TRABALHO PRÁTICO 3

Conrado Santos Boeira e Gabriel Wetzel Rockenbach

Introdução

O objetivo do trabalho é explorar o escalonamento de processos no Linux pela implementação de uma aplicação multithread que utiliza diferentes políticas e prioridades de execução e avaliação de seu funcionamento. Será desenvolvido um programa que demonstra o padrão de execução de múltiplas threads, assim sendo possível comparar as diferentes políticas de escalonamento.

O programa irá utilizar de um buffer global, em qual cada thread irá escrever seu carácter na posição do ponteiro e incrementá-lo, demonstrando sua ordem de execução. Ao executar o programa, utilizaremos a ferramenta trace-cmd para realizar a depuração da execução do programa e analisar a verdadeira ordem de execução dos threads, sendo que seus resultados serão visualizados pela ferramenta KernelShark.

Implementação do Thread Runner

O programa thread_runner funciona criando múltiplas threads que escrevem em um buffer de tamanho especificado. Estas threads utilizarão uma política de escalonamento especificada na entrada. A distribuição é emulada com apenas um núcleo, de modo a observar o funcionamento da política de escalonamento de forma mais simples.

O funcionamento dos threads consiste em escrever no buffer o caractere identificador na posição apontada por um ponteiro global, seguida da incrementação desta posição. A thread continua a executar até que o buffer seja cheio.

Para analisar a troca de execução entre threads, usamos o estado final do buffer para contar o número de mudanças ocorridas ao longo da execução do programa.

Resultados

Testamos a execução do programa utilizando as políticas de escalonamento SCHED_FIFO, SCHED_OTHER e SCHED_RR.

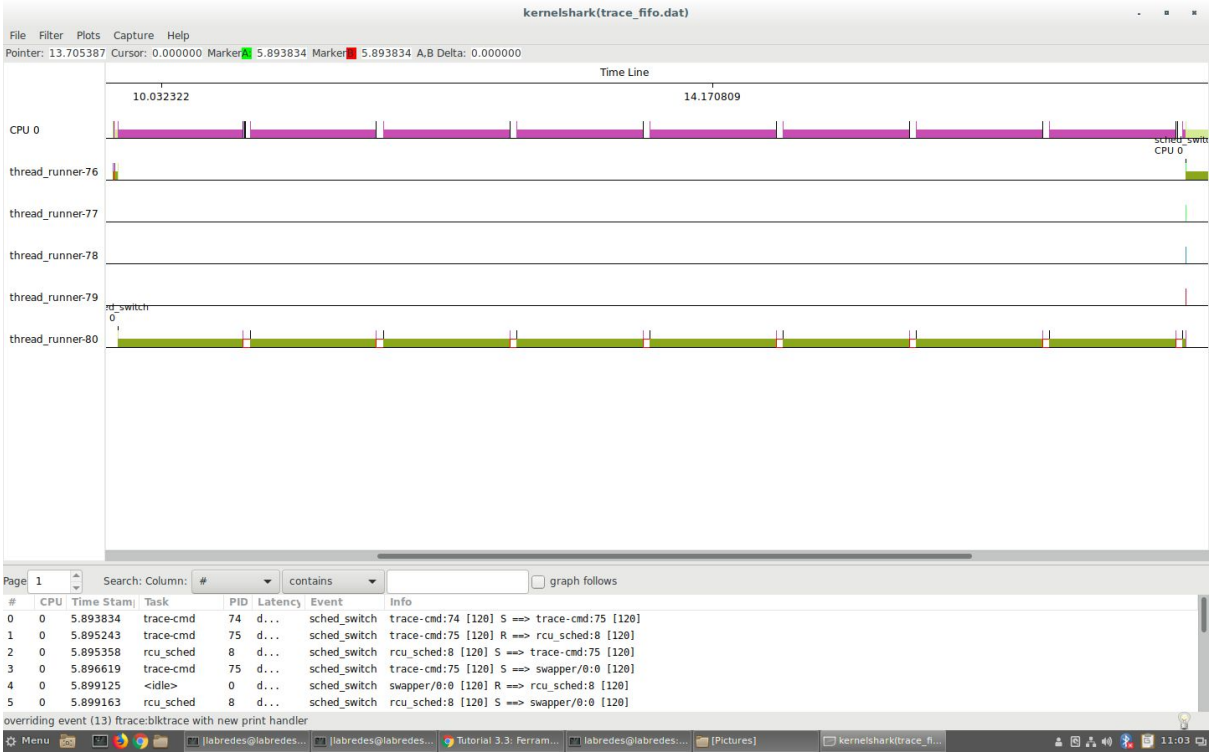
SCHED_FIFO:

A política disponibiliza o acesso à cpu por quanto tempo o processo necessite até sua finalização. Como o nome sugere, vindo do inglês First In First Out, a política usa o conceito uma fila ordenada para escolher quem executa. Ou seja, o primeiro

processo utiliza a cpu até terminar e só então o primeiro processo que chegou após ele irá executar.

FIFO

```
# thread_runner 4 3000 SCHED_FIFO 1
current: SCHED_OTHER PRI_MIN: 0 PRI_MAX: 0
new: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
Iniciando thread D
Iniciando thread C
Iniciando thread B
Iniciando thread A
D
A : 0
B : 0
C : 0
D : 1
```



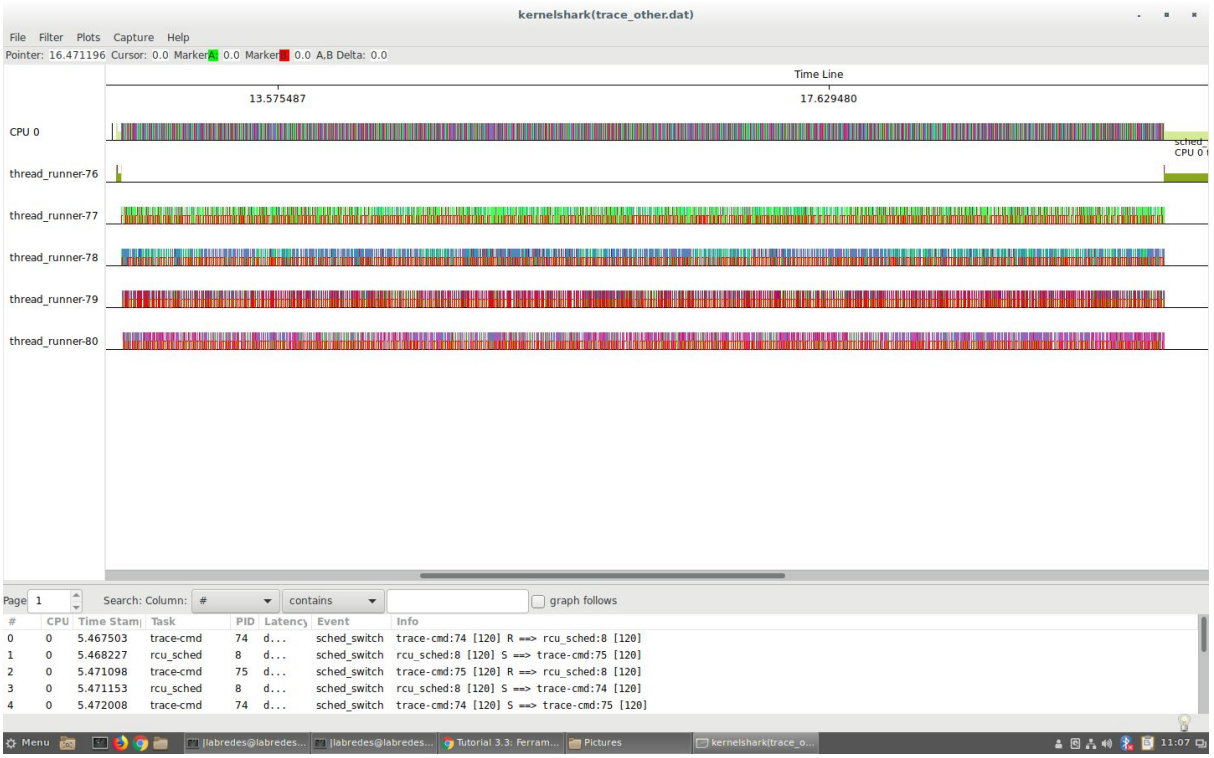
Como é possível ver pelo KernelShark, a primeira thread que chega ao escalonador é a única a executar. Isso acontece pois as threads só param quando o buffer estiver cheio e, já que essa thread específica chegou primeiro, ela pode executar ininterruptamente até usar todo o buffer.

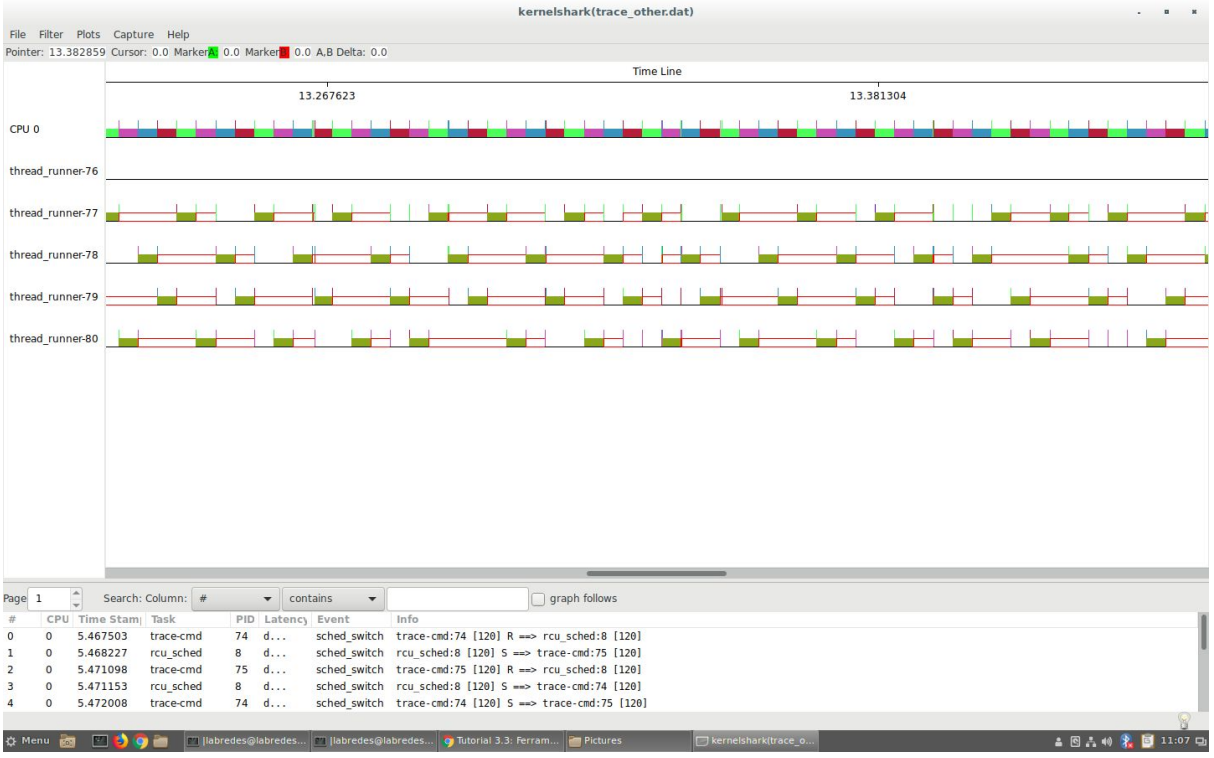
SCHED_OTHER:

Essa política tem como objetivo garantir avanço equilibrado para processos que não requerem funcionamento real-time. Ela funciona atribuindo prioridades dinâmicas para os processos que competem pelos recursos computacionais. Inicialmente, todas threads tem uma prioridade igual ao se valor de *nice*, um atributo específico a cada thread. Porém, essa prioridade vai aumentando toda vez que a thread está pronta para executar mas não é escolhida pelo escalonador. Isso garante uma certa equidade entre o tempo de execução para todas as threads.

OTHER

```
# thread_runner 4 3000 SCHED_OTHER 0
current: SCHED_OTHER PRI_MIN: 0 PRI_MAX: 0
new: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_OTHER PRI_MIN: 0 PRI_MAX: 0
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_OTHER PRI_MIN: 0 PRI_MAX: 0
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_OTHER PRI_MIN: 0 PRI_MAX: 0
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_OTHER PRI_MIN: 0 PRI_MAX: 0
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
Iniciando thread A
Iniciando thread B
Iniciando thread C
Iniciando thread D
ABDCABDABCDABCDACBDACBDACDBADCDBADCDBACDBACBDACDACDBACDBACBDACBACBDACBACDBADBACDBCADBCADCDBCABDCADBCADBCADBCDADBCADBCADBCADBCADBCABCADBCADBCDBCADB
ACDBACDBACDBACBDBACDBACDBACDBADCBACDBCACDBACDBCACD
A : 52
B : 52
C : 54
D : 54
```



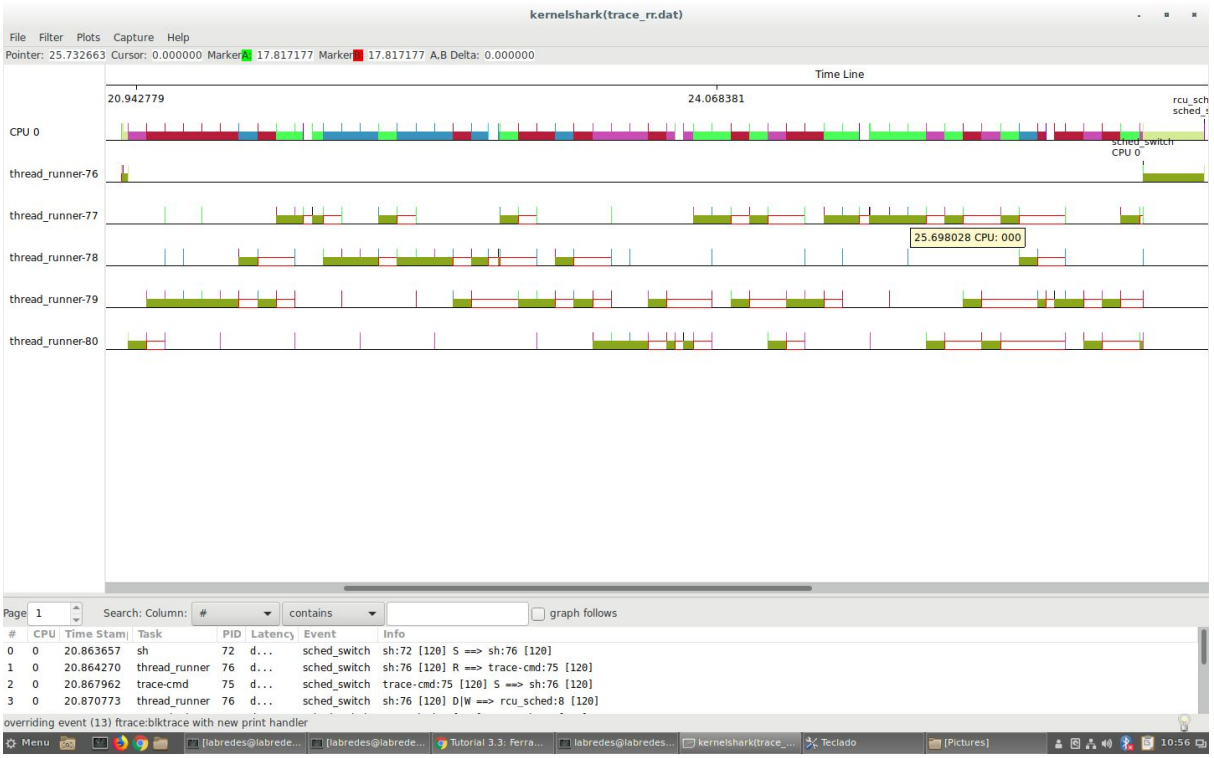


É possível ver tanto na imagem do KernelShark sem zoom quanto na com zoom, que todos os processos são atendidos de maneira a proporcionar a equidade previamente mencionada.

SCHED_RR:
Funciona de modo similar ao SCHED_FIFO, porém com tempo limitado de execução para cada processo. Quando este tempo acaba, o processo é movido para o final da lista de execução e a CPU é disponibilizada ao próximo processo.

RR

```
# thread_runner 4 3000 SCHED_RR 1
current: SCHED_OTHER PRI_MIN: 0 PRI_MAX: 0
new: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_RR PRI_MIN: 1 PRI_MAX: 99
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_RR PRI_MIN: 1 PRI_MAX: 99
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_RR PRI_MIN: 1 PRI_MAX: 99
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_RR PRI_MIN: 1 PRI_MAX: 99
current: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
new: SCHED_FIFO PRI_MIN: 1 PRI_MAX: 99
Iniciando thread D
Iniciando thread C
Iniciando thread B
Iniciando thread A
DCBADCBAD
A : 2
B : 2
C : 2
D : 3
```



Os resultados para essa política acabaram ficando menos claros nos gráficos encontrados. Acreditamos que isso se deve a algum problema durante uma escrita ou leitura que força algumas da threads a esperarem fora da fila para execução.

(OBS.: Para os gráficos mostrados com o KernelShark, não foram usados os mesmo parâmetros das imagens que mostram os resultados das execuções. Para os gráficos foi colocado um tamanho de buffer maior de modo que mais trocas entre as threads pudessem ocorrer.)

Conclusão:

Neste trabalho exploramos alguns dos diferentes algoritmos de escalonamento e pudemos ver eles funcionando na prática com uso de ferramentas como o KernelShark.