GABARITO
P1 - Sistemas Operacionais I
Professor: Leandro Marzulo
2014-1

Nome:

Instruções: Esta prova é composta de quatro 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, entretanto, pedidos de revisão serão considerados apenas para questões respondidas a caneta. BOA PROVA!

1) (4,0) Considere um computador com um sistema operacional superior ao da terceira geração e um processador que leve X u.t. para copiar 1 byte entre a memória principal e a controladora de disco. Esse computador também conta com uma unidade de DMA que leva as mesmas X u.t. para fazer a cópia de dados. Entretanto, para usar o DMA, há um custo de configuração de Y u.t. a cada cópia de até 1000 bytes. O tempo de troca de contexto é irrelevante. Este computador executa um processo A que realiza uma operação de E/S no disco de B bytes. No momento da cópia dos dados provenientes da E/S de A o processo B entra na fila de prontos, que estava vazia. O processo C precisa executar por apenas uma rajada de CPU de Z u.t., terminando logo em seguida. Durante a execução de C, W% do tempo são gastos com acesso à memória de dados (operações de load e store). Nenhum outro processo (além de C) é criado ou desbloqueado até o término completo de A. Quais são as condições para que o uso de DMA seja vantajoso nesse cenário? Qual o tempo de ociosidade do processador durante o tempo da cópia de dados com uso de DMA nesse cenário? Considere que B é múltiplo de 1000 para facilitar as contas

Resp.:

Com o uso de DMA é necessário configurar o controlador para fazer a cópia dos dados.

Só faz sentido usar DMA se:
custo de configuração < custo da cópia de dados.

Custo de configuração = (custo de 1 configuração) * (número de configurações)
Custo de configuração = Y * (número de configurações)

Temos 1 configuração a cada 1000 bytes. Portanto:

Número de configurações = B / 1000

Custo de configuração = Y * B / 1000

Custo da cópia de dados = X * B

Logo, só faz sentido usar DMA se:
Y * B / 1000 < X * B
Y < 1000 * X

Considerando que faz sentido usar o DMA, a ociosidade a parte do tempo da cópia em que não foi possível ocupar o processador. Note que já existe um custo inicial de Y * B / 1000 para a configuração do DMA. Podemos considerar esse tempo como parte do desperdício. Durante o tempo da cópia (X * B), podemos colocar o processo C para executar. Entretanto, como estamos usando DMA, quando a controladora de DMA estiver realizado a cópia de dados, não será possível fazer acesso à memória. Temos portanto, duas possibilidades:

* O tempo de execução de C é inferior ou igual ao tempo de cópia (Z < X * B) :

Nessa situação certamente teremos ociosidade. Qualquer operação de memória de C teria que aguardar o tempo de uma etapa de cópia (de 1000 bytes). Entre uma configuração e outra, C poderia fazer o acesso, dependendo do escalonamento. Entretanto, no pior caso, a ociosidade pode ser total (X * B).

Um cenário de melhor caso, seria uma situação onde todas as operações de memória fosse feitas no final de C. Nessa situação, durante o tempo de cópia, poderíamos executar toda a porção de C que não usa a memória.

O tempo que C gasta com acesso a memória é Z * W /100. Logo, o tempo restante é (1 – W/100) * Z

Logo, no melhor caso, a ociosidade é dada por:

(X * B) – (1 – W/100) * Z

* O tempo de execução de C é superior ao tempo de cópia (Z > X *B)

Nessa situação, no pior caso ainda podemos ter uma operação de memória no início de C, causando ociosidade total (X *B).

Entretanto, no melhor caso (operações de memória no final de C), parte dessas operações (ou até mesmo todas, dependendo do tempo) poderão ser feitas depois da cópia

- Se o tempo gasto com operações normais for superior ao tempo de cópia, eliminaremos a ociosidade. Ou seja:

Se (1 – W/100) * Z >= X*B eliminaremos a ociosidade

- Caso contrário a ociosidade será dada por:

(X * B) – (1 – W/100) * Z

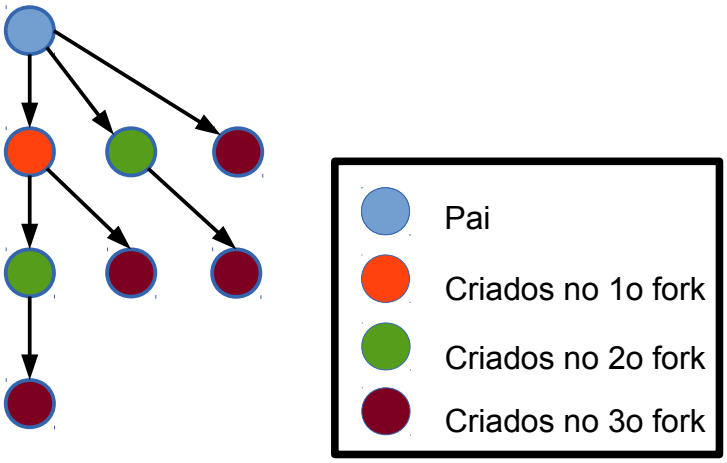
2) (3,0) Qual a importância das interrupções para a multiprogramação?

Resp.: Com a multiprogramação, quando um processo solicita uma operação de E/S, podemos delegar essa tarefa ao controlador do dispositivo e usar o processador para executar outro processo. Quando a E/S termina é necessário informar ao processador para que o SO seja chamado para fazer o tratamento desse evento e colocar o processo solicitante da E/S na fila de prontos. Esse mecanismo é chamado de interrupção. Portanto, sem interrupções não poderíamos ter multiprogramação.

3) (3,0) Considerando o trecho de código abaixo, responda:

```
for (i=0; i<N; i++)
        fork();
```

a) (1,5) Mostre, para N=3, a árvore de processos do processo inicial até que todos os filhos sejam criados (considerando que nenhum deles terminou e assumindo que não houve falha na execução de nenhuma chamada fork())?



b) (1,5) Ao executar este trecho, para um valor qualquer de N, quantos processos são criados, incluindo o processo inicial, assumindo que não houve falha na execução de nenhuma chamada fork()?

Resp.: 2 ^ N, pois a cada chamada fork duplicamos o número de processos.

4) (2,0) Em um computador com apenas um núcleo de processamento e com um sistema operacional com multiprogramação, existiria alguma possibilidade de ganho de desempenho com a criação de múltiplas *threads* de

kernel para executar partes de uma tarefa caso a aplicação fosse totalmente CPU *bound* (não executasse operações de E/S)? E se a aplicação fosse I/O *bound*?

Resp.: Em um sistema com apenas um núcleo de processamento não há execução paralela. Entretanto, pode haver concorrência (pelo uso da multiprogramação). Nesse caso, o ganho se dá pela superposição de E/S com computação. Sendo assim, para que tenhamos ganhos de desempenho com multithread nesse tipo de sistema, é necessário que a thread faça E/S. Portanto, não há ganho algum para uma aplicação totalmente CPU bound. Já para aplicações I/O bound, o ganho pode ser considerável.