Análise do Escalonador do Linux

Experimento de Escalonamento de Processos

May 16, 2025

1 Introdução

Este relatório apresenta uma análise do comportamento do escalonador do Linux (CFS - Completely Fair Scheduler) em diferentes cenários. O experimento foi dividido em quatro partes:

1. Distribuição com N Processos
2. Sobrecarga com N+1 Processos
3. Efeito da Prioridade
4. Processo Bloqueado por Entrada

2 Metodologia

Para cada experimento, criamos processos que executam laços infinitos consumindo CPU e monitoramos seu comportamento usando comandos como `ps`. As métricas coletadas incluem uso de CPU, estado dos processos, tempo de execução e prioridade.

3 Resultados

3.1 Experimento 1: Distribuição com N Processos

Neste experimento, foram criados N processos (onde N é o número de núcleos do processador) executando laços infinitos para consumir CPU continuamente.

3.1.1 Observações

Número de processos ocupados: 0 Uso médio de CPU por processo: 0.00% Desvio padrão do uso de CPU: 0.00% Diferença entre máximo e mínimo uso de CPU: 0.00%

Table 1: Estatísticas dos Processos no Experimento 1

Processo	PID	%CPU	Nice	Estado

3.1.2 Análise

Com N processos em N núcleos, o escalonador CFS do Linux tenta distribuir o tempo de CPU igualmente entre todos os processos. Como cada processo está consumindo 100% de um núcleo, esperamos ver aproximadamente distribuição igual de CPU.

3.2 Experimento 2: Sobrecarga com N+1 Processos

Neste experimento, foram criados N+1 processos (mais do que o número de núcleos) para observar como o Linux gerencia a sobrecarga de processos.

3.2.1 Observações

Número de processos ocupados: 0 Uso médio de CPU por processo: 0.00% Desvio padrão do uso de CPU: 0.00% Diferença entre máximo e mínimo uso de CPU: 0.00%

Table 2: Estatísticas dos Processos no Experimento 2

Processo	PID	%CPU	Nice	Estado

3.2.2 Análise

Com N+1 processos competindo por N núcleos, o escalonador CFS precisa compartilhar os recursos de CPU disponíveis. Idealmente, cada processo deveria receber aproximadamente a mesma fatia de tempo de CPU, resultando em uma distribuição proporcional a N/(N+1) do tempo total disponível para cada processo.

3.3 Experimento 3: Efeito da Prioridade

Neste experimento, foram criados N processos, mas a prioridade de um deles foi aumentada usando `renice -n -10 -p <PID>` para observar o impacto da prioridade no escalonamento.

3.3.1 Observações

Número de processos ocupados: 0 Uso médio de CPU por processo: 0.00% Desvio padrão do uso de CPU: 0.00% Diferença entre máximo e mínimo uso de CPU: 0.00%

Table 3: Estatísticas dos Processos no Experimento 3

Processo	PID	%CPU	Nice	Estado

3.3.2 Análise

No CFS, processos com maior prioridade (nice value mais baixo) devem receber mais tempo de CPU. O escalonador ajusta o tempo virtual de execução (vruntime) para processos com prioridade mais alta, permitindo que eles sejam executados por mais tempo antes de serem preemptados.

3.4 Experimento 4: Processo Bloqueado por Entrada

Neste experimento, foram criados N processos intensivos de CPU mais um processo adicional que aguarda entrada do usuário (bloqueado por I/O).

3.4.1 Observações

Número de processos ocupados: 0 Nenhum processo bloqueado encontrado no monitoramento. Uso médio de CPU por processo intensivo: 0.00% Desvio padrão do uso de CPU: 0.00%

Table 4: Estatísticas dos Processos Intensivos no Experimento 4

Processo	PID	%CPU	Nice	Estado

3.4.2 Análise

Os processos bloqueados por I/O (como esperando entrada do usuário) geralmente entram em estado de sono (estado S) e não consomem CPU enquanto bloqueados. Quando recebem entrada, são acordados pelo escalonador e competem por tempo de CPU.

4 Análise Comparativa

4.1 Comparação do CFS com Outros Algoritmos de Escalonamento

Table 5: Comparação de Algoritmos de Escalonamento

Algoritmo	**Características**
CFS (Linux)	O Completely Fair Scheduler utiliza uma árvore rubro-negra para gerenciar processos e garantir distribuição justa de recursos. Cada processo recebe uma quantidade de tempo proporcional ao seu peso (prioridade). O CFS minimiza a latência de espera, favorecendo processos interativos.
FIFO	First-In-First-Out é um algoritmo não-preemptivo que executa os processos na ordem de chegada até a conclusão. Favorece processos que chegam primeiro, independentemente do tamanho e de requisitos de CPU.
Round Robin	Algoritmo preemptivo que aloca um quantum de tempo para cada processo em uma fila circular. Garante que todos os processos tenham chance de execução, mas pode ser ineficiente com muita alternância de contexto.
SJF	Shortest Job First prioriza processos com menor tempo de execução estimado. Minimiza o tempo médio de espera, mas pode causar inanição (starvation) de processos longos.

4.2 Vantagens do CFS

O Completely Fair Scheduler (CFS) do Linux possui várias vantagens:

- **Justiça**: Garante que todos os processos recebam uma parte justa do tempo de CPU.
- **Escalabilidade**: Utiliza estruturas de dados eficientes (árvore rubro-negra) para suportar muitos processos.
- **Responsividade**: Favorece processos interativos através do modelo de tempo virtual.
- **Priorização flexível**: Permite ajuste de prioridades através de valores nice.

5 Conclusão

Com base nos experimentos realizados, podemos concluir que o escalonador CFS do Linux:

1. Distribui eficientemente o tempo de CPU entre N processos em N núcleos;
2. Mantém uma distribuição justa mesmo quando há mais processos que núcleos disponíveis;
3. Respeita as prioridades dos processos, dando mais tempo aos processos com maior prioridade;
4. Gerencia adequadamente processos bloqueados por I/O, permitindo que eles consumam CPU apenas quando necessário.

O CFS implementa um modelo de justiça que difere significativamente de algoritmos mais simples como FIFO ou Round Robin. Ao utilizar o conceito de tempo virtual de execução (vruntime), o CFS consegue balancear as necessidades de processos com diferentes prioridades e características de execução.