

## Sistemas de Programação II

INF 02828

(Engenharia de Computação / Elétrica)

## Sistemas Operacionais

INF 02780

(Ciência da Computação)

### 2ª. Lista de Exercícios

**Data de Entrega: não há.** O objetivo da lista é ajudar no estudo individual dos alunos. Soluções de questões específicas

leitura a uma base de dados e diversos processos que eventualmente fazem acessos de escrita à mesma base. Vários acessos de leitura podem ocorrer simultaneamente, mas um acesso de escrita não pode ocorrer simultaneamente com acessos de nenhum tipo. Considere o código a seguir para os processos de leitura e de escrita. Suponha que todos os semáforos são iniciados com valor 1:

leitor:

```
...
1 while(1) {
2 P(R);
3 P(M);
4 rc++;
5 if (rc==1) P(W);
6 V(M);
7 V(R);
8 LE;
9 P(M);
10 rc--;
11 if (rc==0) V(W);
12 V(M);
13 ... consome
```

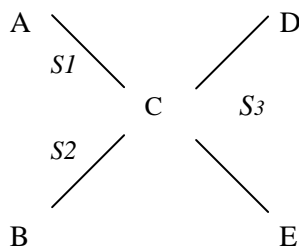
escritor:

```
...
1 while(1) {
2 ...produz
3 P(R);
4 P(W);
5 ESCREVE;
6 V(W);
7 V(R);
8 }
```

(a) Essa solução pode levar a *starvation* de escritores? (pode acontecer de um escritor nunca conseguir o acesso à base devido à seguida chegada de novos leitores?) Explique sua resposta, usando os números das linhas de código para se referir aos passos do programa.

(b) Explique o papel do semáforo M. Dê um exemplo de problema que poderia ocorrer caso as operações sobre ele fossem retiradas.

1. Adicione semáforos ao programa abaixo (S1, S2, S3) e as respectivas chamadas às suas operações, de modo que a precedência definida pelo grafo seja alcançada.



semaphore ...

...

Process *k*

...

do some work *k*

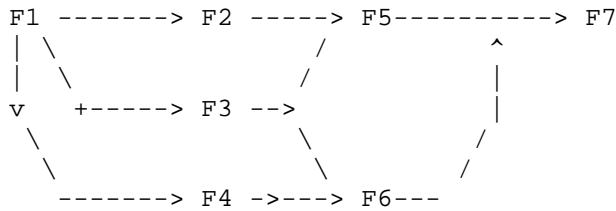
...

end

/\* Comentário \*/

/\* Comentário \*/

2. Considere o seguinte grafo de precedência



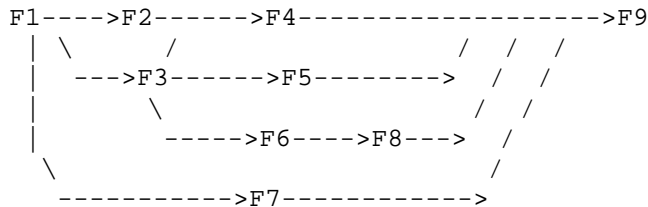
que será executado por três processos, conforme código abaixo:

```

cobegin
  P1: begin F1; F3; end;
  P2: begin F2; F5; F7; end;
  P3: begin F4; F6; end;
coend
  
```

Adicione semáforos a este programa, e as respectivas chamadas às suas operações, de modo que a precedência definida acima seja alcançada.

3. Expresse o grafo de precedência seguinte como um programa concorrente usando fork e wait:



4. Desenhe o gráfico de precedência relativo ao programa abaixo:

```

/* Programa principal */
main () {
  int f1; /* Identifica processo filho 1 */
  int f2; /* Identifica processo filho 2 */
  printf ("Alô do pai\n");
  f1:= fork(codigo_do_filho); /* Cria filho 1 */
  wait(f1);
  printf ("Filho 1 terminou\n");
  f2:= fork(codigo_do_filho); /* Cria filho 2 */
  wait(f2);
  printf ("Filho 2 terminou\n");
  exit(); }

/* Função executada pelos dois processos filho */
codigo_do_filho () {
  printf ("Alô do filho\n");
  exit(); }
  
```

5. Um *semáforo múltiplo* estende semáforos de maneira a permitir que as primitivas *up* e *down* operem em vários semáforos simultaneamente. Isto é útil para requisitar e liberar vários recursos através de uma operação atômica. Como você implementaria tais primitivas usando os semáforos estudados no curso? (Observe que você, como implementador da primitiva, pode acessar o valor do contador associado ao semáforo diretamente. Ou seja, você tem disponível na hora de implementar Down (R1, R2, ..., Rn) ou Up (R1, R2, ..., Rn) uma rotina ```valor_do_contador (R)```, que devolve o valor do contador associado a um semáforo).
6. Resolva usando semáforos: “Três fumantes se encontram em uma sala com um vendedor de suprimentos para fumantes. Para preparar e usar um cigarro, cada fumante precisa de três ingredientes: tabaco, papel e fósforo, coisas que o vendedor tem à vontade no estoque. Um fumante tem o seu próprio tabaco, o segundo tem seu próprio papel, e o outro tem seu próprio fósforo. A ação se inicia quando o vendedor coloca à venda dois ingredientes na mesa, de forma a permitir que um dos fumantes execute esta prática dita como não muito saudável. Quando o tal fumante termina, ele acorda o vendedor, que escolhe então outros dois ingredientes (aleatoriamente) e coloca a venda, portanto desbloqueando outro fumante.”
7. Resolva usando semáforos: “Existem  $n$  passageiros, que repetidamente aguardam para entrar em um carrinho da montanha russa, fazem o passeio, e voltam a aguardar. Vários passageiros podem entrar no carrinho ao mesmo tempo, pois este tem várias portas. A montanha russa tem somente um carrinho, onde cabem  $C$  passageiros ( $C < n$ ). O carrinho só começa seu percurso se estiver lotado.”
8. “Suponha que um grupo de  $N$  canibais come jantares a partir de uma grande travessa que comporta  $M$  porções. Quando alguém quer comer, ele(ela) se serve da travessa, a menos que ela esteja vazia. Se a travessa está vazia, o canibal acorda o cozinheiro e espera até que o cozinheiro coloque mais  $M$  porções na travessa. Desenvolva o código para as ações dos canibais e do cozinheiro – rotinas *canibal*, *seserve*, (chamada por canibal), *cozinheiro* e *enchetravessa* (chamada por cozinheiro). A solução deve evitar deadlock e deve acordar o cozinheiro apenas quando a travessa estiver vazia. Suponha um longo jantar, onde cada canibal continuamente se serve e come, sem se preocupar com as demais na vida de um canibal...”
9. Resolva usando semáforos: “Considere a seguinte situação. Em um determinado stand de uma feira, um demonstrador apresenta um filme sobre a vida de Hoare. Quando 10 pessoas chegam, o demonstrador fecha o pequeno auditório que não comporta mais do que essa platéia. Novos candidatos a assistirem o filme devem esperar a próxima exibição. Esse filme faz muito sucesso com um grupo grande de fãs (de bem mais de 10 pessoas), que permanecem na feira só assistindo o filme seguidas vezes. Cada vez que um desses fãs consegue assistir uma vez o filme, ele vai telefonar para casa para contar alguns detalhes novos para sua mãe. Depois de telefonar ele volta mais uma vez ao stand para assistir o filme outra vez. Usando semáforos, modele o processo fã e o processo demonstrador, lembrando que existem muitos fãs e apenas um demonstrador. Como cada fã é muito ardoroso, uma vez que ele chega ao stand ele não sai dali até assistir o filme. Suponha que haja muitos telefones disponíveis na feira e, portanto, que a tarefa de telefonar para casa não impõe nenhuma necessidade de sincronização.”

OBS: Observe que o demonstrador só pode começar a exibir o filme quando há 10 pessoas no stand, e que as pessoas que chegam durante uma exibição têm que esperar a próxima. É importante: observe que um fã só pode ir telefonar para a mãe depois que acaba a exibição do filme! Isso tem que estar modelado na sincronização entre os processos demonstrador e fãs.”

10. Em um sistema que suporta programação concorrente apenas através da troca de mensagens, será criado um Servidor para controlar o uso das portas seriais. Quando um processo Cliente deseja usar uma porta serial, ele envia uma mensagem “Aloca” para o Servidor. Existem N portas seriais, todas equivalentes, mas cada uma pode ser usada somente por um Cliente de cada vez. O Servidor informa ao Cliente a porta que ele vai usar através da mensagem “Porta p”. Ao concluir o uso, o Cliente envia para o Servidor a mensagem “Libera p”. Suponha que exista mais do que N processos Clientes. Mostre o algoritmo do Servidor, em português estruturado. Supor “receive” bloqueante.
11. Utilizando as primitivas de *semáforos*, explique como implementar um *monitor*.
12. Considere a seguinte modelagem, por monitor, para o problema do produtor-consumidor com  $n$  produtores e  $m$  consumidores.

```
monitor buffer {
  int itens=0; cond temItens, temEspacos;
  ...
  int pega() {
    while (1) {
      if (!itens) wait(temItens);
      pega item no buffer
      itens--;
      signal(temEspacos);
      return (item);
    }
  }

  void coloca() {
    while (1) {
      if (itens==MAX) wait(temEspacos);
      coloca item no buffer
      itens++;
      signal(temItens);
    }
  }
}
```

Suponha que esse monitor funciona com a disciplina “sinaliza e continua”, dada em sala (mas observe que ele admite filas separadas por variáveis de condição). Explique por que essa solução não funciona corretamente.