Relatório - EXP04

MC504 - 2s2024

Alunos: Luiz Felipe Corradini Rego Costa - 230613
Pablo Henrique Almeida Mendes - 230977

1. Introdução

Nesta atividade, utilizamos o Xv6 para aplicar métricas de validação de eficiência para avaliar diferentes otimizações a serem realizadas no sistema operacional. Utilizamos a implementação do sistema operacional em questão na arquitetura x86, a partir do repositório xv6-public, disponível no GitHub.

Para obter tais métricas, executamos X rodadas com processos CPU-bound e 20 - X processos I/O-bound, sendo X definido por usuário. Para garantir que as otimizações no escalonador fossem pertinentes, seguimos a seguinte pipeline no algoritmo que executa tais funções:

(Dentro de loop com max(X, 20-X) iterações)

1) **Primeiro Fork**
    a) Processo Filho executa a função CPU-bound, e logo em seguida é finalizado.
    b) Processo Pai segue na execução

2) **Segundo Fork**
    a) Processo filho executa a função I/O-bound, e logo em seguida é finalizado.
    b) Processo Pai segue na execução

Com isso, garantimos que os dois processos disputarão por tempo, e com isso, as otimizações devem alterar o desempenho geral do sistema. Após todas as execuções, as métricas são calculadas para cada uma das rodadas, e é mostrado para o usuário o índice da rodada atual, o valor de cada métrica, e o desempenho geral do sistema.
Além disso, condições foram adicionadas antes de cada fork para garantir que o número correto de execuções *CPU* e *IO* fossem realizadas.

A função chamada está no arquivo ***run_experiment.c,*** enquanto a implementação da pipeline exemplificada acima está no arquivo ***rodada.c***.

2. Tipos de Processos

Nessa tarefa, executamos dois tipos diferentes de processos: ***CPU-bound e I/O-bound.***

a. CPU-bound

Para o processo CPU-bound, começamos a gerar 1000 digrafos aleatórios. Tanto o número de vértices quanto as arestas do grafo são geradas por um algoritmo pseudo-aleatório de implementação própria. Para definir uma aresta, são gerados dois valores entre *(0, número de nós)*, e, caso sejam distintos, uma aresta entre eles é inserida.
Depois, implementamos o algoritmo de Dijkstra, e o executamos para retornar o caminho mínimo em cada um dos 1000 grafos gerados.
Todas as implementações estão no arquivo *graph.c,* na raiz do repositório principal.

b. I/O-bound

Já para o I/O bound, implementamos um programa que gera 100 caracteres aleatórios por 100 vezes, e os escreve em uma linha de um arquivo *.txt*. Depois, realizamos 50 permutações nessas linhas, guardando-as num buffer, alterando a posição, e as reescrevendo no arquivo.
Vale ressaltar que foi necessário alterar o nome do arquivo para cada rodada, para evitar que processos distintos tentassem alterar os mesmos arquivos.
Todas essas implementações estão no arquivo *lines.c,* na raiz do repositório principal.

3. Métricas

Para metrificar as alterações realizadas nos algoritmos de escalonamento, algumas métricas foram utilizadas para realizar a avaliação. As métricas utilizadas para análise de desempenho foram: **Vazão, Justiça entre Processos, Eficiência do Sistema de Arquivos e Memory Overhead.**

a) Vazão

Calculamos a quantidade de processos que ocorrem a cada segundo. Obtemos a média dessa quantidade e a normalizamos.

b) Justiça entre Processos

Nessa métrica, verificamos a equidade do algoritmo de escalonamento utilizado pelo escalonador. Para isso, utilizamos o tempo total de execução e o número de processos.

c) Eficiência do Sistema de Arquivos

Cálculo do tempo total para leitura, escrita e remoção de arquivos, nos processos *io bound*. Dividimos 1 pela soma desses tempos para obter o valor da Eficiência.

d) Memory Overhead

Cálculo do tempo de acesso, alocação e desalocação de memória. Foram avaliados os tempos nos processos *cpu bound,* e dividindo 1 pela soma dos tempos conseguimos obter o valor final.

Por fim, calculamos o **Desempenho Geral do Sistema** ponderando igualmente as 4 métricas. Vale ressaltar que consideramos um algoritmo de escalonamento "mais eficiente" que outro apenas quando o seu desempenho geral foi superior.

4. Otimizações

Para a avaliação de otimizações, utilizamos conceitos vistos em aula correlacionados com melhoras no fluxo de escalonamento de processos com o objetivo de aumentar a justiça entre estes e, por conseguinte, almejar um impacto na eficiência de processamento geral. Sendo assim, optamos por utilizar três algoritmos diferentes de otimização no escalonamento de processos: **Weighted-Round-Robin, Lottery e Multilevel Feedback Queue**, a fim de realizar um processo comparativo de desempenho-médio geral de cada um deles. A implementação foi realizada a partir da modificação do escalonador presente no arquivo **proc.c**, além da modificações de atributos de cada processo (tal como

peso, tickets e prioridade) no arquivo **proc.h.** Seguem as características de cada algoritmo:

WRR (Weighted Round Robin)

Como Funciona:

É uma extensão do Round Robin, onde cada processo recebe um peso que determina a quantidade de tempo de CPU que ele recebe em cada rodada. Os processos são executados em ciclos, e o tempo de execução é proporcional ao peso do processo.

Pontos Positivos:

1. Justo e equilibrado: Todos os processos têm uma chance de execução, ajustada por seu peso.
2. Priorização: Processos mais importantes podem ter maior peso, recebendo mais tempo de CPU.
3. Menos Latência: Ideal para sistemas interativos, pois todos os processos recebem a chance de ser executados regularmente.

Pontos Negativos:

1. Configuração de pesos: Caso a definição de pesos não seja de forma equilibrada entre os processos de diferentes prioridades, o desempenho geral é afetado como um todo .
2. Overhead de Context Switching: Se o número de processos for alto, o overhead tende a ser significativo devido às trocas de contexto frequentes.

Lottery Scheduling

Como Funciona:

Cada processo recebe um número de "bilhetes" de loteria. A cada turno, um bilhete é sorteado aleatoriamente, e o processo que recebe o bilhete é selecionado para execução. Processos com mais bilhetes têm uma probabilidade maior de serem escolhidos, permitindo, portanto, a implementação de prioridades.

Pontos Positivos:

1. Simples e flexível: Fácil de implementar e permite ajustes dinâmicos nas prioridades dos processos.
2. Justiça Probabilística: Processos com mais bilhetes têm mais chance de execução, mas todos têm alguma chance, mesmo com poucos bilhetes.
3. Adaptação Dinâmica: Facilmente ajustável para responder a mudanças na carga de trabalho.

Pontos Negativos:

1. Imprevisibilidade: O comportamento aleatório pode levar a uma variabilidade significativa no tempo de resposta dos processos.
2. Ineficiência em Cargas Previsíveis: Não se adapta tão bem a sistemas onde a carga é mais previsível e determinística.
3. Overhead de Gerenciamento: Precisa de uma forma eficiente de sorteio para não causar overhead significativo.

MLFQ (Multilevel Feedback Queue)

Como Funciona:

Utiliza múltiplas filas, cada uma representando um nível de prioridade diferente. Quando um novo processo chega, ele geralmente é inserido na fila de maior prioridade. O MLFQ é projetado para ser dinâmico, adaptando a prioridade dos processos com base no seu comportamento e uso de CPU.

Pontos Positivos:

1. Adapta-se ao Comportamento dos Processos: Ajusta a prioridade dos processos com base no seu uso de CPU, tornando-o eficaz para cargas mistas (processos curtos e longos).
2. Reduz Latência para Processos Curtos: Processos que usam pouca CPU recebem alta prioridade, proporcionando tempos de resposta rápidos.
3. Evita Inanição: O mecanismo de boosting impede que processos de baixa prioridade fiquem famintos por CPU.

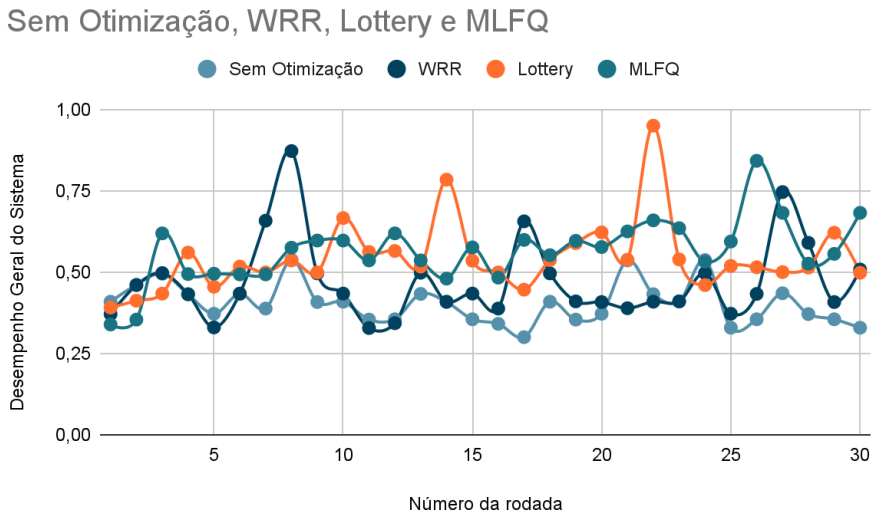
Pontos Negativos:

1. Complexidade de Implementação: Requer uma lógica sofisticada para movimentar processos entre filas e ajustar prioridades.
2. Possível Overhead: Movimentar processos entre filas e ajustar prioridades pode causar overhead adicional.
3. Dificuldade de Configuração: Definir o número de filas, os critérios de movimentação e os tempos de boost pode ser desafiador.

Seguindo as otimizações descritas, as aplicamos no xv6 juntamente com as métricas previamente estabelecidas, a fim de gerar uma análise comparativa entre os diferentes métodos. Os resultados estão descritos na seção seguinte.

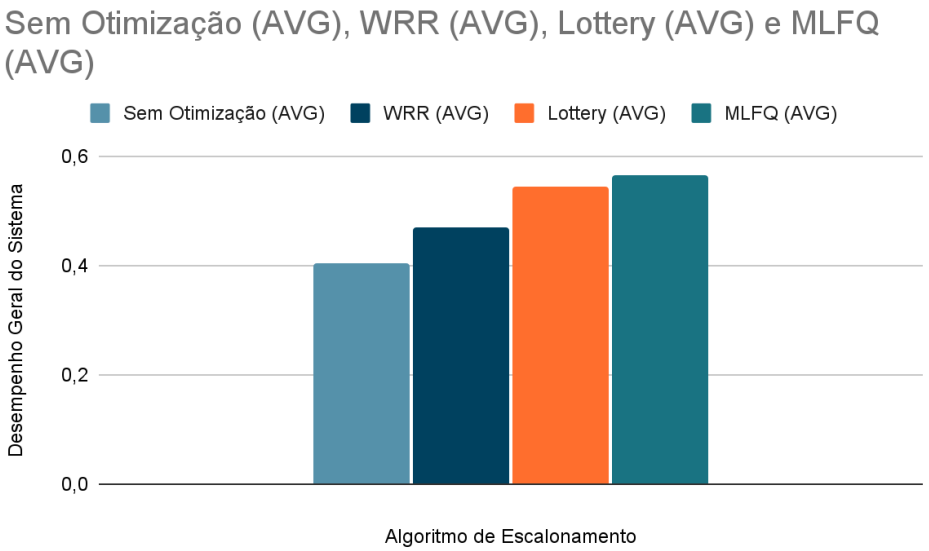
5. Resultados

Por fim, como mencionado anteriormente, registramos o desempenho por rodada de cada otimizador no formato de gráfico, como mostra a figura abaixo.



A partir desta visualização, é possível notar que, como esperado, o desempenho do escalonador padrão *(Sem Otimização)* se mostrou inferior às demais otimizações implementadas. Isso pode ter se dado pelo fato do escalonador padrão do xv6 ser apenas uma implementação básica, que seleciona o primeiro processo que estiver disponível, em contrapartida aos demais métodos, que possuem embasamento teórico para o seu funcionamento otimizado.

Além disso, podemos notar que a variância nos desempenhos gerais dos outros algoritmos se mostra mais expressiva, principalmente no WRR e no Lottery. Porém, mesmo assim, apresentam um resultado que aparenta ser superior à implementação média. No gráfico abaixo, é possível visualizar a média no desempenho de cada um dos escalonadores:



Com isso, podemos concluir que as otimizações realizadas se mostraram pertinentes, já que contribuíram significativamente para uma melhoria no desempenho geral do sistema a partir das métricas avaliadas.