DEIN0114 - Sistemas Operacionais I

prof. Antonio de Abreu Batista Júnior

Prova II

São Luís, 8 de março de 2016

Aluno(a): Alexandre da Silva Soares 20130029S5

Considere a seguinte definição de semáforos para as questões 1 e 2:

```
typedef struct{
  int value;
  struct processo * list;

} semaphore;

semaphore s;


void wait(s){
    if (s->value > 0) {
        s->value --;
    }
    else {
        adicione esse processo na lista de espera do semaforo (s->list);
        block();
    }
}

void signal (s)
{
    if (existe pelo menos um processo bloqueado no semaphore s) {
        remova um processo P da s->list;
        adicione o processo P na lista de pronto;
    }
    else
        s->count++;
}
```

Semaphore s = 0
u p = 0

Questão 1.

Insira semáforos para satisfazer as propriedades:

- imprima A antes de imprimir F.
- imprima F antes de imprimir C.

A
sig - s

E
wait s
wait p
A
(signal p)
wait p

P_1	P_2
Imprima(A);	Imprima(E);
Imprima(B);	Imprima(F);
Imprima(C);	Imprima(G);

Imp A
Sig. s
Imp b
wait p)

Imp E
wait(s).
Imp. F
signal p)
Imp (G)

Questão 2

Suponha que uma quantidade finita de recursos do mesmo tipo tivesse de ser gerenciada. Os processos podem solicitar vários desses recursos e - quando terminam - os devolvem. Como exemplo, muitos pacotes de software comerciais fornecem uma determinada quantidade de licenças, indicando quantas aplicações podem ser executadas concorrentemente. Quando a aplicação é iniciada a contagem de licenças é decrementada. Quando a aplicação é encerrada, a contagem de licenças é incrementada. Se todas as licenças estiverem em uso, solicitações para iniciar a aplicação serão negadas. Essas solicitações só serão atendidas quando um detentor corrente de uma licença encerrar a aplicação e a licença for devolvida.

O segmento de programa a seguir é usado no gerenciamento de uma quantidade finita de intâncias de um recurso disponível. A quantidade máxima de recursos e a quantidade de recursos disponíveis foram declaradas como descrito abaixo:

```
#define MAX_RESOURCES 5
int available_resources =MAX_RESOURCES;
```

Quando um processo quer obter recursos, ele invoca a função *decrease_count*():

```
/*decrease available_resources by count resources*/
/*return 0 if sufficient resources available, */
/*otherwise return -1 */

int decrease_count(int count) {
  if (available_resources < count)
    return -1;
  else {
    available_resources = available_resources-count;
    return 0;
  }
}
```

Quando um processo quer devolver recursos, ele invoca a função *increase_count*():

```
/* increase available_resources by count */
int increase_count(int count) {
  available_resources = available_resources + count;
  return 0;
}
```

O segmento de programa anterior produz uma condição de corrida. Faça o seguinte:

1- Identifique os dados envolvidos na condição de corrida.

2- Identifique a localização (ou localizações) no código onde a condição de corrida ocorre.

3- Usando um semáforo, corrija a condição de corrida. Não há problema em modificar a função *decrease_count*() para que o processo que a invocar seja bloqueado até que recursos suficientes estejam disponíveis.

	Alocação				Max				Disponível			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	1	1	0	0	2	1	0	1	5	2	0
P_1	1	2	3	1	1	6	5	2				
P_2	1	3	6	5	2	3	6	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Questão 3.

Considere o seguinte estado do sistema:
Responda às perguntas a seguir usando o algorítmo do banqueiro:

A. Qual é o conteúdo da matriz *Necessidade*?

B. Demonstre que o sistema está em um estado seguro.

C. Se uma solicitação na forma (2,1,1,0) for feita pelo processo P_1, ela poderá ser atendida imediatamente?

Questão 4.

O deadlock pode ser definido formalmente como se segue: "Um conjunto de processos está em situação de deadlock, se cada processo do conjunto estiver esperando por um evento que somente outro processo pertencente ao conjunto poderá fazer acontecer".
O artigo de Coffman et al. (1971), citado por TANNENBAUM (1996), mostra que quatro condições devem ocorrer para que se configure uma situação de deadlock. Agora, considere o deadlock de tráfego mostrado na Figura 1.

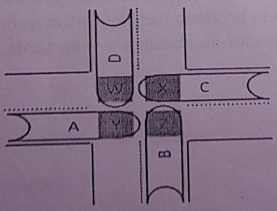

Figure 1: deadlock de tráfego.

A. Demonstre que as quatro condições necessárias para a ocorrência do deadlock estão presentes neste exemplo.

B. Defina uma regra simples para impedir deadlocks nesse sistema.

Exclusão mútua, posse espera, Fila circular, não-preempção

Questão 5.

A condição de inanição de um processo pode ser definida como uma condição em que o processo:

A. conclui antes do tempo necessário.

B. está em deadlock.

C. está travado pelo semáforo de outro processo.

D. não precisa de qualquer recurso do sistema para seguir executando.

E. nunca tem acesso ao recurso necessário.

Questão 6.

Um conjunto de estratégias foram propostas, por Havender, com o intuito de prevenir a ocorrência de deadlocks, analise cada uma delas e estabeleça a relação entre as estratégias propostas e as condições que elas asseguram que não vão acontecer se forem implementadas.

1 - Cada processo deve requisitar todos os recursos de que precisa de uma vez só e não pode continuar até que todos tenham sido concedidos.

( ) Exclusão mútua

2 - Se for negada mais uma requisição a um processo que retém certos recursos, ele deve liberar seus recursos originais e, se necessário, requisitá-los novamente, junto com os recursos adicionais.

2 (1) Posse e espera

1 (2) Não-preempção

3 - Deve ser imposta uma ordenação linear de recursos a todos os processos; se um processo recebeu certos recursos, ele somente poderá requisitá-los,novamente mais tarde, conforme a ordem.

(3) Espera circular