



Gerenciamento de Recursos I

2º Trabalho em Grupo

O problema dos leitores/escritores, apresentado a seguir, consiste em sincronizar processos que consultam/atualizam dados em uma base comum. Pode haver mais de um leitor lendo ao mesmo tempo; no entanto, enquanto um escritor está atualizando a base, nenhum outro processo pode ter acesso a ela (nem mesmo leitores).

```
VAR  Acesso: Semaforo := 1;  
      Exclusao: Semaforo := 1;  
      Nleitores: integer := 0;
```

```
PROCEDURE Escritor();  
BEGIN  
    ProduzDado();  
    DOWN (Acesso);  
    Escreve();  
    UP (Acesso);  
END;
```

```
PROCEDURE Leitor();  
BEGIN  
    DOWN (Exclusao);  
    Nleitores := Nleitores + 1;  
    IF ( Nleitores = 1 ) THEN DOWN (Acesso);  
    UP (Exclusao);  
    Leitura();  
    DOWN (Exclusao);  
    Nleitores := Nleitores - 1;  
    IF ( Nleitores = 0 ) THEN UP (Acesso);  
    UP (Exclusao);  
    ProcessaDado();  
END;
```



Gerenciamento de Recursos I

Semáforos

- a) Suponha que exista apenas um leitor fazendo acesso à base. Enquanto este processo realiza a leitura, quais os valores das três variáveis?
- b) Chega um escritor enquanto o leitor ainda está lendo. Quais os valores das três variáveis após o bloqueio do escritor ? Sobre qual(is) semáforo(s) se dá o bloqueio?
- c) Chega mais um leitor enquanto o primeiro ainda não acabou de ler e o escritor está bloqueado. Descreva os valores das três variáveis quando o segundo leitor inicia a leitura.
- d) Os dois leitores terminam simultaneamente a leitura. É possível haver problemas quanto à integridade do valor da variável nleitores? Justifique.
- e) Descreva o que acontece com o escritor quando os dois leitores terminam suas leituras. Descreva os valores das três variáveis quando o escritor inicia a escrita.
- f) Enquanto o escritor está atualizando a base, chegam mais um escritor e mais um leitor. Sobre qual(is) semáforo(s) eles ficam bloqueados? Descreva os valores das três variáveis após o bloqueio dos recém-chegados.
- g) Quando o escritor houver terminado a atualização, é possível prever qual dos processos bloqueados (leitor ou escritor) terá acesso primeiro à base?
- h) Descreva uma situação onde os escritores sofram starvation (adiamento indefinido).



Gerenciamento de Recursos I

Problema do Barbeiro Dorminhoco

```
Semaforo clientes=?;  
Semaforo barbeiro=?;  
Semaforo mutex=?;  
int sentados=0;      /* #clientes sentados */  
  
barbeiros() {  
    while(1) {  
        down(clientes); /* existem clientes? se não adormece */  
        down(mutex);  
        sentados --; /* menos um cliente à espera */  
        up(barbeiro); /* menos um barbeiro adormecido */  
        up(mutex);  
        cortar();  
    }  
}  
  
clientes() {  
    down(mutex); /* se não existem cadeiras livres */  
    if (sentados < NCads) { /* vai embora; se existem entra */  
        sentados ++; /* mais um cliente à espera */  
        up(clientes); /* acorda barbeiro se necessário */  
        up(mutex); /* liberta zona crítica */  
        down(barbeiro); /* adormece se não há barbeiros livres */  
        sentar_e_cortar();  
    } else  
        up(mutex);  
}
```

Quais são os valores de:

- clientes =

- barbeiro =

- mutex =

a) Considere apenas 1 processo
barbeiros ativo.

b) Considere vários processos
barbeiros ativos.



Gerenciamento de Recursos I

Problema da Padaria

```
#define N 5
typedef struct
{
    int        numero; /* número do cliente que deve ser atendido */
    SEMA       pedido; /* semáforo utilizado para bloquear o vendedor */
                /* enquanto o cliente não está pronto. */
} TVENDEDOR;

TVENDEDOR vendedor [N] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int        senha = 0; /* senha de atendimento */
int        proximo = 0; /* próximo número a ser atendido */
MUTEX      mutex_senha = 1; /* semáforo de exclusão mútua */
MUTEX      mutex_proximo = 1; /* semáforo de exclusão mútua */

void vendedor (int id) { /* id – identificação do vendedor */

    while (1) {
        /* vê o próximo número a ser atendido */
        DOWN (&mutex_proximo);
        vendedor[id].numero = proximo ++;
        UP (&mutex_proximo);

        /* aguarda o cliente pedir */
        DOWN (&vendedor[id].pedido);
        atende_cliente ();
    }
}
```



Gerenciamento de Recursos I

Problema da Padaria

```
void cliente (void) {  
  
    int numero, i;  
  
    /* pega um número */  
    DOWN (&mutex_senha);  
    numero = senha ++;  
    UP (&mutex_senha);  
  
    /* verifica se há um vendedor para atendê-lo */  
    atendido = 0;  
    while (! atendido) {  
        for (i = 0; i < N; i ++)  
            if (vendedor [i].numero == numero) {  
                atendido = 1;  
                break;  
            }  
    }  
    UP (&vendedor[i].pedido);  
    faz_pedido (i);  
}
```

- a) Inicialmente são criados 5 processos representando os 5 vendedores. Estes processos ficam bloqueados? Se a resposta for afirmativa, em que semáforos?
- b) Chega um cliente (processo cliente é criado). Ele é atendido por um vendedor. Qual o valor das variáveis (*senha*, *proximo*, *mutex_senha* e *mutex_proximo*) quando o processo cliente está executando a função *faz_pedido* e o processo vendedor (que o atendeu) está executando a função *atende_cliente*? É possível afirmar qual processo vendedor o atendeu (número do processo vendedor)? Justifique.
- c) Existe alguma situação neste algoritmo que possa causar starvation? Justifique.
- d) Enquanto o primeiro cliente está sendo atendido (executando *faz_pedido*) chegam a padaria mais 9 clientes (mais 9 processos clientes são criados). Quatro destes clientes serão atendidos logo pelos vendedores. Os outros cinco ficarão esperando. Se estes cinco clientes forem embora (os processos forem “mortos”), o que acontece? Justifique.



Gerenciamento de Recursos I

Produtor & Consumidor - Troca de Mensagens

Observe o trecho de pseudo-código abaixo:

```
program so1_2003;
```

```
const null = " ";  
var i, c: integer;
```

```
procedure produtor;
```

```
var p1: message;
```

```
begin
```

```
    repeat
```

```
        receive(produz, p1);
```

```
        p1 := produto;
```

```
        send(produz, p1);
```

```
    forever
```

```
end;
```

```
Procedure consumidor;
```

```
var p2: message;
```

```
begin
```

```
    repeat
```

```
        receive(consome, p2);
```

```
        produto := p2;
```

```
        send(consome, null);
```

```
    forever
```

```
end;
```

```
begin
```

```
    create_mailbox(produz);
```

```
    create_mailbox(consome);
```

```
    for i = 1 to c do send(consome, null);
```

```
        parbegin
```

```
            produtor;
```

```
            consumidor;
```

```
        parend;
```

```
end.
```



Produtor & Consumidor - Troca de Mensagens

- a) Existem algumas instruções erradas. Mostre quais são elas e como devem ser corrigidas para que o programa funcione corretamente, isto é, garanta a exclusão mútua através do uso de mensagens.
- b) Para que o programa funcione corretamente, que características as instruções `send()` e `receive()` precisam satisfazer?
- c) Seria possível usar este mesmo código num contexto com vários produtores e consumidores em paralelo? Justifique sua resposta.
- d) Segundo o código, cada item produzido precisa ser consumido imediatamente? Justifique sua resposta.