GABARITOx
PR - Sistemas Operacionais I
Professor: Leandro Marzulo
2012-2

Nome:

Instruções: Esta prova é composta de cinco 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) (4,0) Considere que cinco 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	7	12	3	5	3
P2	6	5	5	1	4
P3	10	-	-	2	0
P4	7	100	9	4	20
P5	8	-	-	3	80

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	7+10 = 10	12
P2	6+5 = 11	5
P3	10	0
P4	7+9 = 16	100
P5	8	0

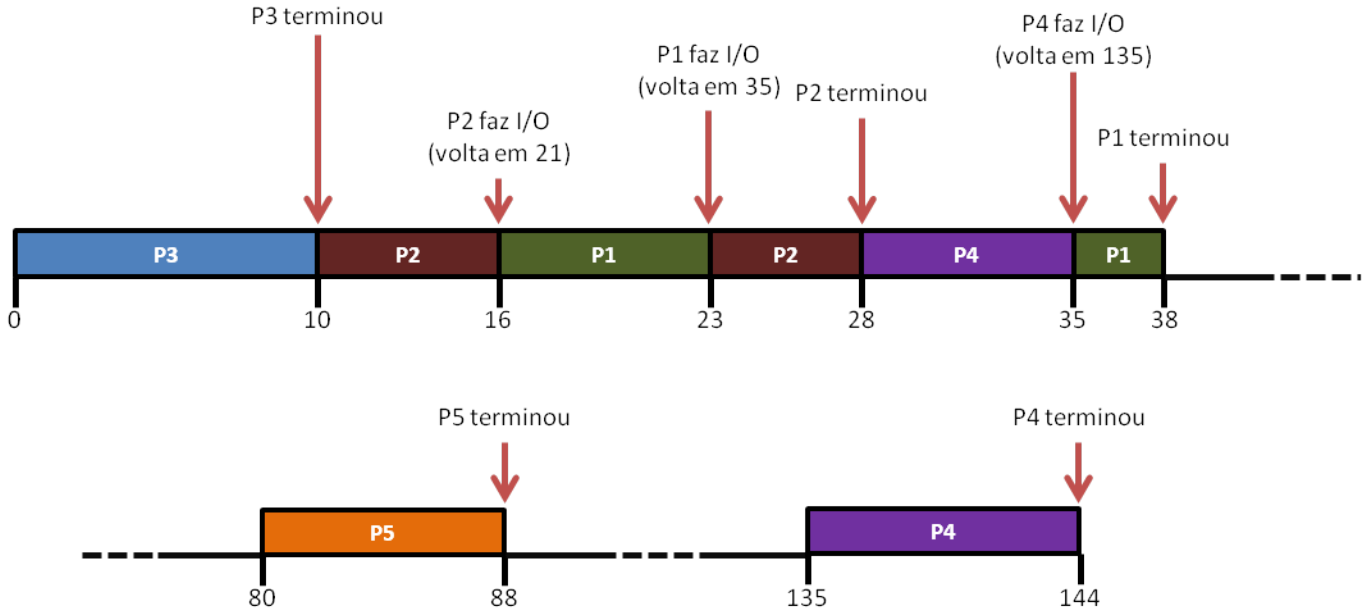
Além disso, sabemos que:

Tempo de turnaround = Instante do término - Instante de criação

e que, portanto:

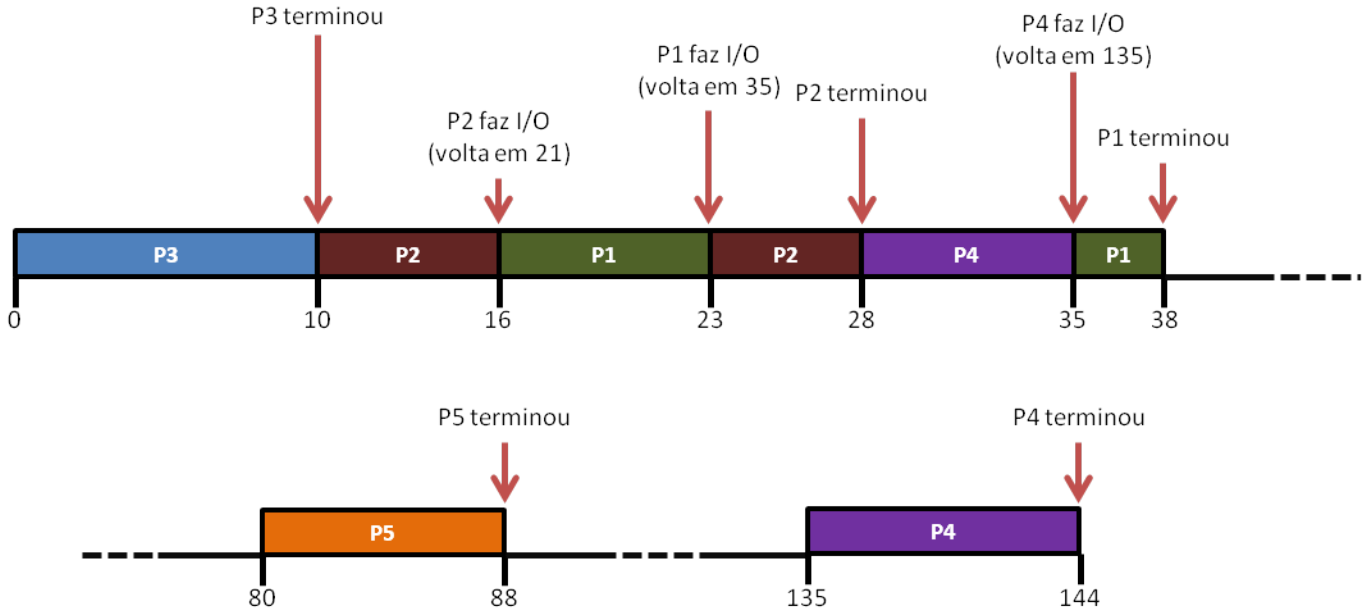
Tempo de Espera = Tempo de turnaround - Tempo de Execução no processador - Tempo de I/O

a) (2,0) Trabalho mais curto primeiro



Processo	Tempo de Turnaround	Tempo de Espera
P1	38-3 = 35	35-12-10 = 13
P2	28-4 = 24	24-5-11 = 8
P3	10-0 = 10	10-0-10 = 0
P4	144-20 = 124	124-100-16 = 8
P5	88-80 = 8	8-0-8 = 0
Média	-	(13+8+0+8+0)/5 = 29/5 = 5,8

b) (2,0) Trabalho restante mais curto primeiro



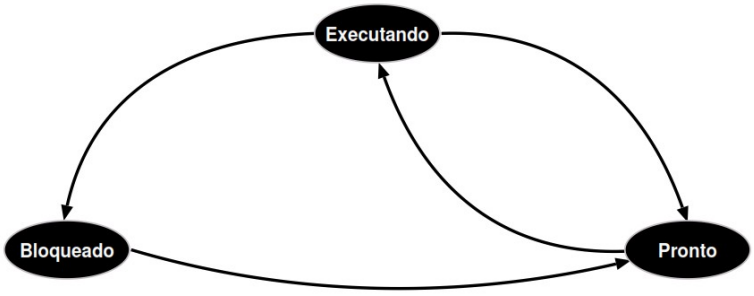
Processo	Tempo de Turnaround	Tempo de Espera
P1	38-3 = 35	35-12-10 = 13
P2	28-4 = 24	24-5-11 = 8
P3	10-0 = 10	10-0-10 = 0
P4	144-20 = 124	124-100-16 = 8
P5	88-80 = 8	8-0-8 = 0
Média	-	(13+8+0+8+0)/5 = 29/5 = 5,8

Repare que o resultado foi o mesmo, pois não tivemos preempção.

2) (2,0) O que é multiprogramação e por que a ela é importante para as últimas gerações de computadores?

Com o passar das gerações, a velocidade de processamento dos computadores se tornou cada vez maior. A velocidade dos dispositivos físicos também aumentou, mas muito mais lentamente do que a velocidade de processamento. Com isso, o tempo de ociosidade do processador quando o processo em execução fazia operações de E/S ficou, com o passar das gerações, cada vez maior, pois cada processo era executado até terminar, sem interrupções. Além disso, existia uma grande demora para se obter os resultados dos processos. Para evitar esses problemas, o conceito de multiprogramação, que permite que mais de um processo esteja em execução no sistema através da divisão do tempo de processamento entre os processos, foi definido a partir da terceira geração.

3) (2.0) Na figura a seguir damos os estados de um processo e as possíveis transições entre estes estados. Pela figura, vemos que um processo não pode passar do estado pronto para o bloqueado e nem passar do estado bloqueado para o executando. Por que estas transições não existem na figura?



Quando um processo está no estado pronto, ele está esperando a sua vez de executar em um dos processadores da máquina. Logo, a transição do estado pronto para o bloqueado não tem sentido, pois um processo passa ao estado bloqueado quando está em execução e não pode continuar a executar, pois passou a esperar pela ocorrência de um evento externo. Já a transição do estado bloqueado para o executando não existe porque o processo que foi desbloqueado, apesar de agora está pronto para a execução, deverá esperar pela próxima vez que o escalonador for chamado pelo sistema operacional. Note que a transição não deve existir mesmo se o processo for mais prioritário do que os outros não bloqueados, pois neste caso o escalonador pode ser imediatamente chamado após o processo ser desbloqueado.

4) (2.0) Um processo que realize uma computação estritamente sequencial pode ser visto como um único fluxo de controle. No entanto, no caso mais geral, é possível que um processo possa ter mais de um fluxo de controle, correspondendo a ações executadas concorrentemente. Cada um desses fluxos de controle é chamado de thread (fio). Assim como os processos, as threads são escalonadas para execução pelo sistema operacional. Porém, o escalonamento das threads é mais leve que o dos processos, já que seu contexto é bem menor. A pilha deve ou não fazer parte desse contexto a ser salvo para cada uma das threads? Justifique a sua resposta.

O objetivo da pilha é o de armazenar várias informações referentes ao fluxo de execução das instruções no processador. Dentre estas informações, temos as variáveis locais do procedimento atualmente em execução no processador, e os endereços de retorno das chamadas que foram executadas e que ainda não foram terminadas. Como um processo com múltiplas threads possui múltiplos fluxos de controle, e como cada um deles pode armazenar variáveis locais na pilha e chamar procedimentos, deveremos ter uma pilha para cada thread do processo, que deverá ser salva no contexto desta thread. Note que enquanto o escalonador estiver escolhendo threads do mesmo processo, somente o contexto da thread deve ser salvo, pois todas as threads de um processo compartilham o contexto deste processo. O contexto do processo somente deverá ser salvo quando o escalonador escolher uma thread de um outro processo, pois esta thread não compartilhará o contexto do processo da thread que estava em execução. Logo, o escalonamento de threads do mesmo processo é mais leve do que o de threads pertencentes a processos diferentes, pois o contexto a ser salvo é menor. Note também que se a pilha não fosse salva no contexto, deveríamos executar as threads do processo sequencialmente, o que não seria lógico, pois o objetivo de termos múltiplas threads é exatamente o de permitir que elas executem concorrentemente. Perceba que no caso de termos mais de um processador ou de o único processador possuir múltiplos contadores de programa, as threads poderão de fato executar paralelamente.

5 (2.0) Considere um sistema com um total 10 instâncias de um determinado recurso. Temos 3 processos A, B e C que executam concorrentemente. A já alocou 3 instâncias e precisa de mais 6 para terminar a sua execução; B alocou 1 e precisa de mais 8; C alocou 4 e precisa de mais 1. Considerando o conceito de sequência de segurança e estado de segurança, responda:

a) (1,0) Mostre pelo menos uma sequência de segurança válida e explique o que a torna válida.

Queremos manter o estado de segurança, ou seja, o sistema pode alocar recursos a cada processo (até o máximo que este necessita) em alguma ordem e continuar evitando deadlocks. Deve existir, portanto, uma sequência de segurança <P1, P2, ..., Pn> onde, para cada Pi, as solicitações de recursos restantes para Pi podem ser satisfeitas pelos recursos disponíveis e pelos recursos detidos por outros processos Pj (sendo j<i). Estado inseguro significa que há possibilidade de deadlock. Uma sequência de segurança válida para este cenário é C-A-B. Neste cenário, temos um total de 8 recursos alocados e 2 disponíveis. se C pegar todos os recursos de que necessita (apenas 1 recurso), vai poder executar e liberar 5 recursos. Passamos a ter, portanto, 6 recursos disponíveis no sistema. Isto nos permite atender as solicitações de A, que executa e

libera 9 recursos. O sistema fica com 9 recursos disponíveis. Podemos então atender B (que precisa de mais 8). Com essa sequência, atendemos todos os processos.

b) (0,5) Se A fizesse a requisição de um recurso e fosse atendido, ainda teríamos um estado de segurança válido? Em caso afirmativo, qual? Em caso negativo, o que aconteceria no sistema?

Se alocamos um recurso para A, este ainda precisará de 5 recursos. Depois desta alocação, ficamos com 1 recurso disponível no sistema, o que nos permite atender a necessidade de C. Depois da execução de C, teremos 5 recursos disponíveis, o que nos permite atender a necessidade de A. Depois da execução de A, teremos 9 recursos disponíveis no sistema. Isto nos permite atender a necessidade de C. Sendo assim, a sequência de segurança para este caso permanece C-A-B.

c) (0,5) Considerando ainda o cenário original do enunciado, se B fizesse a requisição de um recurso e fosse atendido, ainda teríamos um estado de segurança válido? Em caso afirmativo, qual? Em caso negativo, o que aconteceria no sistema?

Se alocamos um recurso para B, este ainda precisará de 7 recursos. Depois desta alocação, ficamos com 1 recurso disponível no sistema, o que nos permite atender a necessidade de C. Depois da execução de C, teremos 5 recursos disponíveis, o que é insuficiente para atender a demanda de A ou B. Sendo assim, não temos uma sequência de segurança válida. A não ser que houvesse preempção de recursos ou algum processo fosse terminado, teríamos um deadlock.