UNIVERSIDADE FEDERAL DE SERGIPE
CAMPUS ITABAIANA
DEPARTAMENTO DE SISTEMAS DE INFORMAÇÃO
PROF: ANDRÉ LUIS MENESES SILVA

1ª Lista de Sistemas Operacionais - Escalonamento

1. Um algoritmo de escalonamento de CPU não preemptivo determina uma ordem para a execução dos processos incluídos na fila de prontos. Dados nos processos a serem designados para execução em um processador, quantos escalonamentos diferentes são possíveis? Forneça uma fórmula em função de n.

R- Podemos usar a fórmula:
n!, onde “n” representa o número de processos na fila. A notação “n!” indica o produto de todos os inteiros positivos de 1 a n. Logo, a fórmula em função de “n” para o calcular o número de escalonamentos diferentes possíveis em um algoritmo de escalonamento de CPU não preemptivo é apenas “n!”.

2. Explique a diferença entre scheduling com preempção e sem preempção.

R- A principal diferença entre escalonamento com preempção e sem preempção é que, no escalonamento com preempção, um processo pode ser interrompido antes de terminar a sua execução, enquanto no escalonamento sem preempção, um processo só pode ser interrompido quando ele termina a sua execução ou quando ele é bloqueado.

No escalonamento com preempção, o escalonador pode interromper um processo para dar a CPU a outro processo. Isso pode ser feito por uma variedade de motivos, como:

Prioridade: o processo interrompido pode ter uma prioridade menor do que o processo que está sendo escalonado.
Tempo de execução: o processo interrompido pode levar muito tempo para ser executado.
Estado de bloqueio: o processo interrompido pode ter sido bloqueado por um dispositivo de E/S.
O escalonamento com preempção pode melhorar o desempenho do sistema, pois pode garantir que os processos de alta prioridade sejam executados o mais rápido possível. No entanto, também pode aumentar o overhead do sistema, pois o escalonador precisa trocar de contexto entre os processos com mais frequência.

No escalonamento sem preempção, o processo que está executando na CPU só pode ser interrompido quando ele termina a sua execução ou quando ele é bloqueado. Isso significa que o processo pode ser executado até o fim, mesmo que ele não seja o processo mais importante no momento.

O escalonamento sem preempção é mais simples de implementar do que o escalonamento com preempção, pois o escalonador não precisa trocar de contexto com tanta frequência.

No entanto, também pode ser menos eficiente, pois pode permitir que processos de baixa prioridade monopolizem a CPU.

A escolha entre escalonamento com preempção e sem preempção depende de uma variedade de fatores, incluindo os requisitos de desempenho e eficiência do sistema.

3. Suponha que um algoritmo de escalonamento favoreça os processos que usaram o menor tempo do processador no passado recente. Por que esse algoritmo favorece programas limitados por I/O e, ao mesmo tempo, impede que programas limitados por CPU fiquem em estado de permanente starvation?

R- Um algoritmo de escalonamento que favorece os processos que usaram o menor tempo do processador no passado recente é conhecido como algoritmo de escalonamento de tempo de espera. Esse algoritmo favorece programas limitados por I/O porque esses processos gastam a maior parte do seu tempo esperando por recursos de I/O. Como resultado, eles terão um tempo de espera relativamente baixo, o que os tornará mais propensos a serem escolhidos pelo escalonador.

Esse algoritmo também impede que programas limitados por CPU fiquem em estado de permanente starvation porque, eventualmente, todos os processos terão um tempo de espera baixo. Isso ocorre porque os processos limitados por CPU eventualmente terminarão de usar a CPU e passarão a usar recursos de I/O. Quando isso acontecer, seu tempo de espera diminuirá e eles se tornarão mais propensos a serem escolhidos pelo escalonador.

Explicação mais detalhada de como esse algoritmo funciona:

* Processos limitados por I/O: Esses processos gastam a maior parte do seu tempo esperando por recursos de I/O. Como resultado, eles terão um tempo de espera relativamente baixo. Isso ocorre porque o tempo de execução de um processo limitado por I/O é determinado pelo tempo que ele leva para receber os recursos de I/O que precisa. Como esse tempo é geralmente imprevisível, o tempo de execução do processo também é imprevisível. No entanto, o tempo de espera do processo é determinado pelo tempo que ele passa esperando por esses recursos, que geralmente é relativamente curto.
* Processos limitados por CPU: Esses processos gastam a maior parte do seu tempo executando na CPU. Como resultado, eles terão um tempo de execução relativamente alto. Isso ocorre porque o tempo de execução de um processo limitado por CPU é determinado pela quantidade de trabalho que ele precisa fazer. Como esse trabalho é geralmente conhecido, o tempo de execução do processo também é geralmente conhecido. No entanto, o tempo de espera do processo é determinado pelo tempo que ele passa esperando por acesso à CPU, que pode ser longo.

No algoritmo de escalonamento de tempo de espera, o escalonador escolhe o processo com o menor tempo de espera na fila de prontos. Como os processos limitados por I/O têm um tempo de espera relativamente baixo, eles são mais propensos a serem escolhidos pelo escalonador. Isso significa que esses processos terão a oportunidade de executar e, eventualmente, concluir suas tarefas.

No entanto, os processos limitados por CPU também receberão sua oportunidade de execução. Isso ocorre porque, eventualmente, todos os processos terão um tempo de espera baixo. Quando isso acontecer, eles serão mais propensos a serem escolhidos pelo escalonador. Isso significa que os processos limitados por CPU não ficarão em estado de permanente starvation.

4. Por que é importante que o escalonador diferencie programas limitados por I/O de programas limitados por CPU?

R- É importante que o escalonador diferencie programas limitados por I/O de programas limitados por CPU porque esses dois tipos de programas têm necessidades diferentes em termos de acesso à CPU.

* Processos limitados por I/O: Esses processos gastam a maior parte do seu tempo esperando por recursos de I/O. Como resultado, eles não precisam de acesso constante à CPU. Na verdade, dar a esses processos acesso excessivo à CPU pode ser prejudicial, pois pode aumentar o tempo de espera de outros processos.
* Processos limitados por CPU: Esses processos gastam a maior parte do seu tempo executando na CPU. Como resultado, eles precisam de acesso constante à CPU para concluir suas tarefas.

Um escalonador que não diferencia esses dois tipos de processos pode causar problemas de desempenho. Por exemplo, um escalonador que dá prioridade a processos limitados por CPU pode causar um aumento no tempo de espera de processos limitados por I/O. Isso ocorre porque os processos limitados por I/O terão que esperar mais tempo para obter acesso à CPU, mesmo que estejam prontos para executar.

Um escalonador que diferencia esses dois tipos de processos pode melhorar o desempenho do sistema ao dar prioridade aos processos limitados por I/O. Isso significa que esses processos terão a oportunidade de executar e, eventualmente, concluir suas tarefas.

Aqui estão alguns exemplos específicos de como um escalonador pode diferenciar programas limitados por I/O de programas limitados por CPU:

* Algoritmo de escalonamento de tempo de espera: Este algoritmo escolhe o processo com o menor tempo de espera na fila de prontos. Como os processos limitados por I/O têm um tempo de espera relativamente baixo, eles são mais propensos a serem escolhidos pelo escalonador.
* Algoritmo de escalonamento de prioridade baseado em I/O: Este algoritmo atribui uma prioridade mais alta aos processos limitados por I/O. Como os processos com prioridade mais alta são mais propensos a serem escolhidos pelo escalonador, isso garante que os processos limitados por I/O tenham acesso à CPU quando necessário.
* Algoritmo de escalonamento de quantum dinâmico: Este algoritmo ajusta o tamanho do quantum de tempo, que é o período de tempo que um processo pode executar antes de ser interrompido, com base no tipo de processo. Os processos limitados por I/O geralmente recebem um quantum de tempo maior do que os processos limitados por CPU. Isso permite que os processos limitados por I/O executem sem serem interrompidos com frequência.

5. O comando nice é usado para estabelecer a prioridade de um processo Linux. Explique por que alguns sistemas permitem que qualquer usuário atribua a um processo um valor refinado > = 0, mas apenas o usuário root atribua valores < 0.

R- Os valores >= 0 do comando nice aumentam a prioridade do processo, enquanto os valores < 0 diminuem a prioridade. Uma prioridade mais alta significa que o processo é mais propenso a ser escolhido pelo escalonador para execução. Uma prioridade mais baixa significa que o processo é menos propenso a ser escolhido pelo escalonador para execução.

Permitir que qualquer usuário atribua a um processo um valor >= 0 é geralmente seguro, pois isso não afetará o desempenho de outros processos. No entanto, permitir que qualquer usuário atribua a um processo um valor < 0 pode ser prejudicial, pois pode causar starvation de outros processos.

Por exemplo, um usuário malicioso pode atribuir um valor muito baixo a um processo, o que faria com que o processo tivesse prioridade muito baixa e, portanto, fosse executado com pouca frequência. Isso poderia impedir que outros processos importantes recebessem acesso à CPU.

Para evitar esse problema, alguns sistemas operacionais permitem que apenas o usuário root atribua valores < 0. O usuário root tem mais experiência e responsabilidade do que os usuários regulares, portanto, é menos provável que abuse desse privilégio.

Aqui estão alguns exemplos específicos de como um usuário malicioso pode usar o comando nice para prejudicar outros processos:

* Um usuário malicioso pode atribuir um valor muito baixo a um processo que está usando muita memória ou recursos de rede. Isso poderia causar um aumento no tempo de espera para outros processos que precisam desses recursos.
* Um usuário malicioso pode atribuir um valor muito baixo a um processo que está executando uma tarefa importante, como uma atualização do sistema ou uma cópia de backup de dados. Isso poderia causar atrasos ou falhas nessas tarefas.

É importante usar o comando nice com cuidado e estar ciente dos possíveis riscos.

6. Qual dos algoritmos de escalonamento pode resultar em starvation?
a. Primeiro-a-chegar, primeiro-a-ser-atendido
b. Menor-job-primeiro
c. Round-robin
d. Por prioridades

R- A resposta é (d).
Por o escalonamento por prioridade focar sempre nos processos de maior prioridade, pode ocorrer de processos com menor prioridade sofrerem de starvation caso a demanda de processos de prioridade alta seja constante.

7 .A seguir, são apresentadas tabelas que possuem um conjunto de tarefas periódicas. Para cada conjunto, desenha a execução dos processos até o instante X, quando utilizados os algoritmos Shortest Remaining Time Next, Round Robin, Escalonamento por prioridade, Rate Monotonic e EDF. Avalie se há perda de deadline em algum deles.

a.) X = 100

Tarefas Periódicas	Período	Tempo de Computação	Chegada	Deadline	Quantum	Prioridade
Tarefa A	10	4	0	20	5	2
Tarefa B	20	8	0	40	5	1
Tarefa C	30	12	0	60	5	0

b.) X = 100

Tarefas Periódicas	Período	Tempo de Computação	Chegada	Deadline	Quantum	Prioridade
Tarefa A	10	5	0	20	5	2
Tarefa B	20	10	0	40	5	1
Tarefa C	30	15	0	60	5	0

c.) X = 20

Tarefas Periódicas	Período	Tempo de Computação	Chegada	Deadline	Quantum	Prioridade
Tarefa A	3	1	9	3	0.5	1
Tarefa B	6	0.75	0	6	0.5	2
Tarefa C	2	0.5	3	2	0.5	0
Tarefa D	4	0.75	0	4	0.5	3

R- Houve perda de deadline na tarefa C em todos os algoritmos.

d.) X = 100

Tarefas Periódicas	Período	Tempo de Computação	Chegada	Deadline	Quantum	Prioridade
Tarefa A	10	5	5	10	5	3
Tarefa B	15	8	5	15	5	1
Tarefa C	25	10	0	25	5	0
Tarefa D	30	15	0	30	5	2

R- Não houve perda de deadline em nenhum dos casos.

e.) X= 80

Tarefas Periódicas	Período	Tempo de Computação	Chegada	Deadline	Quantum	Prioridade
Tarefa A	8	4	8	8	4	2
Tarefa B	16	8	0	16	4	1
Tarefa C	24	12	0	24	4	1
Tarefa D	32	16	24	32	4	2

R- Não houve perda de deadline em nenhum dos casos.

f.) X= 30

Tarefas Periódicas	Período	Tempo de Computação	Chegada	Deadline	Quantum	Prioridade
Tarefa A	3	1	0	3	2	2
Tarefa B	6	3	0	6	2	1
Tarefa C	9	5	0	9	2	0
Tarefa D	15	8	0	15	2	3

R- Não houve perda de deadline em nenhum dos casos.

g.) X= 70

Tarefas Periódicas	Período	Tempo de Computação	Chegada	Deadline	Quantum	Prioridade
Tarefa A	7	2	0	7	2	1
Tarefa B	7	2	0	7	2	2
Tarefa C	14	5	0	14	5	0
Tarefa D	28	12	0	28	12	3

R- Houve perda de deadline no escalonamento por prioridade, a tarefa C não executa até o instante 70.

h.) X= 30

Tarefas Periódicas	Período	Tempo de Computação	Chegada	Deadline	Quantum	Prioridade
Tarefa A	4	1	0	4	0.5	1
Tarefa B	5	0.75	0	5	0.5	2
Tarefa C	3	0.5	0	3	0.5	0
Tarefa D	4	0.75	0	4	0.5	3

R- Houve perda de deadline no escalonamento por prioridade, na tarefa B, e no SRTN nas tarefas B e D