INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA CEARÁ

ENGENHARIA DE COMPUTAÇÃO
SISTEMAS OPERACIONAIS
SEMESTRE 2015.2 – Prova 1ª Etapa
PROF. FERNANDO PARENTE GARCIA

ALUNO: Ilsei Moreira de Albuquerque

1,8

1) (2,0 Pontos) Sobre conceitos básicos de sistemas operacionais, responda:
   a) Qual é a finalidade de uma chamada de sistema (*system call*) em um sistema operacional?
   b) Explique os seguintes conceitos: programa, processo e *thread*.

2,0

2) (2,0 Pontos) Sobre processos e threads, responda:
   a) Um algoritmo que execute diversos cálculos independentes concorrentemente (por exemplo, multiplicação de matrizes) seria mais eficiente se usasse *threads* ou se não os utilizasse?
   b) Que problema ocorre se for definido um *timeslice* muito pequeno para os processos? Justifique.

2,0

3) (2,0 Pontos) Suponha que os seguintes processos chegaram para execução nos tempos indicados. Cada processo rodará a quantidade de tempo listada na tabela.

Processo	Tempo de Chegada (ms)	Tempo de Execução (ms)	Prioridade
A	5	20	2
B	8	12	4
C	13	10	4
D	15	4	5
E	20	14	2

Qual o tempo médio de espera para estes processos quando são utilizados os algoritmos de escalonamento abaixo. Considere que o sistema operacional gasta 1 ms para realizar a troca de contexto.
   a) Escalonamento circular preemptivo com prioridade estática e timeslice de 4 ms.
   b) SJF.

4) (2,0 Pontos) Considere que em um determinado instante, a situação das estruturas de dados usadas pelo sistema apresenta os valores mostrados abaixo. Considere que o ID dos processos inicia em **P0** e dos recursos em **R0**. O estado atual é seguro ou inseguro? Justifique.

2,0

Recursos existentes:

6	7	12	12

0	0	1	2
2	0	0	0
0	0	3	4
2	3	5	4
0	3	3	2

Alocação Corrente

0	0	1	2
2	7	5	0
6	6	5	6
4	4	5	6
0	7	5	2

Requisição Máxima

2,0

5) (2,0 Pontos) *Problema da Montanha Russa*: Suponha que há **n** processos *passageiros* e um processo *vagão*. Os passageiros repetidamente esperam para andar no vagão, que comporta v passageiros, onde v<n. O vagão só pode percorrer a montanha russa quando está cheio. Os passageiros devem dormir durante a viagem do vagão. Verifique se os algoritmos propostos abaixo estão corretos, e caso contrário faça as devidas correções.

SEMAPHORE FULL = 0;
SEMAPHORE FIM_VIAGEM = ~~X~~ = 0
SEMAPHORE EMPTY = V;
INT PASSAGEIROS = 0;
SEMAPHORE MUTEX = 1;

PROCESSO PASSAGEIRO:

```
WHILE (TRUE) {
        DOWN(EMPTY);
        DOWN(MUTEX);
        ENTRA_VAGAO();
        PASSAGEIROS++;
        SENTAR_VAGAO();
        IF (PASSAGEIROS == V) UP(FULL);
        UP(MUTEX);
        DOWN(FIM_VIAGEM);
        SAIR_DO_VAGAO();
        DOWN(MUTEX);
        PASSAGEIROS--;
        IF (PASSAGEIROS == 0)
           FOR (I=0;I<V;I++) UP(EMPTY);
        UP(MUTEX);
    }
```

ROCESSO VAGÃO:

```
WHILE (TRUE) {
        DOWN(FULL);
        PERCORRER_MONTANHA();
        ~~UP(FIM_VIAGEM); 1 V~~
    }   for(i=0; i<V; i++)
            up(FIM_VIAGEM)
```

BOA PROVA!

Lucas Moura de Albuquerque

a) A system call é o modo que o sistema usa para receber solicitações do usuário para acessar recursos. Num sistema computacional em camadas o S.O. faz o intermédio entre o usuário e o hardware, sempre que o usuário deseja um recurso ele solicita ao S.O. através de system calls.

b) Programa é um algoritmo (sequência de passos) escrito em linguagem de alto nível que resolve um problema.

Um processo é um programa em execução. ~~Associado com~~ o modelo de processo também diz que um processo é um container que tem um contexto de software, contexto de hardware e espaço de endereçamento.

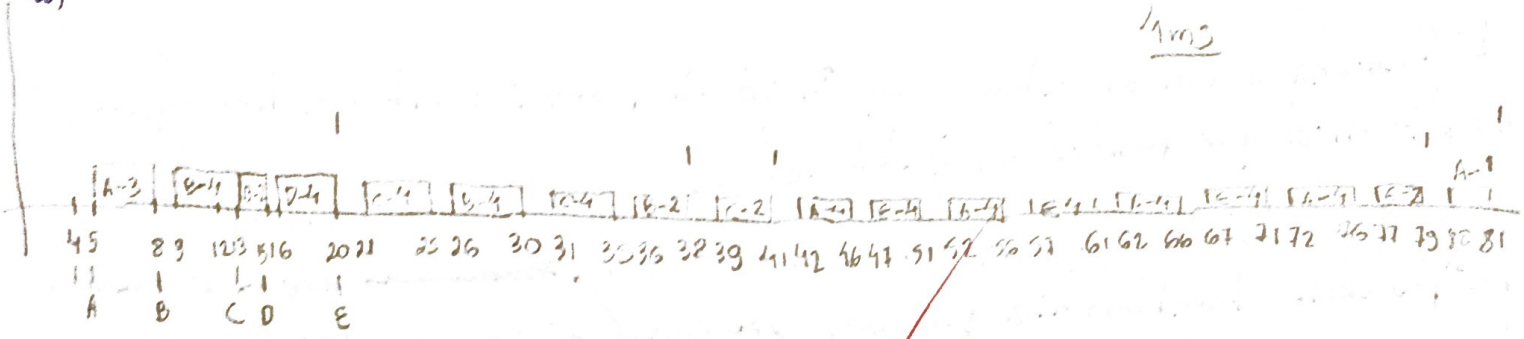
Uma thread é um fluxo de execução que pode está associado à um processo (threads de usuário) compartilhando o espaço de endereçamento com outras threads daquele processo, nesse caso cada thread tem seu próprio contexto de hardware. Threads também podem ester associadas ao S.O. (threads de núcleo).

2ª a) Esse algoritmo possuiria várias tarefas CPU bound. O uso de threads com tarefas CPU bound só é eficiente em CPUs multicore. Isso se dá pelo fato de a execução em CPU de 1 único core exigir a troca de contexto entre as tarefas, o que degradaria a performace em tarefas CPU bound. Então, caso houvesse mais cores, seria eficiente sim usar threads.

b) Um timeslice muito pequeno degradaria a performance pois a CPU gastaria muito tempo com trocas de contexto entre os processos. Assim a CPU estaria gastando mais tempo com algo que não "avança" a execução dos processos.

3)

a)

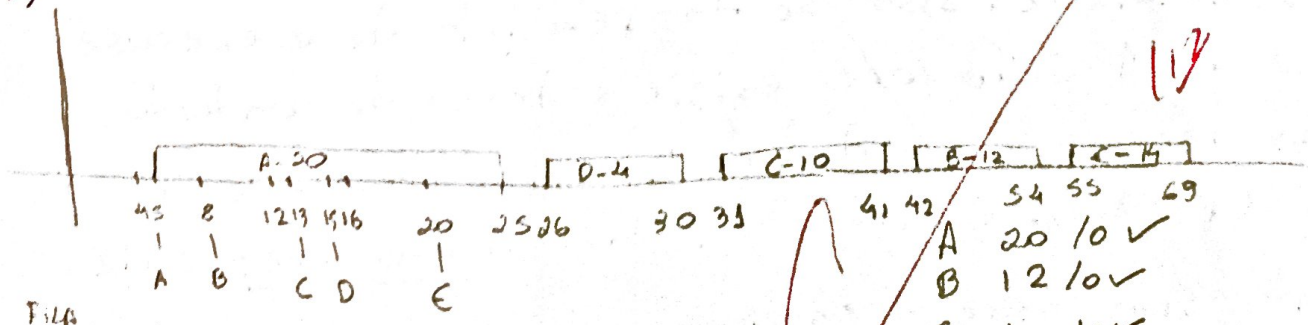


Tempos=
A = 76
B = 30
C = 28
D = 5
E = 59

$T_{médio} = \frac{76+30+28+5+59}{5} = \frac{198}{5} = 39,6$

A 20/17/13/9/5/1 0✓
B 12/8/6/2/0✓
C 10/6/2/0✓
D 4/0✓
E 14/10/6/2/0✓

b)



Tempos
A = 20
B = 46
C = 28
D = 15
E = 49

$T_{médio} = \frac{20+46+28+15+49}{5} = \frac{158}{5} = 31,6$

A 20/0✓
B 12/0✓
C 30/0✓
D 4/0✓
E 14/0✓

Montando as matrizes:

Levi Moreira

Matriz C (corrente)

	R0	R1	R2	R3
P0	0	0	1	2
P1	2	0	0	0
P2	0	0	3	4
P3	2	3	5	4
P4	0	3	3	2

Matriz R (Requisições) = Matriz Máx. − Matriz C

	R0	R1	R2	R3
P0	0	0	0	0 ✓
P1	0	7	5	0
P2	6	6	2	2
P3	2	1	0	2 ✓
P4	0	4	2	0 ✓

C 2,0

Vetor E (Existentes)

R0	R1	R2	R3
6	7	12	12

Vetor A (Disponíveis)

R0	R1	R2	R3
2	1	0	0 ✓

Analisando o Vetor A e a Matriz R, vemos que P0 roda, ao finalizar ele devolve os recursos para A, que se torna A = {2, 1, 1, 2}, com esses valores P3 roda, simulando a entrega, ele finaliza. A vai se tornar A = {4, 4, 6, 6}. Com esses valores P4 roda, ao finalizar A = {4, 7, 9, 8}. Com esses valores P1 roda e ao finalizar P2 poderá rodar, logo é uma situação segura.

5ª

ERRO 1: SEMAPHORE FIM_VIAGEM = 1;

Deveria ser igual a 0, caso contrário o primeiro passageiro que entrar nesse semáforo, dará o down e imediatamente sairá do vagão sem esperar que este faça a viagem.

2,0

ERRO 2: UP(FIM_VIAGEM);

Deveria ser for(i=0; i<v; i++) UP(FIM_VIAGEM), caso contrário só o 1º passageiro será acordado depois que o passeio for concluído.

SISTEMAS OPERACIONAIS – 2015.2

PROF. FERNANDO PARENTE GARCIA

EXERCÍCIO DE SEMÁFOROS

ALUNO: Levi Moreira de Albuquerque

Problema do precipício: Suponha que há um grupo de pessoas (aventureiros) que utiliza uma corda para atravessar um grande precipício. Até três pessoas podem atravessar o precipício simultaneamente, desde que elas estejam cruzando na mesma direção. Se uma pessoa cruzando para o leste encontra uma pessoa cruzando para oeste no meio da corda, elas podem se atrapalhar e cair. Além disso, a corda só é forte o suficiente para suportar até três pessoas. Usando semáforos, desenvolva o código para as ações das pessoas de modo que elas consigam atravessar o precipício e não morram.

```
Semáforo MAXL=3 */ controla o max de pessoas que vão
                   de leste – Oeste /*
Semáforo MAXO=3 */ controla o max de pessoas que vão
                   de Oeste – Leste /*
Semáforo corda=1 */ controla o fluxo da corda (ponte)/*
Semáforo mutex1=1 */ Protege PL de múltiplos acessos
                     simultaneos /*
Semáforo mutex2=1 */ Protege PO de múltiplos acessos
                     simultâneos /*
Int PL=0, PO=0 */ controla a quantidade de pessoas /*
```

THREAD PLESTE

```
while(true){
    down(MAXL)
    down(mutex1)
    PL++
    Se PL=1 down(corda)
    up(mutex1)
    CruzaPonte()
    down(mutex1)
    PL--
    Se PL=0 up(corda)
    up(mutex1)
    up(MAXL)
}
```

THREAD POESTE

```
while(true){
    down(MAXO)
    down(mutex2)
    PO++
    Se PO=1 down(corda)
    up(mutex2)
    CruzaPonte()
    down(mutex2)
    PO--
    Se PO=0 up(corda)
    up(mutex2)
    up(MAXO)
}
```