Sistemas Operacionais

André Luiz da Costa Carvalho

andre@icomp.ufam.edu.br

Aula 05 - Escalonamento por Prioridade

Aula de Hoje

Escalonamento

Na aula passada vimos duas métricas para medir a qualidade de um escalonamento: Tempo de execução (turnaround) e Tempo de resposta.

Vimos que o SJF/STCF eram a melhor opção para tempo de execução, mas eram muito ruins em tempo de resposta.

Vimos também que o Round Robin é ótimo para tempo de resposta mas terrível para tempo de execução.

Escalonamento

Round Robin, por ser muito ruim em turnaroud, não é prático em sistemas reais.

Variantes do Shortest Job tem um problema grave que é: em sistemas reais, é impossível medir o tempo de execução de um processo exatamente.

O ideal para um sistema seria um escalonador onde:

- O Tempo de Execução fosse otimizado.
- O sistema seja responsivo, ou seja, os processos comecem a rodar e respondam rapidamente para os usuários. Idealmente

O grosso de pesquisa em escalonamento vai no sentido de atender estas demandas (entre outras).

Aula de Hoje

1. MLFQ
2. Prioridade
   - Exemplo 1
   - Exemplo 2

Multi-Level Feedback Queue

O MLFQ, ou fila de prioridades multi-nível, tenta resolver estes dois problemas ao mesmo tempo.

A base dele é várias filas, uma para cada nível de prioridade. Um processo pronto sempre estará apenas em uma fila por vez.

O MLFQ sempre escolhe um processo com a prioridade mais alta pra rodar.
Se tiver mais de um com a mesma prioridade, se aplica um Round Robin na fila.

- Ou seja, as filas são circulares.

Regras básicas do MLFQ

1. Se $Prioridade(A) > Prioridade(B)$, rode A (e não B).
2. Se $Prioridade(A) == Prioridade(B)$, rode A e B em round robin.

Regras básicas do MLFQ

1. Se $Prioridade(A) > Prioridade(B)$, rode A (e não B).
2. Se $Prioridade(A) == Prioridade(B)$, rode A e B em round robin.

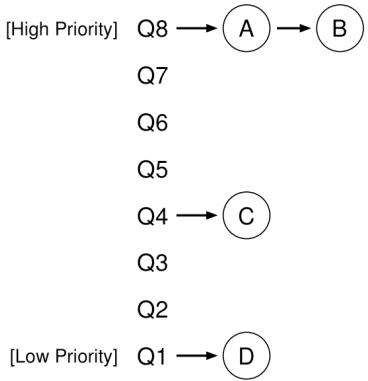
O cerne do MLFQ está em **como determinar a prioridade de cada processo.**

Na prática, o MLFQ altera a prioridade de um processo de acordo com o seu comportamento.

- Se ele está normalmente largando a CPU para esperar por E/S, ele pode ter prioridade alta.
- Se ele normalmente usa seu quantum até o final, ele tem prioridade mais baixa.

MLFQ tenta aprender sobre o processo, usando o histórico para prever o futuro.

Exemplo



Qual o problema desse exemplo?

Aula de Hoje

Alterando a prioridade

Pro MLFQ funcionar, temos que ter critérios bons para alterar a prioridade de um processo enquanto ele está ativo.

Para isto, é necessário entender o workload; ele é normalmente dividido em:

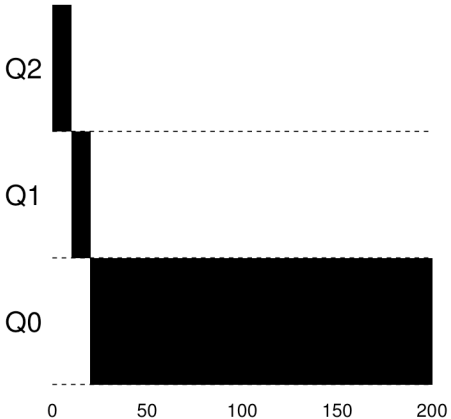
- **CPU-Bound**: processos que constantemente precisam da CPU; que rapidamente carregam os dados necessários para a memória para computar durante um bom tempo.
- **IO-Bound**: processos que estão constantemente fazendo Entrada/Saída, processos que dependem fortemente da interação dos usuários.

Novas Regras

3. Sempre que um processo é criado, ele tem a prioridade máxima.
4. Se o processo usar todo o seu quantum, a prioridade dele é reduzida.
5. Se o processo sair da CPU antes do fim do quantum, ele mantém o mesmo nível de prioridade.

Vocês conseguem explicar o porquê?

Um processo apenas



Dois processos

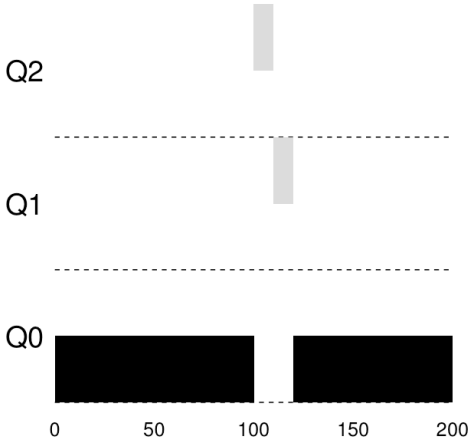
Esse exemplo é moleza. Vamos pensar num exemplo mais complicado, e um que mostra como o MLFQ aproxima o SJF.

Imagine um exemplo com dois processos:

- A, que roda por muito tempo usando a CPU.
- B, que é curto e interativo (depende muito de entrada e saída).

O que vai acontecer?

Exemplo 2



Exemplo 2

Assim, fica fácil entender o objetivo do algoritmo:

Como ele não sabe qual o perfil do processo, ele assume que será curto, dando prioridade alta.

Se ele for realmente curto, ok. Senão ele vai descendo em prioridade conforme a longevidade dele, lembrando o SJF.

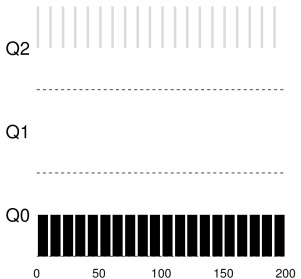
E Entrada e Saída?

De acordo com as regras, um processo que larga a CPU sozinho mantém a prioridade. Por que?

E Entrada e Saída?

De acordo com as regras, um processo que larga a CPU sozinho mantém a prioridade. Por que?

Pois processos interativos (I/O Bound) largam a CPU facilmente, então eles causam poucos problemas se pegarem a CPU.



Problemas

Obviamente esta formulação do MLFQ parece interessante, mas ao mesmo tempo tem **problemas óbvios**. Quais?

Problemas

Obviamente esta formulação do MLFQ parece interessante, mas ao mesmo tempo tem **problemas óbvios**. Quais?

1. Starvation.

Problemas

Obviamente esta formulação do MLFQ parece interessante, mas ao mesmo tempo tem **problemas óbvios**. Quais?

1. Starvation.
2. Programas feitos pra ENGABELAR o S.O.

Problemas

Obviamente esta formulação do MLFQ parece interessante, mas ao mesmo tempo tem **problemas óbvios**. Quais?

1. Starvation.
2. Programas feitos pra ENGABELAR o S.O.
3. Mudanças no comportamento de um programa (**MUITO SÉRIO**).

Mudança de prioridade

Starvation é um problema grave. Como resolver? (lembre que isso é cada vez que o quantum acaba, então não pode ser um algoritmo complexo).

Mudança de prioridade

Starvation é um problema grave. Como resolver? (lembre que isso é cada vez que o quantum acaba, então não pode ser um algoritmo complexo).

Boost de prioridade em processos periódico:

6. Após um período de tempo S, todos os processos voltam para a fila mais alta.

Quais as vantagens disso?

Vantagens

Fim do Starvation, pois de tempo em tempo os processos pesados terão sua vez na CPU.

Se um processo muda de CPU-Bound para I/O-Bound, quando ele voltar para a prioridade mais alta ele se manterá lá.

Exemplo

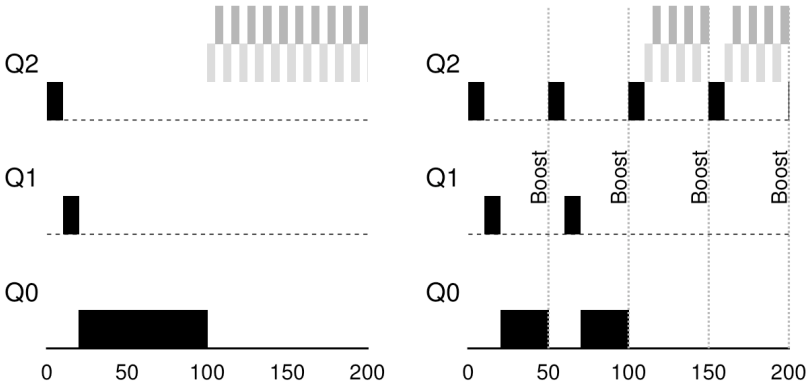


Figure 8.5: **Without (Left) and With (Right) Priority Boost**

Mas e o S?

Um problema é qual o valor bom de S para o seu sistema?

Mas e o S?

Um problema é qual o valor bom de S para o seu sistema?

Constantes vudu.

Se for muito longo, Starvation. Se for muito longo, processos interativos tem tempo de resposta ruim.

Evitando safadenhos

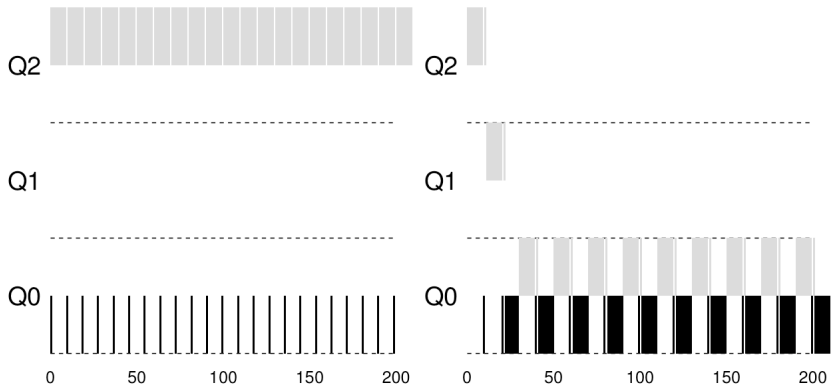
É importante para o sistema evitar que alguns processos, sabendo do funcionamento do escalonador, tentem engabela-lo.

Ao invés de contar quanto tempo o processo passou na CPU da última vez, contar o tempo que ele passou em um determinado nível de prioridade.

Evita que processos usem 99% do quanta antes de sair e se manter no mesmo nível.

Exemplo

Assim que um processo terminar de usar o seu quantum (independente de quantas vezes ele teve que passar pela CPU) ele diminui sua prioridade:



Ajustando o MLFQ

Uma coisa que parece clara é que o MLFQ tem muitos parâmetros.

Como definir estes parâmetros?

- Número de filas
- Quantum
- Boost de processos

Isso não é fácil, e depende muito do **workload**.

Ajustando o MLFQ

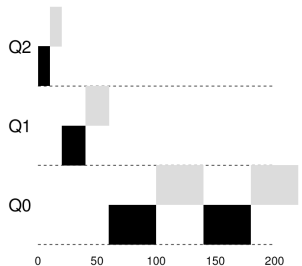
Normal que filas de alta prioridade tenham quantum baixo, e vice-versa.

Por que?

Ajustando o MLFQ

Normal que filas de alta prioridade tenham quantum baixo, e vice-versa.

Por que?



Exemplos de MLFQ

Solaris: Fácil de configurar, com tabelas explícitas com como a prioridade de um processo muda a cada instante.

ADM pode mudar isso de acordo com a carga do sistema específico.

FreeBSD: Não usa exatamente as regras mostradas aqui, mas sim uma fórmula pra calcular a prioridade de um processo, baseada no uso de CPU, em adição a um decaimento com o passar do tempo.

Escalonadores também podem ter outras restrições, como reservar níveis de prioridade mais altos para processos do SO, ou ter interfaces para o próprio usuário indicar mudanças de prioridade a qualquer momento (como no comando `nice`)

Escalonamento por prioridade

1. se prioridade(A)>prioridade(B), A roda antes de B
2. se prioridade(A)==prioridade(B), rode A e B em Round Robin
3. Assim que um processo é criado, ele tem prioridade máxima
4. Assim que um processo gasta seu quantum em um nível, ele é rebaixado.
5. Após um período S, todos os processos voltam para o topo.