Programação Concorrente – Aula 13

Gilmar Gimenes Rodrigues

1 Continuação de Monitores.

Com uma variável de condição.

```
monitor bounded_buffer{
    T[N] buf;  // T é de um tipo qualquer.
    int inicio = 0;
    final = 0;
    int conta = 0;

    cond evento;  // Não vazio ou não cheio.

    void poe(T item) {
        while(conta >= n)
            wait(evento);
        buf[final] = item;
        final = final + 1) % N;
        conta++;
        signal (evento); // Não vazio.
    }

    T pega() {
        while(conta <= 0)
            wait(evento);
        T val = buf[inicio];
        inicio = (inicio + 1) % N;
        conta--;
        signal(evento);  // Não cheio.
        return val;
    }
}
```

2 Problema do Barbeiro (Sleeping Barber)

```
monitor Barbearia {
    void corta_cabelo();     // Método chamado pelos clientes para cortar cabelo.
    void proximo_cliente();
    void corte_terminado();  // Chamado pelo barbeiro.
}

thread cliente() {
    for(;;) {
```

```
        barbearia.proximo_cliente();
        ...  // corta o cabelo do cliente.
        barbearia.corte_terminado();
    }
}

thread cliente() {
    for(;;) {
        barbearia.corta_cabelo();
        ... //faz outras coisa.
    }
}

monitor Barbearia {
    int barbeiro = 0;  // 0 ou 1, indicando disponibilidade do barbeiro.
    int cadeira = 0;   // 0 ou 1, indicando se está ocupada ou não.
    int saida = 0;     // 0 ou 1, indicando se a saida está aberta.

    cond barbeiro_disponivel;
    cond cadeira_ocupada;
    cond saida_aberta;
    cond cliente_saiu;

    void corta_cabelo() {
        while(barbeiro == 0)
            wait(barbeiro_disponivel);
        barbeiro--;
        cadeira++;
        signal(cadeira_ocupada);
        while(saida == 0)
            wait(saida_aberta);
        saida--;
        signal(cliente_saiu);
    }

    void proximo_cliente() {
        barbeiro++;
        signal(barbeiro_disponivel);
        while(cadeira == 0)
            wait(cadeira_ocupada);
        cadeira--;
    }

    void corte_terminado() {
        saida++;
        signal(saida_aberta);
        while(saida > 0)
            wait(cliente_saiu);
    }
}
```

3 Pthreads

POSIX threads;

Implementação de threads padronizada.
Biblioteca C
Linuxthreads implementa o padrão Pthreds
Todas as chamadas da Pthreads começam com "phread_..."
Para isso deve-se usar: #include<pthread.h>
No link, passar -lpthread

- pthread_... Gerenciamento de threads
- pthread_attr_... Manipula atributos de threads (propriedades)
- pthread_mutex_... para fazer exclusão mutua
- pthread_mutex_attr_... para atributos de mutex (opcional)
- pthread_cond_... variáveis de condição. Para sincronização condicional.
- pthread_cond_attr_... atributos de variáveis de condição (opcional)

```
#include <pthread.h>}
pthread_create(pthread_t *tid, cond pthread_attr_t *attr, void *(*func)(void), arq);
```

pthread_t *tid: Ponteiro para função/cod. da thread. Valor é retornado.
cond pthread_attr_t *attr: Normalmente null.
void *(*func)(void *): função que recebe como argumentos um ponteiro para void e retorna um ponteiro para void.