# Exame Preliminar de Doutorado em Sistemas Operacionais

Duração: 4 horas

22 de fevereiro de 1999

**Atenção:** Apesar da soma dos pontos das questões ser 14, tente responder corretamente o número máximo de questões, a fim de maximizar as suas chances de uma boa avaliação. A nota é o máximo das somas das notas de questões, tomado sobre os subconjuntos de questões cujo valor soma 10. Seja claro e preciso em suas respostas. Se quiser, use figuras para ilustrar suas respostas.
**Boa sorte!**

## Questão 1 *(1 ponto)*

O núcleo do sistema operacional Windows NT consiste basicamente de um *kernel* e um *executive*, com as seguintes tarefas cada:

**kernel** escalonamento de threads; tratamento de interrupções e exceções; sincronização em muitiprocessadores; objetos básicos

**executive** gerenciamento de I/O, de processos e threads, gerenciamento de cache; memória virtual; monitor de segurança e funções da interface gráfica

Quais foram as principais razões para projetar o núcleo nestas duas camadas?

## Questão 2 *(1 ponto)*

Qual é, aproximadamente, o tempo (em milisegundos) necessário para realizar um `fork` de um processo em Unix nas seguintes condições:

- tamanho dos segmentos: texto = 100K, dados = 20K, pilha = 10K;

- tamanho da tabela de processos = 1K e da estrutura do usuário (user structure) = 5K;
- cada trap (com retorno) leva aproximadamente 1 ms e precisa-se de 500 nanosegundos para copiar uma palavra de 4 bytes.

### Questão 3 *(1 ponto)*

Considere um algoritmo para escalonamento com as seguintes características:

- preemptivo;
- baseado em prioridades;
- números maiores indicam maior prioridade;
- no momento em que um processo entra na fila de prontos, fica com prioridade 0;
- enquanto um processo está esperando pela CPU na fila de prontos sua prioridade vai caindo numa taxa $\alpha$;
- enquanto o processo está executando na CPU sua prioridade muda com uma taxa $\beta$;
- As taxas de mudança dinâmica de prioridade $\alpha$ e $\beta$ são definidas em tempo de compilação do algoritmo de escalonamento.

Os valores atribuídos a $\alpha$ e $\beta$ podem levar a muitos algoritmos distintos de escalonamento.

- Que algoritmo resulta de $\alpha = -\beta$ ?
- Que algoritmo resulta de $\alpha < \beta < 0$?
- Que algoritmo resulta de $\alpha > \beta > 0$?

### Questão 4 *(3 pontos)*

Uma loja de brinquedos está vendendo kits para carros controlados remotamente por um computador que executa um programa de controle para todos os carrinhos. O kit inclui:

- uma única pista sobre a qual os carrinhos podem se movimentar. A largura da pista comporta no máximo um carrinho e pode ser percorrida nos dois sentidos.
- duas garagens, ou seja, dois cercados de onde todos os carrinhos devem partir. Cada garagem deve ser acoplada a uma ponta da pista;

- N carrinhos, cada um com um número indicando a sua prioridade (configurada de fábrica).Números maiores indicam maior prioridade.
- uma biblioteca de funções para a linguagem C, que oferece as seguintes opções de controle:
  - `int avança(int i)`, que causa o movimento de 1 cm do carrinho $i$ para a frente em cima da pista.
    Se o carrinho não está em cima da pista, a rotina retorna -1. Se o carrinho encontra algum obstáculo no meio da pista (isto é, colide com outro carrinho em movimento no sentido contrário), ele pára o movimento e retorna com 0. Se o movimento está desempedido, o carro percorre 1 centímetro; se chegou na extremidade da pista retorna o valor 2, senão retorna o valor 1;
  - `int vai_para_largada(int i )`, que faz com que o carrinho $i$ se movimente da garagem para a ponta de entrada da pista, retornando o valor 1. Se o carrinho não se encontra na garagem, a rotina devolve -1; se a ponta de largada já estava ocupada ocorre uma colisão entre os carrinhos, com valor retornado sendo 0;
  - `int volta_para_garagem(int i)`, que faz com que o carrinho $i$ se movimente da ponta da pista para a garagem. Se o carrinho não está na ponta da pista, a rotina não tem efeito algum e retorna -1. Em caso de sucesso no retorno à garagem, a rotina retorna 1;
  - `void vire(int i)`, que faz com que o carrinho $i$ rotacione 180 graus (ou seja, vire o sentido do movimento) se o carrinho estiver parado em algum ponto da pista. A rotina não tem efeito algum se o carrinho estiver na garagem, na ponta de entrada da pista ou em movimento.

Sua tarefa é escrever um programa que, usando a biblioteca disponível no kit, controle o movimento da coleção de $n$ carrinhos ($c_1, c_2, \ldots, c_n$, com prioridades associadas $p_1, p_1, \ldots, p_n$) de tal forma que todos os carrinhos consigam chegar à garagem oposta à sua garagem de origem. Cada carrinho inicialmente se encontra em uma das duas garagens (aleatoriamente) e antes de percorrer a pista deve se posicionar na ponta da mesma. A prioridade entre os carrinhos deve ser respeitada:

- a saída da garagem e posicionamento na ponta da pista deve ser feita pelo carrinho de maior prioridade ainda na garagem;
- se um carrinho de prioridade inferior está se movimentando sobre a pista enquanto um carrinho de prioridade superior se posiciona na ponta contrária da pista, o carrinho em movimento deve recuar e retornar à garagem, liberando a pista para o carrinho de maior prioridade.

Cada ponta da pista comporta um único carro de cada vez. Vários carrinhos indo na mesma direção podem percorrer a pista ao mesmo tempo. Se em um dado momento carrinhos com sentidos contrários estiverem percorrendo a pista uma colisão ocorrerá, o que danifica irremediavelmente o kit.

Implemente a rotina `void carro(int i, int minha_garagem)` (minha_garagem é 0 para a da ponta direita, 1 para a garagem da ponta esquerda) a ser executada por cada carrinho de forma a evitar colisões de carrinhos na pista e respeitar as regras de prioridade no uso da pista.

Não é necessário lidar com *fairness* (justiça), ou seja, garantir que os carrinhos de mesma prioridade númerica sejam tratados igualmente. Defina as variáveis globais que considerar necessárias, e seus valores iniciais. Use semáforos na sua implementação.

## Questão 5 *(1 ponto)*

Explique por que é necessário executar um programa verificador da consistência do sistema de arquivos (no Unix é o `fsck`) depois de um *system crash.*

## Questão 6 *(1 ponto)*

Comente os prós e contras das diferentes formas de se implementar um diretório (p.ex: lista encadeada vs. tabela de hash contendo ponteiros para os i-nodes).

## Questão 7 *(2 pontos)*

Em sistemas que possibilitam fazer a E/S de dados armazenados no espaço de endereçamento de processos (em modo usuário):

1. Qual é o problema que pode ocorrer devido à paginação por demanda?
2. Quais são as possíveis soluções para evitar este problema?

## Questão 8 *(1 ponto)*

Discuta a eficácia do seguinte algorítmo de remoção de páginas de memória (*page replacement*): "Cada página contém um contador de referências. A cada interrupção do relógio, todas as páginas na memória são percorridas e para aquelas que foram referenciadas desde o última interrupção, incrementa-se o contador de referências. Quando ocorre um page fault, a página com o menor contador é escolhida para remoção."

## Questão 9 *(1 ponto)*

Suponha que um sistema use o algorítmo SSTF (Shortest Seek Time First) de escalonamento de acesso a disco e que o acesso blocos contendo entradas de diretórios é muito frequente. Em que região do disco (p.ex. trilhas e setores) deveriam estar localizados estes blocos para diminuir o tempo médio de acesso a informação de diretório?

## Questão 10 *(1 ponto)*

Onde é armazenado, no sistema de arquivos do Unix o indicador da posição corrente de um aqruivo? Por que?

## Questão 11 *(1 ponto)*

No Unix, qual é a diferença entre as sequências de caracteres filtradas (*cooked character stream*) e de caracteres crús (*raw character stream*) e qual é o elemento transformador entre as sequências?