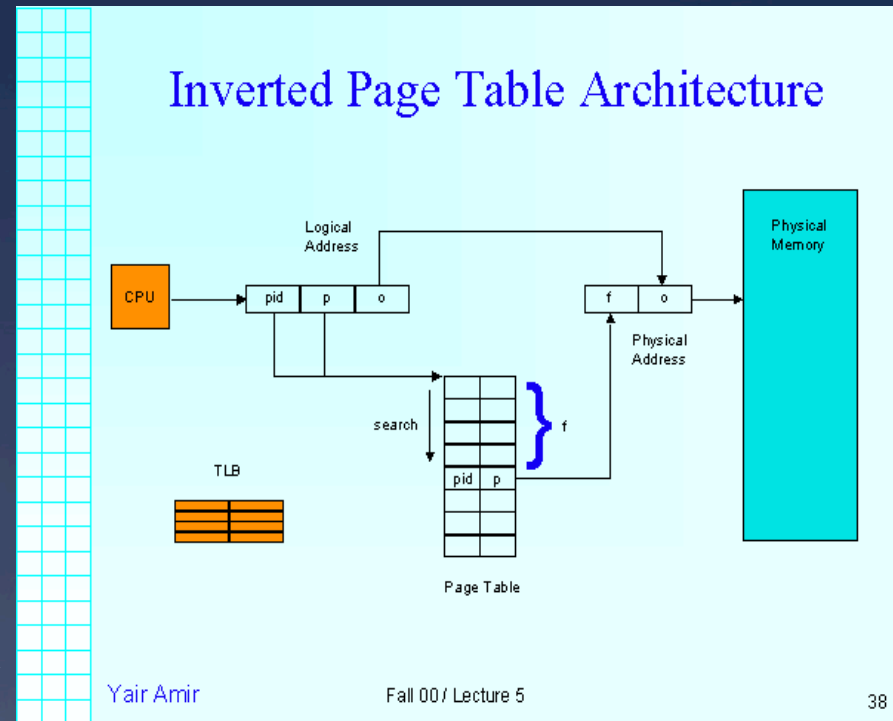
- * Paginação em múltiplos níveis
 - * Uma tabela de página por processo pode usar muita memória (páginas de 4k, 32 bits, 20 bits para número de página, 1 milhão de entradas)
 - * Tabela pode ser dividida em tabela primária e várias tabelas secundárias (e terciárias, quaternárias, etc.)
 - * Endereço dividido em 3 ou mais seções, cada uma usada para indexar uma das tabelas
 - * Mantidas apenas tabelas secundárias utilizadas pelo programa.



fonte: <https://www-soc.lip6.fr/trac/tsar/wiki/VirtualMemory>

Outros mecanismos de paginação

- * Tabela de páginas invertida
 - * Tabela global, não por processo, cada entrada deve conter número da pág. Virtual e id do processo.
 - * Tabela indexada por número da página real
 - * Para procurar endereço virtual E_v para processo PID, procurar entrada na tabela com este conteúdo
 - * Se estiver na entrada i da tabela, a página real é a i -ésima página
 - * Implementações reais usam tabela de hash para busca
 - * Arquitetura Itanium.



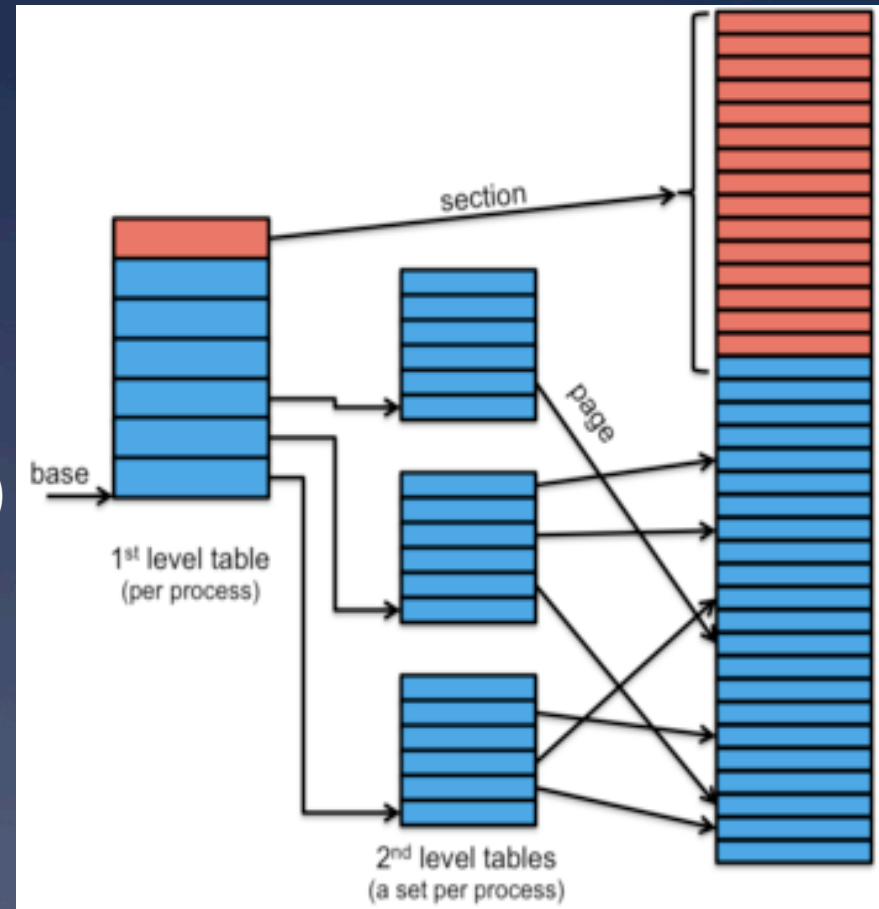
Fonte: <http://www.cs.jhu.edu/~yairamir/cs418/os5/sld038.htm>

Esquemas de memória em processadores atuais: ARMv7

- * Processador RISC (reduced instruction set) de 32 bits
- * Telefones celulares, tablets, consoles de jogos (CortexA8 – Motorola Droid, Ipad, iPhone 3GS e 4; CortexA9 – Ipad2)

Esquemas de memória em processadores atuais: ARMv7

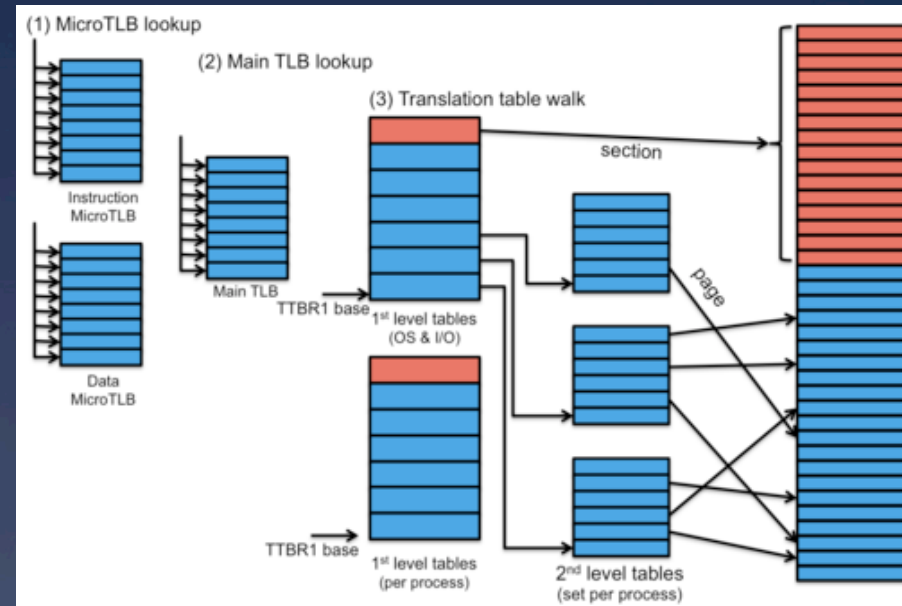
- * Memória paginada,
- * Dois níveis para estrutura da tabela de páginas
 - * Tabela de primeiro nível (por processo) que contém ponteiro para superseção, seção ou tabela secundária
 - * Superseções: blocos de memória de 16MB (offset de 24 bits)
 - * Seções: blocos de memória de 1MB (offset 20 bits)
 - * Páginas grandes: 64kb (offset 16 bits)
 - * Páginas pequenas: 4kb (offset 12 bits)
- * Estrutura flexível MMU (memory management unit)
 - * pode ser configurada para usar os vários tamanhos de página bem como seções e superseções.
 - * Seções e superseções podem ser combinadas com páginas
 - * SO pode ou não usar
- * Seções e superseções
 - * Possibilidade de endereçar com apenas uma nível de hierarquia grandes porções de memória (por exemplo, para o SO)
 - * Uso com páginas introduz problemas comuns à segmentos de tamanho variável



Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html>

Esquemas de memória em processadores atuais: ARMv7

- * TLB em 2 níveis
- * microTLB
 - * Menor e mais rápida
 - * Uma para instruções (32 entradas) uma para dados (32 ou 64 entradas)
 - * Totalmente associativa
 - * Lookup em um ciclo apenas -> sem penalidade de acesso na prática
 - * Address Space Identifier (ASID) – permite entrada de vários processos na TLB
 - * 30 bits chave de endereço, 8 bits ASID
 - * Entradas Globais (compartilhadas) e locais (por processo)
 - * Páginas tem bits de proteção verificada a cada acesso
 - * Reposição round-robin (default) ou randomico



Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html>

Esquemas de memória em processadores atuais: ARMv7

- * TLB em 2 níveis
- * Main TLB
 - * Utilizada quando não há hit na micro TLB
 - * Apenas 1 por processador, usado para instruções e dados
 - * 4 entradas totalmente associativas, que podem ser travadas (nota: SO deve travar também páginas na memória real)
 - * 64 ou 128 entradas em estrutura 2-associativa (2x32 ou 2x64)

Esquemas de memória em processadores atuais: Intel IA-32

- * Arquitetura segmentada
- * 3 modos de endereçamento
 - * Flat memory – endereçamento linear de 4Gb
 - * Segmentado:
 - * Endereço – seletor de segmento + deslocamento
 - * Endereço de memória 16bits (segmento)+32 bits (deslocamento)
 - * 16k segmentos, cada um com até 4Gb
 - * Alguns segmentos com funções específicas (código, pilha, etc.)
- * Modo real
 - * Característica da série intel x86 é compatibilidade
 - * Endereço de 20 bits (1Mb): 4 bits para segmento + 16 bits para deslocamento (16K bytes max)

Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html>

Esquemas de memória em processadores atuais: Intel IA-32

* Endereçamento segmentado-paginado

* 2 tabelas de segmentos, cada uma contém endereços base de até 8.191 segmentos

* Local descriptor table (LDT): segmentos privados ao processo

* Global descriptor table: segmentos compartilhados entre todos os processos (e.g. SO)

* Seletor de segmentos carregado em registrador de segmentos

* Instruções têm seletor de segmentos implícito ou prefixo de segmento antes da instrução

* Seletor de segmento automaticamente seleciona GDT ou LDT

* Endereço linear:

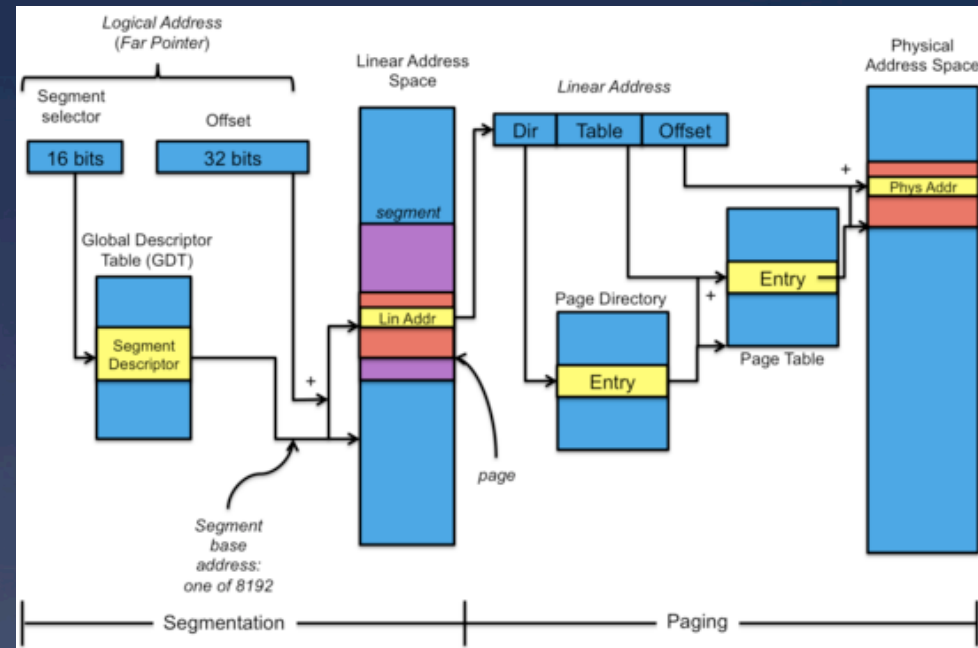
* $LDT[\text{seletor_segmento}] + \text{offset}$

* $GDT[\text{seletor_segmento}] + \text{offset}$

* Endereço linear tratado como endereço virtual paginado

* Tabela de páginas em 2 níveis.

* Paginação pode ser desabilitada



Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html>

Esquemas de memória em processadores atuais: Intel IA-32

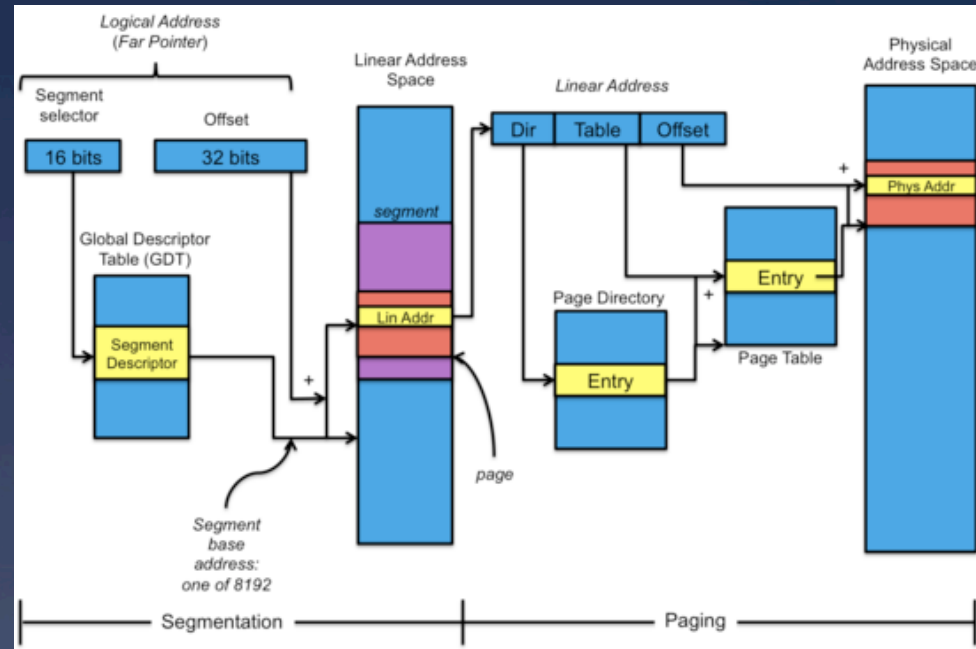
* Detalhes da segmentação

* Entrada na GDT contém

- * Sflag (código ou dados?)
- * Accessed (access bit)
- * Dirty Bit
- * Data/write-enable (read only ou read/write?)
- * Data/expansion direction (por default segmentos no topo do segmento, bit muda direção da expansão)
- * Code/execute-only or execute/read
- * Conforming (indica se execução pode continuar mesmo se nível de privilégio é elevado)

* Detalhe da paginação

- * Páginas de 4Kb ou 4Mb



Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html>

Esquemas de memória em processadores atuais: Intel IA-64

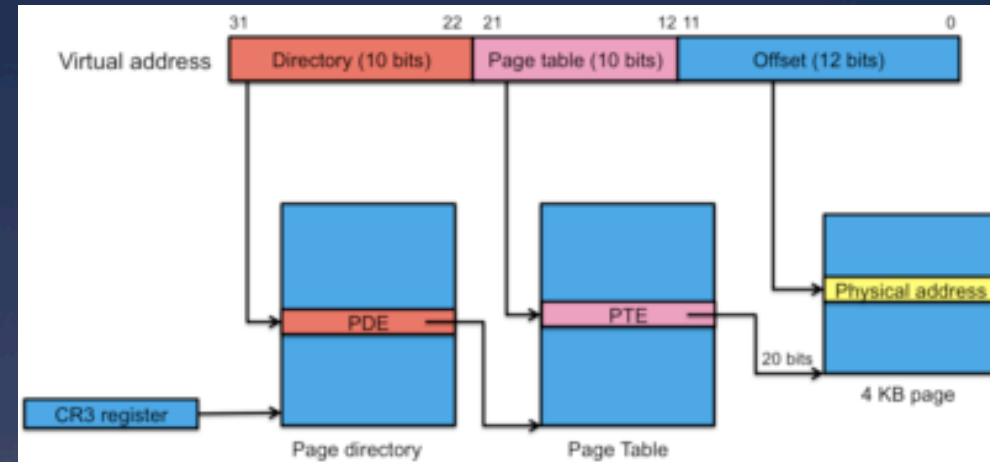
- * Apenas paginação (segmentação tem suporte no modo emulação de IA-32 -> compatibilidade)
- * Três modos de paginação
 - * Paginação 32 bits
 - * PAE (Physical Address Extension mode – IA-32)
 - * Paginação IA-32e – 48 bits

Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html>

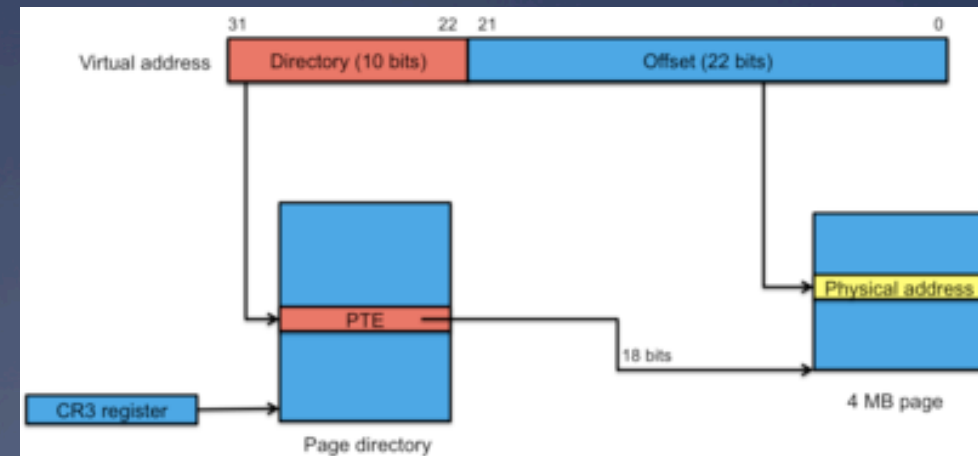
Esquemas de memória em processadores atuais: Intel IA-64

Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html>

- * Paginação 32 bits
- * Páginas de 4Kb ou 4Mb
- * Páginas de 4Kb:
 - * paginação em 2 níveis (10Bits+10bits+12bits)
 - * Tabela de páginas (nível 2) contém base de 20 bits
- * Páginas de 4Mb:
 - * paginação em 1 nível (10bits+22bits)



Página de 4KB



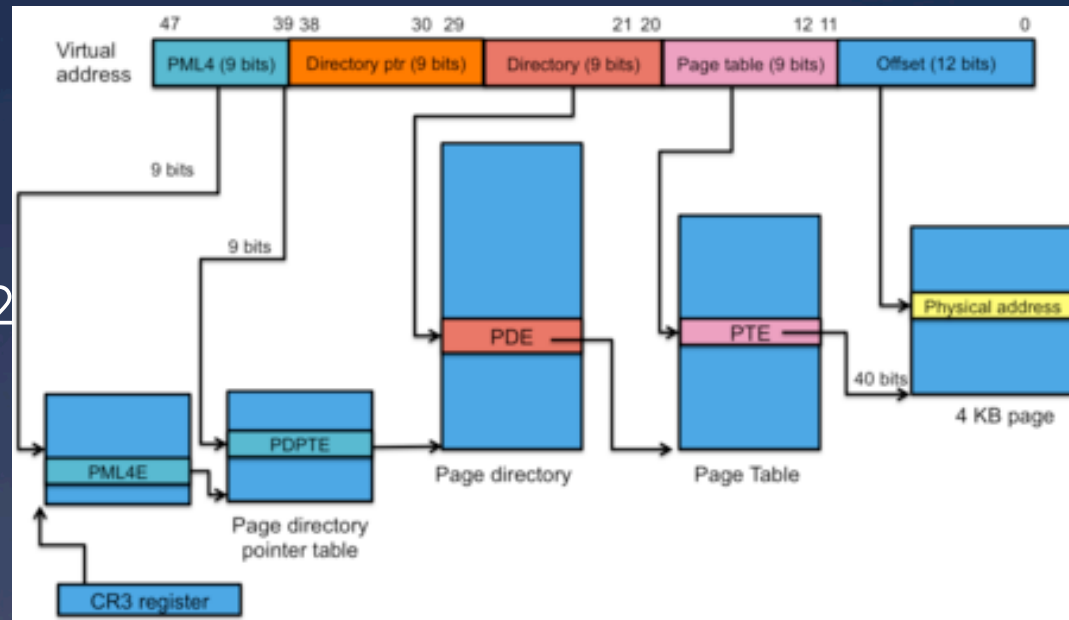
Página de 4MB

Esquemas de memória em processadores atuais: Intedl IA-64

- * Paginação IA-32e
- * Endereço de 48 bits (256 terabytes) gera endereço real de 52bits (4K terabytes)
- * Endereços virtuais de 32 bits
- * Gera endereços virtuais de até 52 bits
- * páginas de 4Kb, 2Mb ou 1Gb

Esquemas de memória em processadores atuais: Intedl IA-64

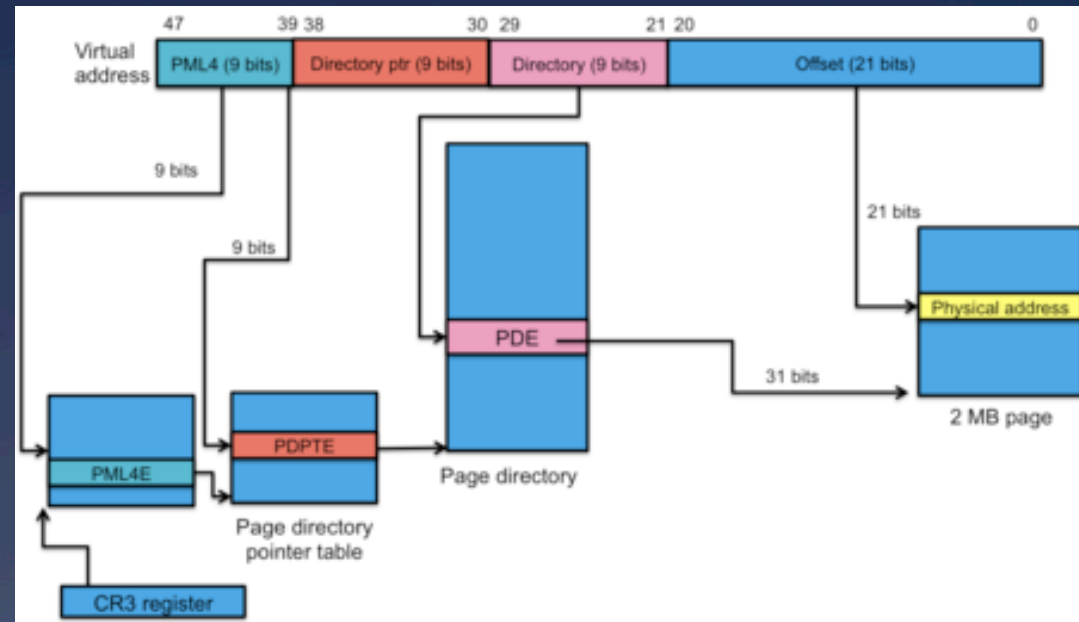
- * Paginação IA-32e
- * Páginas de 4k
- * Paginação em 4 níveis
- * 9bits+9bits+9bits+9bits+12
- * Cada nível com 512 entradas



Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html>

Esquemas de memória em processadores atuais: Intedl IA-64

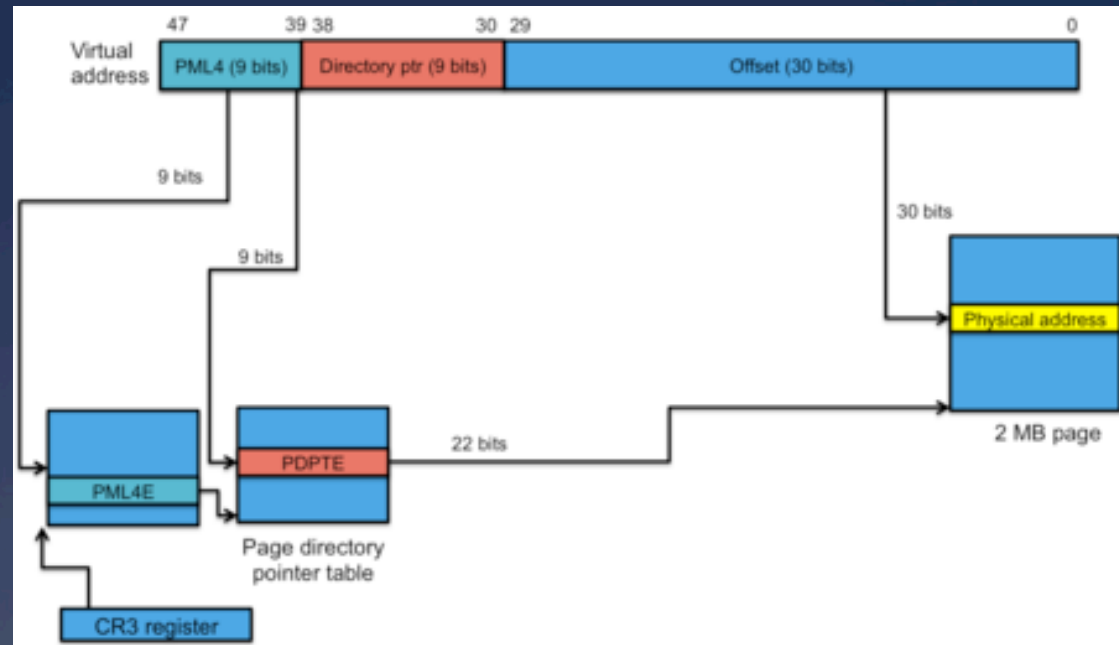
- * Paginação IA-32e
- * Páginas de 2Mb
 - * Paginação em 3 níveis
 - * 9bits+9bits+9bits+21bits



Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html>

Esquemas de memória em processadores atuais: Intel IA-64

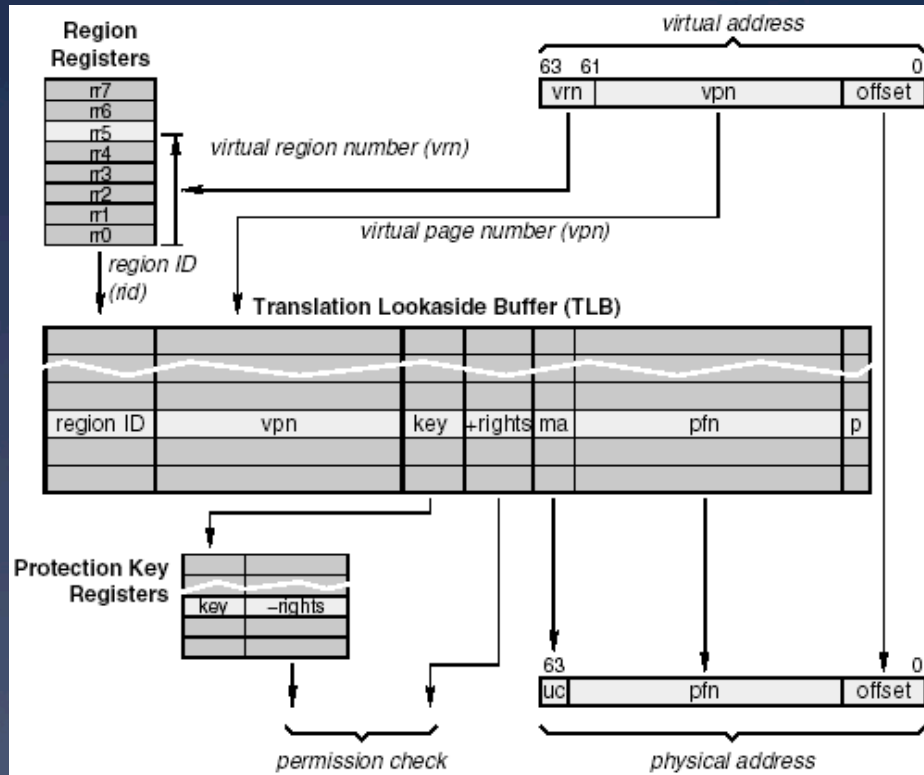
- * Paginação IA-32e
- * Páginas de 1gb
 - * Paginação em 2 níveis
 - * 9bits+9bits+30bits



Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html>

Esquemas de memória em processadores atuais: Intel IA-64 - TLBS

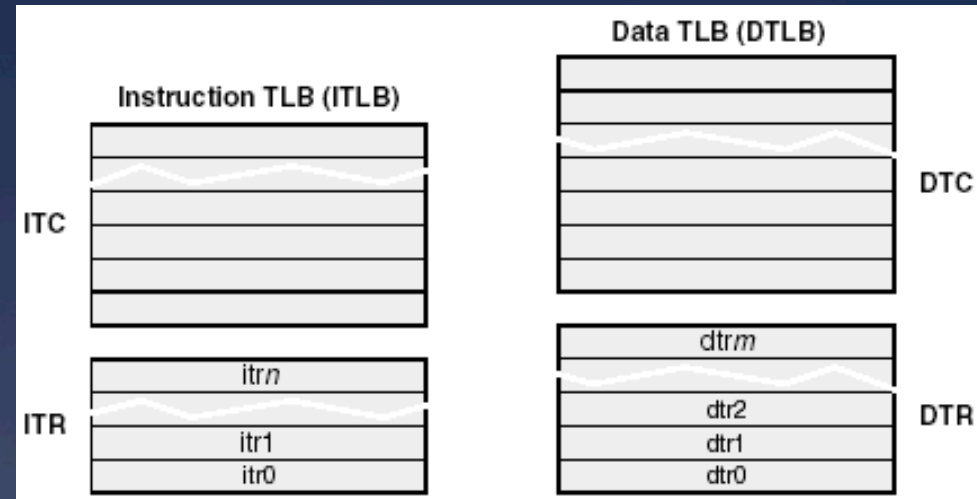
- * Conjugada com 3 outras estruturas
 - * Region registers
 - * Protection key registers
 - * Virtual hash page table walker (VHPT)
- * Mecanismo de tradução da TLB
 - * Endereço em 3 partes
 - * VRN (virtual region number)
 - * VPN (virtual page number)
 - * Page offset
 - * Tabela de VRN gera RegionID que, concatenada com VPN, é enviada à TLB,
 - * TLB contém
 - * PFN (page frame number - página real)
 - * MA - indica se endereço pode ir para a cache (determina valor do bit UC)
 - * +rights (read/write/execute e nível de privilégio para cada - user/kernel)
 - * Key enviado para os protection key registers, onde o registrador com mesma chave indica direitos negativos para completar permissões (pode cancelar os direitos em +rights)



Fonte: <http://www.informit.com/articles/article.aspx?p=29961&seqNum=4>

Esquemas de memória em processadores atuais: Intel IA-64

- * 4 unidades lógicas
- * ITLB (instruções)
- * DTLB (dados)
- * Cada uma tem:
 - * *translation caches* (ITC, DTC)
 - * reposição determinada pelo hardware(SO)
 - * Mínimo 1 entrada – Itanium 96(ITC) +128(DTC)
 - * *translation registers* (ITR, DTR)
 - * reposição determinada pelo software
 - * Mínimo 8, muitas vezes temos opção em determinar que translation caches sejam usadas como translation registers
- * Para mais detalhes (incluindo VHPT)
 - * <http://www.informit.com/articles/article.aspx?p=29961&seqNum=4>



Fonte: <http://www.informit.com/articles/article.aspx?p=29961&seqNum=4>

Sistemas de Arquivos

- * Pode-se argumentar que melhor maneira de armazenar informação no computador seria deixá-la toda no espaço de endereçamento (ex. 2^{64} bytes: 2^{32} segmentos de 2^{32} bytes)
- * Usando proteção e compartilhamento manteria-se toda a informação na memória virtual.
- * Alguns segmentos poderiam guardar diretórios que garantiriam estrutura hierárquica da informação.
- * Teríamos espaço de endereçamento padrão inicializado pelo SO com todas as informações compartilhadas
- * MULTICS tentou esta abordagem (1964, MIT sistema rodou até 10/2000)

Sistemas de Arquivos

- * Esta abordagem, por sua complexidade não foi mais utilizada
- * Maioria dos sistemas tem uma *hierarquia de memória*.
 - * Caches
 - * Memória primária
 - * Memória secundária (dispositivos)
- * “Sistemas de arquivos” provêm a organização da memória secundária. Geralmente persistente e único para uma instância de um SO

Funções dos sistemas de arquivos

- * Criação, eliminação e modificação de arquivos
- * Compartilhamento de arquivos
- * Controle de acesso
- * Backup e recuperação de informação
- * Acesso simbólico (por nomes) aos arquivos
- * Estruturação dos arquivos (ex. Sequencial, aleatório, etc.)
- * Prover visão lógica (não física) dos arquivos: independência de dispositivo
- * Segurança (criptografia)
- * Integridade de arquivos

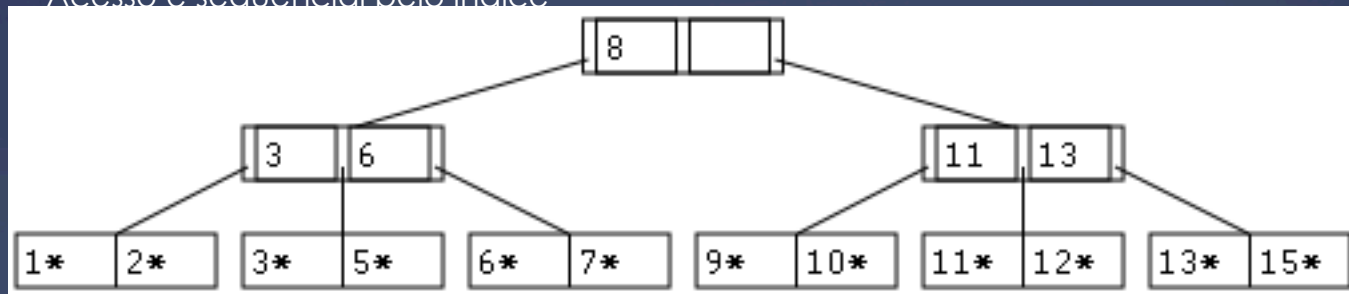
Organização de arquivos

- * Blocos
 - * Registros (comum em mainframes, principalmente os antigos)
 - * Bytes (Unix, Minix)
- * Métodos de acesso
 - * Sequencial (Unix/Minix – character)
 - * Indexado
 - * Aleatório (Unix/Minix - block)

*

Organização de arquivos: organização interna

- * Unix block:
 - * Sequencia de blocos, mas endereçada por bytes
 - * read/write/seek
- * QSAM (Queued Sequential Access Method)
 - * Arquivos organizados por blocos físicos
 - * Acesso do usuário por registro lógico definido pelo usuário
- * ISAM (Indexed Sequential Access Method)
 - * Organização em árvore: blocos com número fixo de registros
 - * Apontadores de outros blocos entre registros
 - * Acesso é sequencial pelo índice



- * Outros métodos (IBM): XDAP, BDAM, BSAM, BPAM, VSAM, OAM
- * Em sistemas derivados dos computadores pessoais, em geral acesso aleatório e organização por blocos.

Tipos de arquivos

- * Regulares
 - * Arquivos de caracter (ASCII,...)
 - * Linhas terminadas por “line feed” ou “carriage return” ou ambos
 - * Linhas de tamanho variável
 - * Arquivos ASCII úteis porque codificação padrão de informação, facilitam uso de “pipes” para conectar saída de um arquivo com entrada de outro.
 - * Arquivos binários
 - * Código: todos os SO's precisam reconhecer arquivos formatados para código executável (TOPS-20 – DEC – inclusive verificava se código estava em dia com o “fonte” recompilava se necessário.
 - * Campos:
 - * Header: Número mágico (descreve arquivo executável), tamanho da área de texto, tamanho da área de dados, tamanho do BSS, tamanho da tabela de símbolos, ponto de entrada, flags
 - * Texto, Dados, Bits de relocação (para colocar o programa na memória), tabela de símbolos (depuração)

Tipos de arquivos

- * Archive (UNIX)
 - * Coleção de módulos de biblioteca, compilados mas não linkados
 - * Cada módulo, tem entradas com
 - * header com seu nome, data de criação, dono, código de proteção e tamanho
 - * “object module”
- * Arquivos “fortemente tipados” podem ser problema
 - * Ex: pré- processador de C (C++) que gera como resultado programa C em arquivo .dat (Livro, pág 487)
 - * Compilador C não aceita .dat
 - * Sistema impede cópia de .dat para .c

Atributos comuns em arquivos

- * Proteção: indica quem pode acessar arquivo e como
- * Senha específica para o arquivo
- * Criador (usuário)
- * Dono (usuário)
- * Flag de leitura/escrita (No Unix em proteção)
- * Flag de “hidden file”
- * System flag
- * ASCII/Binary flag
- * Archive flag (o arquivo tem backup?)
- * Random access flag (acesso sequencial apenas?)

Atributos comuns em arquivos

- * Temporary flag (indica se arquivo pode ser eliminado na saída do processo)
- * Lock flag
- * Tamanho do registro – arquivos formatados
- * Key position (posição da chave em cada registro) – para arquivos formatados
- * Key length
- * Criação
- * Último acesso
- * Última alteração
- * Tamanho atual
- * Tamanho máximo (incomum em sistemas modernos)

Operações em arquivos

- * Criação
- * Remoção
- * Abertura
 - * Verifica disponibilidade (existe?, tem alguém usando?)
 - * Verifica proteção
 - * Associa chave ao arquivo para acesso não ser mais simbólico
 - * Cria área para guardar bloco
 - * Mantém registro para controlar acesso (posição do bloco lido em disco, etc.)
- * Fechamento (close): libera registro de controle, buffer, blocos lidos, grava últimas alterações

Operações em arquivos

- * Leitura – usuário pode ter que especificar tamanho da leitura (se arquivo não for formatado em registros), e deve fornecer área para armazenamento do resultado (e.g. `Read(bytes, &buffer)`)
- * Escrita – grava no buffer, se atingiu final do bloco escreve no disco; se arquivo está no final, aumenta seu tamanho, se está no meio sobrescreve dados
- * Append: forma restrita de escrita, só adiciona dados no final do arquivo
- * Seek : para arquivos de acesso randômico, muda apontador de leitura/escrita

Operações em arquivos

- * Pega atributos: retorna registro com atributos do arquivo; usado , por exemplo, para make
- * “Seta atributos” – modifica atributos dos arquivos, em geral operação restrita
- * Renomear: pode ser substituído por cópia e remoção (mais overhead)
- * Lock – usado para restringir temporariamente acesso a um arquivo ou parte dele (registro?)

Diretórios

- * Arquivos são agrupados em conjuntos chamados diretórios ou pastas (*directories*, *folders*)
- * Diretórios podem, por sua vez, serem arquivos (Unix)
- * Variações
 - * Um diretório por sistema
 - * Um diretório por usuário
 - * Estrutura hierárquica (árvore ou DAG)
 - * Nomes absolutos (`/usr/local/bin/perl`) vs. nomes relativos (`fotos/noisNaFoto.jpg`)
 - * Nomes relativos implicam “diretório corrente” mantido para cada processo

Operações em Diretórios

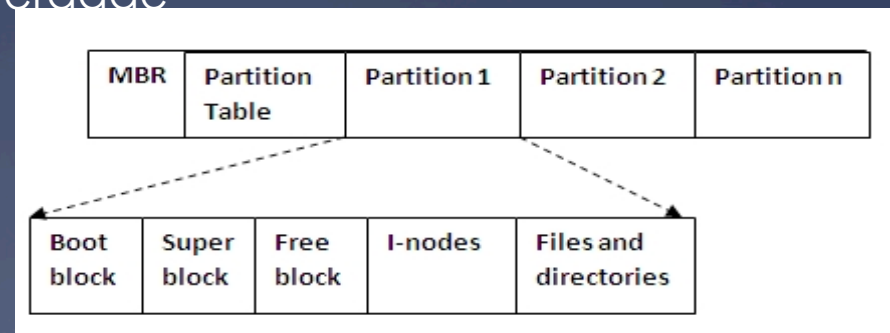
- * Criação: cria diretório vazio (em Unix tem `.` e `..`)
- * Remoção (em Unix precisa estar vazio)
- * Abertura: diretórios podem ser lidos, para isso precisa ser aberto
- * Fechamento
- * Leitura (*readdir*) -
 - * Lê próxima entrada do diretório
 - * poderia ser leitura comum, mas usuário precisaria conhecer formato interno do diretório
- * Renomear
- * Link
 - * Hard links:
 - * duas entradas apontam para mesmo descritor de arquivos
 - * Facilita compartilhamento, arquivos removidos apenas quando não são apontados por ninguém
 - * Segurança; bom (não vou perder arquivo), ruim (alguém podem manter meu arquivo vivo)
 - * Limitado a um sistema de arquivos
 - * Soft links
 - * Entrada especial com endereço absoluto/relativo de outra entrada em diretório
 - * Pode ser usado em sistemas de arquivos diferentes
 - * Segurança: bom (remoção do link apenas elimina entrada no diretório, remoção do arquivo elimina acesso a ele) vs ruim (posso ficar com apontador inválido, ou arquivo pode mudar)
- * Unlink: entrada é removida

Implementação de sistemas de arquivos

- * Layout de sistemas de arquivos
 - * Em discos: Master Boot Record seguido de tabela de partições
 - * Partições primárias (sistemas PC tem apenas 4 entradas na tabela de partições)
 - * Partições lógicas: SO divide partições em sub-partições, neste caso partição contém tabelas de subpartições
 - * Nem todos os dispositivos tem partições: BIOS lê primeiro setor de um disco e procura “número mágico” para ver se é executável (MBR)
 - * Cada partição organizada independentemente e tem seu “boot block”

Implementação de sistemas de arquivos

- * Organização das partições varia de sistema para sistema
- * Unix:
 - * BootBlock – se tiver número mágico correto, pode ser programa executável
 - * SuperBlock – parâmetros básicos do sistema de arquivos, carregado na memória durante o boot ou quando sistema de arquivos usado pela primeira vez
 - * Free space management
 - * I-nodes – descritores de arquivos
 - * RootDir – em posição fixa para fácil acesso
 - * Arquivos e diretórios – os dados de verdade



Fonte: <http://www.moreprocess.com/operating-systems/file-system-layout-in-operating-system>

Implementação de sistemas de arquivos

Alocação de arquivos Contínua

- * Blocos alocados em sucessão física
- * Fácil de implementar, pouca informação por arquivo
- * Ótima performance de leitura – apenas um seek
- * Problema: fragmentação – desfragmentação envolve alto *overhead*
- * Problema: precisamos saber tamanho do arquivo na criação, aumento pode envolver relocação
- * Era muito usado para fitas magnéticas, caiu em desuso,
- * CUIDADO, volta para o passado é comum em SO
 - * razoável para CDROM/DVD/BLUE-RAY

Implementação de sistemas de arquivos

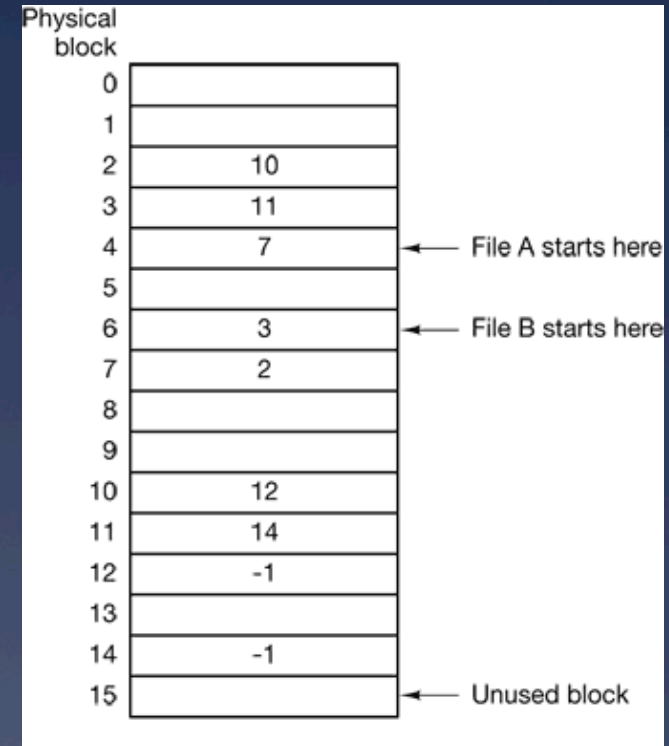
* Alocação de arquivos em lista ligada

- * Cada bloco inicia (termina) com ponteiro para endereço no disco do próximo bloco do arquivo
 - * Elimina problema de fragmentação
 - * Acesso sequencial ok
 - * Péssima performance em acesso aleatório
 - * Fragmentação interna pois tamanho do bloco não é mais potência de 2 (ponteiro ocupa espaço), dificultando relação bloco/setor

Implementação de sistemas de arquivos

tabela (FAT)

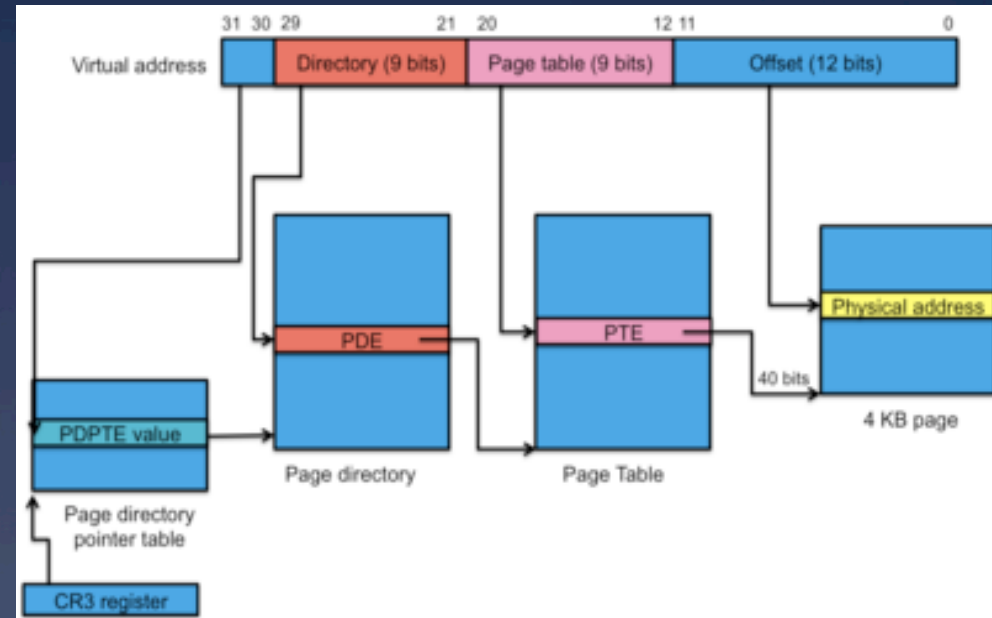
- * Alguns blocos guardam indexador, que é carregado na memória
- * Uma entrada para cada bloco da partição do disco
- * Entrada no diretório contém número do primeiro bloco
- * Cada entrada na FAT contém índice do próximo bloco do arquivo
- * Como tabela é residente na memória acesso aleatório é rápido
- * Problema tamanhos
 - * FAT-12, FAT-16, FAT-32 (máx 4G blocos)



Fonte: tannembraum

Esquemas de memória em processadores atuais: Intel IA-64

- * PAE – Physical address extension
- * Emula PAE do IA-32
- * Endereços virtuais de 32 bits
- * Gera endereços virtuais de até 52 bits
- * páginas de 2Mb ou 4Kb
- * Com páginas de 4Kb, tabela de páginas com 3 níveis (figura)
 - * 2bits+9bits+2bits+12bits
 - * Page Directory Pointer Table (PDPT) com 4 entradas (2 bits)
 - * Page directory
 - * Page table: entradas com 40bits (que somadas aos 12 bits de deslocamento totalizam 52bits)



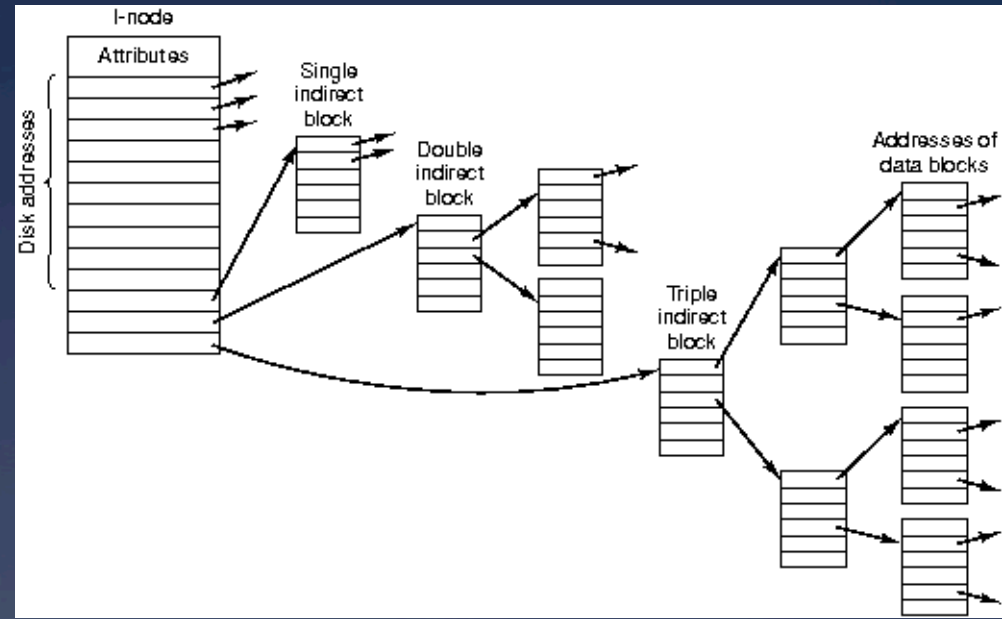
Fonte: <http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html>

Implementação de sistemas de arquivos

Fonte: tannembaum

I-Nodes (Sistemas Unix)

- * Cada arquivo tem um registro que contém ponteiros para os primeiros blocos e descrição do arquivo (veja em atributos)
- * Arquivos maiores usam esquemas de acesso indireto
 - * Single indirect
 - * Double indirect
 - * Triple indirect
- * Pouco overhead para arquivos pequenos
- * Acessos indiretos envolvem novas leituras do disco
- * Acesso totalmente aleatório de arquivos grandes é lento: princípio da localidade ameniza problema
- * Tamanho de arquivo MUITO escalável:
 - * 32bits endereçam $4 \cdot 10^9$ blocos
 - * bloco 4k, 16Tb disco, arquivo 4tb
 - * Bloco 8k 32TDisco, arquivo 32Tb.



Implementação de diretórios

Fonte: tannembaum

CP/M

- * Sistema para computadores de 8 bits
- * 1 diretório para todos os arquivos
- * Entrada tamanho padrão
 - * Cód usuário (1 byte)
 - * Nome (8 bytes)
 - * Extensão (3 bytes)
 - * Extent (1 byte) – para mais de uma entrada para o mesmo arquivo, diz número da entrada
 - * Livre (2 bytes)
 - * Blocos em uso (1 byte)
 - * Apontadores para disco (16 bytes)
 - * Arquivos grandes repetem entradas

Implementação de diretórios

Fonte: tannembaum

DOS (precursor do Windows)

- * Originalmente para sistemas de 6 bits, disco de 10Mb
- * Diretório raiz é fixo (112 entradas no floppy)
- * Outros são arquivos normais
- * Estrutura hierárquica de diretórios
- * Sem usuário
- * Entrada tamanho padrão
 - * Nome (8bytes)
 - * Extensão (3 bytes)
 - * Atributos (1 byte)
 - * Livre (2x2 bytes)
 - * Primeiro bloco na FAT(2 bytes)
 - * Tamanho (4 bytes)

Implementação de diretórios

Fonte: tannembaum

Windows 95

- * 2 tipos de entrada (para compatibilidade)
- * Entrada tamanho padrão
 - * Nome (8bytes)
 - * Extensão (3 bytes)
 - * Atributos (1 byte)
 - * Campo NT (1 byte) – para múltiplas entradas
 - * Sec (1 byte)
 - * Data e horário da criação (4 bytes)
 - * Ultimo acesso (2 bytes)
 - * 16 bits superiores do primeiro bloco (2 bytes)
 - * Data e horário da última escrita (4 bytes)
 - * 16 bits inferiores do primeiro bloco (2 bytes)
 - * Tamanho (4 bytes)

* Windows 95

- * Entradas adicionais para nomes grandes antes da entrada basica
- * Entrda para nomes grandes
 - * Novas entrdas associadas ao mesmo arquivo
 - * Sequencia(1 byte)
 - * 5 caracteres (10 bytes)
 - * Atributo (1 byte) – valor inválido para DOS não compromete lookup an
 - * Zero (um byte)
 - * Checksum (1 byte)
 - * 6 caracteres (12 bytes)
 - * Zero (2 bytes)
 - * Dois caracteres (4 bytes)

Implementação de diretórios

Fonte: tannembaum

Unix

- * Tabela de I-nodes em local fixo na partição (local no superbloco)
- * localização do I-node a partir do número é trivial
- * Diretórios são todos arquivos normais
- * I-node para diretório raiz na primeira entrada
- * I-node contém toda a informação sobre o arquivo, não o diretório
- * Entrada
 - * Número do I-node (2 bytes)
 - * Nome (14 bytes)
- * Formato varia, mas sempre I-node e nome
- * Busca: I-node zero, carrega diretório, busca, pega inode do primeiro passo, e repete para cada passo do caminho (pg 507 Tanenbaum)

Implementação de diretórios

Fonte: tannembaum

* NTFS (Windows 2000 e depois)

- * Em Unix arquivo é conjunto de bytes
- * Nos sistemas windows modernos, conjunto de atributos
- * Cada atributo é um stream de bytes
- * Estrutura básica é o Master File Table (MFT)
- * MFT é um arquivo
- * Até 16 atributos, cada um com até 1kb
- * Atributos não residentes (um atributo pode apontar para um arquivo com mais atributos)

Administração de espaço

Fonte: tannebaum

Questão importante é como o espaço em disco é administrado

- * Sequência contígua de bytes
- * Sequência de blocos (não necessariamente contínuos)

Tamanho do bloco

* Candidatos

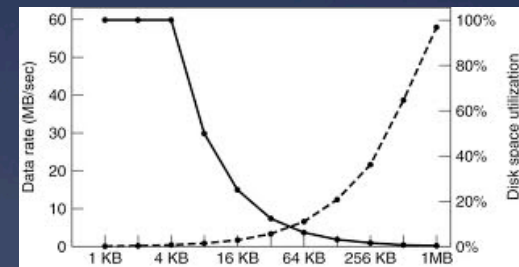
* Setor, trilha, cilindro

* Desperdício de espaço vs. Taxa de transferência de dados (latência de movimento de braço + rotação + taxa de transferência de cada bloco)

* Tamanho dos arquivos

* Unix (2055) mediana do tamanho dos programas é 2k

* Windows em sistemas científicos (Cornell), idem



<http://granite.sru.edu/~whit/cpsc464/Notes/ch4.html>

Administração de espaço

Fonte: tannembaum

↳ Espaço livre

- * Blocos com lista ligada de blocos livres
 - * -ultima entrada é próximo bloco da lista
 - * Apenas 1 bloco na memória de cada vez
 - * simples implementação
 - * Disco de 1Tb, 32 bits para número do setor, setor de 1Kb, lista livre ocupa até 4Gb
- * Vetor de bits
 - * 1 bit por bloco do disco
 - * Tamanho fixo
 - * Muito compacto, 1 bit por bloco
 - * Bloco de 1k contém indexação de 8 mil blocos (8Mb)
 - * Disco de 8Tb indexado com 125Mb de lista
 - * Implementação de busca mais complexa

Compartilhamento de arquivos

Fonte: tannembaum

Quando usuários diferentes compartilham arquivo, é conveniente se arquivo aparece simultaneamente em diretórios diferentes

Se é possível acessar arquivos por caminhos diferentes, estrutura de diretórios vira DAG (não mais árvore)

Novas conexões são chamadas de “links”

2 maneiras

- * Hard links – entradas em diretórios distintos compartilham descritor do arquivo
 - * Assim lista de blocos não faz parte da entrada do diretório
 - * UNIX
- * Links simbólicos – tipo especial de arquivo contendo caminho com nome da localização real do arquivo

Compartilhamento de arquivos

Fonte: tannembaum

* Problemas

- * Links simbólicos
 - * Sobrecarga de acesso – apenas na abertura
 - * Entradas órfãs de arquivos eliminados
- * Hard links
 - * Remoção de arquivo deve cuidar para que arquivo real seja removido apenas quando não há mais “links” (l-node tem contador de links)
 - * Usuário original continuará sendo cobrado pelo espaço
 - * Arquivo removido pelo dono continua sendo usado
 - * Ciclos
- * Geral
 - * Back-ups devem cuidar para não duplicar trabalho e reconstituição deve manter unicidade

Confiabilidade do sistema de arquivos

Fonte: tannembaum

Arquivos mais valiosos que o hardware

Localização dos setores ruins do disco

- * Hardware – setor do disco contém lista dos blocos ruins com lista dos blocos substitutos. Controlador usa substitutos automaticamente
- * Software – sistema de arquivos constrói arquivo contendo blocos ruins

Atualização atômica

- * Garantia que atualização parcial nunca acontece, mantendo arquivos sempre consistentes
- * Se atualização falhar antes de completar, basta rodar de novo
- * Discos com atualização atômica tolerante a falhas implementados usando 2 ou mais discos físicos por um lógico.

Confiabilidade do sistema de arquivos:RAID

- * Esquemas para operação de vários discos como se fosse apenas 1 criando “disco virtual
- * Original “RAID5”- 1978 (Norman Ken Ogushi, IBM – “System for recovering data stored in failed memory unit
- * Implementação por hardware (placa) ou software

RAID 0

- * 2 ou mais discos se juntam para criar um disco maior, sem redundância ou recursos de recuperação
- * Aumento de capacidade de um sistema de arquivos
- * Leitura e escrita mais rápidas

RAID 1

- * Disco espelho
- * Falha simples recuperada
 - * Uma das cópias de setor com paridade ruim
 - * Um disco falha
- * Leitura mais rápida

RAID 5

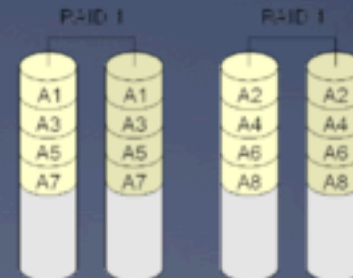
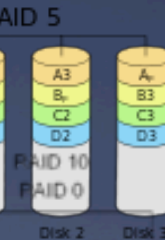
- * “stripping” de blocos com paridade distribuída
- * Conteúdo de bloco ausente pode ser recuperado com blocos restantes
- * Resiste a falhas em qualquer disco (mínimo 3 discos)
- * Leitura escrita mais rápidas ((n-1)X)
- * Performance com falha é reduzida
- * Disco com falha pode ser reconstituído

RAID 6

- * Semelhante a RAID5
- * Paridade Dupla distribuída (min. 4 discos)
- * Até duas falhas
- * Leitura e escrita mais rápida ((n-2) X)

Outros RAID

- * Bit stripping, byte stripping (RAID2, RAID3)
- * RAID5 com 2 paridades (RAID6)
- * RAID5 com 3 paridades (RAID7)
- * RAID5 com 4 paridades (RAID8)
- * RAID5 com 5 paridades (RAID9)
- * RAID5 com 6 paridades (RAID10, etc)



Fonte: wikipedia

Confiabilidade do sistema de arquivos

Fonte: tannebaum

Controle de Concorrência

- * Unix tradicional – requisição de leituras e escritas executadas na ordem em que chegam
- * Pode gerar problemas quando exclusão mútua é necessária
- * Solução mais comum são travas (“locks”)
 - * 1 por arquivo
- * Mesmos problemas que exclusão mútua (ex. Usuário “pifa”)
- * Diferença importante – ênfase nos dados (ao invés de código)

Confiabilidade do sistema de arquivos

Fonte: tannembaum

Transações

- * Travamento automático + atualização atômica
- * Begin_transaction
 - * Atualizações
- * End_transaction
- * Nada acontece até “end_transaction”

NTFS (<http://msdn.microsoft.com/en-us/magazine/cc163388.aspx>)

Confiabilidade do sistema de arquivos: cópias de segurança (backups)

Dois objetivos

- * Recuperação de desastres
- * Recuperação de erros

Potencialmente lentos e alto uso de espaço

O que recuperar?

- * Dump Físico vs Dump Lógico

Dump físico

- * Disco inteiro é copiado
- * Simples mas custoso
- * Cuidado com blocos inválidos
 - * Se mantidos pelo hardware, ok
 - * Se mantidos pelo SO, programa de backup deve ter acesso a estruturas e evitar copiá-los

Confiabilidade do sistema de arquivos

Fonte: tannembraum

Dump lógico

- * Muitos sistemas não fazem backup de executáveis, arquivos temporários, arquivos especiais (/dev/ é até perigoso)
- * Em geral é interessante especificar diretórios a serem guardados
- * Começa em um ou mais diretórios especificados e recursivamente percorre a estrutura de diretórios salvando os itens encontrados
- * não copiar arquivos especiais (pipes, etc.)

Dumps incrementais

- * Não tem sentido fazer novo backup de arquivos não mudados
- * Dump completo + incrementos (e.g. Mês + dia)
- * Complica recuperação
- * Mesmo diretórios não modificados devem ser salvos para facilitar recuperação
 - * Domingo - Full dump
 - * Segunda - Backup de /usr/local/ndr2/xpto/arq1.txt
 - * Terça - quero remover /usr/local/ndr2
 - * Quarta - como recuperar /usr/local/ndr2/xpto/arq1.txt? -> preciso recriar ndr2 e xpto

Confiabilidade do sistema de arquivos

Fonte: tannembaum

Outras questões

- * Unix pode ter “buracos” nos arquivos
 - * “open, write, seek, write”,
 - * core dumps tem espaço entre código e pilha
 - * não queremos “buracos” preenchidos na recuperação.

Cuidado com links para evitar duplicação (e loops)

Consistência do sistema de arquivos

- * Sistemas de arquivos lêem/criam blocos, podem modificá-los e depois são salvos
- * E se programa morre antes dos blocos serem salvos?
 - * Mais grave se bloco é i-node
- * Maioria dos SOs têm programas para verificar consistência do sistema de arquivos.
 - * Unix – fsck
 - * Windows – chkdsk

Consistência do sistema de arquivos: fsck

* Consistência de Blocos

- * Duas tabelas com contadores para cada bloco, inicializados com zero
 - * Quantas vezes um bloco está presente em um arquivo
 - * Quantas vezes um bloco está presente na lista livre
- * Programa lê todos os i-nodes e percorre lista de blocos
 - * Toda vez que bloco é encontrado atualiza primeira tabela
- * Programa percorre lista livre
 - * Toda vez que bloco é encontrado atualiza segunda tabela
- * Blocos bons => (1,0) ou (0,1)
- * Missing block (0,0) => adicionado à lista livre
- * Bloco com mais de uma ocorrência em lista livre (0,n) – reconstrói a lista livre
- * Blocos presentes em mais de um arquivo (n,0)
 - * Situação mas grave – deve gerar depois (n,m) ou (0,m)
 - * Faz cópia do bloco e insere a cópia em um dos arquivos
 - * Quase com certeza um bloco está corrompido

Consistência do sistema de arquivos: fsck

- * Consistência de arquivos e diretórios
 - * Tabela de contadores, um por arquivo
 - * Percorre árvore de diretórios
 - * Incrementa contador toda vez que um i-node é encontrado (lembrem-se que arquivos podem ser apontados por mais de um diretório por “hard links”).
 - * Compara contadores com número de links nos i-nodes respectivos
 - * Link count muito alto
 - * Sistema não atualizou contador após remoção
 - * Arquivo ficaria no sistema mesmo após dever ser removido
 - * Atualiza contador de links
 - * Link count muito baixo
 - * Erro mais grave, provocaria remoção prematura do arquivo
 - * Atualiza contador de links

Consistência do sistema de arquivos: fsck

- * Outras ocorrências suspeitas que podem ser reportadas
 - * Diretórios com muitos arquivos (e.g. Mais de mil)
 - * Permissões estranhas (e.g. 0007)
 - * Arquivos em diretório de usuário mas pertencentes ao root e com setuid ligado

Consistência do sistema de arquivos: fsck

- * Outras ocorrências suspeitas que podem ser reportadas
 - * Diretórios com muitos arquivos (e.g. Mais de mil)
 - * Permissões estranhas (e.g. 0007)
 - * Arquivos em diretório de usuário mas pertencentes ao root e com setuid ligado

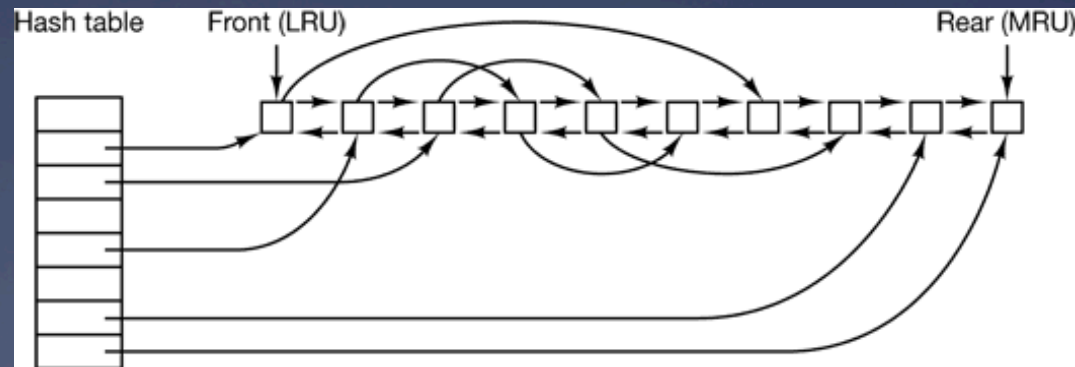
Performance do sistema de arquivos

* CACHES

- * Blocos sempre carregados na área de cache antes de serem lidos
- * Tabela de hash (dispositivo+número do bloco) e lista ligada
- * Reposição semelhante aos algoritmos de memória virtual
 - * Como caches lidas com menos frequência (e sempre chamada de sistema) é viável o uso de LRU
 - * Hash+ lista livre (quando bloco usado vai para fim)

* Considerações adicionais

- * Blocos mais antigos em geral modificados
- * i-nodes são importantes para consistência do sistema



Fonte: livro tanenbaum

Performance do sistema de arquivos

* Caches

- * Blocos devem ser divididos em categorias (i-nodes, blocos indretos, diretórios, blocos de dados completos, blocos de dados parciais)
 - * Blocos que não devem ser usados tão cedo vão na frente, outros no final (como blocos que são de arquivos abertos para escrita e que estão parcialmente completos)
- * Blocos que são essenciais para consistência do sistema (i.e. Todos menos os blocos de dados)
 - * Devem ser implementados write-through.
- * Mesmo assim, não deveríamos deixar blocos modificados sem serem escritos por muito tempo
 - * UNIX – syncs periódicos (30 segundos?)
 - * Windows – write-through cache. (i.e. USB drives – FAT – em geral são seguros)
 - * Porque? (Unix – discos rígidos, Windows – disquetes que podem ser removidos pelo usuário)

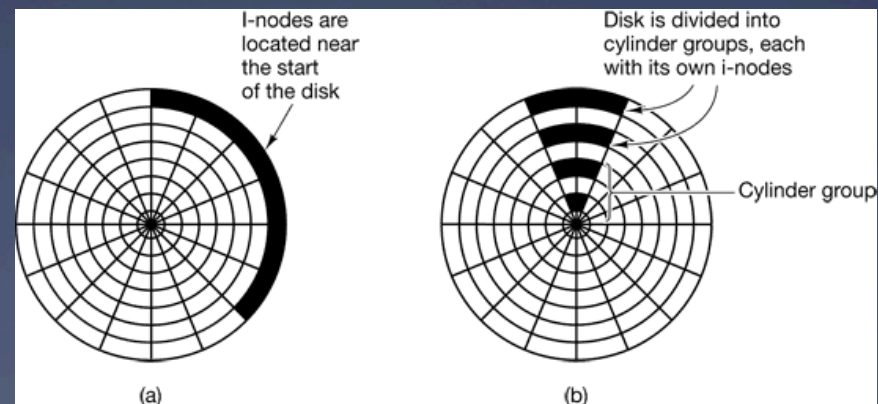
Fonte: livro tanenbaum

Performance do sistema de arquivos

- * Caches: Leitura preventiva de blocos
 - * Tentar colocar blocos na cache antes de sua leitura/escrita ser requisitada
 - * Ex. Maioria dos arquivos lido sequencialmente
 - * FS pode ver, quando block K é requisitado, se bloco K+1 está lá, se não estiver, pode requisitar leitura preventiva
 - * Como? – supor sequencial, quando seek é chamado, supor randomico (sem leitura preventiva),

Performance do sistema de arquivos

- * Reduzindo movimento do braço do disco
 - * Colocar blocos que podem ser utilizados sequencialmente próximos um do outro, de preferencia no mesmo cilindro
 - * E.g. Se lista livre como bitmap, podemos pegar bloco o mais próximo possível do último escrito
 - * Alocar mais de um bloco de cada vez para arquivo
 - * I-nodes + arquivos: distribuir i-nodes por grupos de cilindros e alocar blocos dos arquivos preferencialmente próximos.
 - * Variação – inodes no meio do disco



Fonte: livro Tanenbaum

Sistemas de arquivos estruturados como logs

- * Log-structured file systems –LSU (Berkeley)
 - * Aumento da diferença de performance motivou aumento das caches
 - * Enorme aumento das memórias
 - * Caches devem conter porcentagem cada vez maior dos acessos
 - * Maioria dos acessos serão para escrita
 - * Porém – escritas em geral são pequenas (pense em um print)
 - * Ex: criar arquivo em Unix
 - * O i-node do diretório, o bloco do diretório, o i-node do arquivo e o arquivo precisam ser gravados no primeiro write
 - * Adiar estas escritas pode comprometer consistência.

Fonte: livro tanenbaum

Sistemas de arquivos estruturados como logs

- * Log-structured file systems –LSU (Berkeley)
 - * Sistemas de arquivos são estruturados como buffers circulares
 - * Sist. De arquivos organizado como segmentos
 - * Periodicamente (ou quando necessário) todas as escritas pendentes são coletadas em um segmento que é escrito de uma vez só e de maneira contínua no disco.
 - * Cada segmento inclui descritor indicando o que está armazenado no segmento
 - * Se segmento tiver aprox. 1Mb, em geral se usa completamente a capacidade de fluxo do disco
 - * I-nodes ainda existem, mas distribuídos no log
 - * Uma vez encontrado o i-node, localizar blocos é feito da maneira usual
 - * Sistema mantém tabela de i-nodes no disco com cache na memória
 - * Abertura de arquivo agora envolve localizar segmento do i-node
 - * Blocos continuam no disco, mas em segmentos.
 - * Neste esquema segmentos podem conter blocos obsoletos.
 - * Cleaner
 - * Inicialmente verifica no segmento quais i-nodes e arquivos estão lá (no sumário)
 - * Verifica mapa dos i-nodes para descobrir quais i-nodes do segmento ainda válidos e quais blocos ainda ativos.
 - * Copia informação ainda válida em novo segmento, livra o segmento atual.
 - * Para reconstruir sistema de arquivos após crash, basta iniciar do último ponto consistente do log
 - * Para livrar espaço atrás: pula versões antigas, se encontra última versão, move para o início.

Fonte: livro tanenbaum

Segurança

- * Segurança vs. Proteção
 - * Segurança é o problema geral
 - * Proteção são os mecanismos para garantir a informação no sistema
- * Objetivos
 - * Confidencialidade dos dados
 - * Sistema deve garantir que dados de usuário só serão conhecidos com sua permissão
 - * Integridade dos dados
 - * Usuários não autorizados não devem poder modificar dados sem permissão (inclui remoção e adição de dados)
 - * Disponibilidade
 - * Ninguém deve conseguir perturbar o sistema a ponto de tornar acesso aos dados impossível (denial of service)
 - * Privacidade
 - * Proteger usuários do uso indevido de informações a seu respeito (padrões de uso, etc.)

Fonte: livro tanenbaum

Segurança

- * Intrusos
 - * Usuários não autorizados no sistema
 - * Passivos vs. Ativos: apenas olham os dados ou tentam modificá-los
- * 4 tipos de intruso
 - * Curiosos ocasionais: pessoas com pouco conhecimento que xeretam arquivos não protegidos (e.g. em Unix é comum arquivos serem criados com permissão de leitura universal)
 - * “xeretas descolados”: pessoas que são usuários do sistema que consideram desafio pessoal conseguir quebrar a segurança da informação. Em geral são muito habilidosos e dispostos a usar muito tempo na tarefa
 - * Estelionatários: programadores em bancos. Ameaças incluem truncar (e não arredondar) transações, uso de fundos de contas inativas, chantagem
 - * Espionagem comercial ou militar: esforços sérios e muito bem financiados para roubar programas, informações, tecnologia, etc. Iniciativas podem envolver escuta de linhas de transmissão, antenas para detectar sinais eletromagnéticos do computador
- * Esforço dedicado à segurança deve ser proporcional à qualidade do possível intruso e do potencial dano a ser infringido.

Fonte: livro tanenbaum

Segurança

- * Programas maliciosos
 - * Virus: pedaço de código que pode se reproduzir introduzindo uma cópia de si mesmo em outros programas e causar vários danos
 - * Destruição de informação
 - * Envio de informação privada
 - * Inserção de informação
 - * 'Denial of service': consome recursos do sistema a ponto de torná-lo inutilizável (CPU, disco, canais de comunicação, serviços...)
 - * DDOS (Distributed Denial of Service): vírus tem duas fases, reprodução, onde se espalha pela rede, e ataque. Em data pré determinada todas as cópias "acordam" e iniciam ataque (ex. Webpage específica).
 - * Key logger: monitora teclado procurando padrões (e.g. Conta de email seguida de sequencia de caracteres.....)

Fonte: livro tanembaum

Segurança

* Programas maliciosos

- * Worm: diferente do vírus, que está incluído em outro programa/ arquivo, é um programa independente
 - * Worm pode se colocar no “startup diretctory”
- * Cavalo de Troia: programa que desempenha uma função mas que esconde funcionalidade secundária (lançar um worm, inserir vírus, bisbilhotar sistema de arquivos, etc. Em geral baixados voluntariamente)
- * Bomba lógica: programa elaborado por programador do sistema que exigem senha periódica. Se não receber a senha a bomba é “detonada” ações podem envolver todo o tipo de dano como remoção de arquivos (aleatória ou não), encriptação de arquivos, mudanças difíceis de detectar em programas importantes, etc.
- * Spyware: obtido visitando-se sites na web,
 - * Cookies usado para identificar o usuário e coletar informações sobre quais sites ele visitou, etc.
 - * Plugins maliciosos instalados pelo usuário

Fonte: livro tanenbaum

Segurança: um pouco de história

- * Multics (cavalo de tróia)
 - * Não havia nenhuma segurança pra processamento BATCH (sistema foi feito para usuários interativos, processamento BATCH adicionado depois)
 - * Qualquer um podia submeter programa batch que lia conjunto de cartões (sim, cartões de computador, faz tempo....) e copiava seu conteúdo no diretório de QUALQUER usuário
 - * Para roubar arquivos de um usuário era só modificar fontes de um editor de texto, recompilá-lo e substituir código objeto do editor original no diretório bin do usuário
 - * seria possível fazer isso no seu diretório? – CUIDADO COM PERMISSÕES DE DIRETÓRIO

Fonte: livro tanenbaum

Segurança: um pouco de história

- * TENEX (popular nos sistemas DEC-10)
 - * Um sistema com memória paginada, mas para auxiliar usuários em monitoração era possível mandar o S.O. Chamar uma rotina de usuário a cada falha de página
 - * Acesso a arquivos era controlado por senhas, que eram verificadas um caracter por vez, parando no primeiro caracter inválido.
 - * Para violar senhas o usuário deveria colocar uma senha com o primeiro caracter em uma página da memória e caracteres seguintes na próxima página.
 - * Em seguida usuário deveria certificar-se que próxima página não estava na memória (bastava fazer várias referências a outras páginas distintas)
 - * Quando programa tentava abrir um arquivo, se primeiro caracter estivesse errado, sistema retornaria mensagem de senha inválida., senão quando ocorresse a falha de página a rotina do usuário informaria o fato
 - * No máximo 128 tentativas para adivinhar cada caracter da senha
 - * $N \cdot 128$ tentativas (ao invés de 128^N)

Fonte: livro tanenbaum

Segurança: um pouco de história

* OS 360

- * Era possível começar uma transferência de dados de uma fita e continuar computando enquanto transferência era feita em paralelo
- * O truque era começar uma transferência e então emitir uma chamada de sistema para ler/modificar um arquivo dando seu nome e a senha, fornecidos na área de usuário
- * Sistema de arquivos primeiro verificava se a senha estava correta e, em seguida, lia o nome do arquivo a ser aberto (não guardava internamente o nome do arquivo).
- * Truque estava em sincronizar transferência da fita para que o nome do arquivo fosse substituído na memória pelo nome de um outro arquivo qualquer no intervalo entre a verificação da senha e a releitura do nome.
- * Cronometragem correta não era muito difícil de se conseguir e podia ser tentada automaticamente por programas

Fonte: livro tanenbaum

Segurança: um pouco de história

* Unix:

- * Arquivos de senhas tinham leitura pública. Algoritmo “one way” usado para encriptar senhas. Usuários podiam olhar arquivos e testar listas de senhas prováveis aplicando algoritmo e vendo se codificação da senha batia (nomes, dicionários, datas...)
- * Lpr tem uma opção para remoção do arquivo após sua impressão. Versões iniciais do Unix permitiam que qualquer um imprimisse e depois removesse arquivo de senhas.
- * Usuário podia linkar arquivo com nome “core” no seu diretório ao arquivo de senhas. O intruso então forçava um “core dump” de um programa com SETUID. O sistema escrevia o dump no arquivo “core”, substituindo o arquivo de senhas do sistema
- * Mkdir foo : mkdir é um programa com SEUID cujo dono é o root. Mkdir primeiro cria o i-node para foo, e após isso muda o “dono” do arquivo do UID efetivo (root) para o real (usuário). Se o sistema fosse lento era possível ao usuário remover o i-node de “foo” após o mknod, mas antes do *chown* e trocá-lo por outro no arquivo de diretório. Invasor podia assim se apropriar de arquivos de sistema (login, senhas, etc.). Colocando programas numa shellscript tentativa podia ser repetida até funcionar

Fonte: livro tanenbaum

Segurança: um pouco de história

- * 1988 – Robert T. Morris (Cornell) – verme com 2 programas: o comando de bootstrap e o verme em si.
 - * Programa de bootstrap tinha 99 linhas em C, compilado e executado no sistema sob ataque
 - * Uma vez rodando, conectava-se à máquina de origem e carregava o verme principal, executando-o
 - * Após algumas etapas de camuflagem consultava tabelas de roteamento de seu hospedeiro e tentava espalhar o comando de bootstrap nestas.
 - * Na máquina hospedeira tentava quebrar senhas de usuário, usando técnicas de artigo clássico da área de 1979. A cada usuário invadido, espalhava-se pelas máquinas onde este usuário tinha contas.
 - * Lotou a rede em 1988 e derrubou inúmeros sistemas
 - * Defesa argumentou que não foi malicioso e que era apenas um experimento (!!!)
 - * Motivou a criação do CERT (Computer Emergency Response Team)

Fonte: livro tanenbaum

Segurança: ataques

- * Todo o S.O. tem lacunas. Maneira normal de se testar é uso de times de invasão (peritos contratados ou desafio)
- * Ataques mais comuns
 1. Requisitar páginas de memória, espaço em disco e simplesmente investigar o conteúdo: muitos sistemas não apagam o conteúdo, apenas reutilizam
 2. Tentar chamadas de sistema ilegais, ou chamadas legais com parâmetros legais, mas improváveis.
 3. Faça o login e então pressione DEL, RUBOUT, BREAK no meio da sequência de login, o sistema pode matar o programa de login e deixá-lo entrar
 4. Tente modificar estruturas do SO mantidas no espaço de usuário (caso existam)
 - * Em alguns sistemas, por exemplo, ao abrir arquivo, vários parâmetros mantidos em área de usuário. Modificação destes parâmetros DURANTE a leitura pode afetar segurança

Fonte: livro tanenbaum

Segurança: ataques

* Ataques mais comuns

1. Criar programa also de login que registra username e senha do usuário antes de fazer um login.
2. Olhe nos manuais onde diz: “não faça X” e tente todas as possíveis variações de X.
3. Porta dos fundos: covença (\$\$\$\$) programador de sistema para liberar controle de segurança de sua conta
4. Suborno (diretor do centro de computação, secretária)

Fonte: livro tanenbaum

Segurança: princípios de segurança

- * Design de um sistema deve ser público:
 - * Supor que o intruso não vai saber como funciona o sistema é uma ilusão (pior ainda nos dias de hoje, com web)
- * A ação padrão deve ser sempre negar o acesso. Erros nos quais um acesso autorizado é negado são relatados mais rapidamente do que casos de permissão de acesso não autorizado
- * Faça verificação contínua de permissões
 - * Ex. Usuário pode abrir arquivo e mantê-lo aberto por semanas, mesmo depois que o dono mudar as permissões.
- * A cada processo deve ser dado o mínimo possível de privilégios. Se um editor só puder acessar os arquivos que estão sendo editados, por exemplo, ataques tipo “cavalo de tróia” são impedidos
- * Mecanismo de proteção deve ser simples, uniforme e implantado nas camadas mais internas do sistema
- * Mecanismos devem ser de fácil aceitação: usuário não adotam procedimentos muito complexos e mesmo assim ficam contrariados quando acontece acesso não autorizado.

Fonte: livro tanenbaum

Segurança: controle de acesso

* Senhas

- * Fácil de quebrar pois difícil forçar usuário a escolher boas senhas
- * Bloqueio de senhas fáceis
- * Substituição periódica obrigatória
- * Variação: perguntas e respostas armazenados no sistema
- * Variação: resposta a desafio – algoritmos pré escolhidos (ex x^2), computador faz perguntas periódicas e espera resposta da aplicação do algoritmo.

* Identificação física

- * Digitais
- * Comprimento de dedos
- * Identificação de retina

Fonte: livro tanenbaum

Segurança: contramedidas

- * Delays no processo de login dificultam tentativas exaustivas. Delays crescentes melhor
- * Registro de todos os logins: ataques podem ser detectados por logins falhos sucessivos de alguma fonte
- * Armadilhas:
 - * login fácil com senha fácil (silva, 1234): login nestas contas alertam administradores que podem monitorar uso
 - * Bugs fáceis de achar

Fonte: livro tanenbaum

Mecanismos de Proteção: domínios de proteção

- * Domínio = (objeto, direitos de acesso)
- * Objeto: entidade do S.O. (arquivo, etc.)
- * Direitos: (permissões para efetuar cada operação relativa ao objeto)
- * A cada momento processo executa em um domínio de proteção:
 - * Há coleção de objetos que ele pode acessar, cada um com conjunto de direitos.
 - * Domínio pode ser mudado
 - * Dois processos com mesmo domínio tem mesmo acesso
- * Ex: Unix:
 - * domínio é UID e GID.
 - * Setuid muda domínio
- * Conceitualmente existe grande “matriz de domínios”

Fonte: livro tanenbaum

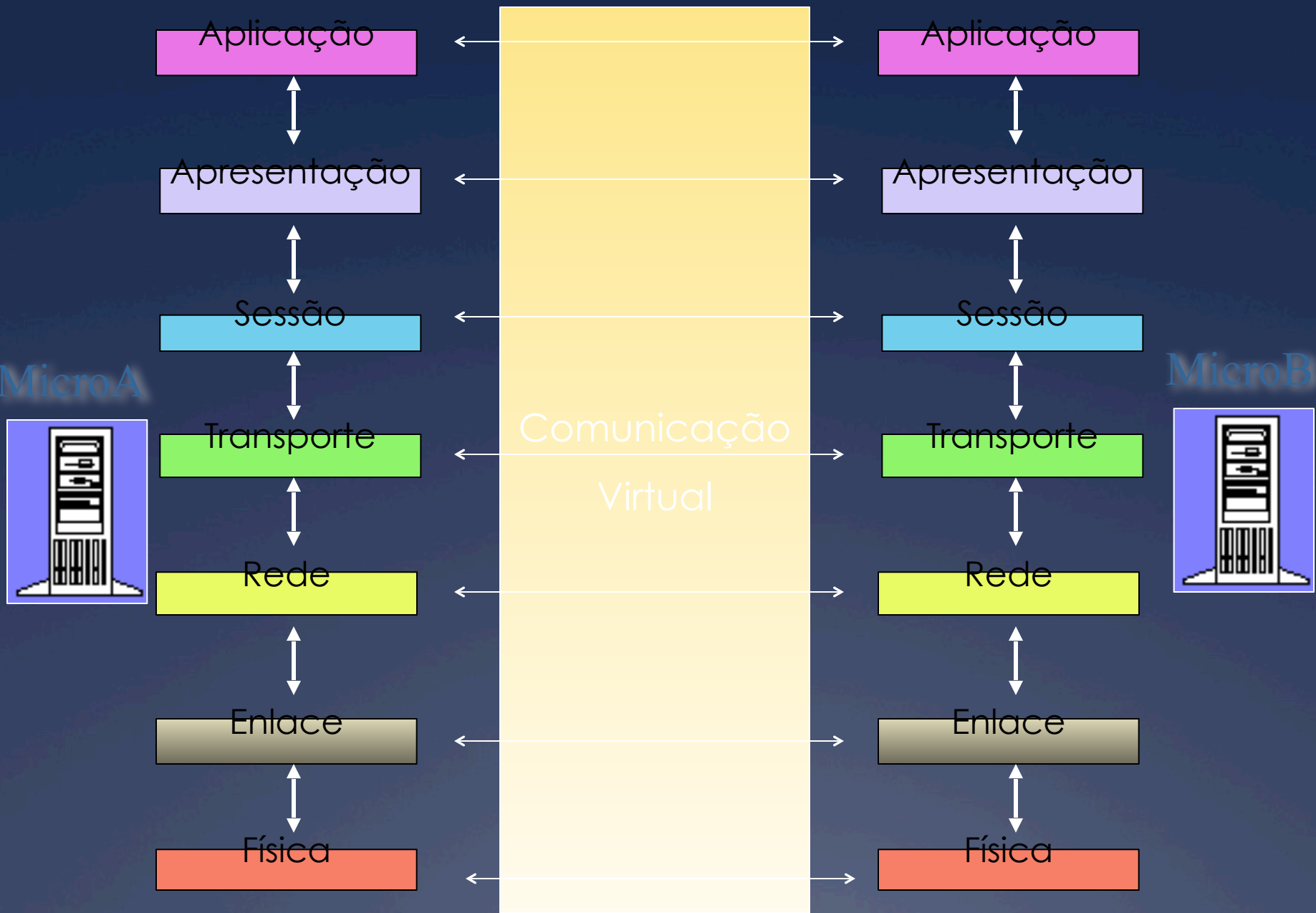
TCP/IP

O MODELO DE REFERÊNCIA OSI

- tráfego na rede é gerado quando ocorre uma solicitação na rede. A Solicitação tem de ser alterada daquilo que o usuário vê para um formato que possa ser utilizado na rede.
- Essa transformação é possível por meio do modelo de referência do Open Systems Interconnection (OSI), desenvolvido pela International Organization for Standardization (ISO)
- O tráfego da rede é gerenciado na forma de pacotes de dados:
 - Um pacote de dados é a informação de um usuário transformado em um formato entendido pela rede.
- O modelo OSI tem 07 Camadas, e é utilizado como uma diretriz pelos desenvolvedores de programas de rede.
- Embora vários fabricantes alterem o modelo, ele ainda é a fundação utilizada no desenvolvimento.

O Modelo OSI

- As 7 camadas do modelo OSI, operam como blocos de construção para os pacotes de dados.
- Cada camada adicionará informações ao pacotes de dados, contudo o pacote de dados não é alterado.
- As informações adicionadas a um pacote são chamadas de cabeçalho.
- Cabeçalho de uma camada é simplesmente a informação que detalha o formato do pacote de dados.
 - Esse cabeçalho é recebido na camada correspondente do cliente receptor e é utilizado para entender o formato do pacote.
- Todas as camadas se comunicam com as camadas que lhe são adjacentes.
 - Isso significa que qualquer camada pode ser comunicar com a camada diretamente acima ou abaixo dela..



O MODELO OSI

O Modelo OSI: função das camadas

- **Aplicação:** faz a interface entre o protocolo de comunicação e o aplicativo.
- **Apresentação:** codifica o dado para um formato entendido pelo protocolo. É nesta camada que pode, casualmente, ser feita a compressão de dados e criptografia.
- **Sessão:** permite que duas aplicações de computadores diferentes estabeleçam uma sessão de comunicação.
- **Transporte:** é responsável pela fragmentação no transmissor e montagem no receptor, dos dados em pacotes.
- **Rede:** converte os endereços lógicos em endereços físicos e determina qual rota vai ser seguida pelos pacotes.
- **Link de dados (Enlace):** adiciona informações como endereço da placa de rede de origem, de destino, dados de controle e o CRC, um esquema para detecção de erros na transmissão.
- **Física:** transforma a informação em sinais compatíveis com o meio por onde os dados devem ser transmitidos, elétrico ou óptico, por exemplo.



Remetente



Receptor

Dados



Dados



+ Cabeçalho de aplicativo

Dados

Aplicação



Dados

- Cabeçalho de aplicativo

+ Cabeçalho de Apresentação

Dados

Apresentação

Dados

- Cabeçalho de Apresentação

+ Cabeçalho de Sessão

Dados

Sessão

Dados

- Cabeçalho de Sessão

+ Cabeçalho de Transporte

Dados

Transporte

Dados

- Cabeçalho de Transporte

+ Cabeçalho de Rede

Dados

Rede

Dados

- Cabeçalho de Rede

+ Cabeçalho de Enlace

Dados

Enlace

Dados

- Cabeçalho de Enlace

+ Cabeçalho Físico

Dados

Física

Dados

- Cabeçalho Físico

FLUXO DE DADOS UTILIZANDO O MODELO OSI

A Camada de Aplicativo

- A **Camada 7**, a mais alta no modelo OSI é a camada de Aplicativo.
- É responsável por interagir com o aplicativo do usuário;
 - ela aceita os dados do aplicativo a partir do aplicativo de software
 - fornece o serviço de aplicativo de rede que é responsável pela solicitação do usuário.
- Alguns exemplos de transformação de dados na camada de Aplicativo incluem o seguinte:
 - Envio de um e-mail, a camada de aplicativo, fornecerá acesso ao serviço do Simple Mail Transfer Protocol (SMTP)
 - Uma transferência de arquivos pode ser realizada utilizando o File Transfer Protocol (FTP). O Serviço de FTP é uma responsabilidade da camada de Aplicativo.

A Camada de Apresentação

- A **Camada de Apresentação** tem como principal função a conversão, principalmente de linguagens.
- A linguagem utilizada em redes é o protocolo.
- Se 2 clientes utilizarem protocolos diferentes é necessário a conversão, que é responsabilidade da Camada de Apresentação.
- A Camada de Apresentação também é responsável pelo gerenciamento de todos os dados. Ela fornece a conversão de um conjunto de caracteres, criptografia de dados e compactação de dados. Também é responsável pelo redirecionamento das solicitações de entrada e de saída.
- Tanto a Camada de Aplicativo, quanto a de Apresentação, fornecem serviços de rede, mais cada uma um tipo de serviço diferente, o termo serviço de rede, pode ser confundido, para tanto lembre-se que:
 - Os serviços de aplicativos de rede são invocados pelo usuário e se comunicam diretamente com a camada de aplicativo. Um e-

A CAMADA DE SESSÃO

- Também conhecida como Camada de conexão de Aplicativo
- permite que aplicativos idênticos que operam em dois clientes diferentes comuniquem-se.
- Estabelece uma conexão virtual, com base no nome do usuário, nome de computador ou credenciais de rede do cliente.
- Faz o gerenciamento de uma conexão virtual emitindo pontos de verificação nos dados que ela recebe.

A camada de Transporte

- Responsável por verificar erros e fazer o controle do fluxo de dados
- Nesta camada dois protocolos são em geral usados para transmissão de dados: Transmission Control Protocol (TCP) e User Datagram Protocol (UDP)
- **TCP (Transmission Control Protocol)** é orientado para conexão, enquanto o **UDP (User Datagram Protocol)** é
 - TCP rastreia todos os dados enviados, exigindo confirmação para cada octeto (geralmente).
 - O UDP não usa nenhum reconhecimento e geralmente é usado para protocolos em que alguns datagramas perdidos não importam.
- Por causa de confirmações, o TCP é considerado um protocolo confiável de transferência de dados. Ele garante que nenhum dado seja enviado para o aplicativo da camada superior que esteja fora de ordem, duplicado ou que tenha partes faltantes. Pode até gerenciar transmissões para tentar reduzir o congestionamento.
- Os principais usos do UDP incluem anúncios de serviço, como atualizações de protocolo de roteamento e disponibilidade de servidor, aplicativos multicast um-para-muitos e aplicativos de streaming, como voz e vídeo, O datagrama é muito menos importante

A camada de Rede

- responsável pelo endereçamento e roteamento da rede.
- O Internet Protocol (IP) é comumente utilizado para pacotes de endereçamento e especificará o endereço de origem (remetente) e endereço de destino (receptor) dos pacotes de dados.
- O Roteamento é realizado na camada de rede para determinar o melhor caminho ou rota do destino.
- Protocolos comuns de roteamento que operam na camada de Rede incluem o Routing Information Protocol (RIP), Open Shortest Path First (OSPF) e Border Gateway Protocol (BGP).

A camada de enlace

- dividida em duas subcamadas: **Controle de Enlace Lógico e Controle dos Meios Físicos de Acesso** (Central)
- Subcamada de Controle de Enlace Lógico é responsável por inserir um cabeçalho e um trailer (CRC - Cyclical Redundancy Check) que realiza um cálculo de paridade do pacote de dados e coloca o resultado no trailer.
 - Quando o receptor recebe os pacotes, um CRC faz uma comparação ao CRC do remetente.
 - Se os resultados forem consistentes então são válidos, e são passados à próxima camada.
 - Caso contrário serão descartados
- A subcamada de Controle dos meios físicos (MAC - Media Access Control) coloca o endereço físico da placa de rede no cabeçalho que é adicionado ao pacote de dados.
 - Um endereço MAC é um número hexadecimal único de 12 algarismo que está em todas as placas

A camada Física

- Responsável principalmente por colocar os dados brutos na rede.
- Os dados brutos são representados por um formato binário ou conjunto de 1s e 0s.
- Referida como Camada de Hardware, estabelece e mantém conexões entre o remetente e o receptor.
- Em resumo define como um cabo de rede é anexado à placa de rede e como os dados devem ser formatados para a transmissão.

Os Protocolos TCP/IP

- O conjunto de protocolos do TCP/IP é baseado em quatro camadas.
- Estas quatro camadas juntas definem o modelo de rede do DOD (Department of Defense).
- Cada camada é responsável por executar funções específicas de rede e envolve diversos protocolos.
- Abaixo encontramos o modelo do TCP/IP sua interligação com o modelo OSI, onde vemos equivalências de funções entre as camadas:
- Veja agora, a quais camadas pertencem os protocolos e topologias de comunicação:

Modelo OSI Versus TCP/IP

Aplicação

Apresentação

Sessão

Transporte

Rede

Enlace

Física

Aplicação

Aplicativos que funcionam dentro da Internet : HTTP , FTP , TELNET, GOPHER.

Transporte

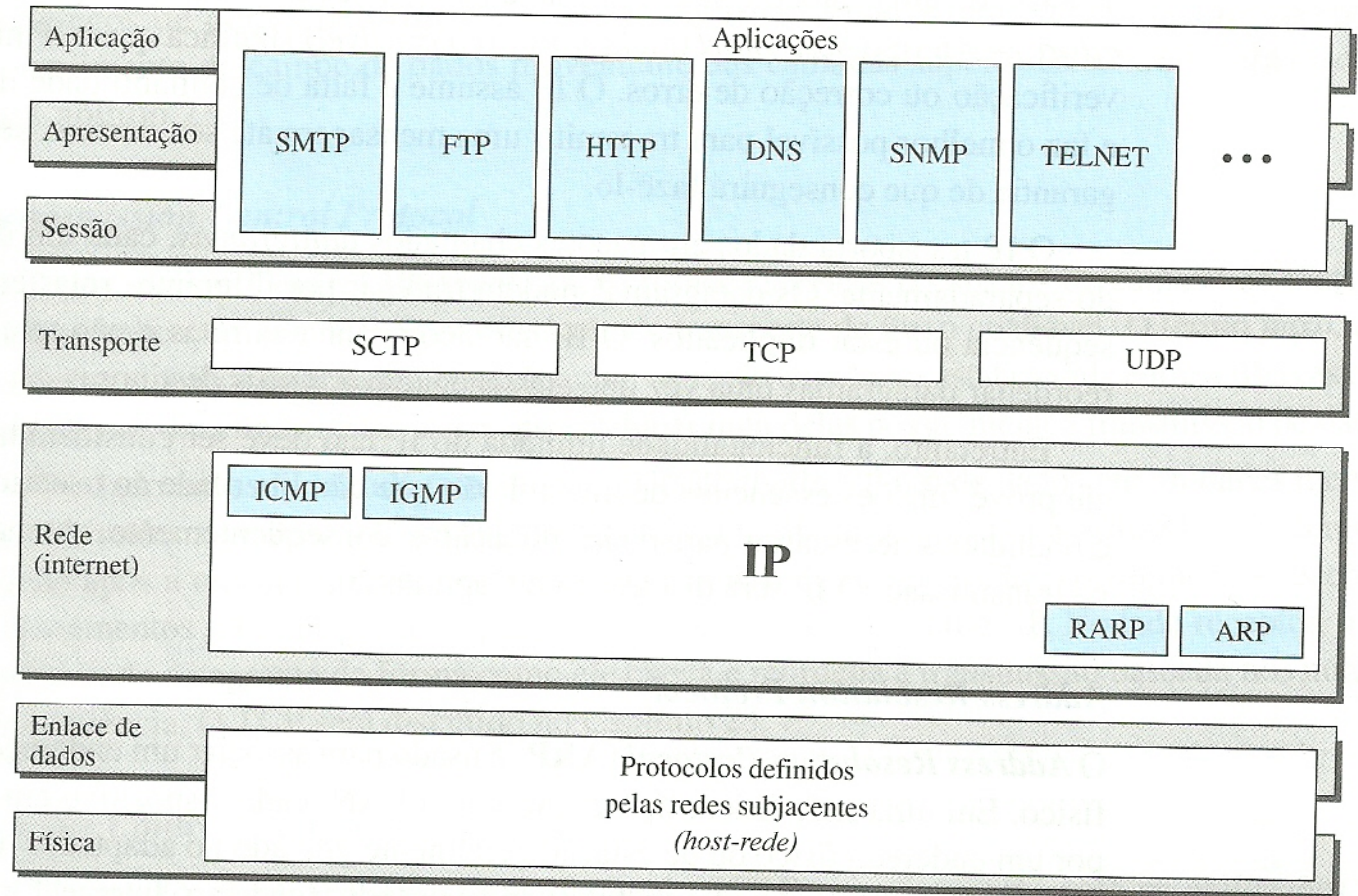
TCP e o UDP

Inter-Redes / Internet

IP , o ICMP, o ARP e o RARP

Interface com Rede

Gateways ou roteadores.

Figura 2.16 Modelo OSI e TCP/IP

A CAMADA DE INTER-REDE / INTERNET

A Camada Inter-Rede funciona quase da mesma maneira como a Camada de Rede do Modelo OSI. **É a principal responsável pelo endereçamento e roteamento de rede.** Além disso, essa camada é responsável pela **fragmentação do pacote.** OS pacotes de dados são **montados e remontados para transmissão nessa camada.**

Vários protocolos operam na camada de Inter-Rede, mais os mais comuns são:

- Internet Protocol (IP)
- Internet Control Message Protocol (ICMP)
- Address Resolution Protocol (ARP)
- Internet Group Management Protocol (IGMP)

O Protocolo ARP

- * *Address Resolution Protocol* ou ARP é um protocolo usado para encontrar um endereço da camada de enlace (ETHERNET, por exemplo) a partir do endereço da camada de rede (como um endereço IP).
- * O emissor difunde em *broadcast* um pacote ARP contendo o endereço IP de outro host e espera uma resposta com um endereço MAC respectivo.
- * Cada máquina mantém uma tabela de resolução em cache para reduzir a latência e carga na rede.
- * O ARP permite que o endereço IP seja independente do endereço Ethernet, mas apenas funciona se todos os *hosts* o suportarem.
- * O ARP foi implementado em vários tipos de redes; não é um protocolo restrito a redes IP ou Ethernet e pode ser utilizado para resolver endereços de diferentes protocolos de rede.
- * Porém devido a prevalência de redes IPv4 e Ethernet, ARP é utilizado primordialmente para traduzir Endereço IP para Endereço MAC. Também é utilizado em outras tecnologias de IP sobre LAN, como Token Ring, FDDI ou IEEE 802.11, e para redes IP sobre ATM.

Protocolo RARP

- * **Reverse Address Resolution Protocol (RARP)** ou **Protocolo de Resolução Reversa de Endereços** associa um endereço MAC conhecido a um endereço IP.
- * Permite que os dispositivos de rede encapsulem os dados antes de enviá-los à rede.
- * Um dispositivo de rede, como uma estação de trabalho sem disco, por exemplo, pode conhecer seu endereço MAC, mas não seu endereço IP.
- * O RARP permite que o dispositivo faça uma solicitação para saber seu endereço IP. Os dispositivos que usam o RARP exigem que haja um servidor RARP presente na rede para responder às solicitações RARP.
- * Os protocolos ARP e RARP encontram-se na camada de nível três (rede) do modelo OSI e na camada de nível um (link layer) do modelo TCP/IP.

Protocolo ICMP

- * ICMP, sigla para o inglês *Internet Control Message Protocol*, é
 - * utilizado para fornecer relatórios de erros à fonte original.
 - * Qualquer computador que utilize IP precisa aceitar as mensagens ICMP e alterar o seu comportamento de acordo com o erro relatado.
 - * Os *gateways* devem estar programados para enviar mensagens ICMP quando receberem datagramas que provoquem algum erro.
- * As mensagens ICMP geralmente são enviadas automaticamente em uma das seguintes situações:
 - * Um pacote IP não consegue chegar ao seu destino (i.e. Tempo de vida do pacote expirado)
 - * O Gateway não consegue retransmitir os pacotes na frequência adequada (i.e. Gateway congestionado)
 - * O Roteador ou Encaminhador indica uma rota melhor para a máquina a enviar pacotes.

Protocolo ICMP

Algumas das possíveis mensagens são:

- Rede fora de alcance;
- Nó fora de alcance;
- Porta fora de alcance;
- Nó desconhecido;
- Rede destino desconhecida;
- Tempo de vida do pacote excedido;
- Pedido de eco (ping);
- Resposta de eco (pong);
- O protocolo ICMP não detecta erros, leva a informação apenas ao seu destino, não aos nós intermediários.

Camada de rede PROTOCOLO IP

Roteia pacotes de um nó para o outro, na mesma rede ou em redes distintas.

Usa comutação por pacotes com datagrama não-confiável. Ou seja, não garante a entrega dos pacotes e é não-orientado à conexão.

Define também o endereçamento dos nós, o endereço IP.

IPv4 - 4 bytes (octetos)

IPv6 (em fase de adoção)

1) Endereços IP

O endereço IP é identificação de um equipamento conectado a Internet. Todos os equipamentos devem ter um endereço IP associado e único que será utilizado na comunicação entre os equipamentos. A definição de um endereço IP segue uma série de especificações que são definidas pela NIC (Network Information Center), que atribui e controla os endereços IP pelo mundo para garantir a segurança e unicidade dos endereços.

2) Formato e Categorias de Endereço IP

O endereço IP é constituído por 4 octetos (4 grupos de 8 bits) que servem para identificar a rede (Net ID) a qual o equipamento pertence e o próprio equipamento ou host (HOST ID).

- Existem 5 classificações para os endereços IP: Classe A, B, C, D, E. Porém utilizamos apenas os endereços de classe A, B e C.
- que os diferencia é a divisão do número de octetos que serão utilizados para identificação de rede e para identificação de host:
- Atualmente, a grande maioria de endereços IP são de Classe C. Utilizando o conceito de sub-rede a capacidade de combinações numéricas para formação dos endereços aumenta consideravelmente.
- Apesar de serem configurados através de números binários (0 e 1), os endereços IP são apresentados com notação decimal, como: 200.246.128.241, 192.168.10.123, etc.

Divisão em classes dos endereços IP

- * Cada classe de rede permite um número fixo de hosts.
- * **Rede classe A,**
 - * o primeiro octeto é atribuído à rede, restando os três últimos octetos (24 bits) para serem atribuídos aos hosts,
 - * portanto o número máximo de hosts é 2^{24} (menos 2 endereços reservados para rede e broadcast), ou seja, 16.777.214 hosts.
- * **Rede classe B,**
 - * os primeiros dois octetos são atribuídos à rede, restando os dois últimos octetos (16 bits) para serem atribuídos aos hosts,
 - * portanto o número máximo de hosts é 2^{16} (menos 2), ou seja, 65.534 hosts.
- * **Rede classe C,**
 - * os primeiros três octetos são atribuídos à rede, restando o último octeto (8 bits) para ser atribuído aos hosts,
 - * portanto o número máximo de hosts é 2^8 (menos 2), ou seja, 254 hosts.
- * O primeiro endereço em cada rede é reservado para o endereço de rede real (ou número de rede), e o endereço final em cada rede é reservado para broadcast.

Padrões de bits do endereço IP

nº de bits	1	7	24
------------	---	---	----

Classe A:	0	REDE x	HOST x
-----------	---	--------	--------

nº de bits	1	1	14	16
------------	---	---	----	----

Classe B:	1	0	REDE x	HOST x
-----------	---	---	--------	--------

nº de bits	1	1	1	21	8
------------	---	---	---	----	---

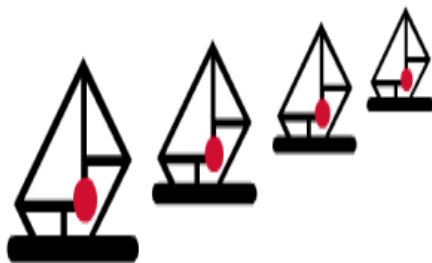
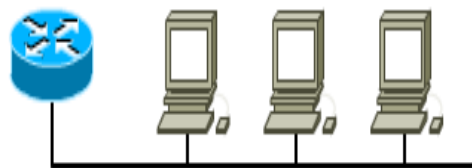
Classe C:	1	1	0	REDE x	HOST x
-----------	---	---	---	--------	--------

Máscara de sub-rede

- * A máscara de sub-rede aponta quais bits são da rede, e quais são das máquinas.
- * Exemplo: Máscara: 255.255.255.0
 - * 3 octetos para rede (255.255.255) e 1 octeto para máquinas (0) = 256 possibilidades.

C L A S S E	INTERVALO	ID DE REDE/ HOST	MÁSCARA SUB-REDE PADRÃO	NÚMERO DE REDES	HOSTS END. UTILIZA DOS
A	1 - 126	R,H.H.H	255.0.0.0	126 ($2^7 - 2$)	$2^{24} - 2 =$ 16.777.214
B	128-191	R.R.H.H	255.255.0.0	$2^{14} - 2 =$ 16.382	$2^{16} - 2 =$ 65.534
C	192-223	R.R.R.H	255.255.255.0	$2^{21} - 2 =$ 2.097.150	$2^8 - 2 = 254$
D	224-239	RESERVADO PARA MULTIDIFUSÃO			
E	240 - 254	EXPERIMENTAL, USADO PARA PESQUISA			

Endereço de broadcast



Visto por todos os dispositivos.

Inserido em todas as caixas de correio para todos verem.

Um endereço de broadcast é um endereço composto totalmente por 1s no campo do host. Quando enviar um pacote de broadcast por uma rede, todos os dispositivos da rede perceberão. Por exemplo, em uma rede com ID 176.10.0.0, um broadcast que chegasse a todos os hosts teria o endereço 176.10.255.255.

ENDEREÇO SEM CLASSE

- * Esgotamento de Endereços
 - * O imenso crescimento da Internet levou ao rápido esgotamento dos endereços disponíveis, embora o número de dispositivos na Internet seja menor que os 2^{32} espaços de endereços.
 - * Ficamos sem endereços de classe A e B, e um bloco de classe C é muito pequeno para a maioria das organizações de porte médio.
 - * Uma solução que reduziu o problema é a ideia de endereçamento sem classes.
- * Endereçamento sem Classes
 - * Nesse método, não existem classes, mas os endereços ainda são concedidos em blocos.
- * *Blocos de Endereços*
 - * No endereçamento sem classes, quando uma entidade, pequena ou grande, precisa ser conectada a Internet, lhe é concedido um bloco (intervalo) de endereços.
 - * O tamanho do bloco (o framer^o de endereços) varia tomando como base a natureza e o tamanho da entidade.
 - * um eletrodoméstico pode receber apenas dois endereços; uma grande organização, milhares de endereços.
 - * Um provedor de acesso a Internet, pode receber milhares ou centenas de milhares de endereços

BLOCO DE ENDEREÇOS E MÁSCARAS

UTILIZAMOS O ESPAÇO DO HOST NAS MÁSCARAS
E TOMAMOS EMPRESTADOS BITS.

REPRESENTAMOS POR NOTAÇÃO EM BARRA OU
CIDR. EXEMPLO:

255.0.0.0 = /8

255.255.0.0 = /16

255.255.255.0 = /24

/27 = 11111111.11111111.11111111.11100000 = 27
1'S

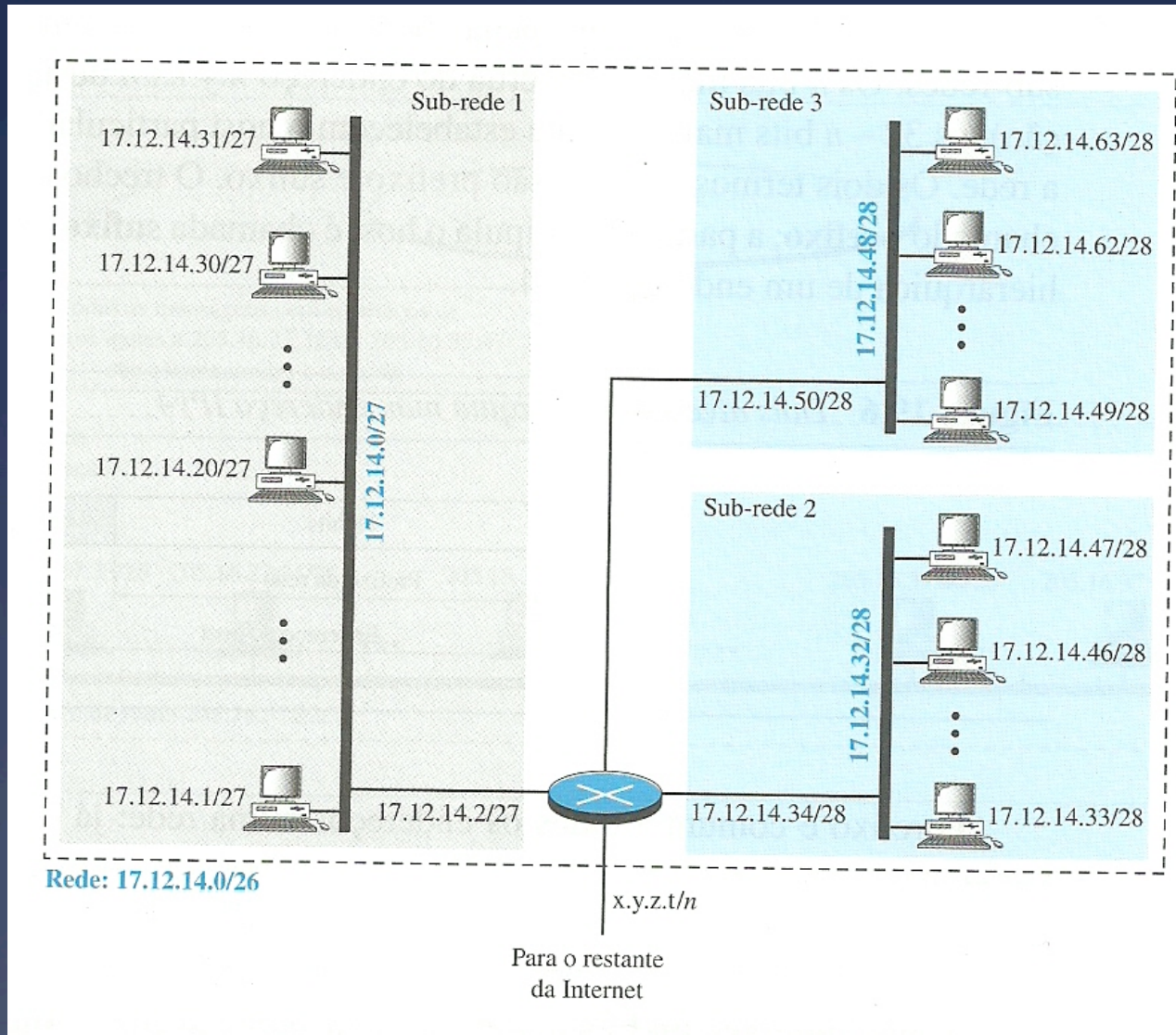
MÁSCARA = 255.255.255.224

Um bloco de 16 endereços concedido a uma pequena empresa

	Bloco	Bloco			
Primeiro	205.1637.32	11001101	00010000	00100101	00100000
	205.1637.33	11001101	00010000	00100101	00100001
	•				
	•				
Ultimo	205.1637.47	11001101	00010000	00100101	00101111

a. Decimal b. Binario

EXEMPLO DE SUB-REDES



REDES PRIVADAS

Para separar os endereços usados internamente na residência ou empresa daqueles utilizados para a Internet, os provedores de Internet reservaram três conjuntos de endereços, denominados privados

ENDEREÇOS DE REDES PRIVADAS:

INTERVALO	TOTAL
10.0.0.0 A 10.255.255.255.	2^{24}
172.16.0.0 A 172.31.255.255	2^{20}
192.168.0.0 A 192.168.255.255	2^{16}

Na máscara de sub-rede da classe A padrão, os bits que representam o endereçamento de rede são ligados (1s) e os bits de host são desligados (0s)

11111111 00000000 00000000 00000000

Se mais redes são necessárias, mais bits na máscara de sub rede precisarão ser utilizados para a rede. Se adicionarmos os quatro bits que serão utilizados para determinar a localização, a máscara de sub-rede fica semelhante a:

11111111 11110000 00000000 00000000

Quando o valor binário é convertido de volta para decimal, a nova máscara de sub-rede parece revelar-se:

11111111 11110000 00000000 00000000

255

240

0

0

O próximo passo é descobrir o endereço IP para cada localização. Que podemos chamar de ID (identificador) de localização.

Como o 10 é parte atribuída ao endereço, somente os valores no segundo octeto mudarão e os últimos dois octetos serão 0.0. para todos os IDs de localização.

0000 = 0

0001 = 1

0010 = 2

0011 = 3

0100 = 4

0101 = 5

0110 = 6

0111 = 7

1000 = 8

1001 = 9

1010 = 10

1011 = 11

1100 = 12

1101 = 13

1110 = 14

1111 = 15

Muitas pessoas têm dificuldade de entender que nem toda alteração no número do segundo octeto será uma sub-rede diferente. No exemplo, as sub-redes 10.14.0.0 e 10.15.0.0 estão ambas na mesma localização; entretanto 10.16.0.0 não está mesma localização.

11111111 00000000 00000000 00000000

Localização  Sub-rede 



Quantidade total de sub-redes na mesma localização

00001110 = 14

00001111 = 15

00010000 = 16

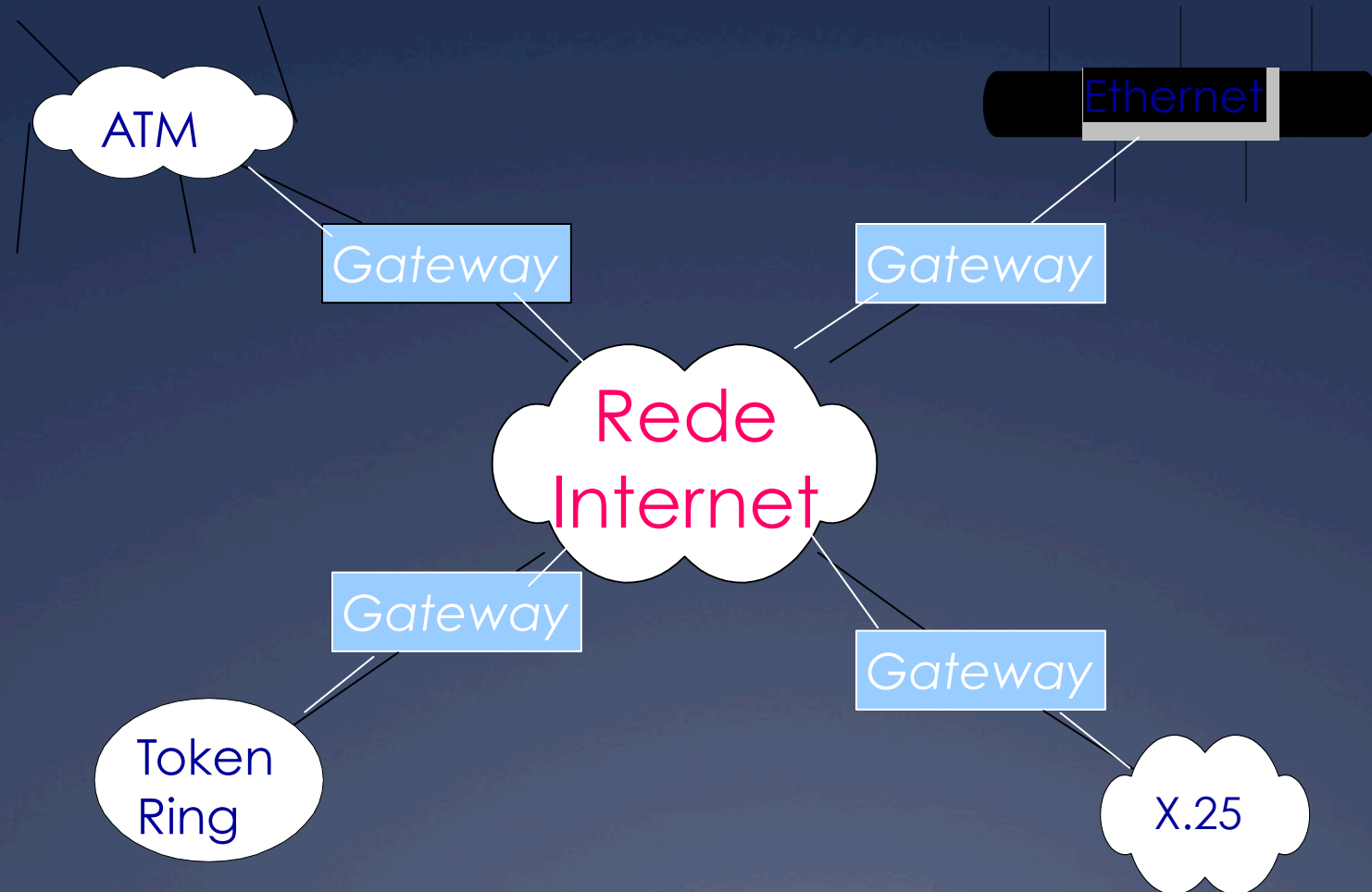
Camada de Transporte

Integração com Diferentes Tecnologias de Redes

- * cada tecnologia de rede possui suas próprias características: protocolos, endereços, interfaces, taxa de transmissão, meios físicos, etc
- * a Internet deve enxergá-las de forma transparente
- * a transparência é através do encapsulamento
- * a compatibilização é realizada pelos gateways que eliminam as diferenças
- * os gateways também implementam roteamento

Camada de Transporte

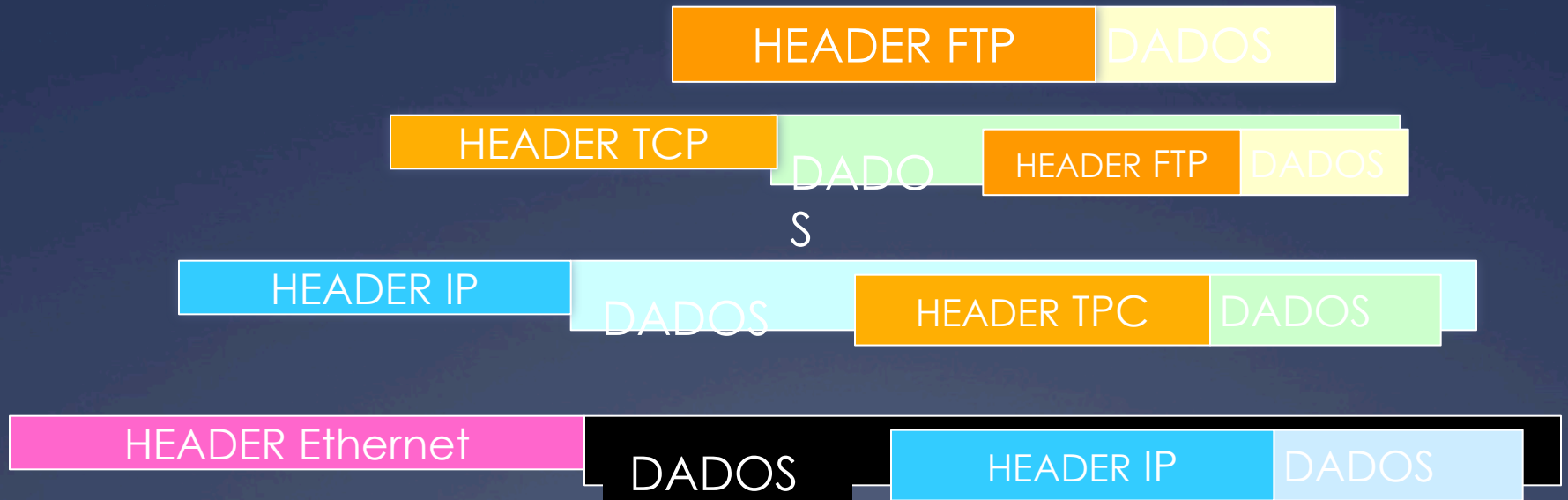
Interconexão de Tecnologias e Redes Diferentes



Não importa a tecnologia e sim a comunicação entre os serviços

Camada de Transporte

Encapsulamento



Camada de Transporte

❑ Características:

- ❑ Comunicação fim-a-fim entre aplicações
- ❑ Controle de fluxo
- ❑ Serviço confiável:
 - ❑ Controle de erro
 - ❑ Controle de seqüência
- ❑ Divisão de mensagens em segmentos
- ❑ Mecanismos de identificação de processos
origem e destino

Camada de Transporte

* Identificação de Processos:

* **Portas:**

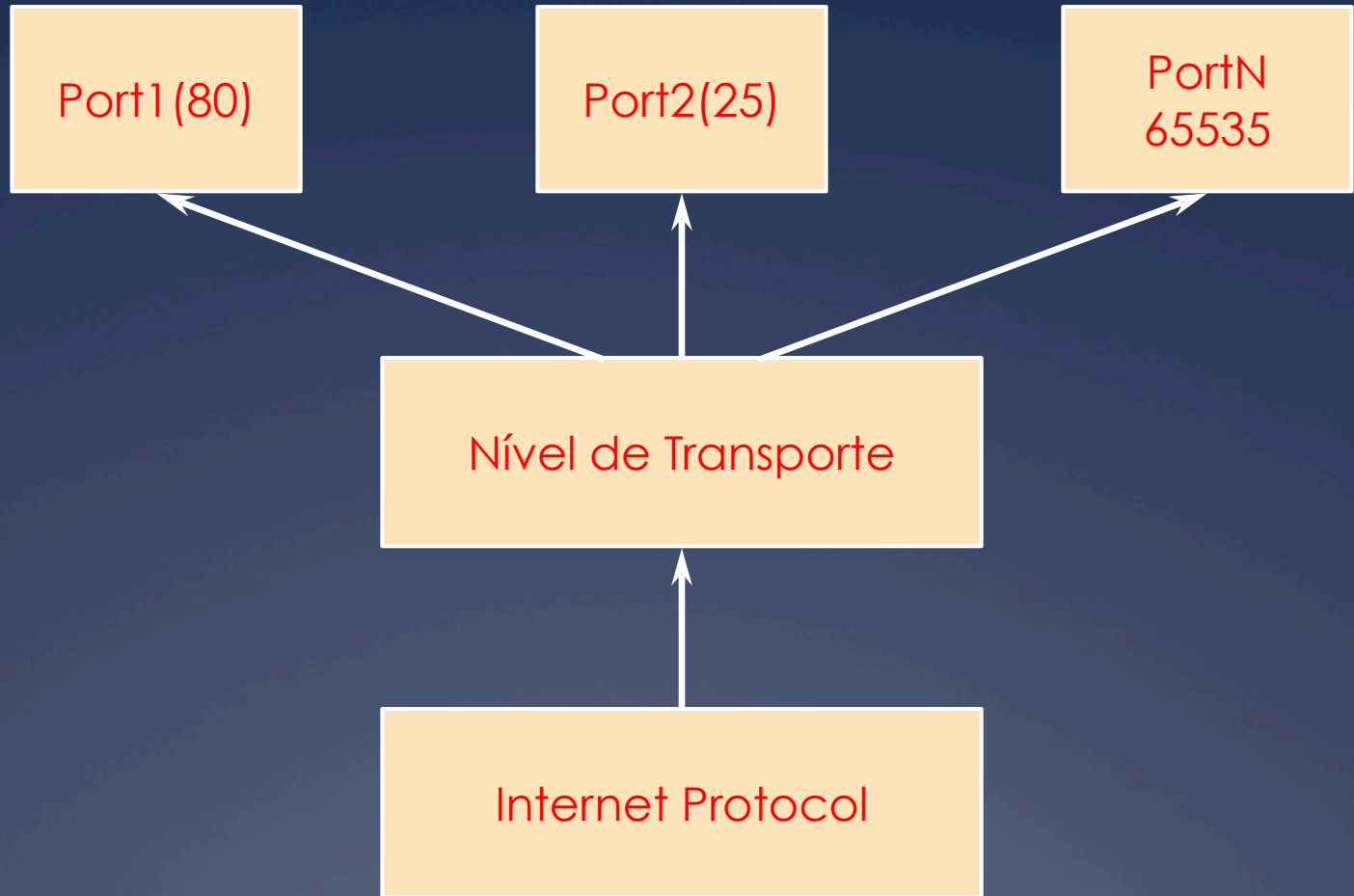
- * Identificam os processos origem e destino
- * Viabilizam a comunicação fim-a-fim
- * Sistema operacional oferece interface que permite às aplicações especificarem ou acessarem portas
- * Permite as aplicações em um determinado host enviar e receber datagramas de forma independente
- * Utilizam buffers de recepção e transmissão

Camada de Transporte

- * Portas
 - * Reservadas
 - * 0 - 1023 (RFC 1070)
 - * Liberadas
 - * 1024 a 65535
 - * automaticamente definida pelo SO para aplicações clientes
 - * portas de servidores de aplicações ainda não reconhecidas formalmente na Internet

Camada de Transporte

* Portas:



Camada de Transporte

- * NÃO ESQUEÇA QUE UM PROCESSO EM UMA PORTA NADA MAIS É QUE :
 - * UM PROCESSO PAI INSTANCIADO E LENDO INFORMAÇÕES, ELE NÃO PROCESSA E SIM REPASSA A UM PROCESSO FILHO QUE ADMINISTRA E PASSA AO EXECUTOR QUE É O PROCESSO ESCRAVO.

Camada de Transporte

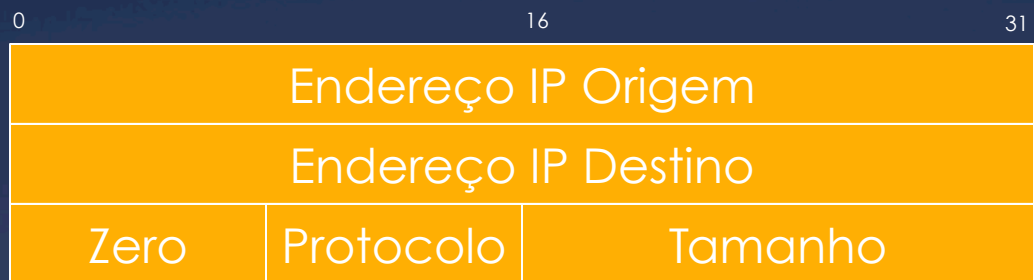
* Dois tipos de serviços:

- * Confiável: Controle de erros, controle de fluxo, multiplexação, seqüenciação
- * não-confiável: Multiplexação e desmultiplexação
 - * voltado para aplicações com entrega imediata
 - * ex. transmissão de vídeo e/ou voz

Camada de Transporte

- * Arquitetura TCP/IP especifica 2 tipos de protocolos:
 - * TCP (*Transmission Control Protocol*)
 - * UDP (*User Datagram Protocol*)
- * TCP é orientado à conexão e garante a transferência confiável de dados
- * UDP é não orientado à conexão, simples extensão do IP e não garante a entrega de dados
- * a utilização de um ou de outro depende das necessidades da aplicação (**SNMP-UDP, FTP-TCP**)

Pseudo-Header



Onde,

Endereço IP Origem e Endereço IP destino são do nível de rede (protocolo IP) utilizadas para a segunda validação do destino do datagrama.

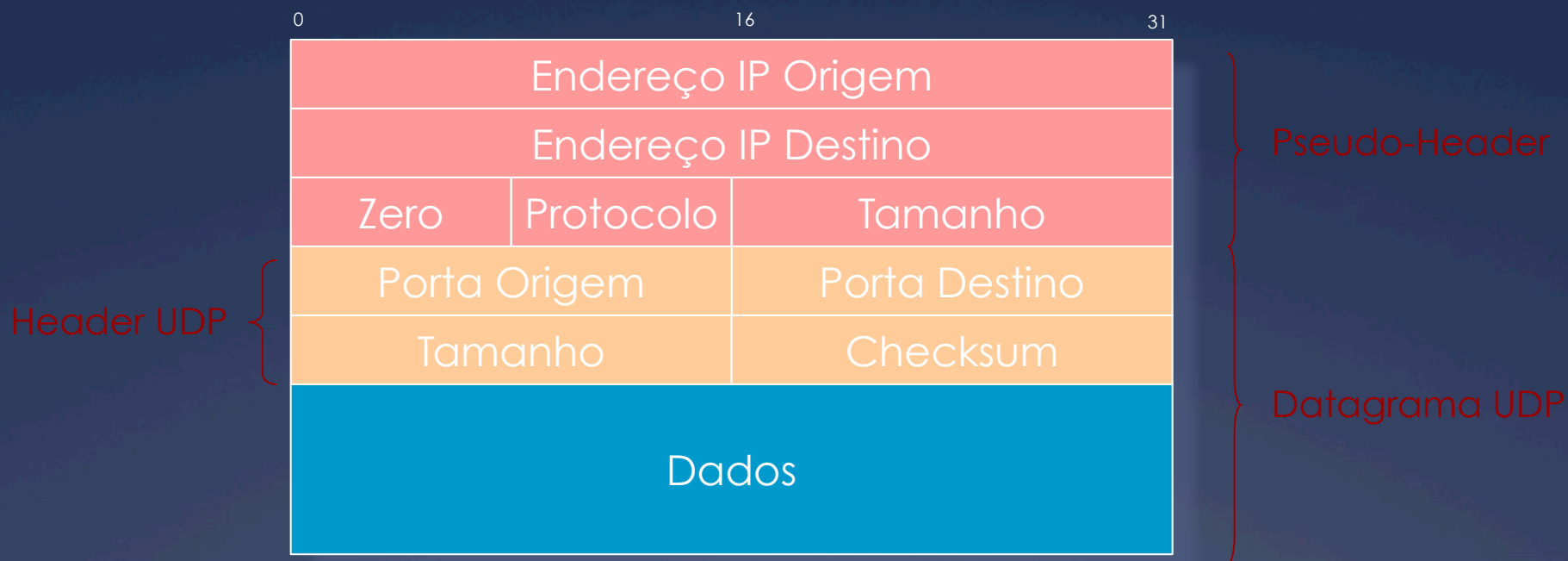
Zero é um campo com valor zero para complementar a estrutura do pseudo-header.

Protocolo indica qual o protocolo de transporte (TCP ou UDP), pois o pseudo-header é utilizado para os dois protocolos.

UDP (User Datagram Protocol)

- * Muito mais rápido
- * O protocolo UDP é bastante simples
 - * Orientado a datagrama
 - * Não orientado à conexão
 - * Não executa controle de fluxo, controle de erro e sequenciamento
 - * Não tem reconhecimento dos datagramas (ACK/NACK)
- * Devido a sua simplicidade é considerado não confiável
 - o Segmentos podem não chegar
 - o Segmentos podem chegar fora de ordem

Ordem de Header para o Checksum do UDP



Atenção!

O Pseudo-Header não é transmitido junto com o datagrama UDP, ele é utilizado apenas para cálculo do Checksum.

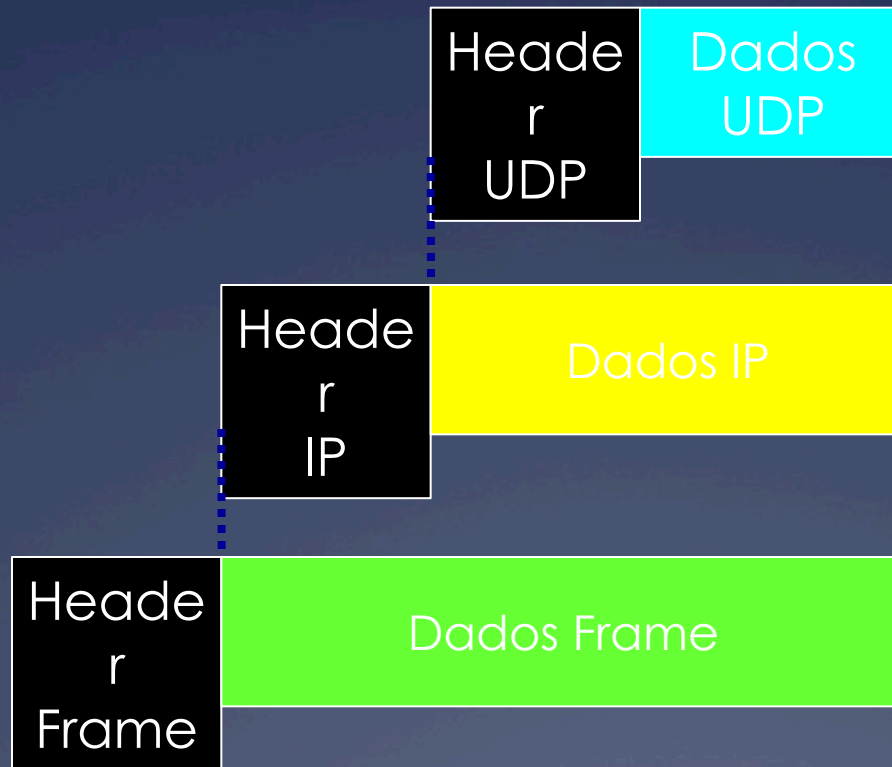
Tamanho Máximo do Datagrama

- * Teoricamente o tamanho máximo é de 64Kb
 - * Porque no IP o campo tamanho total é de 16 bits
 - * Mas deve-se considerar que no IP estão sendo calculado
 - * Tamanho do Header do IP (20 bytes)
 - * Datagrama UDP (8 bytes)
 - * Assim, o tamanho máximo é de 65507 bytes

Tamanho Máximo do Datagrama

- * Outros fatores podem influenciar
 - * Programas de aplicação podem ser limitados pela interface de programação
 - * Implementação do kernel do TCP/IP
- * Truncando Datagramas
 - * Apesar do tamanho máximo, nem todas as aplicações podem estar preparadas para receber um datagrama maior que esperado
 - * Truncar ou não? Depende da implementação de cada interface de programação

Encapsulamento de um Datagrama UDP



UDP

- Utilização
 - Aplicações cliente x servidor
 - Um consulta para uma resposta
 - Aplicações de mídia
 - Aplicações tolerantes a falha

TCP

- Transmission Control protocol
- Serviço confiável
 - Garante que os dados serão entregues na forma que foram enviados
- Orientado a conexão
 - Conexões nos sistemas finais
- Controle de fluxo
 - Transmissor não esgota a capacidade do receptor

TCP

- Controle de congestionamento
 - Transmissor não esgota recursos da rede
- Gerencia timers
 - Necessita de timers para executar o trabalho
- Mais complexo que o UDP
- Principal protocolo
- Utilizado pela maioria das aplicações

TCP

- Recebe fluxo da aplicação e divide em partes
 - Com frequência, temos 1460 para que ele possa caber em um quadro ethernet
- Camada de rede aceita cada parte como um segmento distinto
- Destino restaura o fluxo original

TCP

- Destino retorna um segmento com número de confirmação com o número da próxima sequência a receber
- Se expirar o timer, o segmento será retransmitido

TCP

- Serviço TCP utiliza o conceito de socket
 - Endereço IP + Número da porta
 - Portas abaixo de 1024 - Portas bem conhecidas
- O serviço TCP necessita estabelecimento de sessão
- Fluxo
 - Estabelecimento de sessão
 - Transferência de dados
 - Término da conexão

TCP

- Conexões são identificadas por socket
 - Endereço IP de origem
 - Porta de origem
 - Endereço IP de destino
 - Porta de destino
- Conexões são full-duplex
- Analogia ao endereço de um edifício

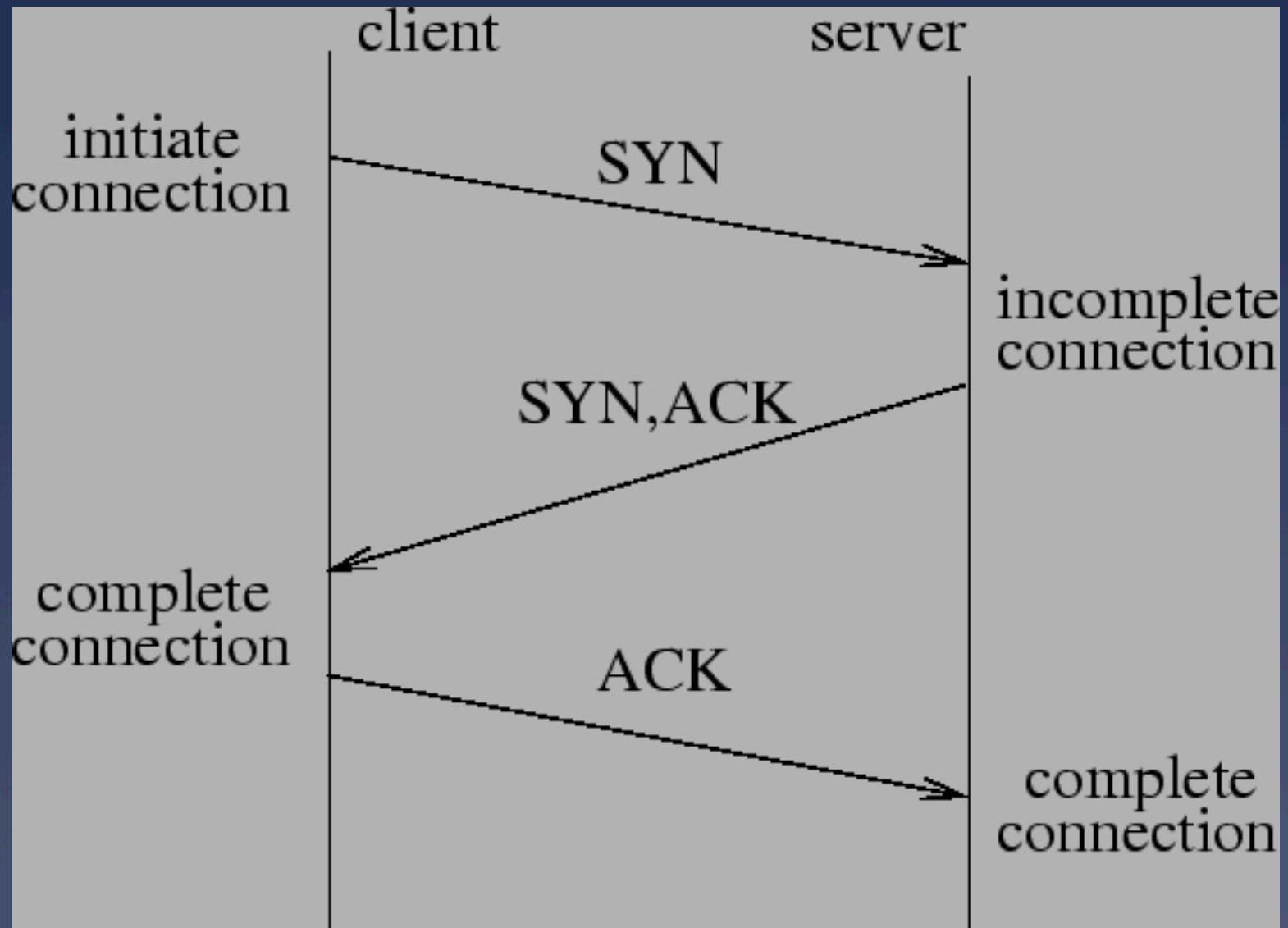
TCP

- Gerenciamento da conexão
 - Gerencia o estabelecimento de conexão entre o transmissor e o receptor
 - Campos: RST, SYN e FIN
 - RST
 - Recusa uma tentativa de conexão
 - SYN
 - Inicia uma conexão
 - FIN
 - Finaliza uma conexão

TCP

- Estabelecimento de conexão
 - Passo 1: Cliente inicia conexão com SYN=1 e ack=0
 - Passo 2: Servidor responde com SYN=1 e ack=1
 - Passo 3: Cliente responde com SYNC=1 e ack=1
 - Famoso: 3-way handshake

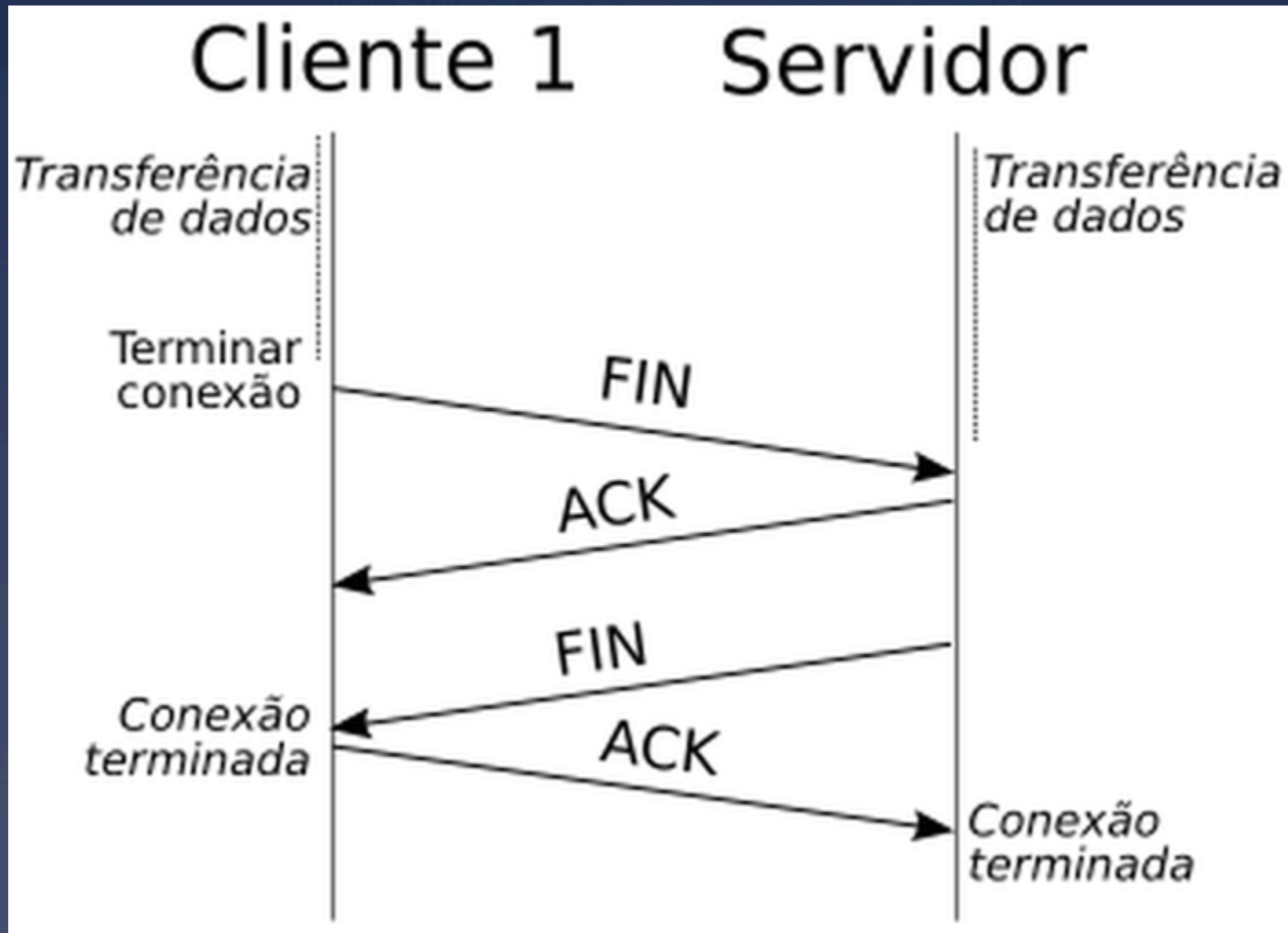
TCP



TCP

- Término de conexão
 - Passo 1: host 1 envia FIN ao host 2
 - Passo 2: host 2 responde FIN para host 1 com ACK
 - Passo 3: host 2 envia FIN ao host 1
 - Passo 4: host 1 responde FIN para host 2 com ACK

TCP



TCP

- Controle de fluxo
 - Evitar que o transmissor esgote a capacidade do receptor
 - No estabelecimento de sessão é alocado um buffer de recepção e informado ao par
 - Em toda confirmação é enviado o espaço disponível no buffer (janela)
 - Campos: Windows size
 - Indica quantos bytes podem ser enviados

TCP

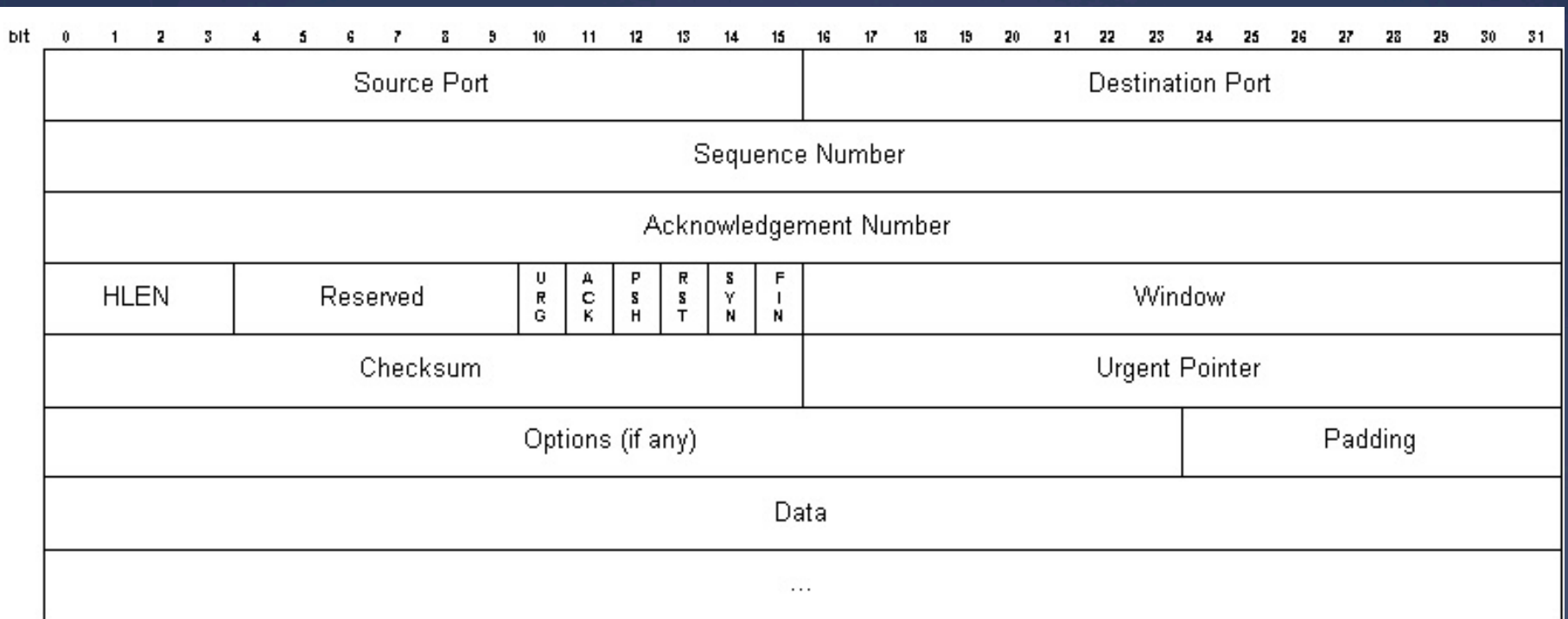
- Controle de congestionamento
 - Quando a carga oferecida a qualquer rede é maior que a capacidade, tem-se um congestionamento
 - Sintomas:
 - Perda de pacotes
 - Atrasos grandes
 - Pode ser piorado se a camada de transporte retransmite dados
 - Pode causar colapso

TCP

- Controle de congestionamento
 - TCP utiliza quantidade de pacotes perdidos como uma medida de congestionamento
 - Reduz a taxa de transmissão quando esse número aumenta
 - A maioria dos timeouts se deve a congestionamento e não por erros de transmissão

TCP

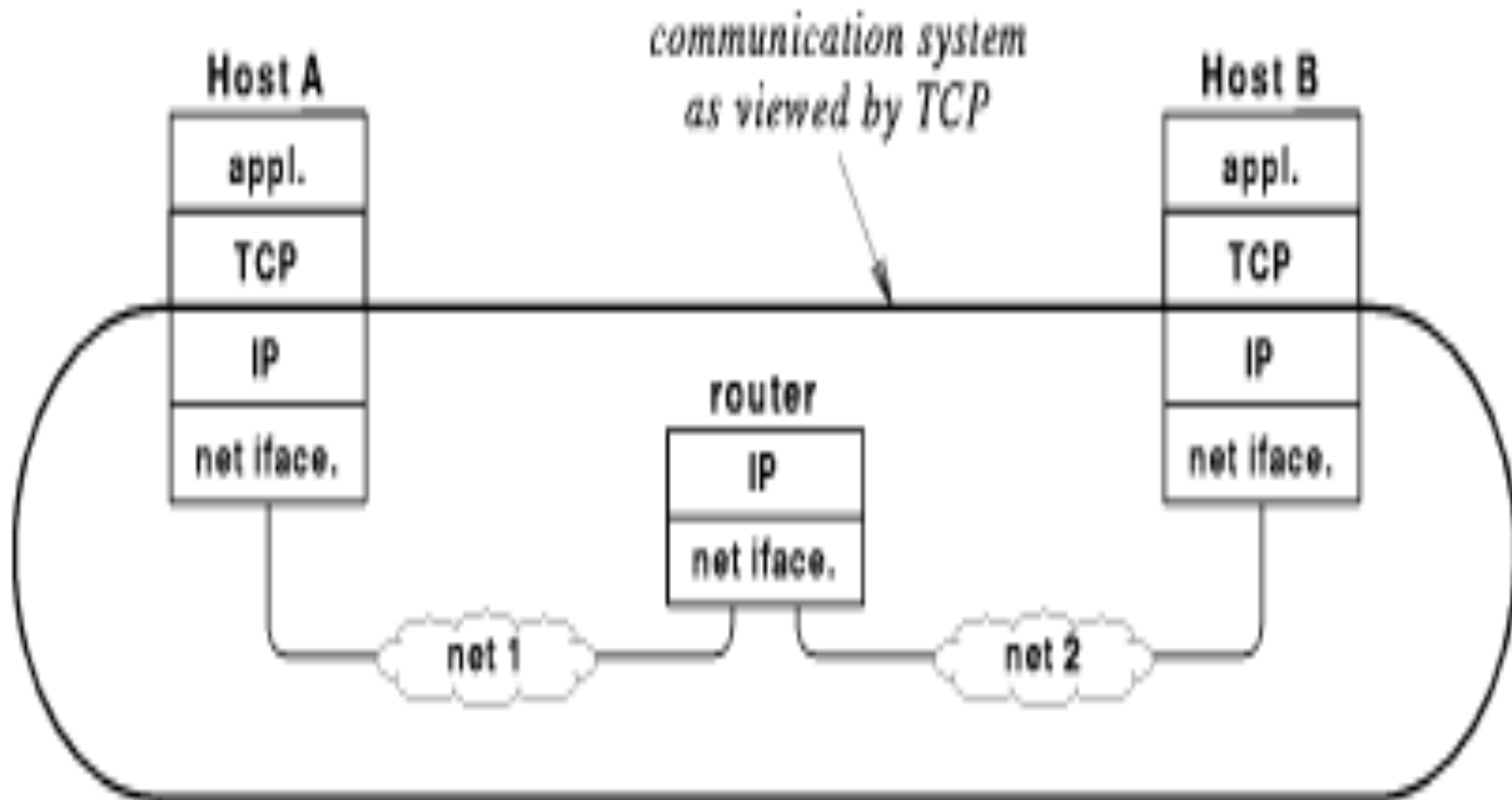
- Segmento



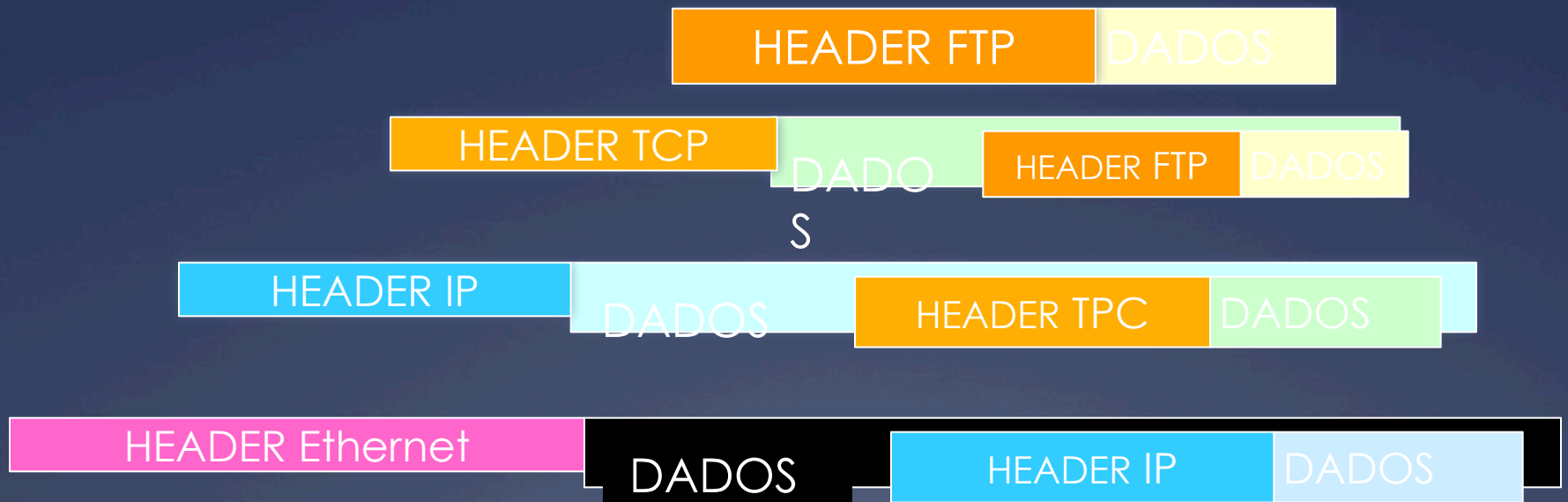
TCP (Transmission Control Protocol)

- * Protocolo de transporte considerado confiável
 - * Orientado à conexão
 - * Controle de erros com retransmissão
 - * Controle de fluxo
 - * Sequenciamento
 - * Entrega ordenada
- * Orientado a “*byte stream*”

TCP - Visão de comunicação

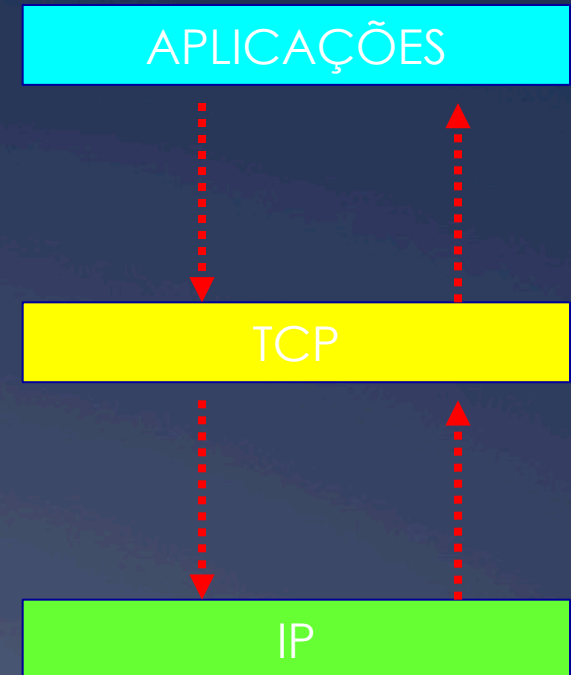


Encapsulamento - TCP



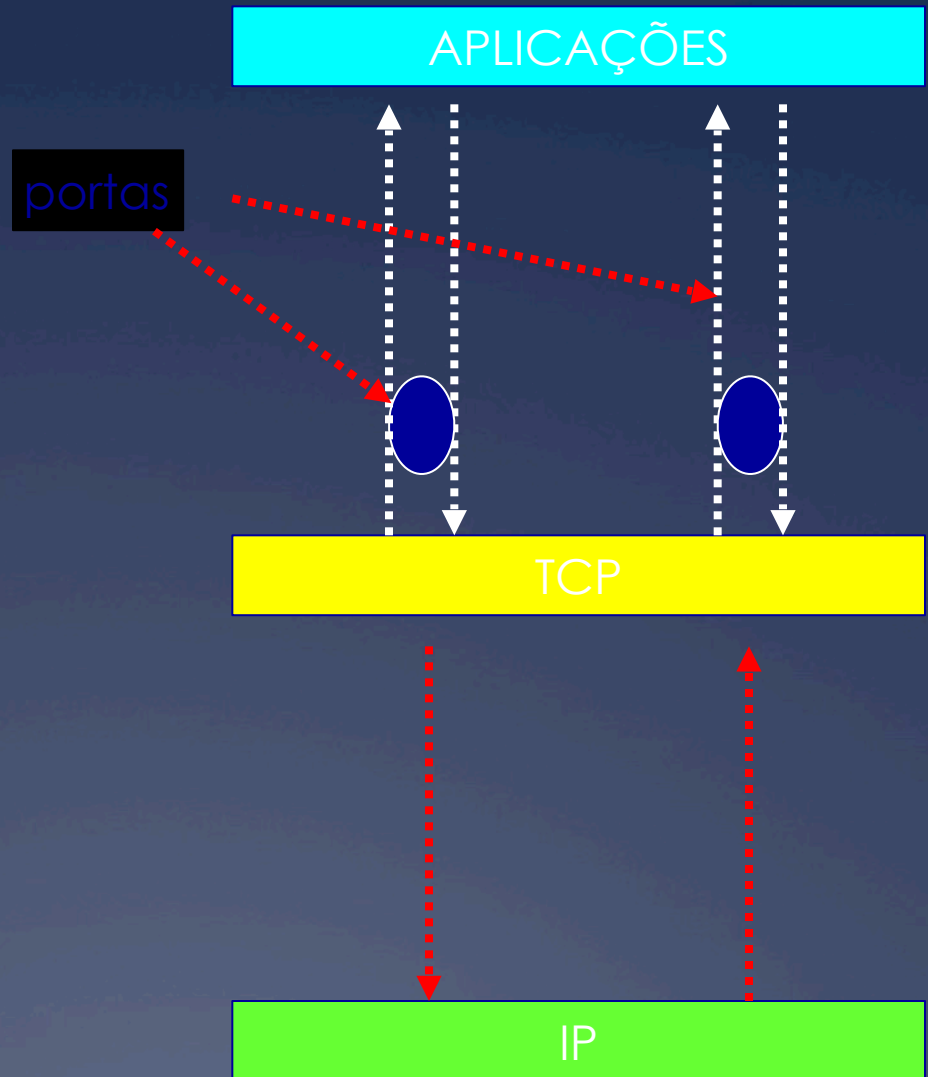
Protocolo TCP

- * foi projetado p/ funcionar c/ base em um serviço de rede s/ conexão e s/ confirmação
- * interage de um lado c/ processos de aplicação e de outro c/ o protocolo da camada de rede
- * a interface c/ os processos de aplicação consiste em um conj de chamadas
- * a interface c/ a camada inferior (rede) é através de um mecanismo assíncrono



Protocolo TCP

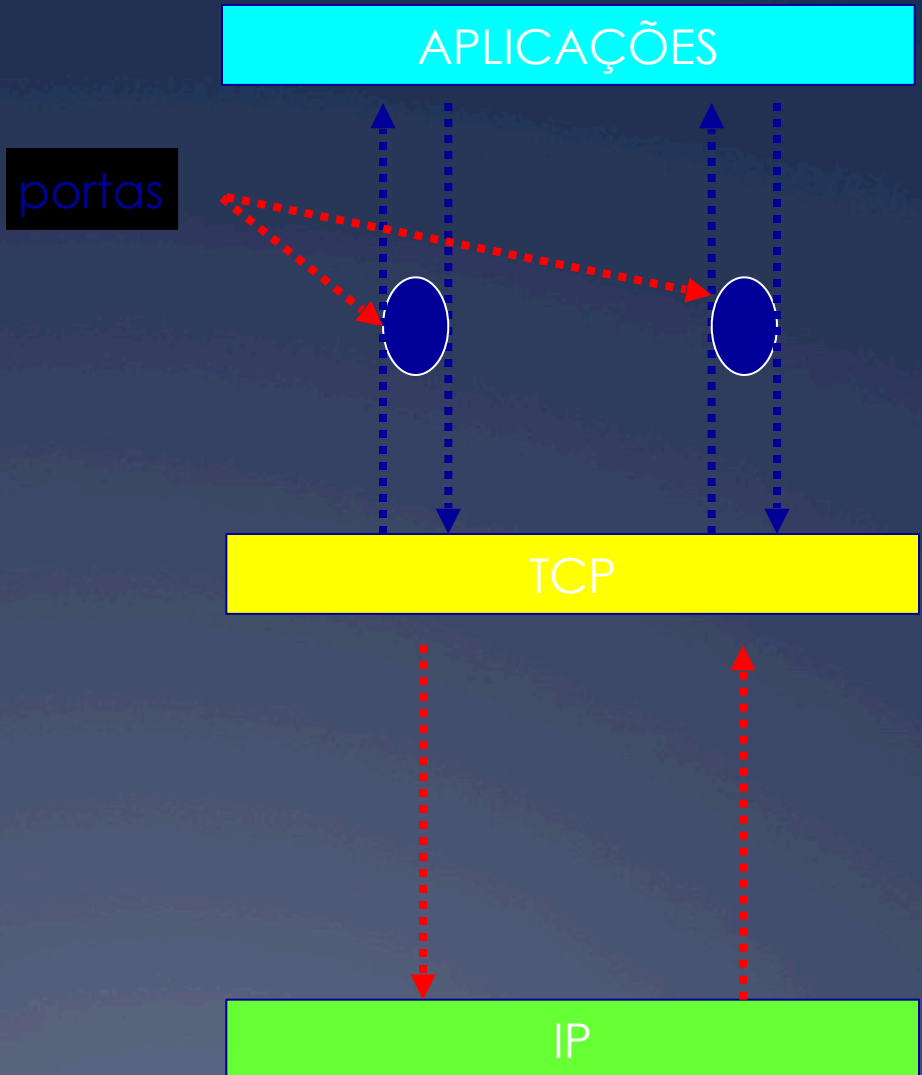
- * p/ que vários processos simultaneamente usem os serviços do TCP, é usado o **conceito de porta**
- * cada processo de aplicação, em um dado momento, é identificado por uma porta diferente



Protocolo TCP - Portas

As portas abaixo de 256 são chamadas portas conhecidas e reservadas p/ serviços-padrão (RFC 1700)

Aplicação	Portas Default
FTP	20 dados 21 controle
TELNET 23	
SNMP	161 162 (trap)



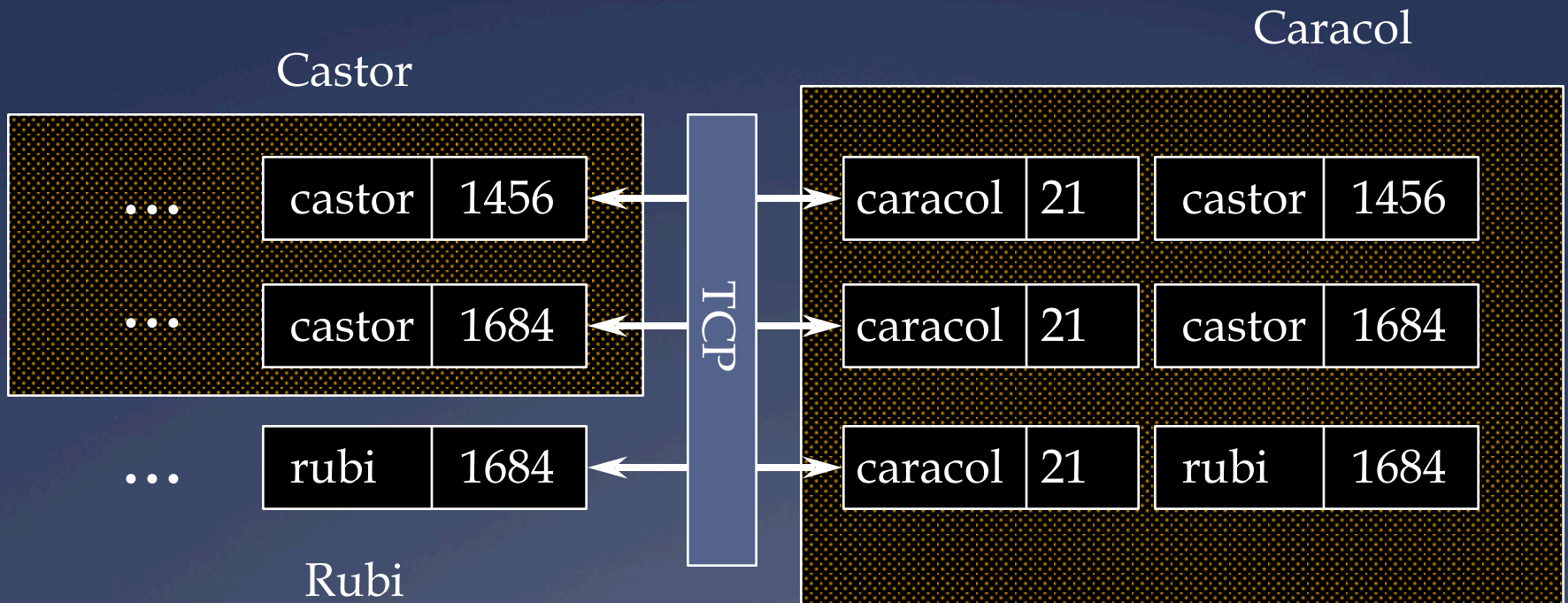
TCP - Parâmetros

- * MSS - Maximum segment Size
- * Padronização de ports
- * Controle de Congestionamento

Port	Serviço
15	netstat
21	ftp
23	telnet
25	smtp (mail)

TCP - Exemplo

* FTP - port 21



Header TCP



Header TCP

Onde,

Porta Origem e Porta Destino identificam o processo de aplicação que está enviando dados e o processo de aplicação que irá receber os dados.

Número de seqüência identifica os bytes enviados. Na prática ele é a identificação do primeiro byte de dados contido no segmento enviado. Os demais são seqüenciados a partir deste byte.

Acknowledgement identifica os bytes que foram recebidos e tratados sem erro pelo destino, bem como a seqüência do próximo byte esperado

Tamanho é representa o tamanho total do frame TCP

Reservado é um campo ainda não utilizado

FLAGS identifica as flags (syn, fin, psh, rst, ack, urg)

Window identifica o tamanho da janela para o controle de fluxo

Checksum destina-se a verificação de erros de transmissão. É calculado usando o pseudo header, o header TCP e também a área de dados

Urgent Poninter é um ponteiro para dados urgentes, contidos na área de dados.

Controle de Conexão TCP

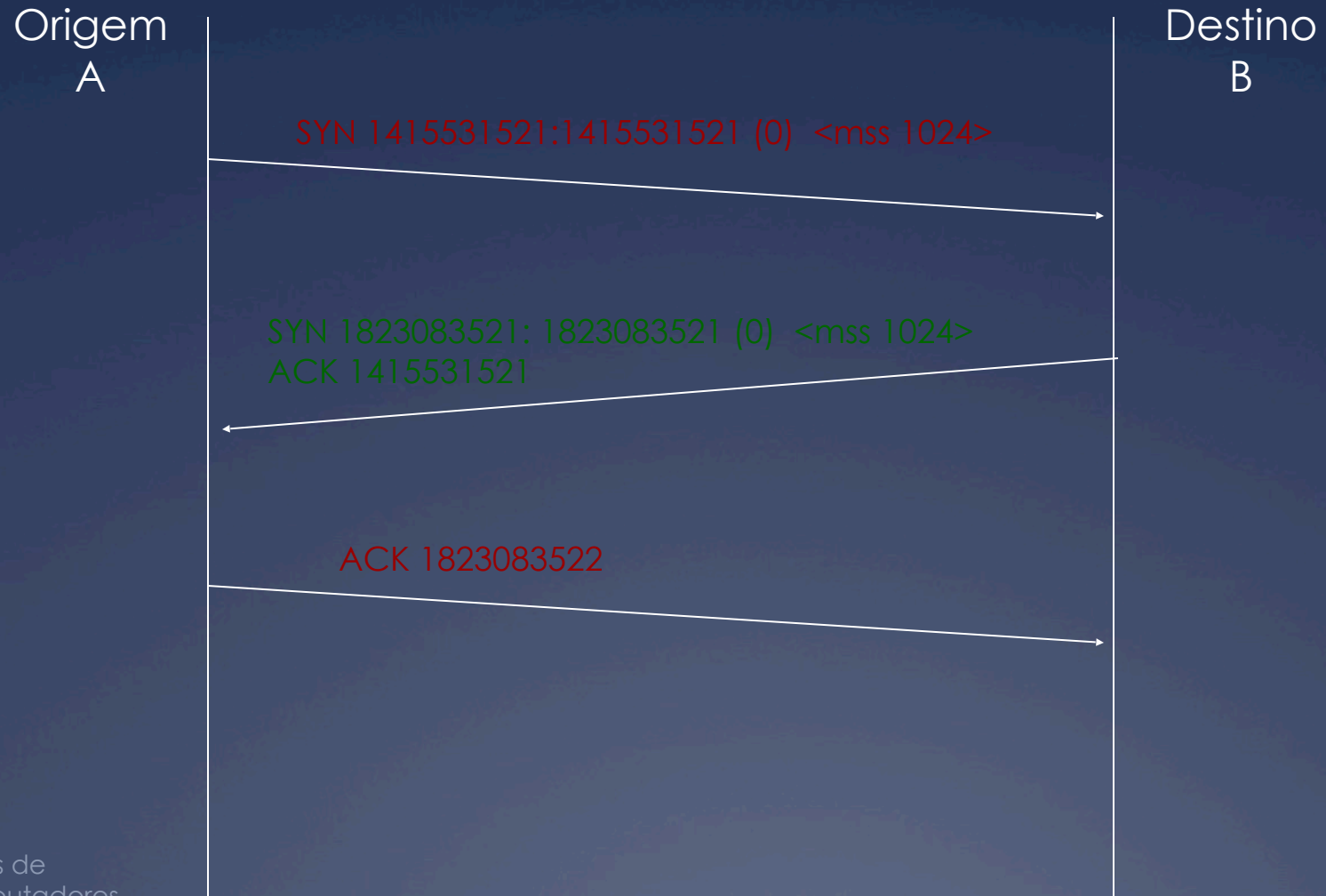
- * Três Fases

- * Estabelecimento da Conexão
- * Transmissão de Dados
- * Encerramento da Conexão

- * Flags

- * SYN – solicitação de conexão
- * FIN – Finalização da Conexão
- * RST – Reset da Conexão
- * ACK – Reconhecimento de recebimento

Estabelecimento da Conexão



Estabelecimento da Conexão

- * Ativo x passivo
 - * A origem da solicitação de conexão executa o “*active open*”
 - * O destino que recebe a solicitação de conexão executa o “*passive open*”
- * Origem e destino enviam seus número de seqüência iniciais para a conexão em curso
 - * Este número deve ser alterado ao longo do tempo e ser diferente de conexão para conexão

[200.163.195.16]	[212.143.22.63]	DLC: Ethertype=8864, size=70 bytes PPPoE: : PPP Session Stage PPP: Internet IP: D=[212.143.22.63] S=[200.163.195.168] LEN=28 ID=33931 TCP: D=80 S=3746 SYN SEQ=3637205993 LEN=0 WIN=16384
[212.143.22.63]	[200.163.195.168]	DLC: Ethertype=8864, size=70 bytes PPPoE: : PPP Session Stage PPP: Internet IP: D=[200.163.195.168] S=[212.143.22.63] LEN=28 ID=55563 TCP: D=3746 S=80 SYN ACK=3637205994 SEQ=4038046534 LEN=0 WIN=163
[200.163.195.16]	[212.143.22.63]	DLC: Ethertype=8864, size=62 bytes PPPoE: : PPP Session Stage PPP: Internet IP: D=[212.143.22.63] S=[200.163.195.168] LEN=20 ID=33932 TCP: D=80 S=3746 ACK=4038046535 WIN=17424
[200.163.195.16]	[212.143.22.63]	DLC: Ethertype=8864, size=251 bytes

```

IP: No options
IP:
TCP: ----- TCP header -----
TCP:
TCP: Source port          = 3746
TCP: Destination port    = 80 (WWW-HTTP)
TCP: Initial sequence number = 3637205993
TCP: Next expected Seq number= 3637205994
TCP: Data offset         = 28 bytes
TCP: Flags               = 02
TCP:      ...0. .... = (No urgent pointer)
TCP:      ...0. .... = (No acknowledgment)
TCP:      .... 0... = (No push)
TCP:      .... .0.. = (No reset)
TCP:      .... ..1. = SYN
TCP:      .... ....0 = (No FIN)
TCP: Window              = 16384
TCP: Checksum            = 8865 (correct)
TCP:
TCP: Options follow
TCP: Maximum segment size = 1452
TCP: No-op
TCP: No-op
TCP: SACK-Permitted Option

```

TCP - Estabelecendo uma conexão - (Three Way Handshake)



1: Envia SYN $ISN=x$

2: Envia SYN $ISN=y$, ACK $x+1$

3: Envia ACK $ISN y+1$

4: Conexão estabelecida

TCP Three Way Handshake

Packet Analysis - Parte 1

Packet 1

TIME: 19:50:32.912582 (0.040960)

LINK: 00:40:05:E3:09:D0 -> 00:00:C5:38:0D:27 type=IP

IP: strife -> testbed hlen=20 TOS=00 dgramlen=44 id=2864

MF/DF=0/0 frag=0 TTL=64 proto=TCP cksum=E641

TCP: port 24616 -> ftp seq=2735221453 ack=0000000000

hlen=24 (data=0) UAPRSF=000010 wnd=512 cksum=FBEC urg=0

DATA: <No data>

The requesting client sends a *SYN* (synchronize) segment specifying the **port number** of the server it wishes to connect to and the client's **ISN** (Initial Sequence Number).

TCP Three Way Handshake

Packet Analysis - Parte 2

Packet 2

TIME: 19:50:32.912792 (0.000210)

LINK: 00:00:C5:38:0D:27 -> 00:40:05:E3:09:D0 type=IP

IP: testbed -> strife hlen=20 TOS=00 dgramlen=44 id=5FF4

MF/DF=0/1 frag=0 TTL=64 proto=TCP cksum=6EB1

TCP: port ftp -> 24616 seq=**2809565737** ack=**2735221454**

hlen=24 (data=0) UAPRSF=010010 wnd=17520 cksum=7FCB urg=0

DATA: <No data>

The server responds with a *SYN* segment including the servers own *ISN*. An **ACK** (acknowledge) is also sent with the clients ISN plus one.

TCP Three Way Handshake

Packet Analysis - Parte 3

Packet 3

TIME: 19:50:32.913768 (0.000976)

LINK: 00:40:05:E3:09:D0 -> 00:00:C5:38:0D:27 type=IP

IP: strife -> testbed hlen=20 TOS=00 dgramlen=40 id=2865

MF/DF=0/1 frag=0 TTL=64 proto=TCP cksum=A644

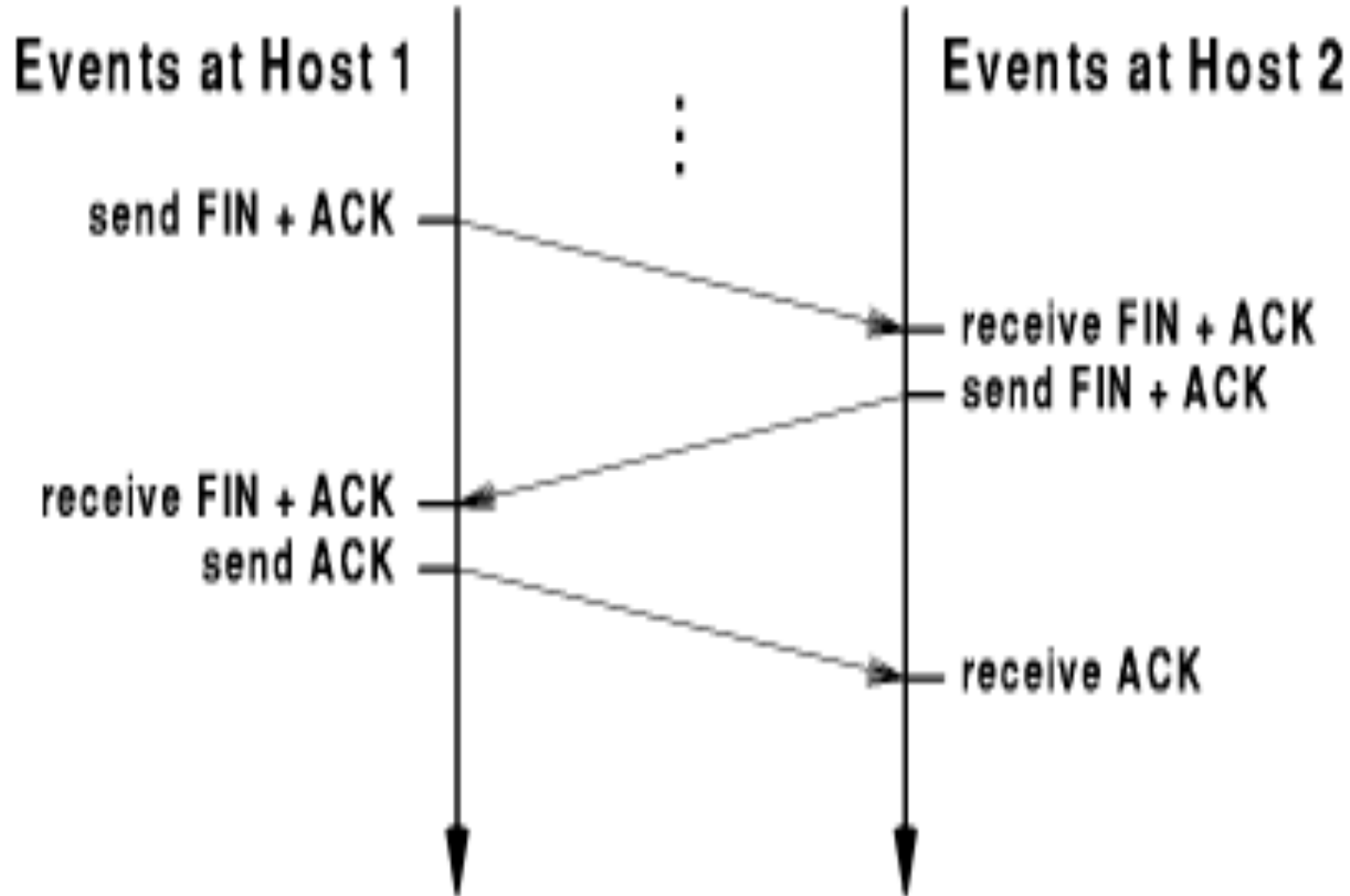
TCP: port 24616 -> ftp seq=2735221454 **ack=2809565738**

hlen=20 (data=0) UAPRSF=010000 wnd=32120 cksum=5E80 urg=0

DATA: <No data>

The client acknowledges the servers *SYN* and sends an **ACK** segment with the Servers *ISN* plus one.

TCP - Estabelecimento de conexão



Inicialização do Número de Seqüência

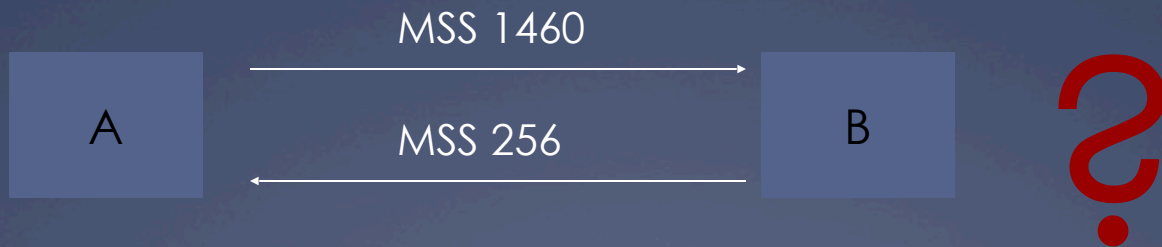
- * RFC 793
 - * Número de 32 bits
 - * É incrementado a cada 4 microsegundos
- * Como escolher o número inicial?
 - * 4.4BSD
 - * Quando sistema é inicializado o número de seqüência é 1 (violação da RFC)
 - * A variável é incrementada de 64.000 a cada $\frac{1}{2}$ segundo
 - * Isso significa que irá retornar a 0 em períodos de 9 horas e $\frac{1}{2}$

MSS (Maximum Segment Size)

- * O MSS representa o tamanho do maior bloco de dados que poderá ser enviado para o destino.
- * Não é negociável, cada host divulga o seu MSS
 - * Default: 536 bytes (20 bytes IP, 20 bytes TCP, para um total de 576 bytes)
 - * Ethernet: 1460 bytes (20 bytes IP, 20 bytes TCP, para um total de 1500 bytes)

MSS...

- * Em geral, quanto maior o MSS melhor, até que ocorra fragmentação
- * Quanto maior a quantidade de dados enviados em um único bloco, menor o overhead de headers do TCP e do IP
- * Exemplo



MSS..

Cliente

```
44 TCP: ----- TCP header -----
TCP:
TCP: Source port          = 3746
TCP: Destination port    = 80 (WWW-HTTP)
TCP: Initial sequence number = 3637205993
TCP: Next expected Seq number = 3637205994
TCP: Data offset         = 28 bytes
TCP: Flags                = 02
TCP:          ...0. .... = (No urgent pointer)
TCP:          ...0. .... = (No acknowledgment)
TCP:          ...0. .... = (No push)
TCP:          ...0. .... = (No reset)
TCP:          ...1. .... = SYN
TCP:          ...0. .... = (No FIN)
TCP: Window              = 16384
TCP: Checksum            = 8865 (correct)
TCP:
TCP: Options follow
TCP: Maximum segment size = 1452
TCP: No-op
```

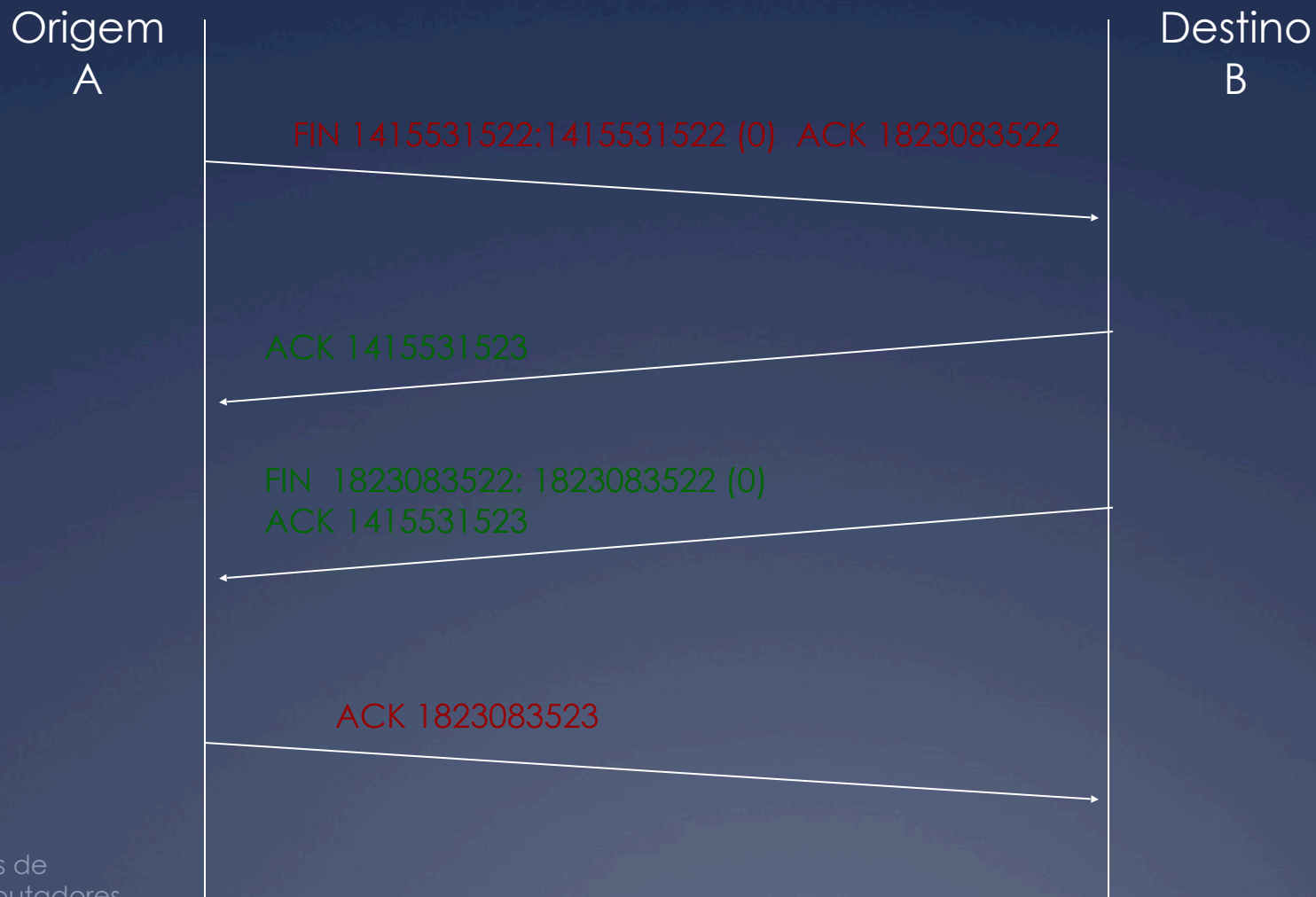
Servidor

```
44 TCP: ----- TCP header -----
TCP:
TCP: Source port          = 80 (WWW-HTTP)
TCP: Destination port    = 3746
TCP: Initial sequence number = 4038046534
TCP: Next expected Seq number = 4038046535
TCP: Acknowledgment number = 3637205994
TCP: Data offset         = 28 bytes
TCP: Flags                = 12
TCP:          ...0. .... = (No urgent pointer)
TCP:          ...1. .... = Acknowledgment
TCP:          ...0. .... = (No push)
TCP:          ...0. .... = (No reset)
TCP:          ...1. .... = SYN
TCP:          ...0. .... = (No FIN)
TCP: Window              = 16384
TCP: Checksum            = E455 (correct)
TCP:
TCP: Options follow
TCP: Maximum segment size = 1460
```

Outras Opções TCP

- * End of option list (1 byte)
- * No operation (NOP) (1 byte)
- * Windows scale factor (3 bytes)
- * Timestamp (10 bytes)
- * MSS (4 bytes)

Encerramento da Conexão



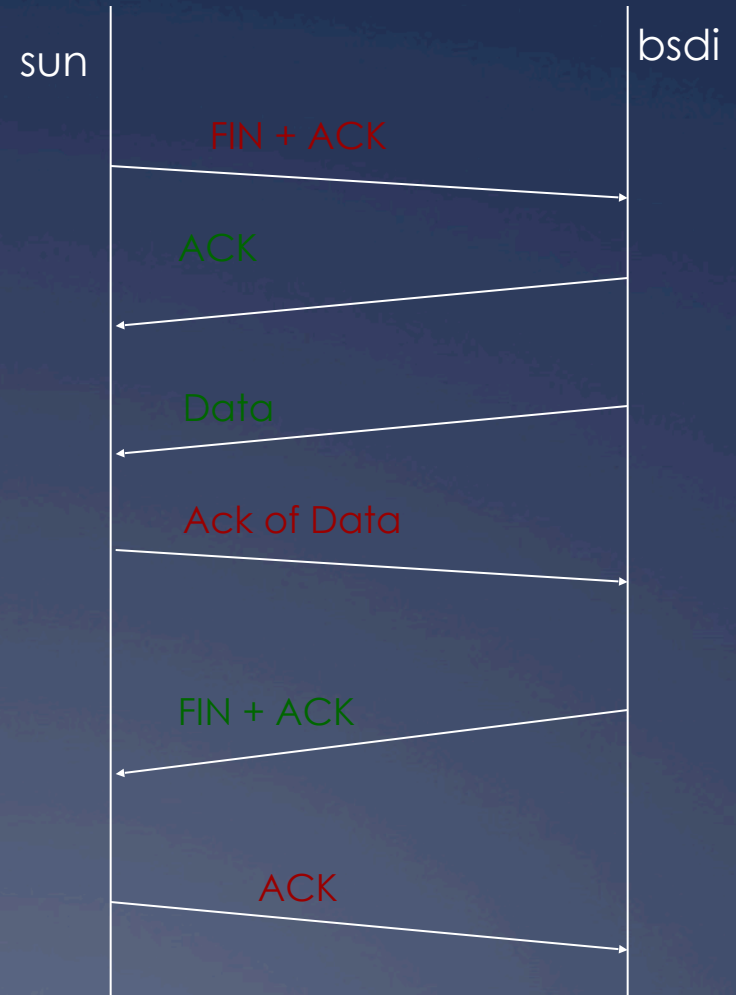
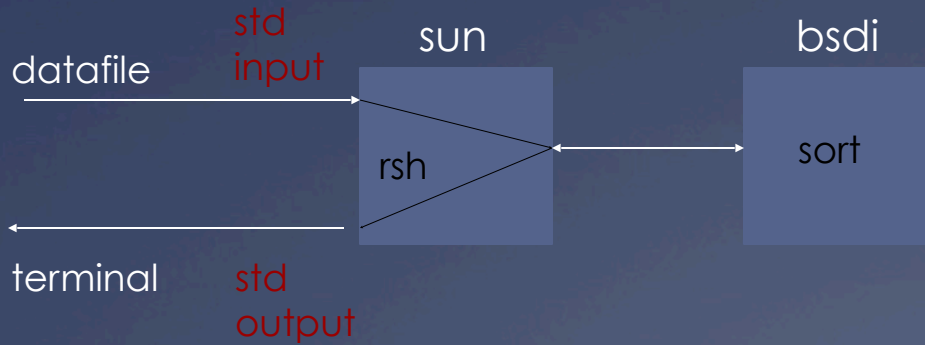
Encerramento da Conexão

- * *Half Close*
 - * Conexões TCP são *full-duplex*, logo cada lado da conexão deve finalizar a conexão de forma independente
 - * Quando um dos lados envolvidos recebe uma solicitação de finalização deve enviar a notificação para a aplicação
 - * Uma aplicação após receber o pedido de finalização ainda pode mandar dados

Half Close - Exemplo

Exemplo:

```
sun% rsh bsd1 < datafile
```



Timeout no Estabelecimento da Conexão

- * Trecho de tráfego monitorado (tcpdump)
- * Importante: tempo entre cada tentativa vs. tempo máximo exigido na RFC
 - * Tempo: 75 segundos
 - * 4.4 BSD: leva 76 segundos
 - * Problema: Timeout

Estados x Mensagens



Reset de Conexões

- * Em geral, um Reset é gerado sempre que é recebido um segmento que não parece estar correto para a conexão identificada.
- * Casos
 - * Solicitações de conexões para portas inexistentes
 - * Aborto de conexões
 - * Solicitações de conexões falsas

Reset de Conexões

Arquivo Exibir Imagem Cores Ajuda

576	[10.0.0.4]	[64.124.33.151]	TCP: D=80 S=1342 RST WIN=0
577	[200.163.195.16]	[64.124.33.151]	TCP: D=80 S=1342 RST WIN=0
578	[10.0.0.4]	[64.124.33.155]	TCP: D=80 S=1353 RST WIN=0
579	[200.163.195.16]	[64.124.33.155]	TCP: D=80 S=1353 RST WIN=0
580	[10.0.0.4]	[216.239.39.99]	TCP: D=80 S=1348 RST WIN=0
581	[200.163.195.16]	[216.239.39.99]	TCP: D=80 S=1348 RST WIN=0

TCP: ----- TCP header -----

- TCP: Source port = 1342
- TCP: Destination port = 80 (WWW-HTTP)
- TCP: Sequence number = 754697933
- TCP: Next expected Seq number = 754697933
- TCP: Data offset = 20 bytes
- TCP: Flags = 04
- TCP: ...0... = (No urgent pointer)
- TCP: ...0... = (No acknowledgment)
- TCP:0... = (No push)
- TCP:1... = Reset
- TCP:0... = (No SYN)
- TCP:0... = (No FIN)
- TCP: Window = 0
- TCP: Checksum = 8F70 (correct)
- TCP: No TCP options
- TCP:

Estabelecimento de Conexões Simultâneas

- * É possível que 2 hosts tentem estabelecer conexão entre eles simultaneamente
 - * Ambos executam um *active open*
 - * Exemplo:
 - * Host A solicita conexão ao Host B na porta 777 e usa como porta local 888
 - * Host B solicita conexão ao Host A na porta 888 e usa como porta local 777
- * TCP foi projetado para suportar estes casos
 - * Apenas uma conexão resulta, não duas

Estabelecimento de Conexões Simultâneas

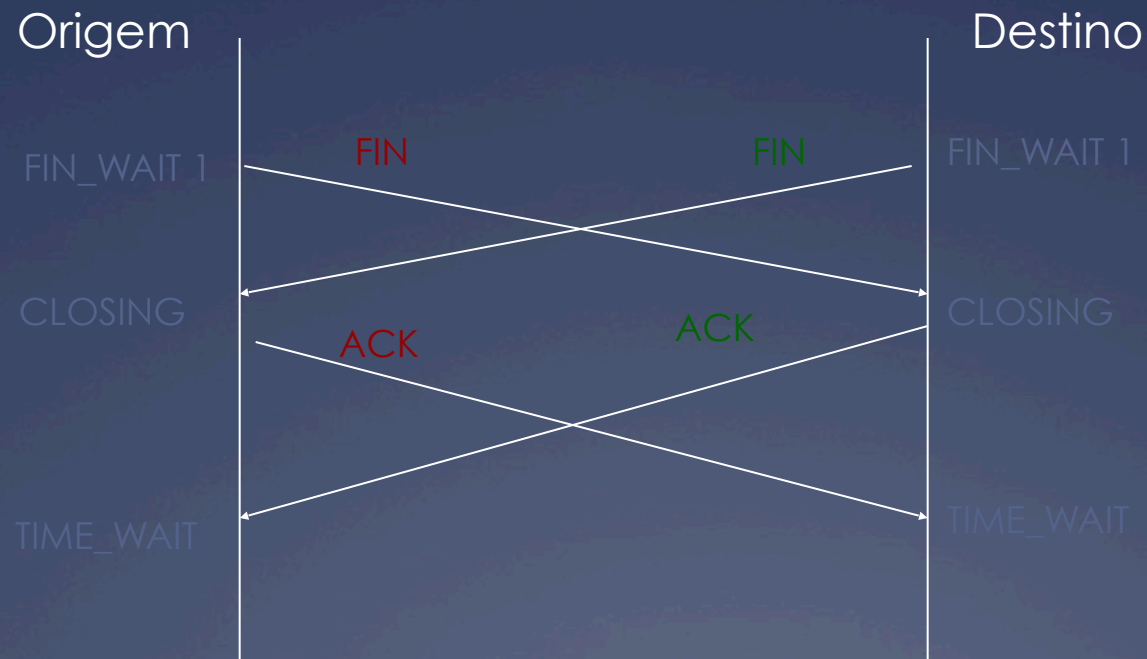
No.	SI	Source Address	Dest Address	Summary
9		[200.163.195.16	doom.cefetpr.br	TCP: D=80 S=3832 SYN SEQ=4176522617 LEN=0 WIN=16384
10		doom.cefetpr.br	[200.163.195.168]	TCP: D=3832 S=80 SYN ACK=4176522618 SEQ=754803995 LEN=0 WIN=5840
11		[200.163.195.16	doom.cefetpr.br	TCP: D=80 S=3832 ACK=754803996 WIN=17424

No.	SI	Source Address	Dest Address	Summary
23		[200.163.195.16	doom.cefetpr.br	TCP: D=80 S=3835 SYN SEQ=4176713861 LEN=0 WIN=16384
24		doom.cefetpr.br	[200.163.195.168]	TCP: D=3835 S=80 SYN ACK=4176713862 SEQ=748884077 LEN=0 WIN=5840
25		[200.163.195.16	doom.cefetpr.br	TCP: D=80 S=3835 ACK=748884078 WIN=17424

```
44 TCP: ----- TCP header -----
TCP:
TCP: Source port          = 3835
TCP: Destination port    = 80 (WWW-HTTP)
TCP: Initial sequence number = 4176713861
TCP: Next expected Seq number = 4176713862
TCP: Data offset         = 28 bytes
TCP: Flags                = 02
TCP:          ...0. .... = (No urgent pointer)
TCP:          ...0. .... = (No acknowledgment)
TCP:          .... 0... = (No push)
TCP:          .... .0.. = (No reset)
TCP:          .... ..1. = SYN
TCP:          .... ....0 = (No FIN)
TCP: Window              = 16384
TCP: Checksum            = FD85 (correct)
TCP:
TCP: Options follow
TCP: Maximum segment size = 1452
TCP: No-op
TCP: No-op
TCP: SACK-Permitted Option
TCP:
```

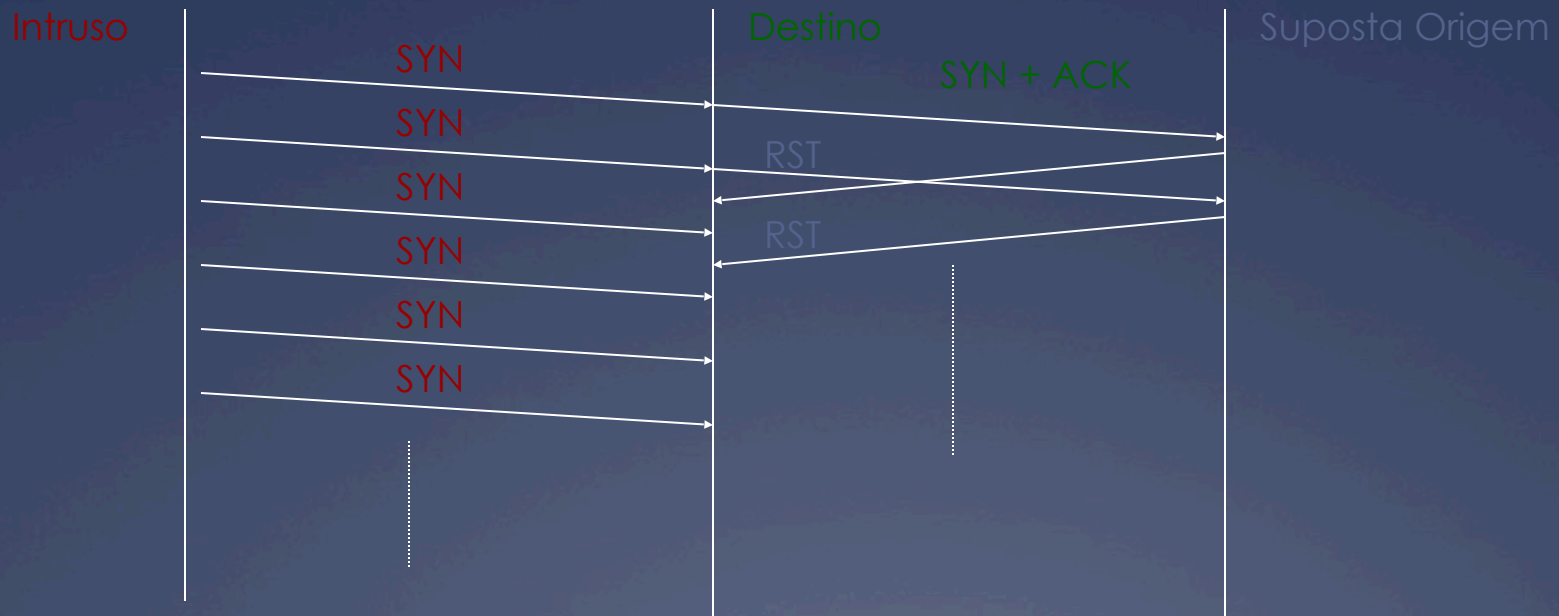
Encerramento de Conexões simultâneas

- * Os hosts também podem tomar a iniciativa de encerrar a conexão simultaneamente



SYN FLOOD

- * Flooding de solicitações de conexão falsas
- * Normalmente usa IP Spoofing

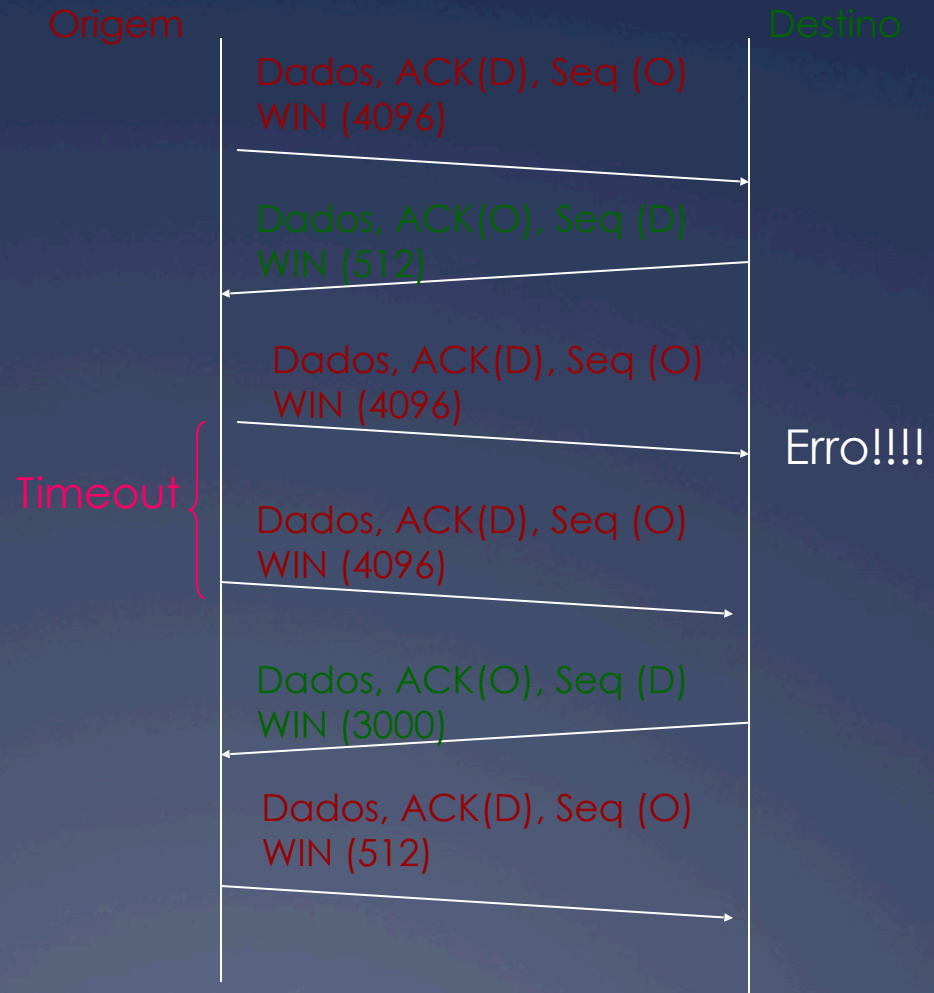


Controle de Erros

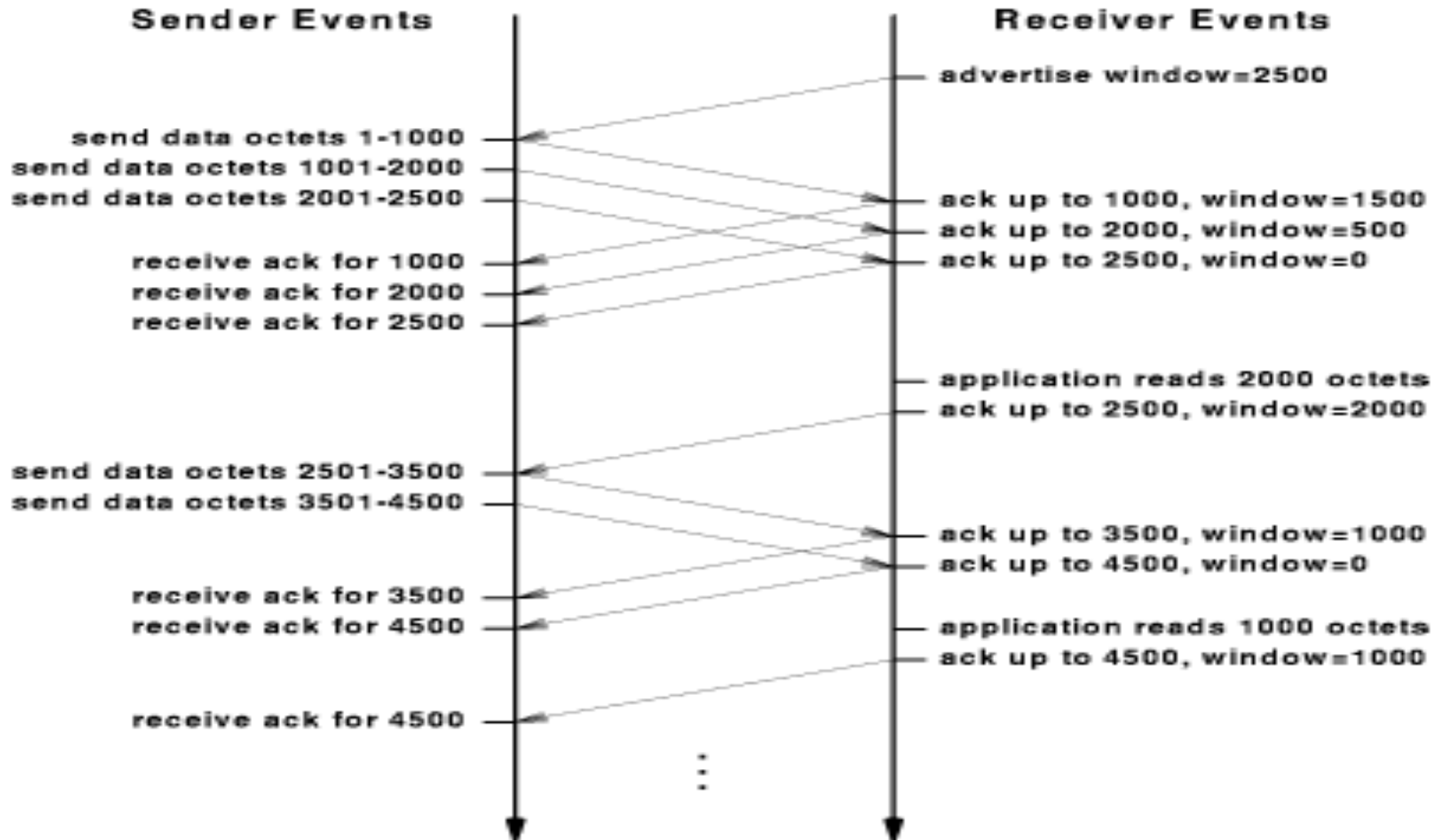
- * O TCP executa controle de erro com retransmissão
 - * Neste caso o checksum não é opcional
 - * Se um segmento TCP é recebido com checksum igual a zero, ele é descartado
 - * O destino envia mensagens de reconhecimento positivo
 - * Não envia NACK
 - * A necessidade de realizar uma retransmissão é detectada pela ausência do ACK

Controle de Fluxo

- * O TCP executa o algoritmo de janela deslizante
- * A cada envio de mensagens o host informa o número de bytes que podem ser recebidos

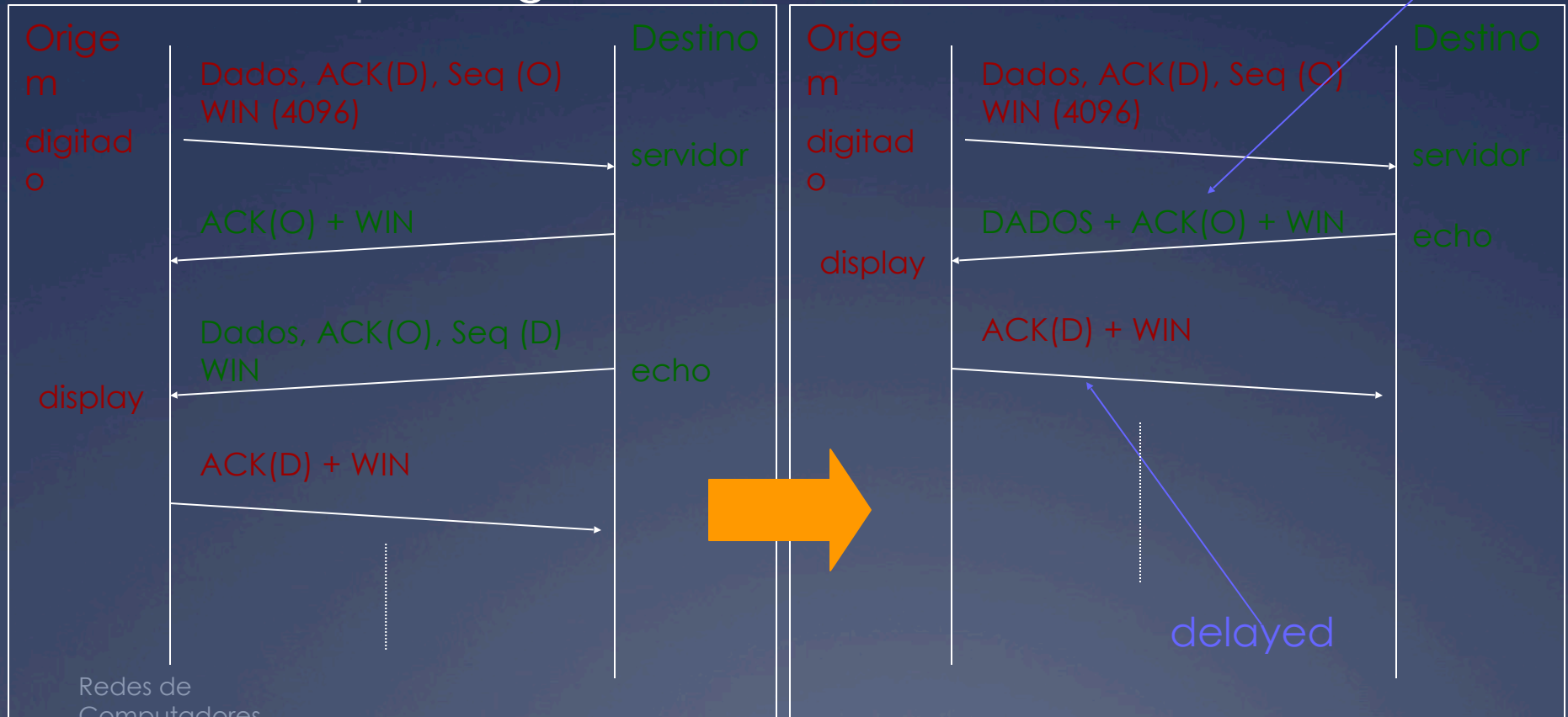


TCP - Janela deslizante



Fluxo Interativo

* Exemplo: rlogin



TCP - Confiabilidade

Events at Host 1

Events at Host 2

send message 1

receive message 1

send ack 1

receive ack 1

send message 2

receive message 2

send ack 2

receive ack 2

send message 3

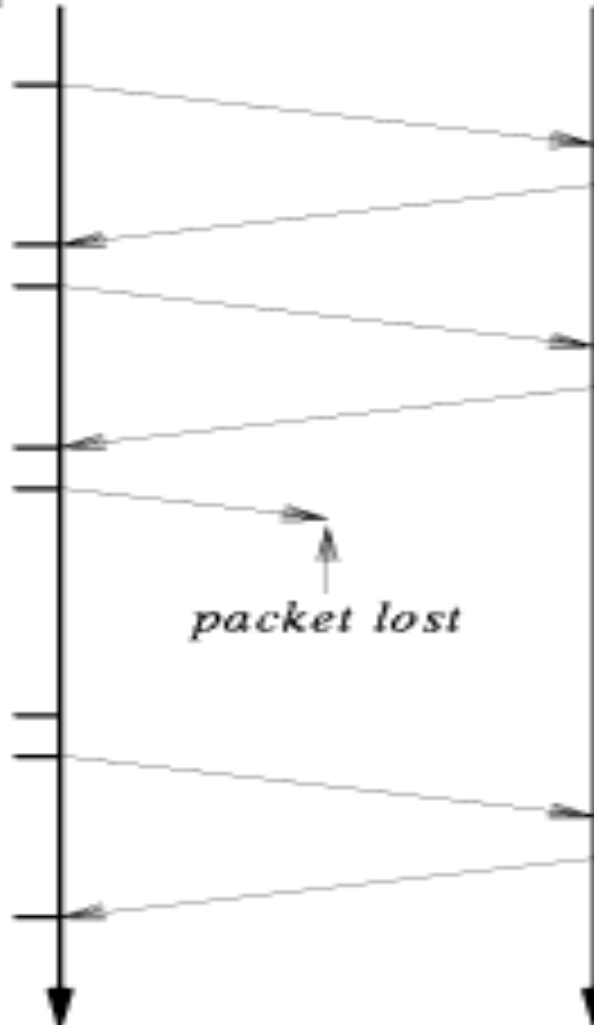
packet lost

retransmission timer expires

retransmit message 3

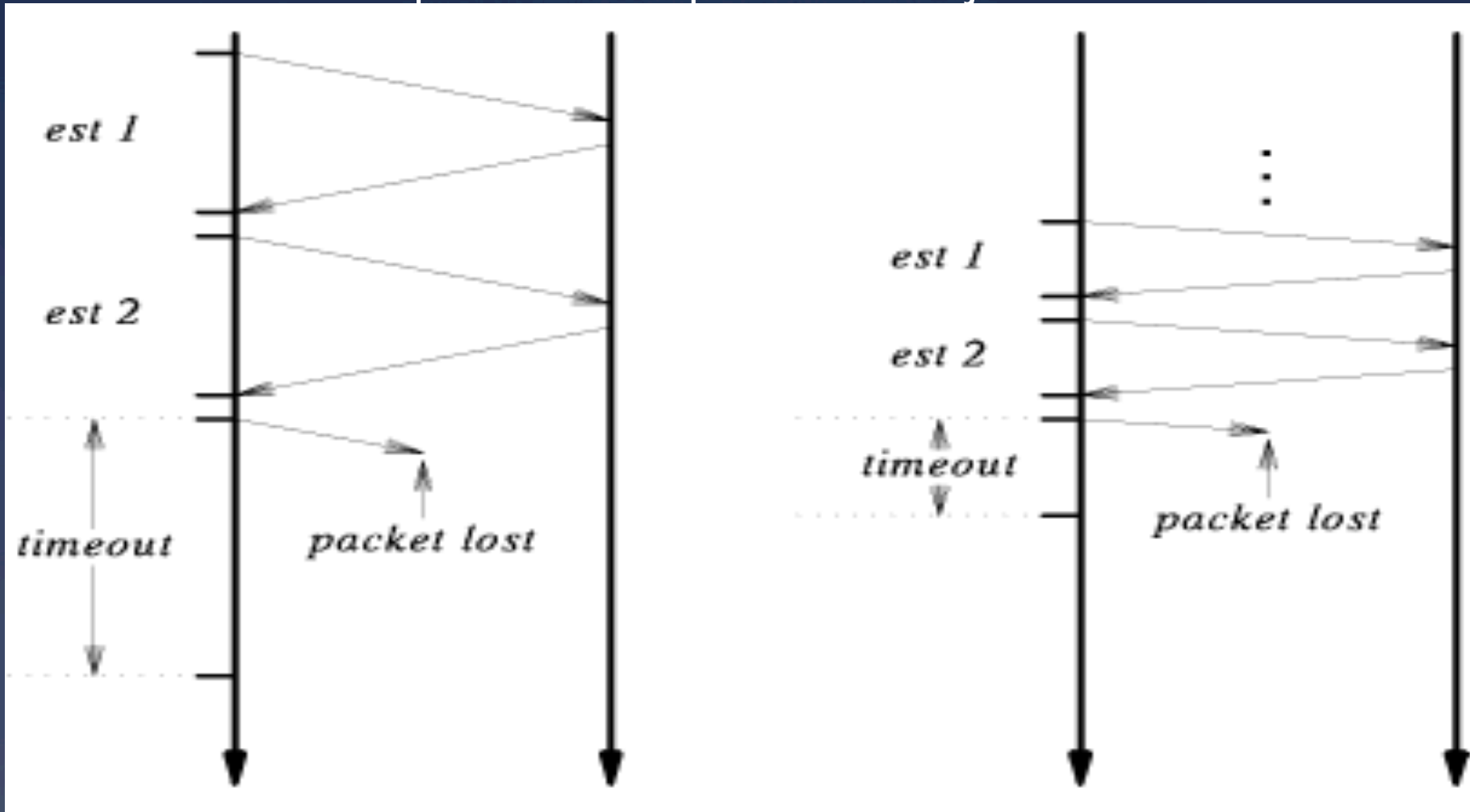
receive message 3

send ack 3



TCP - timeout adaptativo

* Verifica o tempo de resposta e ajusta o timeout



O Nível de Aplicação- Arquitetura TCP/IP

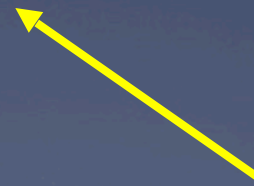
- * aplicações são implementadas de forma isolada
- * não existe um padrão que defina como deve ser estruturada
- * aplicações trocam dados utilizando diretamente a camada de transporte (**TCP ou UDP**) através de chamadas padronizadas API - **Application Program Interface**
 - * **Berkeley Sockets**
 - * **TLI (System V)**

Socket

- * um socket identifica univocamente um usuário TCP
- * permite a associação entre processos de aplicação
- * o identificador da porta é concatenado ao endereço IP, onde a entidade TCP está rodando, definindo um **socket**

socket(23,200.18.5.1)

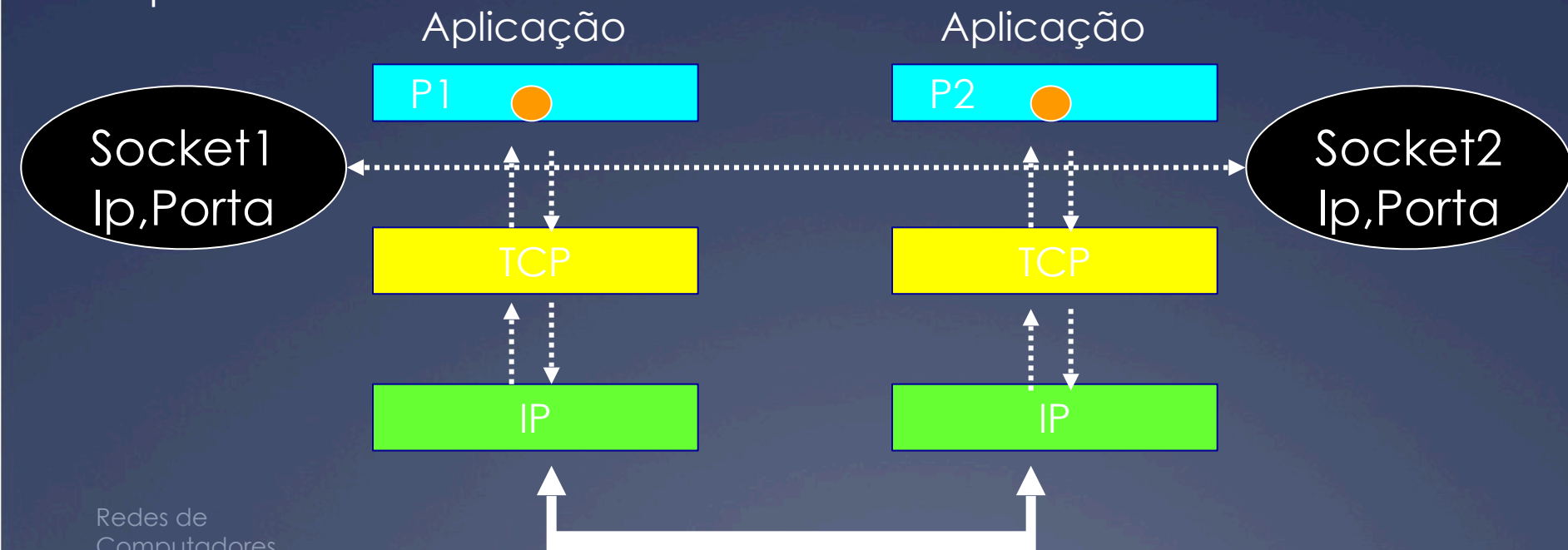
End
Porta



End
IP

Socket

- * A associação entre 2 processos cooperantes (cliente/servidor) é identificada por um par de sockets (socket1, socket2)
- * uma vez estabelecida uma conexão, cada socket corresponde a um ponto final dessa conexão



Socket

- * Uma associação entre 2 processos de aplicação é definida como uma quintupla:

{protocolo, end local, processo local, end remoto, processo remoto}

- * protocolo: TCP ou UDP
- * end local ou remoto: porta + end IP
- * processo local ou remoto: identifica a aplicação local (PID)

Já temos endereços suficientes

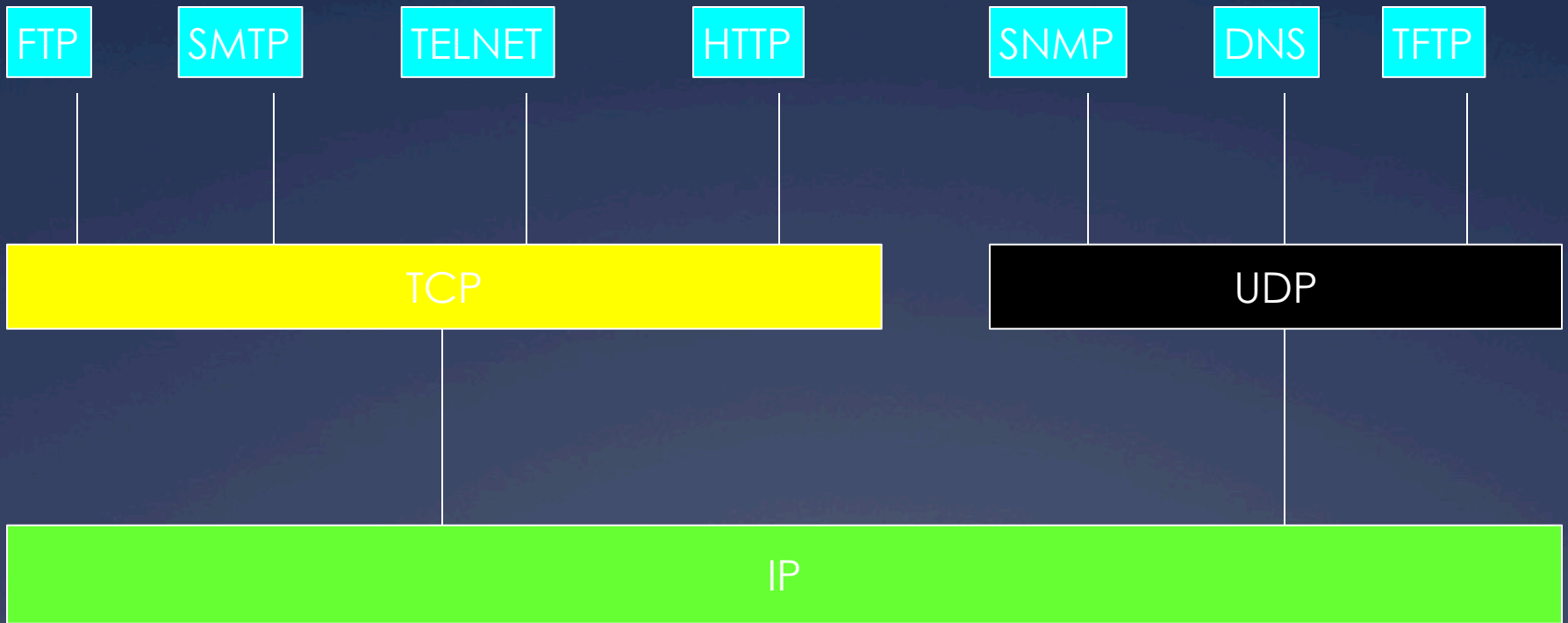
Endereço MAC

Endereço IP

Endereço de Porta

Dados

Aplicações TCP/IP



Protocolos da Camada de aplicação

TELNET - O Telecommunications Network possibilita a um usuário trabalhar em um host distante. Ele emula um terminal especial que faz as conversões necessárias entre dois terminais diferentes, permitindo que se atue remotamente sobre um host sem a necessidade que ambos os hosts tenham um terminal semelhante.

FTP - O File Transfer Protocol é uma poderosa ferramenta para transmitir arquivos pela Internet. Ele define os procedimentos para gerenciar a troca de informações entre os hosts do TCP. Uma conexão do FTP passa por dois processos: Conexão de Controle e Transferência de Dados.

A Conexão de Controle é a primeira etapa do processo de conexão do FTP. Ela serve para realizar o login e definir níveis de segurança e de manipulação de arquivos. FTP - Protocolo de transferência de arquivos - Porta TCP 20 e 21,

Já a Transferência de dados é a etapa onde os arquivos são transmitidos. Ela depende do sucesso da conexão de controle para ser realizada. HTTP - Protocolo de transferência de hipertexto - base da Web, junto com a linguagem HTML e os navegadores. Porta TCP 80.

Protocolos da Camada de aplicação

SMTP - O Simple Mail Transfer Protocol é um aplicativo de nível superior que está ligado à transmissão de correio eletrônico via Internet. Ele é um dos protocolos de nível superior mais utilizados na Internet, que funciona da seguinte forma: O SMTP é constituído de duas partes que são a origem e o destino, sendo que cada uma delas possui acesso a um servidor de armazenamento. Quando a origem envia uma mensagem para o destino, essa mensagem é primeiramente armazenada no servidor de armazenamento da origem. O servidor então tenta enviar as mensagens e, se ocorrer algum problema com o destino, o servidor tentará posteriormente reenviar a mensagem. Se não conseguir, a mensagem será enviada de volta à origem ou ao postmaster. (servidores de E-mail)

DNS – Funcionamento

Analogia com o sistema postal:

Exemplo: lyric.labma.ufrj.br (lyric incluído em labma, que está incluído emufrj, que está incluído em br).

br - domínio por país.

ufrj e labma - sub-domínios.

lyric - máquina.

Estrutura hierárquica dos servidores DNS:

Máquina pede resolução de nome ao DNS.

Se sim, volta o IP.

Se não, o DNS pede ao próximo DNS.

Pode chegar num dos DNS- raiz da Internet.

Se sim, o endereço resolvido volta no sentido contrário. Se não, o DNS propaga uma mensagem de que o endereço não existe.

URL

Universal Resource Locator

Maneira simples e eficiente de apontar onde está um arquivo (rede ou disco local).

Exemplo:

`ftp://fab:123@ftp.arq.com/pub/dados/beltrano.zip`

- | | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
1. Protocolo: http, ftp, wais, gopher, file, etc.
 2. Separador.
 3. Usuário e senha (quando necessário).
 4. Nome de domínio
 5. Caminho dentro do site
 6. Nome do arquivo.

Outros serviços

SNMP - O Simple Network Management Protocol é o padrão mais expressivo em matéria de gerenciamento de redes. Ele é um protocolo que serve para lidar com eventuais falhas nas redes ou nos equipamentos. O *SNMP* ainda é utilizado para monitoramento das redes. Ele é usado principalmente em redes que utilizam TCP/IP.

HTTPS, DNSSEC, SMTPS, POPS, IMAPS, SSH - protocolos que usam cifragem (criptografia) para aumentar a segurança no diálogo cliente-servidor, desmotivando ataques do tipo "homem no meio" (a informação que vai do cliente para o servidor é copiada por um intruso).

Créditos

- * Estes slides foram coletados de apresentações de vários autores, todas disponibilizadas na internet
 - * Breno Parras
 - * Rebeca Carlos Lagos
 - * Rico Van Der Vinne
 - * Roberto Amaral

Threads

Processes have two characteristics:

- * Resource Ownership
 - * Process includes a virtual address space to hold the process image
 - * the OS performs a protection function to prevent unwanted interference between processes with respect to resources
- * Scheduling/Execution
 - * Follows an execution path that may be interleaved with other processes
 - * a process has an execution state (Running, Ready, etc.) and a dispatching priority and is scheduled and dispatched by the OS

Processes and Threads

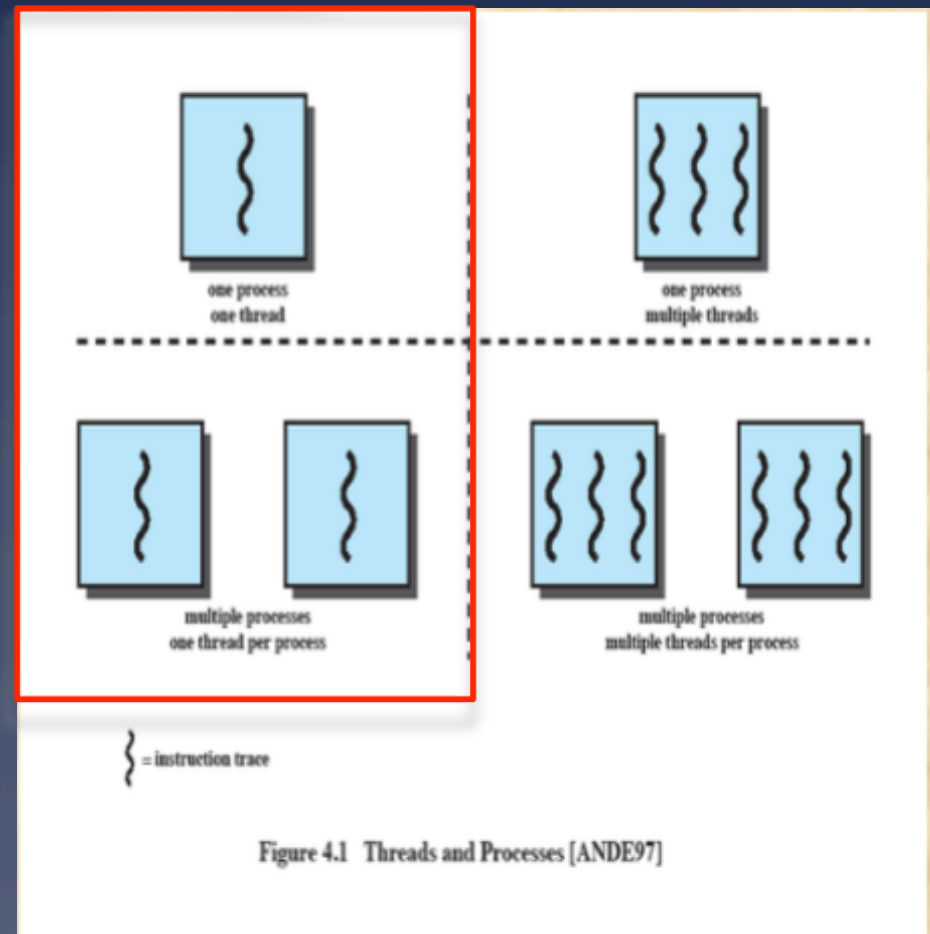
- * The unit of dispatching is referred to as a thread or lightweight process
- * The unit of resource ownership is referred to as a process or task
- * Multithreading - The ability of an OS to support multiple, concurrent paths of execution within a single process

Threads

- * **A process** defines the address space, text, resources, etc.,
- * **A thread** defines a single sequential execution stream within a process (PC, stack, registers).
- * Threads extract the *thread of control* information from the process
- * Threads are bound to a single process.
- * Each process may have multiple threads of control within it.
 - * – The address space of a process is shared among all its threads
 - * – No system calls are required to cooperate among threads
 - * – Simpler than message passing and shared-memory

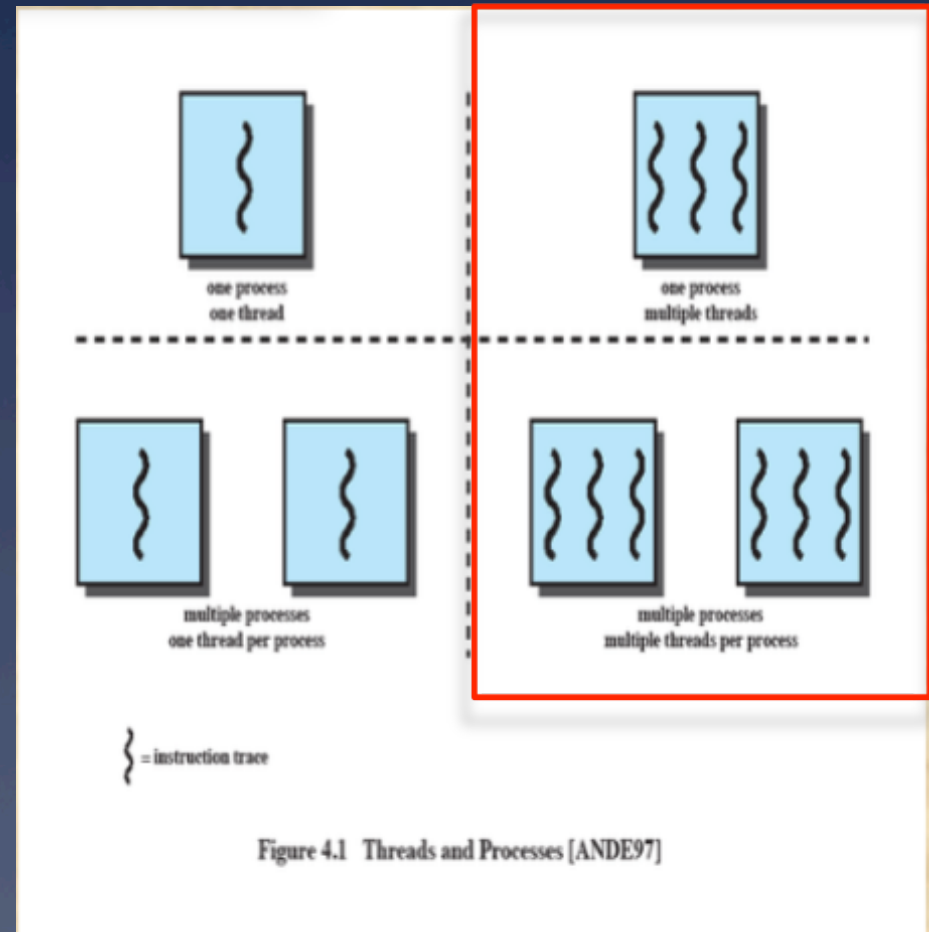
Single Threaded Approaches

- * A single thread of execution per process, in which the concept of a thread is not recognized, is referred to as a single-threaded approach
- * MS-DOS is an example



Multi-threaded approaches

- * The right half of Figure 4.1 depicts multithreaded approaches
- * A Java run-time environment is an example of a system of one process with multiple threads



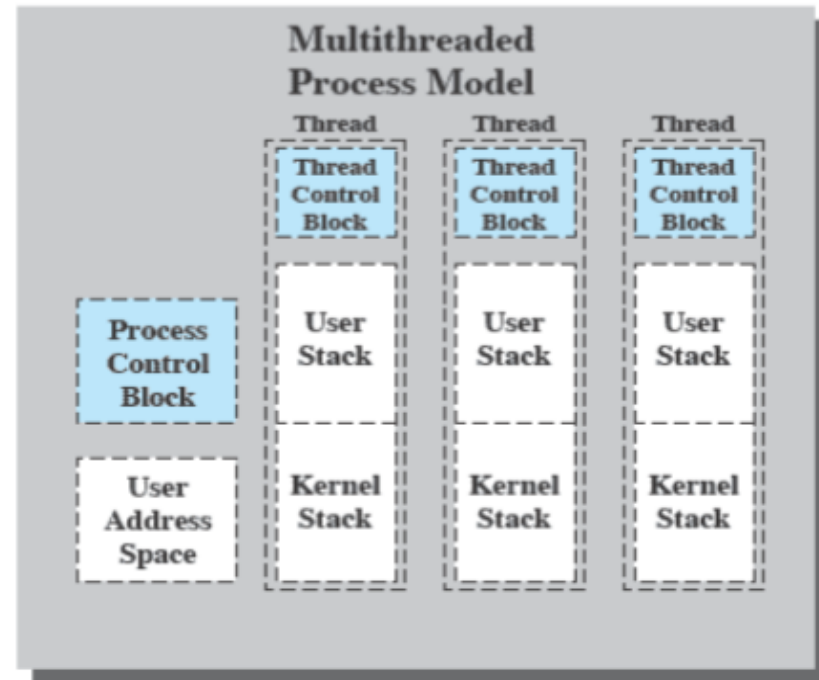
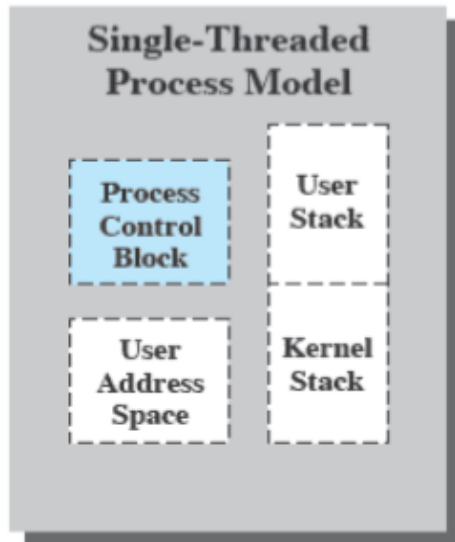
Processes

- * The unit of resource allocation and a unit of protection
- * A virtual address space that holds the process image
- * Protected access to:
 - * Processors
 - * other processes
 - * Files
 - * I/O resources

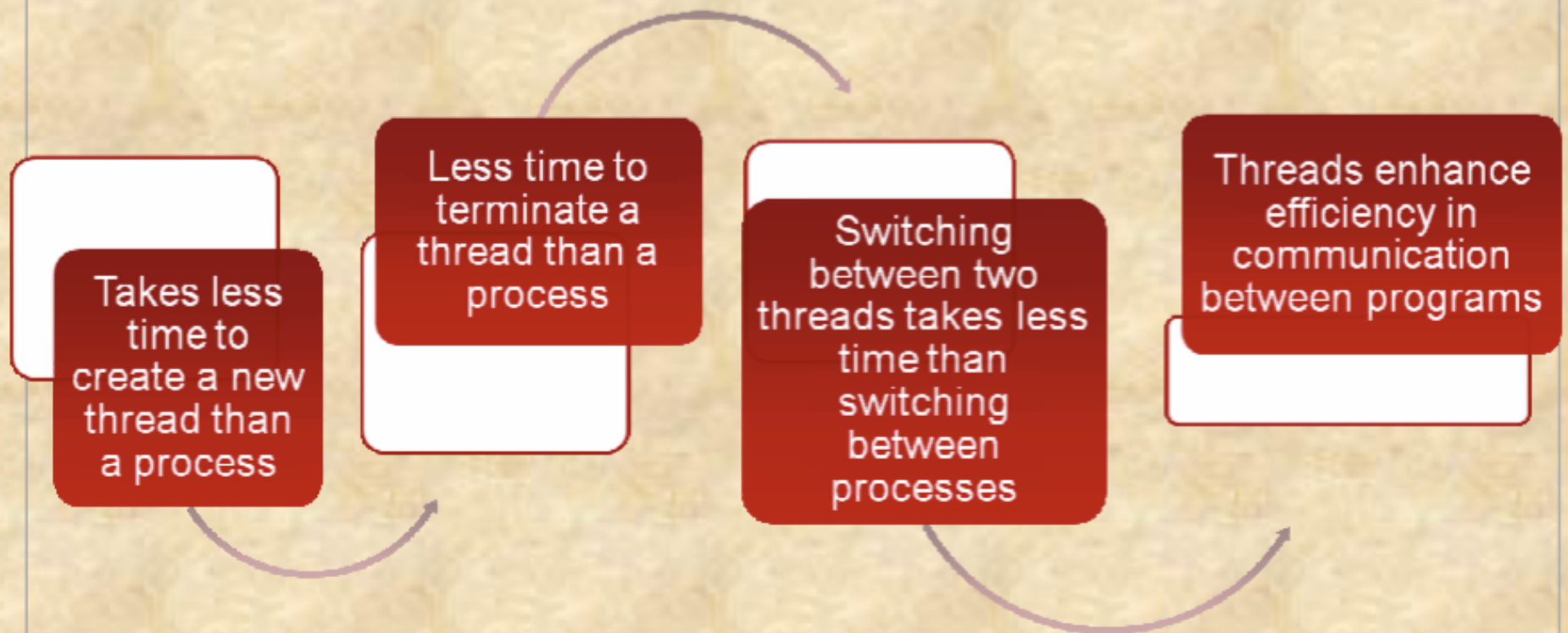
One or more threads in a Process

- * Each thread has
 - * An execution state (running, ready, etc.)
 - * Saved thread context when not running
 - * An execution stack
 - * Some per-thread storage for local variables
 - * Access to the memory and storage to its process (all threads of a process share this)

Threads vs. Processes



Benefits of threads



Thread use in a single-user system

- * Foreground and background work
- * Asynchronous processing
- * Speed of execution
- * Modular program structure

- * In an OS that supports threads, scheduling and dispatching is done on a thread basis
- * Most of the state information dealing with thread execution is maintained in thread-level data structures
 - * suspending a process involves suspending all threads of the process
 - * termination of a process terminates all threads within the process

Thread Execution States

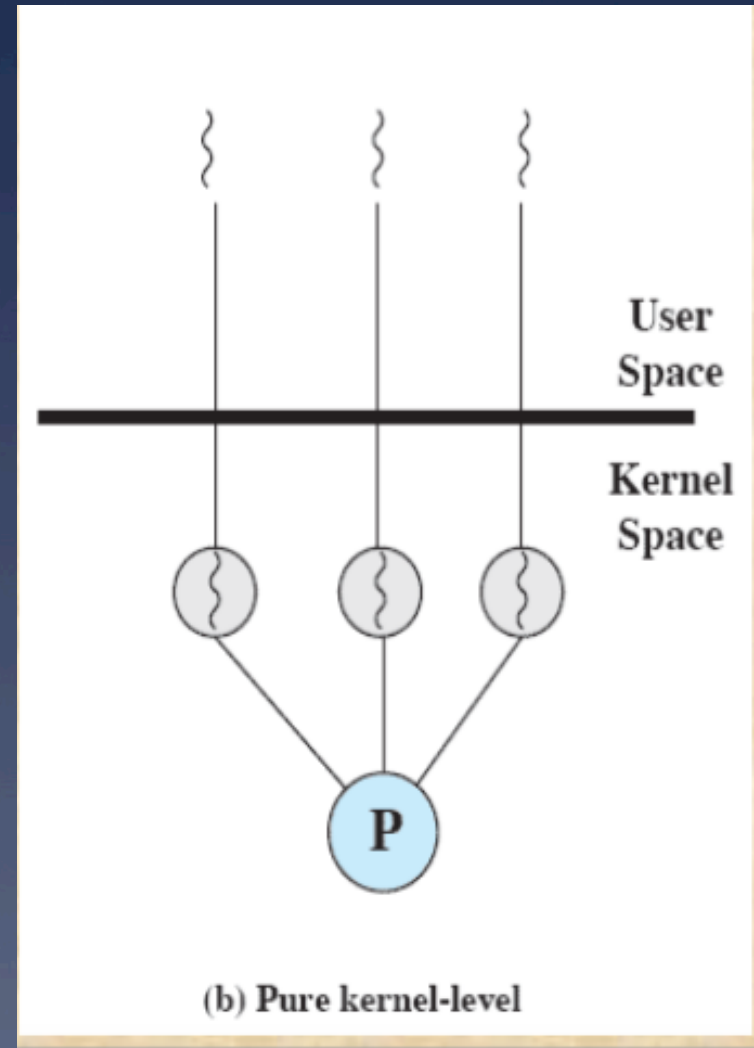
- * Key states in a thread
 - * Running
 - * Ready
 - * Blocked
- Thread operations associated with change in state
 - Spawn
 - Block
 - Unblock
 - Finish

Types of threads

- * User-level Thread (ULT)
- * Kernel-level Thread (KLT)

Kernel-level Thread

- * Thread management is done by the kernel
- * no thread management is done by the application
- * Windows is an example of this approach

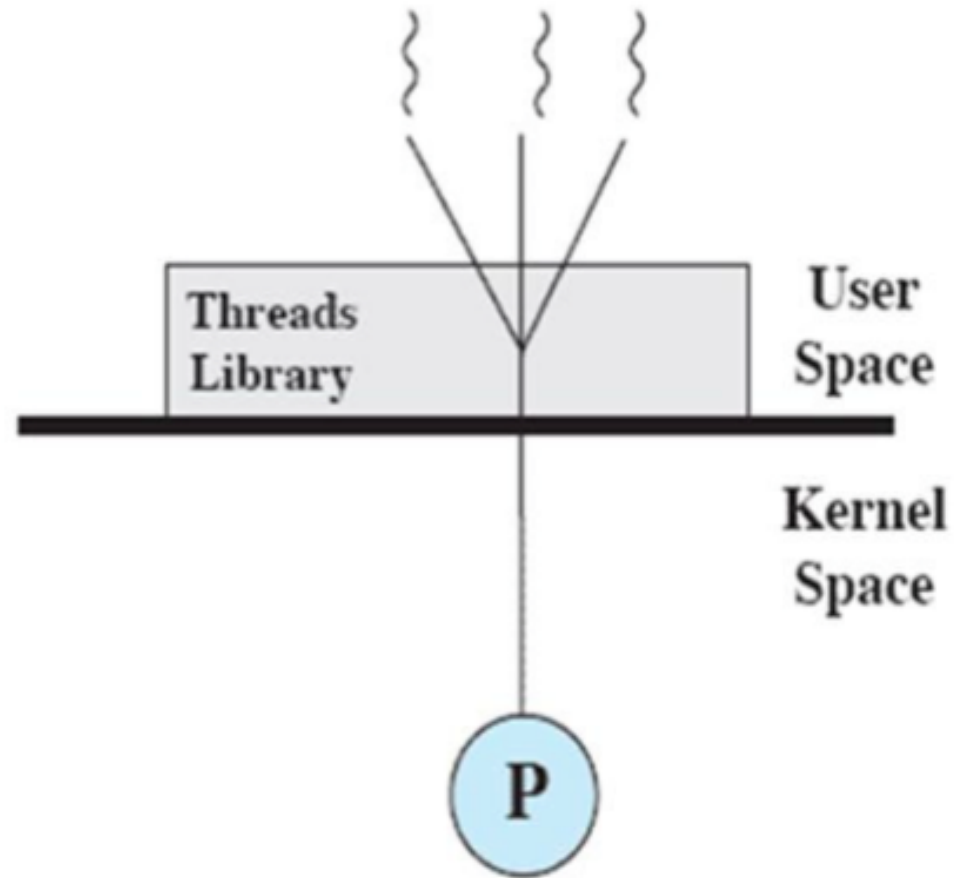


Kernel-level Threads

- * **A kernel thread**, also known as a **lightweight process**, is a thread that the operating system knows about.
- * Switching between kernel threads of the same process requires a small context switch.
 - * The values of registers, program counter, and stack pointer must be changed.
 - * Memory management information does not need to be changed since the threads share an address space.
- * The kernel must manage and schedule threads (as well as processes), but it can use the same process scheduling algorithms.
- * Switching between kernel threads is slightly faster than switching between processes.

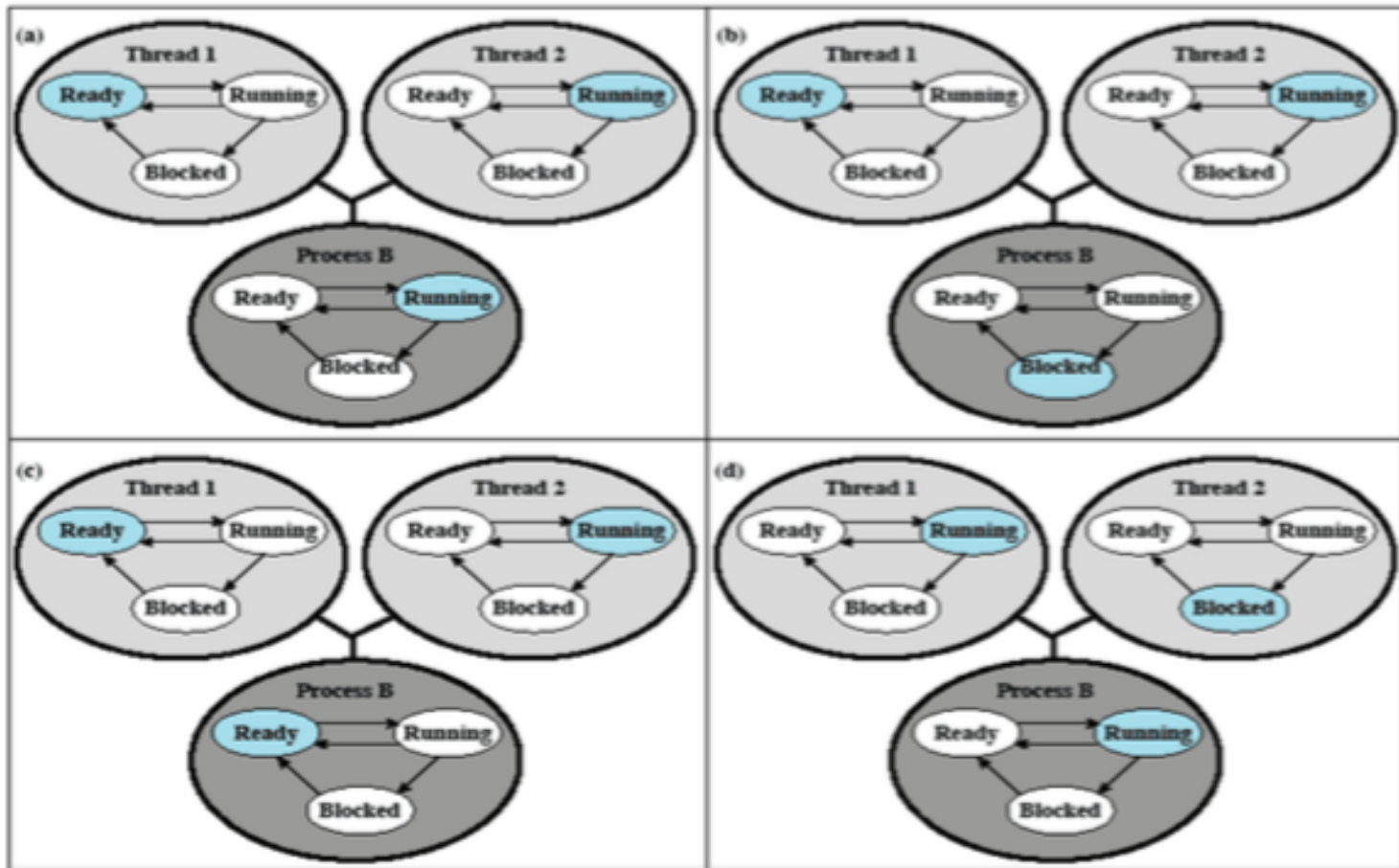
User Level Thread (ULT)

- * All thread management is done by the application
- * The kernel is not aware of the existence of threads



(a) Pure user-level

Processes and User-level threads



Colored state
is current state

Figure 4.6 Examples of the Relationships between User-Level Thread States and Process States

Advantages of ULTs

- * Threads can run on any OS
- * Scheduling can be application-specific
- * Thread switching does not require kernel-mode privileges

Disadvantages of ULTs

- * In a typical OS many system calls are blocking
 - * as a result, when a ULT executes a system call, not only is that thread blocked, but all of the threads within the process are blocked
- * In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing
- * Since the OS does not know about the existence of the user-level threads, it may make poor scheduling decisions:
 - * It might run a process that only has idle threads.
 - * If a user-level thread is waiting for I/O, the entire process will wait.
 - * Solving this problem requires communication between the kernel and the user-level thread manager.
- * Since the OS just knows about the process, it schedules the process the same way as other processes, regardless of the number of user threads.
 - * For kernel threads, the more threads a process creates, the more time slices the OS will dedicate to it

Advantages of KLTs

- * The kernel can simultaneously schedule multiple threads from the same process on multiple processors
- * If one thread in a process is blocked, the kernel can schedule another thread of the same process
- * Kernel routines can be multithreaded

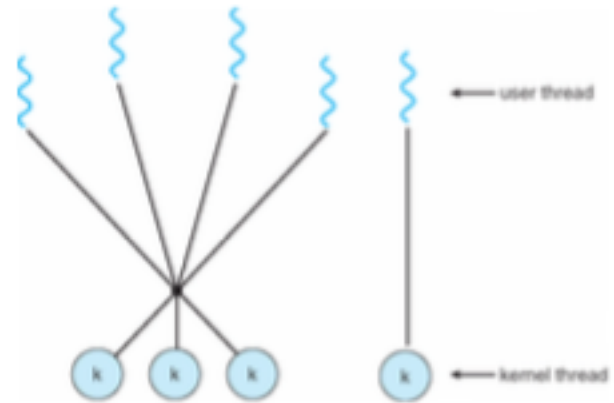
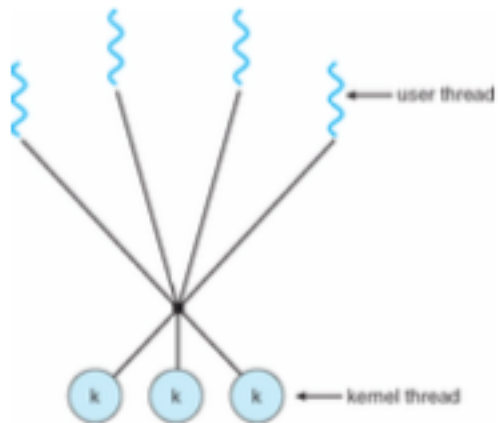
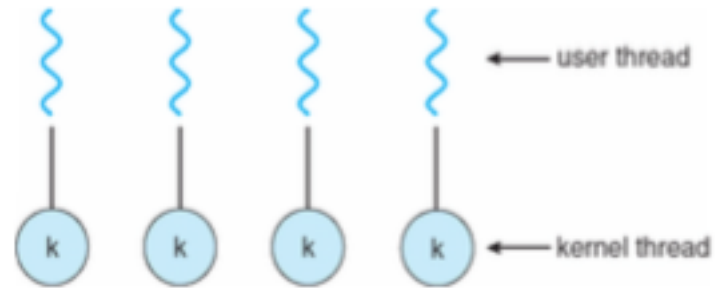
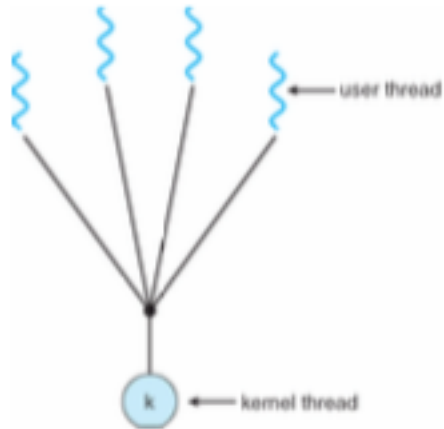
Disadvantages of KLTs

- * TIME: requires a context mode switch to kernel-mode

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

Table 4.1 Thread and Process Operation Latencies (μs)

Threading models



Threading models

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

Table 4.2 Relationship between Threads and Processes

