UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA DE COMPUTAÇÃO GRÁFICA

JOSÉ VICTOR ROCHA DE ALENCAR
LUIZ GUSTAVO DALL'AGNOL CAVALCANTE

Trabalho Final de Sistemas Operacionais

Agosto de 2025
Boa vista - RR

Conteúdo

1. Introdução

Este projeto tem como objetivo estudar, analisar e propor uma política de escalonamento alternativa para o núcleo do sistema operacional Linux, com foco em compreender melhor o funcionamento do agendador e as possíveis melhorias que podem ser incorporadas para cenários específicos.

O escalonamento de processos é uma das funções mais críticas de um sistema operacional, pois é responsável por determinar a ordem de execução dos processos na CPU. Um bom algoritmo de escalonamento deve balancear eficiência, justiça, resposta e utilização da CPU, garantindo que o sistema se comporte de maneira responsiva e previsível, mesmo sob carga intensa.

Dentro desse contexto, este trabalho propõe a criação e avaliação de uma política de escalonamento chamada `SCHED_BACKGROUND`, voltada para processos que não requerem alta prioridade ou respostas em tempo real, como tarefas de manutenção, sincronização ou atualizações em segundo plano.

2. Conceitos Fundamentais

2.1. Definição do Escalonador no Linux

O escalonador é o componente do sistema operacional responsável por decidir qual processo ou *thread* será executado pela CPU em um determinado momento. Dado que vários processos podem estar prontos para execução simultaneamente, o papel do escalonador é distribuir o tempo de CPU de forma eficiente e justa, de acordo com políticas específicas, muitas delas baseadas em prioridades.

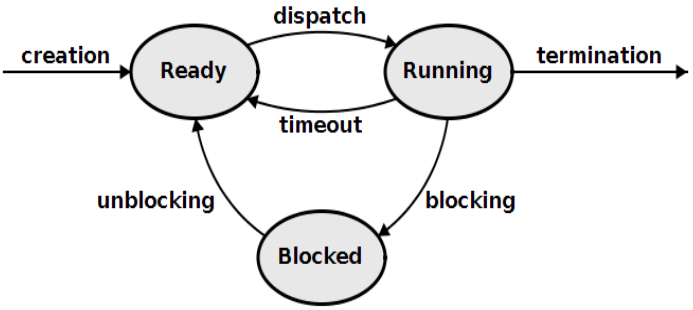
2.2. Funcionamento

O kernel do Linux adota um modelo de escalonamento preemptivo, ou seja, um processo em execução pode ser interrompido antes de ser concluído, caso outro processo com maior prioridade esteja pronto para ser executado. Essa preempção permite melhor gerenciamento dos recursos do sistema, especialmente em sistemas multitarefa.

O escalonador utiliza uma lógica baseada em prioridades para definir a ordem de execução dos processos. Quando a troca de contexto ocorre, o estado do processo anterior é salvo, e o do novo processo é restaurado, permitindo que a execução continue do ponto em que parou. Para tomar suas decisões, o escalonador precisa conhecer o estado atual de cada *thread*, classificando-as em diversas categorias.

2.3. Classificação das Threads

2.3.1. Estados



Cada *thread* pode estar em desses estados:

1 - Pronto (Ready): A *thread* está apta a ser executada, mas aguarda a disponibilidade de um processador e a decisão do escalonador.

2 - Executando (Running): A *thread* está sendo executada pela CPU no momento.

3 - Bloqueado (Blocked): A *thread* está aguardando algum evento externo (como uma operação de entrada/saída) e não pode ser executada até que o evento ocorra.

2.3.2. Recursos Utilizados

As threads também são classificadas com base no uso de recursos:

1 - CPU-Bound: São *threads* que demandam grande capacidade de processamento, utilizando intensamente a CPU (por exemplo, cálculos matemáticos complexos). Esses processos tendem a ser executados por mais tempo uma vez selecionados, mas são preteridos quando há muitos processos I/O-Bound no sistema.

2 - I/O-Bound: São *threads* que fazem uso intenso de operações de entrada e saída, como leitura de arquivos ou interações com dispositivos externos. Como passam muito tempo aguardando essas operações, consomem menos tempo de CPU, e por isso, frequentemente recebem maior prioridade pelo escalonador.

2.3.3. Tempo

O tempo também é um critério para categorizar *threads:*

1 - Tempo Real (Real-Time): As *threads* de tempo real operam sob restrições temporais conhecidas como *deadlines*. Nesses casos, a corretude da execução não depende apenas do resultado obtido, mas também de sua entrega dentro de um prazo específico. Por esse motivo, o escalonador do Linux atribui **alta prioridade** a processos de tempo real.

2 - Tempo Não Real (Non-Real-Time): Processos que não possuem restrições temporais específicas são classificados como tempo não real. Eles incluem processos do tipo *batch*, que realizam grandes volumes de processamento sem intervenção do usuário. Nesses

casos, a rapidez na resposta não é crítica, permitindo que esses processos sejam escalonados conforme os recursos do sistema estejam disponíveis.

2.3.4. Outras Características Relevantes

Além dos fatores já abordados, existem outros critérios que podem influenciar o tratamento das *threads*, dependendo da política de escalonamento utilizada. Entre eles, destacam-se a prioridade (estática ou dinâmica), o tempo que o processo já aguardou para ser executado e, no caso do Completely Fair Scheduler (CFS), o peso atribuído ao processo, que afeta diretamente a quantidade de tempo de CPU recebida.

3. Políticas de Escalonamento no Linux

3.1. SCHED_OTHER

A política padrão usada para processos comuns no sistema. Ela é baseada em um algoritmo de escalonamento conhecido como *Completely Fair Scheduler* (CFS). O CFS tenta compartilhar a CPU de forma igualitária entre as tarefas, levando em consideração fatores como tempo de execução recente e peso de prioridade. Essa política não garante tempo real, mas oferece boa responsividade para aplicações interativas e multitarefa.

3.2. SCHED_FIFO

First-In First-Out é uma política de tempo real. As threads com essa política são escalonadas de acordo com a prioridade fixa atribuída, sem preempção por threads de mesma prioridade. A thread com maior prioridade que estiver pronta será executada até bloquear ou ser interrompida por outra thread de maior prioridade. Isso garante previsibilidade, mas exige cuidado para não causar inanição de threads de prioridade menor.

3.3. SCHED_RR

Round Robin também é uma política de tempo real, semelhante à *First-In First-Out* , porém com adição de fatias de tempo (time slices). Threads com mesma prioridade se alternam na CPU após um tempo limite pré-determinado, promovendo maior justiça entre elas. Ainda assim, threads com maior prioridade preemptam as de prioridade inferior.

3.4. SCHED_IDLE

É uma política para tarefas que só devem ser executadas quando o sistema estiver ocioso. Threads com essa política têm a menor prioridade possível e são usadas para tarefas em segundo plano ou que não dependem do tempo. Elas só recebem tempo de CPU quando não há mais nada para executar.

3.5 SCHED_BACKGROUND

Nova política a ser implementada neste trabalho, menor prioridade

em comparação às demais políticas (exceto idle). Funciona de forma similar ao idle onde só é executada com o sistema ocioso e as demais políticas sem execução pronta, a diferença é que é como a SCHED_OTHER baseada no algoritmo CFS e time slices, útil para por exemplo caches, indexação de arquivos e sincronização com a nuvem, tarefas computacionalmente mais leves mas de segundo-plano.

4. Tutorial para Criação de Políticas de Escalonamento

- **Ubuntu no WSL2 e QEMU**
  Instale o wsl2 (distro pode ser ubuntu ou debian) e o qemu. Atualize usando o 'sudo apt update'.
- **Baixe o código fonte do kernel com wget**
  No terminal " wget https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/snapshot/linux-5.15.tar.gz"
- **Defina a política**
  #define SCHED_BACKGROUND 7
- **Registre a nova classe de escalonamento**
  Em kernel/sched/core.c:
  extern const struct sched_class background_sched_class;
  Localize as linhas:
  static const struct sched_class *sched_class_highest = &stop_sched_class;
  static const struct sched_class *sched_class_lowest = &idle_sched_class;
  Modifique para:
  static const struct sched_class *sched_class_highest = &stop_sched_class;
  static const struct sched_class *sched_class_lowest = &background_sched_class;
- **Implemente a classe nova**
  #include "sched.h"
  const struct sched_class background_sched_class ={//reaproveita o algoritmo CFS}
- **Atualize o mapeamento de política**
  Ainda em kernel/sched/core.c na linha sched_class:
  case SCHED_BACKGROUND:
  return &background_sched_class;
- **Valide**
  #define sched_policy_valid(policy) \
  ((policy) == SCHED_NORMAL || \
  *demais políticas
- **Compile**
  make defconfig
  make -j$(nproc)
- **Execute no QEMU**
  qemu-system-x86_64 \
  -kernel arch/x86/boot/bzImage \
  -append "console=ttyS0" \
  -nographic \
  -m 1G

- **Teste a política**
  dentro de #define _*GNU*_SOURCE
  gcc test_bg.c -o test_bg

5. Implementação da Política SCHED_BACKGROUND

Foi criado dois arquivos, um ‘benchmark.c’ onde o computador testa o funcionamento e desempenho das políticas de escalonamento definidas, e serve como base para comparar o desempenho. Em conjunto foi criado ‘schedbckg.c’ para testar as políticas de escalonamento, esse arquivo configura a política de escalonamento a ser utilizada podendo ser alterada via syscall ou no kernel. Os seguintes processos foram utilizados para o benchmark:
sudo ./schedbckg OTHER ./benchmark
sudo ./schedbckg FIFO ./benchmark
sudo ./schedbckg RR ./benchmark
sudo ./schedbckg BACKGROUND ./benchmark

6. Conclusão

A implementação da política de escalonamento SCHED_BACKGROUND demonstrou o comportamento esperado de uma política de baixa prioridade e que tenta ao máximo operar em “segundo-plano”, seu impacto na performance é mínimo porém perceptível. O tempo de wallclock é grande em comparação a por exemplo SCHED_FIFO, quando executada de forma isolada ela demonstrou uma performance similar ao SCHED_OTHER devido a reutilização do CFS fazendo com que processos compitam pela CPU.

Essa política mostrou-se útil para executar tarefas que não demandam resposta imediata, indexação e sincronização, por exemplo. Possíveis ajustes seriam a inclusão de um contador de starvation para prevenir que fique tempo demais sem executar, pois devido a sua baixa prioridade a política está propícia a condições de starvation.

7. Referências Bibliográficas

1 - OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da Silva; TOSCANI, Simão Sirineo. *Sistemas operacionais*. 3. ed. Porto Alegre: Bookman, 2008. Reimpressão. ISBN 9788577803378.

2 - COLOMBO, Víctor Cora. *Linux kernel QEMU setup*. São Carlos: Blog Cybersecurity, 7 ago. 2021. Disponível em: https://vccolombo.github.io/cybersecurity/linux-kernel-qemu-setup/. Acesso em: 4 ago. 2025.

8. Apêndice

Link do repositório do projeto no Github:
https://github.com/GugaCS50/ProjetoFinal_SO_LuizGustavo_Jos-Victor-