GABARITO

P2 - Sistemas Operacionais I
Professor: Leandro Marzulo
2013-2

Nome:

Instruções: Esta prova é composta de 3 questões totalizando 12 (doze) pontos, sendo a nota máxima 10 (dez). Responda as questões de forma sucinta e clara. O uso de lápis é permitido, no entanto, pedidos de revisão serão considerados apenas para questões respondidas a caneta. BOA PROVA!

1) (6,0) Considere três processos com as características descritas na tabela a seguir:

Processo	Tempo da Rajada de CPU 1	Tempo da Rajada de I/O	Tempo da Rajada de CPU 2	Prioridade	Instante de criação
P1	5	10	3	2	2
P2	7	5	5	1	4
P3	4	9	8	3	1

Considere que cada processo faz I/O em um dispositivo independente (todos os I/Os são paralelos) e que o tempo de troca de contexto é insignificante. Saiba que, para cada processo:

Tempo de Turnarround = Tempo de Execução no processador + Tempo de I/O + Tempo de Espera

Repare que o Tempo de Execução no Processador e o Tempo de I/O de cada processo é independente do mecanismo de escalonamento. Além disso, o tempo de turnarround de cada processo pode também ser definido como o tempo decorrido entre o instante de criação do processo e o instante do seu término.

Para cada um dos mecanismos de escalonamento a seguir, desenhe o diagrama de Gantt ilustrando o escalonamento dos processos, além de calcular seus respectivos tempos de turnaround e tempo médio de espera, segundo as políticas especificadas a seguir.

Vamos calcular, para cada processo, o "Tempo de Execução no processado" e o "Tempo de I/O", pois estes são independentes do mecanismo de escalonamento. O Tempo de Execução no processado" de um processo é a soma dos tempos de rajada de CPU do mesmo. Já o "Tempo de I/O" é soma dos tempos de raja de I/O do mesmo. Temos, portanto:

Processo	Tempo de Execução no processador	Tempo de I/O
P1	5+3 = 8	10
P2	7+5 = 12	5
P3	4+8=12	9

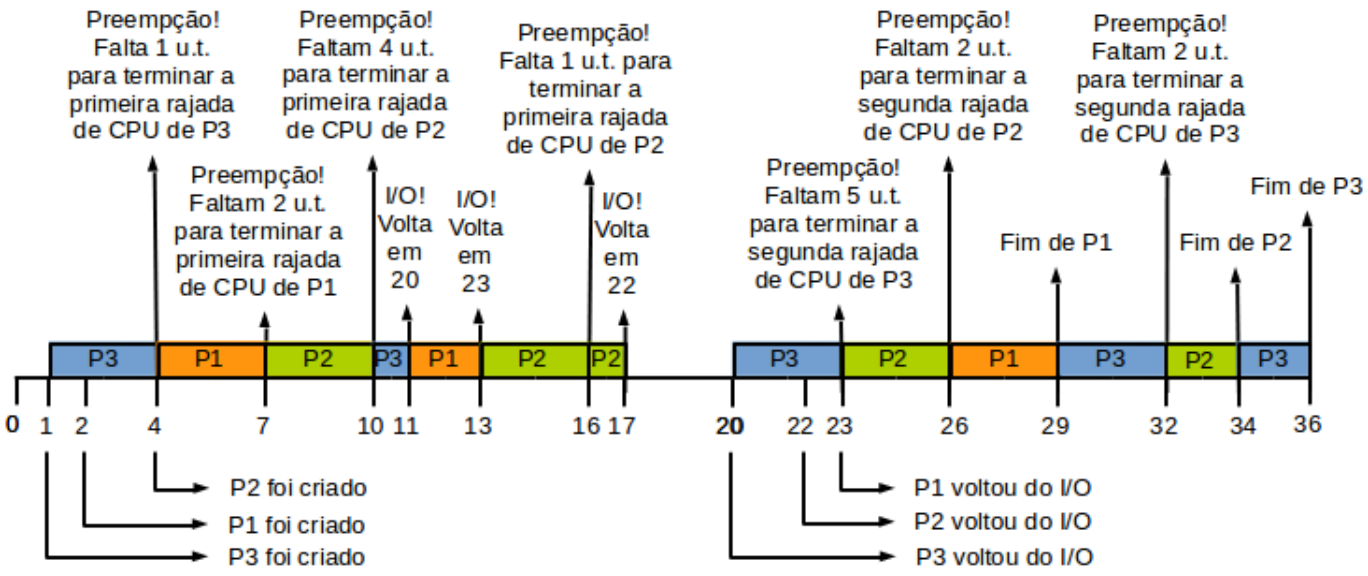
Além disso, sabemos que:

Tempo de turnaround = Instante do término - Instante de criação

e que, portanto:

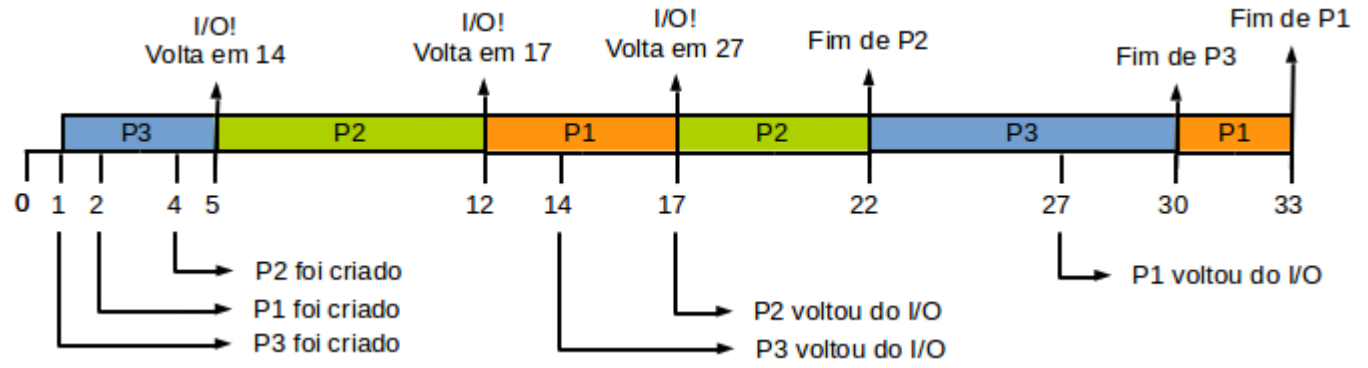
Tempo de Espera = Tempo de turnaround - Tempo de Execução no processador - Tempo de I/O

a) (2,0) Round Robin com quantum=3u.t.



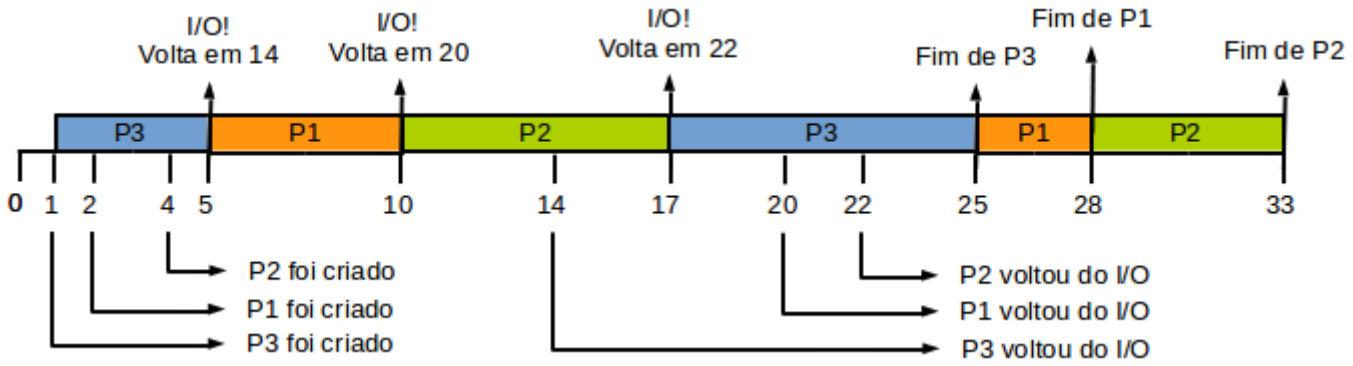
Processo	Tempo de Turnaround	Tempo de Espera
P1	29-2 = 27	27-8-10 = 9
P2	34-4 = 30	30-12-5 = 13
P3	36-1 = 35	35-12-9 = 14
Média	-	(9+13+14)/3 = 36/3 = 12

b) (2,0) Prioridade sem preempção (número menor implica prioridade maior)



Processo	Tempo de Turnaround	Tempo de Espera
P1	33-2 = 31	31-8-10 = 13
P2	22-4 = 18	18-12-5 = 1
P3	30-1 = 29	29-12-9 =8
Média	-	(13+1+8)/3 = 22/3 = 7,34

c) (2,0) Trabalho mais curto primeiro



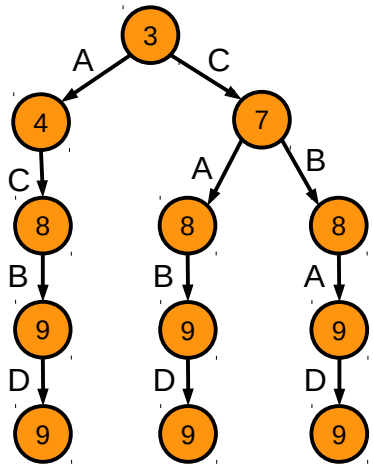
Processo	Tempo de Turnaround	Tempo de Espera
P1	28-2 = 26	26-8-10 = 8
P2	33-4 = 29	29-12-5 = 12
P3	25-1 = 24	24-12-9 = 3
Média	-	(8+12+3)/3 = 23/3 = 7,67

2) (3,0) Considere um sistema com um total 9 instâncias de um determinado recurso. Temos 4 processos A, B, C e D que executam concorrentemente. A já alocou 1 instâncias e precisa de mais 3 para terminar a sua execução; B alocou 1 e precisa de mais 6; C alocou 4 e precisa de mais 2; D não alocou nenhuma e precisa de 9. Considerando o conceito de sequência de segurança e estado de segurança, responda:

a) (1,0) Mostre todas as sequências de segurança válidas (se houver alguma) e explique o que torna as sequência válidas.

Para resolver este problema, vamos construir uma árvore onde o nó raiz contém o número de recursos disponíveis inicialmente. Para cada nó da árvore, verificamos quais processos podem ser atendidos com o número de recursos indicado no nó e criamos filhos com a nova quantidade de recursos depois de atentender um determinado processo. O processo atendido será indicado como rótulo da aresta ligando os dois nós. Se houver um caminho na árvore (partindo da raíz em direção aos filhos) onde apareçam todos os processos, este caminho representa uma sequência de segurança.

Temos 3 recursos disponíveis (9 – 1 – 1 – 4 – 0 = 3).

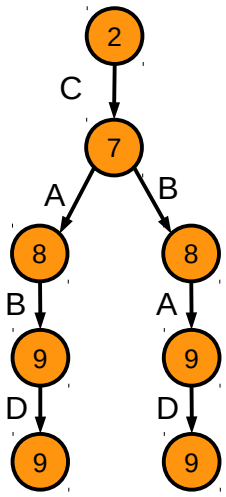


Observando a árvore, podemos concluir que existem as seguintes sequências de segurança válidas:

ACBD
CABD
CBAD

b) (1,0) Se C fizesse a requisição de um recurso e fosse atendido, ainda teríamos alguma sequência de segurança válida? Em caso afirmativo, quais? Em caso negativo, o que aconteceria no sistema?

Neste caso, temos 2 recursos disponíveis inicialmente, já que C alocou 5 recursos e precisa de mais 1.

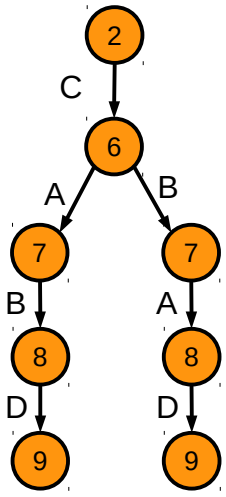


Observando a árvore, podemos concluir que existem as seguintes sequências de segurança válidas:

CABD
CBAD

c) (1,0) Considerando ainda o cenário original do enunciado, se D fizesse a requisição um recurso e fosse atendido, ainda teríamos alguma sequência de segurança válida? Em caso afirmativo, quais? Em caso negativo, o que aconteceria no sistema?

Neste caso, temos 2 recursos disponíveis inicialmente, já que D alocou 1 recurso e precisa de mais 8.



Observando a árvore, podemos concluir que existem as seguintes sequências de segurança válidas:

CABD
CBAD

3) (3,0) Um problema clássico da comunicão de processos é o problema dos filósofos. Neste problema, cada filósofo passa por períodos alternados de comer e de pensar, estando cinco filósofos dispostos em torno de uma mesa circular. Nessa mesa existe um prato de comida frente de cada filósofo e um hashi entre dois pratos consecutivos. Cada filósofo e cada hashi é númerado (de 0 a 4). Quando um filósofo está com fome, ele tenta pegar o hashi à sua esquerda e o da sua direita. Só depois de pegar os dois hashis ele poderá comer e depois recolocará os hashis na mesa, voltando a pensar até ficar novamente com fome. A seguir é dada uma solução para o problema dos filósofos baseada em 5 semáforos binários, sendo que o semáforo hashi[i] está associado ao hashi i ($0 < i < 5$). Os semáforos são todos inicializados com o valor 1. Esta solução funcionará se os processos que implementam os filósofos executarem a função philosopher dada a seguir, sendo que cada um deles está associado a um identificador i, ($0 < i < 5$)? Caso a solução dada não funcione, dê uma solução correta para o problema proposto no enunciado.

```
void philosopher (int i)
{
        while (1)
        {
                pensar ();
                sem_wait(&hashi[i]);
                sem_wait(&hashi[(i+1)%5]);
                comer ();
                sem_post(&hashi[(i+1)%5]);
                sem_post(&hashi[i]);
        }
}
```

A solução é fazer o acesso aos semáforos em ordem crescente, ou seja, para que não ocorra espera circular, um dos filósofos deve inverter a ordem de requisição dos semáforos.

```
void philosopher (int i)
{
        while (1)
        {
                pensar ();
                if (i < ((i+1)%5))
                {
                        sem_wait(&hashi[i]);
                        sem_wait(&hashi[(i+1)%5]);
                }
                else
                {
                        sem_wait(&hashi[(i+1)%5]);
                        sem_wait(&hashi[i]);
                }
                comer ();
                sem_post(&hashi[(i+1)%5]);
                sem_post(&hashi[i]);
        }
}
```