UFRR

UNIVERSIDADE FEDERAL DE RORAIMA

CENTRO DE CIÊNCIAS E TECNOLOGIA

BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

DCC 403 – Sistemas Operacionais (2021.1)

Prof. Herbert Oliveira Rocha

Política de escalonamento para o Kernel do Linux

JOSAFÁ MANDULÃO

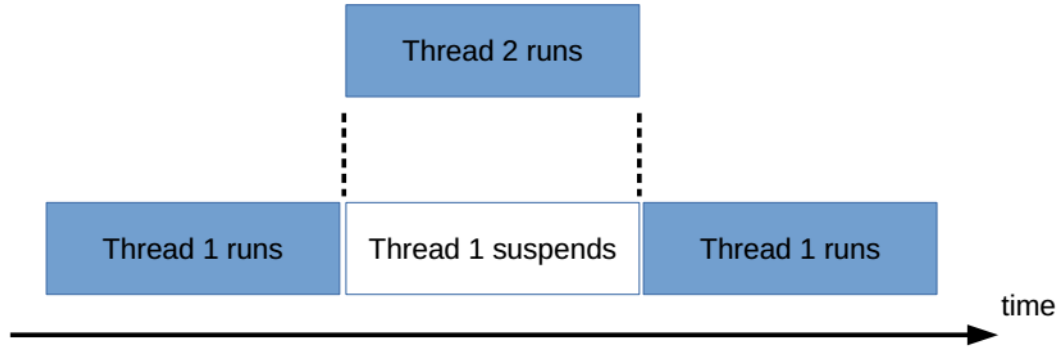
LUCAS BESSA

RAFAEL NÓBREGA

Boa Vista/RR
2021

- **Como funciona o escalonador no Linux**

Antes mesmo de falar como funciona o escalonador precisamos saber sua definição. Em sistemas de computadores modernos pode haver vários threads esperando para serem atendidos, com isso o escalonador é o responsável por ajudar o Kernel a decidir qual thread e por quanto tempo ele deve ser executado.



O kernel do Linux apresenta um agendamento preemptivo, isso significa que em determinada situação uma thread pode ser parada para execução de outra antes de ser concluída, conforme a figura XXX. O que acontece, é que o escalonador do Linux trabalha com base em prioridades, desta maneira as threads ou processos que possuírem uma maior prioridade são executados primeiro e os que possuírem níveis equivalentes de prioridade serão executados de acordo com a política round-robin de escalonamento. Com isso o encadeamento antecipado retoma a thread suspensa após a outra finalizar ou simplesmente ser bloqueada.

Quando ocorre essa intercalação de threads, o contexto do thread antigo é salvo em algum lugar e o contexto do novo thread é carregado. Este procedimento é conhecido como troca de contexto.

Essa troca de contexto ocorre tão rápido que os usuários sentem que as threads estão sendo executados simultaneamente.

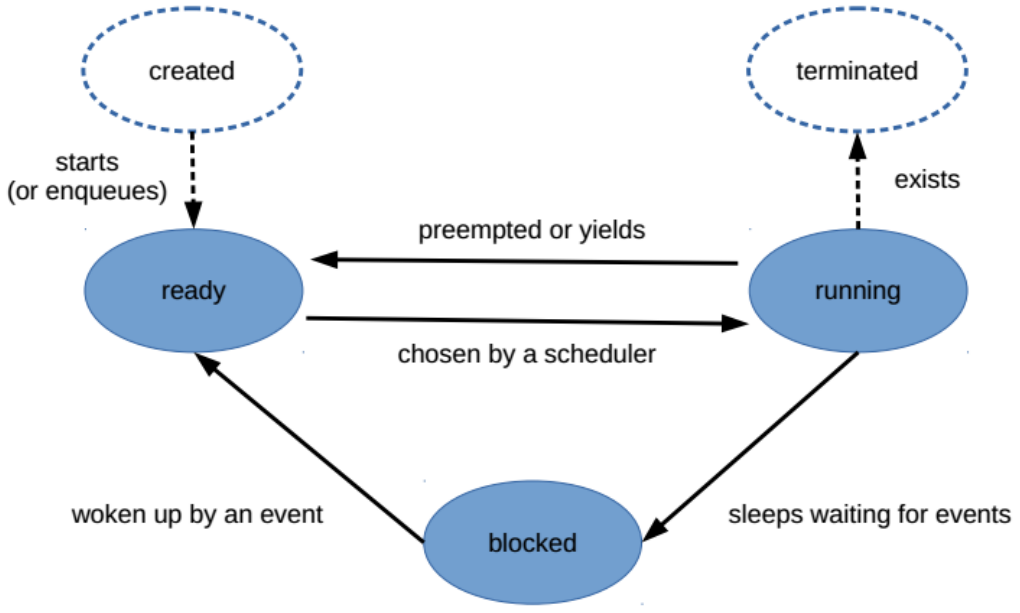
Para ocorrer a escolha o escalonador precisa saber quais são as threads que estão disponíveis para serem executadas naquele momento, por isso cada thread tem um estado atual.

Os estados dos threads são:

Pronto: O encadeamento está pronto para ser executado, mas não tem permissão, ele só terá a permissão após os processadores desocuparem os que estão executando e se o escalonador o escolher;

Executando: A thread está sendo executada em um processador;

Bloqueado :A thread está bloqueada esperando apenas algum evento externo como um I/O Bound ou um sinal. A thread bloqueada não pode ser executada.



Como dito anteriormente, existe o I/O-bound que é uma das classificações de threads. Nesse contexto, ela pode ser classificada de duas maneiras distintas:

- I/O-Bound: São encadeamentos que fazem o uso intensivo de chegadas de entradas (por exemplo, cliques no mouse ou até copiar um arquivo para um pendrive) ou a conclusão de saídas (por exemplo, gravação em discos). Esses processos são conhecidos assim por fazer pouco uso da CPU e por isso eles são processados mais rapidamente e o escalonador considera que ele tenha uma maior prioridade em relação ao CPU-Bound;
- CPU-Bound :São aqueles onde o tempo de processamento depende mais do processador do que as entradas e saídas, fazendo assim com que atrapalhe o tempo total de processamento (como por exemplo, a compilação de programa). As threads que têm vinculação com o CPU são pouco escolhidas, mas em compensação quando são escolhidas elas mantêm a CPU por mais tempo.

Outro critério para categorizar os encadeamentos é saber se eles são real time ou não.

Real-time são aquelas que geralmente trabalham com uma restrição de tempo, um prazo, também conhecido como deadline. A exatidão operacional de um encadeamento não depende apenas do resultado do cálculo, mas também de os resultados serem entregues antes do prazo. Por esse motivo o escalonador deixa com a mais alta prioridade as threads real-time. Estas, são classificadas como hard real-time ou soft real-time (devido à perda de um prazo), Threads hard real-time exigem que todos os prazos sejam

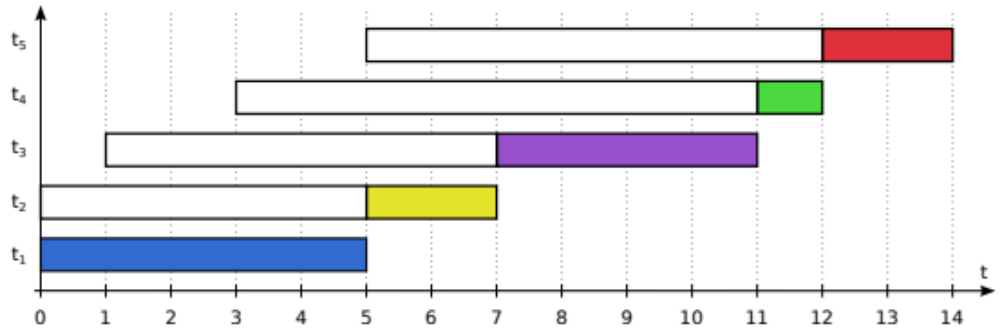
cumpridos sem exceção, caso contrário, um sistema pode cair em uma falha catastrófica. Agora, para threads soft real-time, perder um prazo resulta em degradação da qualidade do serviço pretendido, mas o sistema ainda pode continuar.

Non-real-time não está associado a nenhum prazo. Elas podem ser threads com interação humana ou threads do tipo batch, os threads batch são aquelas que processam uma grande quantidade de dados sem intervenção manual. Para os batch threads, um tempo de resposta rápido não é crítico e, portanto, eles podem ser programados para serem executados conforme os recursos permitirem.

- **Políticas de escalonamento do Linux**

1. **First In, First Out(FIFO) ou First Come, First Served(FCFS)**

É um algoritmo de escalonamento não preemptivo para a estruturas de dados do tipo fila. E tem como principal conceito: “o primeiro elemento a dar entrada na fila é o que será processado primeiro”. Isso quer dizer que ele executará um processo como todo do início ao fim não interrompendo o processo executado até ser finalizado ,então se chegar um novo processo e estiver algo processando esse novo processo vai para a fila de espera até ser atendido pela CPU.



No gráfico acima apresenta o escalonamento baseado no tempo, onde cada bloco colorido é um processo e o bloco branco seria o tempo de espera na fila.

2. **Round-Robin**

É um algoritmo escalonador de tarefas que consiste em dividir o tempo de uso da CPU. Cada processo recebe uma fatia de tempo, esse tempo é

chamado Time-Slice, também conhecido por Quantum. Os processos são todos armazenados em Fila (ou Buffer) circular. O escalonador executa cada tarefa por tempo determinado pelo Time-Slice e ao fim deste período é executada a troca de contexto, onde o próximo processo fila passa a ser executado pela CPU até percorrer o período do Time-Slice. Após percorrer todos os processos da fila, essas atividades se repetem e o escalonador aponta para a primeira tarefa. A figura a seguir ilustra bem todo esse processo.

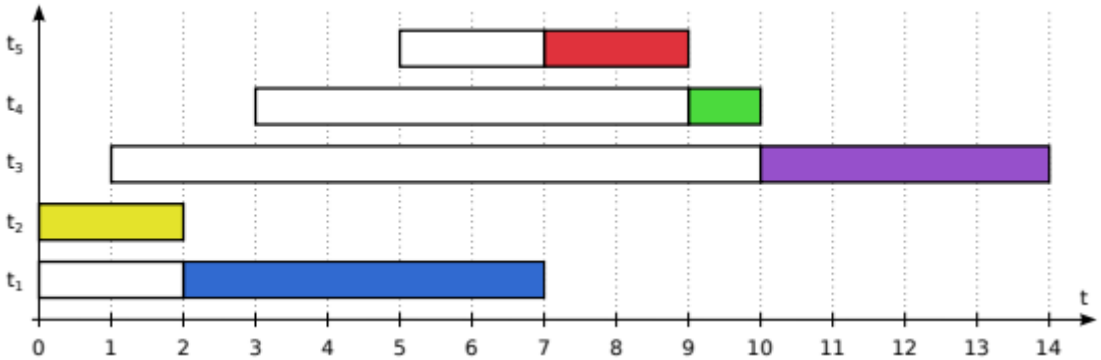
A figura acima ilustra bem como funciona esse tipo de algoritmo, onde cada bloco colorido é um processo a ser executado.

3. Escalonamento por prioridades fixas (PRIOc,PRIOp)

No escalonamento por prioridade, a cada tarefa é associada uma prioridade, geralmente na forma de um número inteiro, que representa sua importância no sistema. Os valores de prioridade são então usados para definir a ordem de execução das tarefas. Esse tipo de escalonamento pode ser cooperativo (PRIOc) ou preemptivo (PRIOp).

PRIOc:

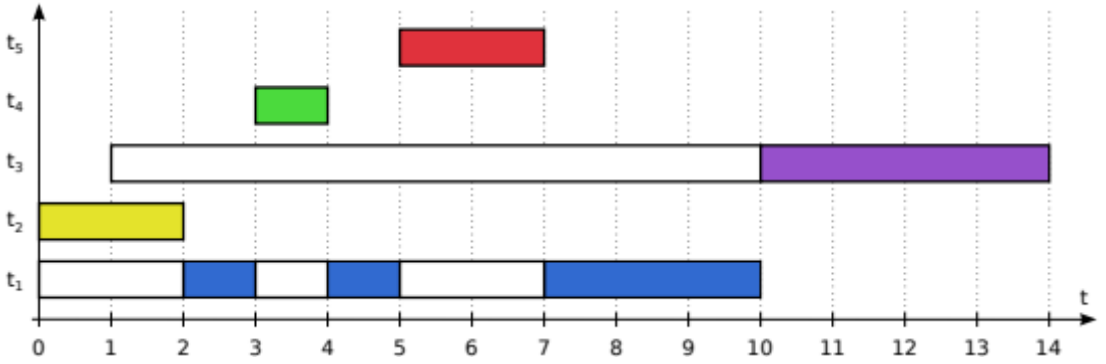
Os valores de prioridade são considerados em uma escala de prioridade positiva, ou seja, valores numéricos maiores indicam maior prioridade.



No gráfico acima os blocos coloridos significam os processos em processamento e os brancos é o tempo de espera, por estar nesse tipo de escalonamento cooperativo a prioridade é de total importância, nota-se que o t1 e t2 iniciaram ao mesmo tempo, mas por t2 ter maior prioridade ele acabou sendo processado primeiro e o t1 ficou em tempo de espera. Esse tipo de comportamento acontece até o final de todos os processos.

PRIOp:

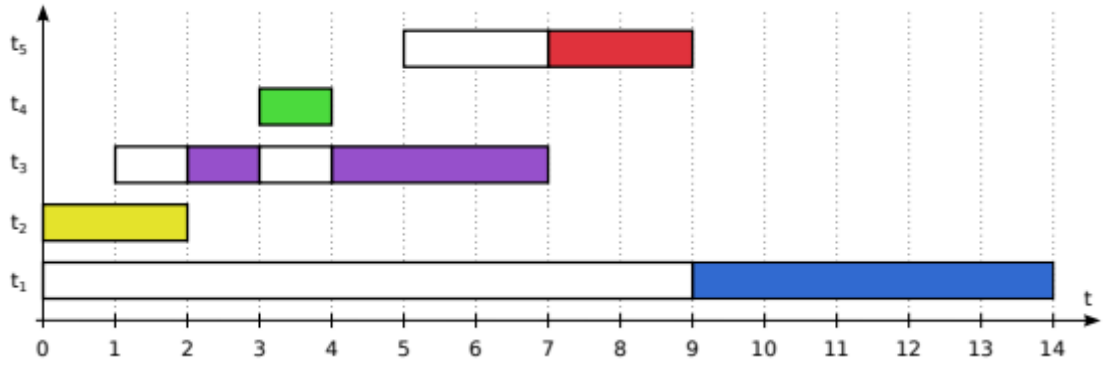
No escalonamento por prioridade preemptivo (PRIOp), quando uma tarefa de maior prioridade se torna disponível para execução, o escalonador entrega o processador a ela, trazendo a tarefa atualmente em execução de volta para a fila de prontas. Em outras palavras, a tarefa em execução pode ser “preemptada” por uma nova tarefa mais prioritária



Já nesse tipo de escalonamento preemptivo, nota-se que a prioridade também é de suma importância, tão importante que pode ocorrer de um processo com maior prioridade cessar o processo atual e tomar o seu lugar para ser processado. Esse tipo de comportamento ocorre com o t1, t4 e t5 de forma que no meio do processamento do t1 o t4 toma seu lugar para ser processado e logo após o t5 faz a mesma coisa e apenas depois, que o t1 termina de ser processado.

4. Shortest Remaining Time First(SRTF)

Esse tipo de algoritmo é cooperativo, ou seja, uma vez que uma tarefa recebe o processador, ela executa até encerrar (ou liberá-lo explicitamente). Em uma variante preemptiva, o escalonador deve comparar a duração prevista de cada nova tarefa que ingressa no sistema com o tempo de processamento restante das demais tarefas presentes, inclusive aquela que está executando no momento. Caso a nova tarefa tenha um tempo restante menor, ela recebe o processador



No caso do gráfico acima, a prioridade não é tão importante para um processo ser executado e sim o menor tempo para ser executado. Nota-se o comportamento de t3, que com a chegada de t4 ele para seu processamento e o t4 pega o seu lugar para ser processado, isso ocorre por o t4 ter o menor tempo de processamento em relação ao t3.

- ## Tutorial

Primeiramente, para implementar a sua política de escalonamento, é necessário baixar o kernel do linux na versão que você deseja modificar. É interessante que a versão do kernel a ser baixada seja a mesma que o seu sistema linux já possui para evitar incompatibilidade.

Após, você deve identificar a pasta em que o escalonador é implementado (normalmente é a pasta /kernel/sched).

Na pasta em questão, estão diversos módulos auxiliares e os dois arquivos que desejamos fazer as devidas mudanças, o sched.c e o sched.h.

Na pasta sched.c estão implementadas todas as ações do sistema com relação às políticas de escalonamento suportadas. Já na pasta sched.h estão definidas as estruturas e rotinas utilizadas no arquivo sched.c.

exemplos de implementação :

Para compilar o kernel, precisamos de algumas ferramentas e bibliotecas auxiliares que podem ser obtidas através do comando apt-get.

comando:

$sudo apt-get install linux-source-2.6.24 kernel-package libncurses5-der fakeroot.

Depois, precisamos abrir um novo shell em modo root para executar os últimos comandos:

comando:

$sudo bin/bash

Mudamos o diretório para dentro do source e instalamos o bunzip caso necessário:

comandos:

$cd /usr/src

$bunzip2 linux-2.6.24.tar.bz2

$tar xvf linux-2.6.24.tar

$ln -s linux-2.6.24 linux

Aplicamos as mudanças de implementação nas pastas sched.c, sched.h e chrtB.c.

Substituímos as pastas modificadas nos seus respectivos diretórios

fazemos uma cópia da configuração do kernel para usar durante a compilação:

comando:

$cp boot/config-2.6.24-generic /usr/src/linux/.config

Fazemos o make clean

comando:

$make-kpkg clean

Agora, compilamos o kernel e reiniciamos o sistema :

comando:

$ fakeroot make-kpkg --initrd --append-to-version=-custom kernel_image kernel_headers.

Após o sistema reiniciar, o kernel já estará pronto para executar tarefas no modo sched_background.

- ## Referências

https://embeddedlinuz.wordpress.com/2011/06/10/how-to-compile-linux-kernel-2-6/

http://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-livro.pdf

https://github.com/rkavi1408/SCHED_BACKGROUND