Construção de Sistemas Operacionais - Trabalho 3

Prof. Sérgio Johann Filho

Descrição

Este trabalho consiste na criação de um módulo de kernel carregável (LKM) que implementa um escalonador de requisições de entrada e saída. Esse escalonador deverá se comportar de acordo com a política C-SCAN e funcionar com o kernel Linux versão 4.13.9, utilizado nas aulas. O trabalho consiste na execução de 5 etapas:

1. Compreensão das técnicas de escalonamento de disco, incluindo as implementações atuais do kernel Linux.
2. Implementação dos tutoriais 3.1 e 3.2 para adquirir noções básicas do subsistema de I/O de disco do Linux.
3. Implementação do módulo de kernel conforme a política C-SCAN e as regras apresentadas, incluindo parametrização.
4. Implementação de aplicação para testes.
5. Avaliação de desempenho da implementação.

Implementação do módulo

O módulo de kernel a ser desenvolvido deverá ser parametrizável, de tal forma que diferentes parâmetros possam ser definidos em tempo de carga do módulo. Os seguintes parâmetros precisam ser definidos, sendo que estes irão influenciar o comportamento do algoritmo C-SCAN:

- Tamanho da fila de requisições (número de requisições): Define o número de requisições que deverão ser enfileiradas antes de se realizar o despacho das mesmas. Esse parâmetro especifica a granularidade com a qual o algoritmo irá trabalhar, ou seja, quando a fila ficar cheia devem ser atendidas as solicitações. Recomenda-se usar um valor entre 5 e 100 requisições.

- Tempo máximo de espera (em milisegundos): Define o tempo máximo em que requisições que estão na fila devem ser atendidas. Passado o tempo de espera, mesmo que a fila não esteja cheia as requisições nela armazenadas deverão ser atendidas. Recomenda-se um valor entre 1ms e 100ms.

- Modo depuração: caso habilitado, mensagens sobre o comportamento do algoritmo deverão ser exibidas no log do kernel. Por exemplo, requisições adicionadas na fila, requisições na ordem em que são atendidas, tempo dos eventos, entre outros.

Para facilitar o processo de desenvolvimento do módulo, está sendo disponibilizado juntamente com o enunciado um esqueleto de código. Esse esqueleto deverá servir como ponto de partida para o seu projeto. Ainda, sugere-se consultar como apoio as outras implementações de escalonadores de disco do Linux. Por exemplo, os escalonadores *noop* e *CFQ* estão disponíveis, respectivamente, em:

```
linux-4.13.9/block/noop-iosched.c
linux-4.13.9/block/cfq-iosched.c
```

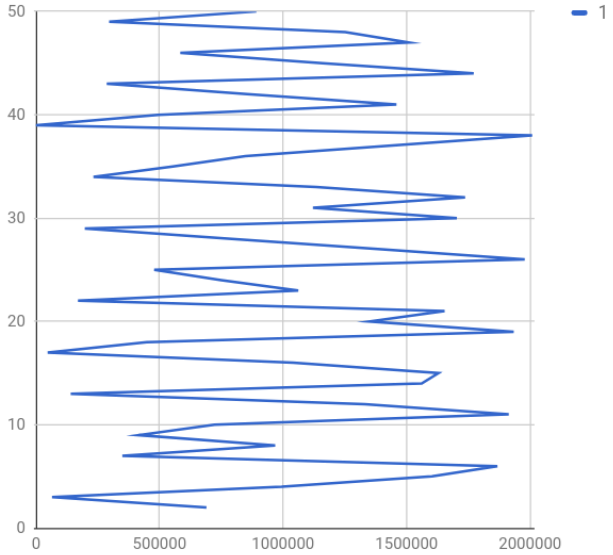
Juntamente com o esqueleto do módulo, é apresentada uma aplicação para acesso direto (raw) ao disco. Tal aplicação deverá ser modificada para gerar múltiplas requisições concorrentes para o sistema de entrada e saída, com o objetivo de simular um ambiente multiprogramado real, onde o sistema operacional precisa atender solicitações de diversos processos. A versão disponível gera apenas algumas requisições de um único processo. Dica: utilize a chamada de sistema *fork()* para criar um grande número de processos que geram requisições de leitura em regiões aleatórias do disco.

Avaliação de desempenho

Como estamos usando um simulador "*instruction accurate*", isto é, que simula o sistema a nível de instruções, não é possível tirarmos medidas efetivas de tempo para comparação entre políticas de escalonamento. Outro agravante, é que as requisições de acesso a disco do Linux emulado estão sujeitas às políticas de escalonamento do SO *host*. Contudo, podemos ter uma ótima ideia da efetividade da nossa implementação quando compararmos a ordem das requisições de acesso a disco recebidas pelo módulo e a ordem em que são atendidas. Por exemplo, considere um conjunto de 50 requisições de acesso a disco geradas na ordem abaixo:

1261960, 689984, 64968, 994448, 1603288, 1867480, 349528, 968120, 415624, 722632, 1913536, 1331304, 140544, 1561288, 1629256, 1040136, 47272, 448320, 1933696, 1350016, 1653904, 169880, 1061512, 749616, 478432, 1978192, 1383664, 792632, 197520, 1703504, 1121120, 1736912, 1133936, 232664, 553800, 847248, 1428072, 2008512, 0, 494056, 1458928, 854608, 286384, 1771728, 1160664, 584152, 1509152, 1251800, 296120, 891896

Isso resulta no gráfico de acesso a disco abaixo, que percorre um total de 37.074.272 setores (FCFS).



Se as requisições forem ordenadas segundo o algoritmo C-SCAN, teremos uma redução drástica no total de setores percorridos, pois o algoritmo atende as solicitações seguindo uma ordem por proximidade das solicitações. Desta forma, a implementação do módulo deve exibir os setores na ordem de chegada e na ordem em que forem sendo atendidos. Assim, é importante gerar um grande número de requisições de disco com a aplicação de teste, evitando que o escalonador fique sem trabalho a fazer. **Importante:** atente-se para gerar requisições de leitura e escrita, e que a faixa de valores das requisições cubram todo o tamanho do disco virtual utilizado. Por exemplo, se um disco de 4MB estiver sendo utilizado e forem considerados blocos com tamanho de 512 bytes, devem ser geradas requisições com valores (número do bloco) entre 0 e 8191.

Este trabalho deverá ser realizado em duplas ou trios e apresentado no dia 29/11 (apresentação de 5 a 10 minutos). Para a entrega, é esperado que seja enviado pelo Moodle um arquivo *.tar.gz* do projeto com o seu nome e contendo:

1. Código fonte do módulo de kernel desenvolvido;
2. Código fonte da aplicação de usuário desenvolvida para validação do módulo;
3. Um relatório simples, apresentando os resultados que ilustram o funcionamento do algoritmo e sua comparação com a política FCFS (noop).