Escalonamento de Processos

Introdução

Em um sistema operacional com multiprogramação, múltiplos processos estão frequentemente na memória e prontos para executar. No entanto, em um sistema com uma única CPU, apenas um processo pode estar em execução em um determinado instante. A decisão de qual processo, dentre os prontos, deve ser alocado à CPU é feita pelo escalonador de processos (ou escalonador de CPU). O escalonamento de processos é uma função fundamental dos sistemas operacionais, pois influencia diretamente o desempenho do sistema, como o tempo de resposta, a vazão (throughput) e a utilização da CPU. Este capítulo explora os conceitos básicos do escalonamento de processos, os critérios utilizados para avaliar algoritmos de escalonamento, diversos algoritmos de escalonamento clássicos e modernos, e como o escalonamento é implementado nos sistemas operacionais Linux e Windows.

Conceitos Básicos de Escalonamento

- **Ciclo de Burst de CPU-E/S (CPU–I/O Burst Cycle):** A execução de um processo consiste em uma alternância de bursts (rajadas) de CPU e bursts de E/S. Um processo executa por um tempo na CPU (CPU burst), depois realiza uma operação de E/S (I/O burst), espera pela conclusão da E/S, e então retorna para outro CPU burst, e assim por diante, até sua conclusão.
- **Escalonador de CPU (CPU Scheduler):** Seleciona um processo da fila de processos prontos (ready queue) para alocar a CPU sempre que ela se torna ociosa.
- **Escalonamento Preemptivo vs. Não Preemptivo:**
  - **Não Preemptivo (ou Cooperativo):** Uma vez que a CPU é alocada a um processo, ele a mantém até que termine sua execução ou mude para o

estado de espera (waiting state), por exemplo, ao solicitar uma operação de E/S. Não há interrupção forçada do processo em execução.
  - **Preemptivo:** O sistema operacional pode interromper um processo em execução (preemptar) e alocar a CPU a outro processo. Isso pode ocorrer, por exemplo, quando um processo de maior prioridade se torna pronto ou quando o quantum (fatia de tempo) de um processo expira em sistemas de tempo compartilhado.
- **Dispatcher:** É o módulo que dá o controle da CPU ao processo selecionado pelo escalonador de curto prazo. Suas funções incluem: trocar o contexto, mudar para o modo de usuário e saltar para o local apropriado no programa do usuário para reiniciar sua execução. O tempo gasto pelo dispatcher é chamado de latência do dispatcher e deve ser o menor possível.

Critérios de Escalonamento

Diferentes algoritmos de escalonamento têm propriedades distintas e podem favorecer um conjunto de processos em detrimento de outros. Vários critérios podem ser usados para comparar algoritmos de escalonamento de CPU:

- **Utilização da CPU (CPU Utilization):** Manter a CPU o mais ocupada possível. Idealmente, varia de 40% (sistemas levemente carregados) a 90% (sistemas fortemente carregados).
- **Vazão (Throughput):** Número de processos completados por unidade de tempo. Para processos longos, a vazão pode ser de um processo por hora; para processos curtos, pode ser de dezenas de processos por segundo.
- **Tempo de Turnaround (Turnaround Time):** O intervalo de tempo desde a submissão de um processo até sua conclusão. É a soma dos períodos gastos esperando para entrar na memória, esperando na fila de prontos, executando na CPU e fazendo E/S.
- **Tempo de Espera (Waiting Time):** A soma dos períodos gastos esperando na fila de prontos. O algoritmo de escalonamento não afeta o tempo que um processo gasta executando ou fazendo E/S; ele afeta apenas o tempo de espera na fila de prontos.
- **Tempo de Resposta (Response Time):** Em um sistema interativo, é o tempo desde a submissão de uma requisição até que a primeira resposta seja produzida (não a saída completa). É o tempo que leva para começar a responder, não o tempo que leva para exibir a resposta.

O objetivo é geralmente maximizar a utilização da CPU e a vazão, e minimizar o tempo de turnaround, o tempo de espera e o tempo de resposta.

Algoritmos de Escalonamento

Existem diversos algoritmos de escalonamento, cada um com suas características:

1. **First-Come, First-Served (FCFS) / Primeiro a Chegar, Primeiro a Ser Servido:**
   - O processo que requisita a CPU primeiro é alocado à CPU primeiro. Implementado com uma fila FIFO (First-In, First-Out).
   - **Não preemptivo.**
   - **Vantagens:** Simples de entender e implementar.
   - **Desvantagens:** O tempo médio de espera pode ser longo, especialmente se processos curtos chegarem após processos longos (efeito comboio - convoy effect).
2. **Shortest-Job-First (SJF) / Trabalho Mais Curto Primeiro:**
   - Associa a cada processo o tamanho do seu próximo burst de CPU. A CPU é alocada ao processo com o menor próximo burst de CPU.
   - Pode ser **não preemptivo** (uma vez que a CPU é dada a um processo, ele não pode ser preemptado até completar seu burst de CPU) ou **preemptivo** (Shortest-Remaining-Time-First - SRTF: se um novo processo chega com um burst de CPU menor que o tempo restante do processo atual, o processo atual é preemptado).
   - **Vantagens:** Provadamente ótimo em termos de minimizar o tempo médio de espera para um dado conjunto de processos.
   - **Desvantagens:** Difícil de conhecer o tamanho do próximo burst de CPU. Pode levar à inanição (starvation) de processos longos se processos curtos continuarem chegando.
3. **Escalonamento por Prioridade (Priority Scheduling):**
   - Uma prioridade é associada a cada processo, e a CPU é alocada ao processo com a maior prioridade (menor número de prioridade geralmente implica maior prioridade).
   - Pode ser **não preemptivo** ou **preemptivo** (se um novo processo de maior prioridade chega, ele preempta o processo atual).
   - SJF é um caso especial de escalonamento por prioridade onde a prioridade é o inverso do próximo burst de CPU previsto.
   - **Desvantagens:** Pode levar à inanição de processos de baixa prioridade. Uma solução é o *aging*, onde a prioridade de processos que esperam por muito tempo é gradualmente aumentada.
4. **Round Robin (RR):**
   - Projetado especialmente para sistemas de tempo compartilhado.
   - Semelhante ao FCFS, mas com preempção adicionada para alternar entre os processos.
   - Uma pequena unidade de tempo, chamada quantum de tempo (time quantum) ou fatia de tempo (time slice), é definida (geralmente de 10 a 100 milissegundos).
   - A fila de prontos é tratada como uma fila circular. O escalonador percorre a fila de prontos, alocando a CPU a cada processo por um intervalo de tempo de até 1 quantum. Se o processo ainda estiver rodando ao final do quantum, ele é preemptado e colocado no final da fila de prontos.
   - **Vantagens:** Bom tempo de resposta para processos curtos.

- **Desvantagens:** O desempenho depende muito do tamanho do quantum. Se o quantum for muito grande, RR se comporta como FCFS. Se for muito pequeno, a sobrecarga de trocas de contexto se torna excessiva.

5. **Escalonamento por Múltiplas Filas (Multilevel Queue Scheduling):**
   - A fila de prontos é particionada em várias filas separadas. Por exemplo, uma fila para processos de primeiro plano (interativos) e outra para processos de segundo plano (batch).
   - Cada fila tem seu próprio algoritmo de escalonamento (e.g., RR para primeiro plano, FCFS para segundo plano).
   - O escalonamento entre as filas também é necessário, geralmente implementado como escalonamento por prioridade fixa (e.g., servir todos os processos da fila de primeiro plano antes dos da fila de segundo plano) ou com divisão de tempo entre as filas.
   - **Desvantagens:** Pouca flexibilidade; processos não podem mudar de fila.
6. **Escalonamento por Múltiplas Filas com Retroalimentação (Multilevel Feedback Queue Scheduling):**
   - Permite que um processo se mova entre as várias filas.
   - A ideia é separar processos com diferentes características de burst de CPU. Se um processo usa muito tempo de CPU, ele será movido para uma fila de prioridade mais baixa. Se um processo espera muito tempo em uma fila de baixa prioridade, ele pode ser movido para uma fila de prioridade mais alta (aging).
   - É o algoritmo de escalonamento de CPU mais geral e também o mais complexo. Os parâmetros que definem um escalonador de múltiplas filas com retroalimentação incluem: o número de filas, o algoritmo de escalonamento para cada fila, o método usado para determinar quando promover um processo para uma fila de maior prioridade, o método usado para determinar quando rebaixar um processo para uma fila de menor prioridade, e o método usado para determinar qual fila um processo entrará quando precisar de serviço.

Escalonamento nos Sistemas Operacionais Linux e Windows

Linux

O escalonador do Linux evoluiu significativamente ao longo do tempo. Versões mais antigas usavam um escalonador O(1) (tempo constante para seleção de processo), enquanto versões mais recentes (a partir do kernel 2.6.23) utilizam o **Completely Fair Scheduler (CFS)** para tarefas normais (não de tempo real).

- **Completely Fair Scheduler (CFS):**
  - Projetado para ser justo, dando a cada processo uma porção proporcional do tempo da CPU.

  - Não usa prioridades fixas ou fatias de tempo da maneira tradicional. Em vez disso, mantém um “tempo de execução virtual” (vruntime) para cada processo. O processo com o menor vruntime é escolhido para executar.
  - O vruntime de um processo aumenta à medida que ele executa. Processos que executam por menos tempo ou que são limitados por E/S terão vruntime menor e, portanto, maior chance de serem escalonados.
  - Usa uma árvore rubro-negra para armazenar os processos prontos, ordenada por vruntime, permitindo a seleção eficiente do próximo processo.
- **Escalonamento de Tempo Real no Linux:**
  - Para processos com requisitos de tempo real, o Linux oferece políticas de escalonamento como SCHED_FIFO (First-In, First-Out, preemptivo, baseado em prioridade) e SCHED_RR (Round Robin, preemptivo, baseado em prioridade com quantum).
  - Esses processos têm prioridade sobre os processos normais gerenciados pelo CFS.

Windows

O Windows utiliza um escalonador preemptivo baseado em prioridades, com 32 níveis de prioridade.

- **Níveis de Prioridade:**
  - Divididos em duas classes principais: classe de tempo real (prioridades 16-31) e classe de prioridade variável (prioridades 1-15). A prioridade 0 é reservada para a thread de página zero.
  - Dentro da classe de prioridade variável, a prioridade de uma thread pode mudar dinamicamente (priority boosting). Por exemplo, quando uma thread completa uma operação de E/S, sua prioridade pode ser temporariamente aumentada para melhorar a responsividade.
- **Quantum:**
  - Cada thread tem um quantum de tempo. Quando o quantum de uma thread expira, se houver outra thread de mesma prioridade pronta, ocorre uma troca de contexto. Se não houver, a thread continua executando.
  - O valor do quantum pode variar dependendo da configuração do sistema (e.g., otimizado para aplicações de servidor ou desktop) e da prioridade da thread.
- **Filas de Prontos:** O Windows mantém uma fila de prontos para cada nível de prioridade. O escalonador sempre escolhe uma thread da fila de maior prioridade não vazia.
- **Afinidade de Processador:** Permite que threads sejam restringidas a executar em um subconjunto específico de processadores em sistemas multiprocessadores.

Conclusão

O escalonamento de processos é um componente crítico dos sistemas operacionais, determinando como o recurso mais fundamental, a CPU, é compartilhado entre as tarefas concorrentes. A escolha de um algoritmo de escalonamento envolve um trade-off entre vários critérios, como utilização da CPU, vazão, tempo de turnaround, tempo de espera e tempo de resposta. Algoritmos como FCFS, SJF, Prioridade e Round Robin fornecem diferentes abordagens para esse problema, cada um com suas próprias vantagens e desvantagens. Sistemas operacionais modernos como Linux e Windows implementam algoritmos de escalonamento sofisticados (CFS no Linux, escalonamento por prioridade multinível no Windows) que tentam equilibrar justiça, eficiência e responsividade para uma ampla gama de cargas de trabalho, incluindo tarefas interativas, batch e de tempo real. A compreensão desses algoritmos e de suas implementações é essencial para otimizar o desempenho do sistema e desenvolver aplicações eficientes.

Referências

- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.
- Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems* (4th ed.). Pearson Education.
- Love, R. (2010). *Linux Kernel Development* (3rd ed.). Addison-Wesley Professional.
- Russinovich, M. E., Solomon, D. A., & Ionescu, A. (2012). *Windows Internals, Part 1* (6th ed.). Microsoft Press.