Segundo Trabalho de INF 1019 – 2017.2

Simulando Memória Virtual e Substituição de Páginas LFU

Data de entrega: 2ªfeira, 11/12 e data da apresentação: 12/12

1- Introdução e Objetivo

Nesse trabalho, você deverá implementar um simulador de memória virtual em linguagem C. Neste simulador você criará as estruturas de dados e os mecanismos de mapeamento de memoria (virtual -> física) necessários para realizar a paginação, e deverá implementar um algoritmo *de Substituição de página global:* o Least Frequently Used (LFU).

O seu programa deverá consistir de um processo pai (GM), e 4 processos filho, P1, P2, P3, e $P4_i$, que chamaremos de user processes. Cada um desses user processes deverá ler um aquivo de endereços de memória virtual supostamente acessados pelo processo. Cada processo terá um arquivo diferente, e todos os 4 arquivos serão fornecidos pelo professor (**compilador.log, matrix.log, compressor.log** e **simulador.log**,)[1] Cada linha dos arquivos será um endereço de 32 bits (8 caracteres em hexadecimal), seguido por uma letra R ou W, para indicar se o acesso é de leitura ou de escrita.

2- Mapeando endereços virtuais para físicos

Como voce sabe, endereços virtuais precisam ser tranformados para endereços de memória física, e, para isso, cada endereço virtual é particionado em um índice de página (i) e um offset (o) dentro da página, que são os bits mais e menos significativos do endereço, respectivamente. E cabe ao gerente de memória mapear (transformar) cada índice (i) para um page frame (F) específico da Memória Física. A memoria física deverá ter apenas 256 page frames, a serem compartilhados por todos os 4 processos.

O mapeamento (end.virtual -> end.físico) está definido na Tabela de páginas de cada processo (PageTable-p_x). No entanto, como nem todas as páginas de todos os processos "cabem na memória física", eventualmente antes de concluir uma tradução de um índice i, será necessário alocar um novo page frame (F) para abrigar essa nova página e copiar a página da área de swap (no HD) para a memória física (*). O seu simulador GM não precisará simular a área de swap, mas deverá suspender o processo que sofreu o Page Fault por 1 segundo. Por outro lado, caso a página (página i) já esteja em um frame da memória, então o processo Px não será suspenso e deverá imprimir diretamente o endereço da memória física obtido, gerando o seguinte output:

Px, F-Number + o, r/w

Onde Px é o numero do processo, F-Number+o é o endereço físico da memoria, composto da concatenação do número do frame e do offset), e r/w deve ser "r" para um acesso de leitura e "w" para um acesso de escrita.

---

[1] Estes arquivos estão disponíveis na página do professor, em http://www.inf.puc-rio.br/~seibel

Note que quando ocorre uma substituição de página, as tabelas de páginas de dois processos precisam ser atualizadas: o processo que causou o page-fault e "ganhou" um novo frame; e o processo que previamente tinha uma de suas paginas na memória e que acabou der "perder" o seu frame. Ambas as atualizações serão feitas por sinais do GM para os 2 processos envolvidos: o "perdedor" e o "ganhador" de um frame.

(*) Caso o frame escolhido estiver ocupado com uma página de um processo que tenha sido modificada (isto é, bit M setado), então esta página do frame escolhido terá que ser primeiro salva na área de swap (antes que a nova página possa ser copiada para o frame). E isso requer mais um acesso ao disco. Nesse caso, simule que o processo que causou o page-fault precisará esperar o dobro do tempo: 2 segundos (devido ao acesso duplo ao disco). Para ter o seu page-fault atendido.

3- Interação entre o GM e os user processes

A tradução de endereços deverá ser requisitada por todos os processos, para cada linha lida do arquivo de log, na forma de uma chamada para um procedimento `trans()` de uma biblioteca VM a ser criada por voce. Nessa chamada, deverão ser passados o número do processo, e a tripla (i, o, r/w). Por exemplo:

```
trans(P3, 0044, e4f8, R)      // leitura na página i=0044 e offset=e4f8
trans(P1, 2f69, 65a0, W)      //escrita em página i=2f69 e offset=65a0
```

Além disso, é na biblioteca VM que o processo deverá criar uma memória compartilhada (shmem) com o GM para compartilhar a Page Table do processo em questão. Assim, a cada vez que a entrada da PageTable indicar que um indice de página i não está com um page frame, em vez de imprimir o endereço físico, o procedimento `trans()` do processo Px deverá emitir um sinal SIGUSR1 para o GM avisando que falta uma página, ou seja, simulando uma interrupção page-fault gerada por uma MMU.
A partir desse ponto, o GM vai:

- dar um SIGSTOP no processo requisitante Px;
- registrar o período de "suspensão" do processo Px, como sendo 1 ou 2 segundos (vide acima)
- executar o algoritmo de substituição de páginas para escolher e substituir uma page frame da memória física, a ser alocada para i.
- Assim que o conteúdo do frame escolhido estiver devidamente trocado (vide acima), irá atualizar as Page Tables (nas memórias compartilhadas) dos processos correspondentes e enviar um SIGUSR2 para o processo que perdeu um frame.
- E fazer o SIGCONT no processo que causou a falta de página.

4- Obtendo o indice de página a partir do endereço lógico

Cada linha do arquivo (.log) conterá um endereço de memória acessado (em representação hexa-decimal), seguido das letras R ou W, indicando um acesso de leitura ou escrita, respectivamente. Por exemplo:

```
0044e4f8 R
0044e500 R
```

```
0044e520 R
0700ff10 R
2f6965a0 W
0044e5c0 R
00062368 R
```

A leitura de cada linha do arquivo poderá ser feita usando o procedimento `da stdio: fscanf(file, "%x %c ", &addr, &rw)`, onde o tipo de `addr` é `unsigned`. Para se obter o índice da página que corresponde a um endereçp lógico, `addr`, basta descartar os *s* bits menos significativos, ou seja, executar um shift para a direita: **`page = addr >> s`**. Por exemplo, se o tamanho da página/frame for 64KB, obtém-se o índice da página com a instrução: **`page = addr >> 16`**.

3- Visão Geral

A Figura 1, resume o funcionamento e a interação entre o Gerente de Memória – GM (o processo pai) e P1, P2, P3, e P4 (os seus quatro processos filhos). Essa interação acontece através da memória compartilhada, onde a tabela de página de cada processo é compartilhada, e através dos sinais SIGSTOP e SIGCONT, SIGUSR1 e SIGUSR2.

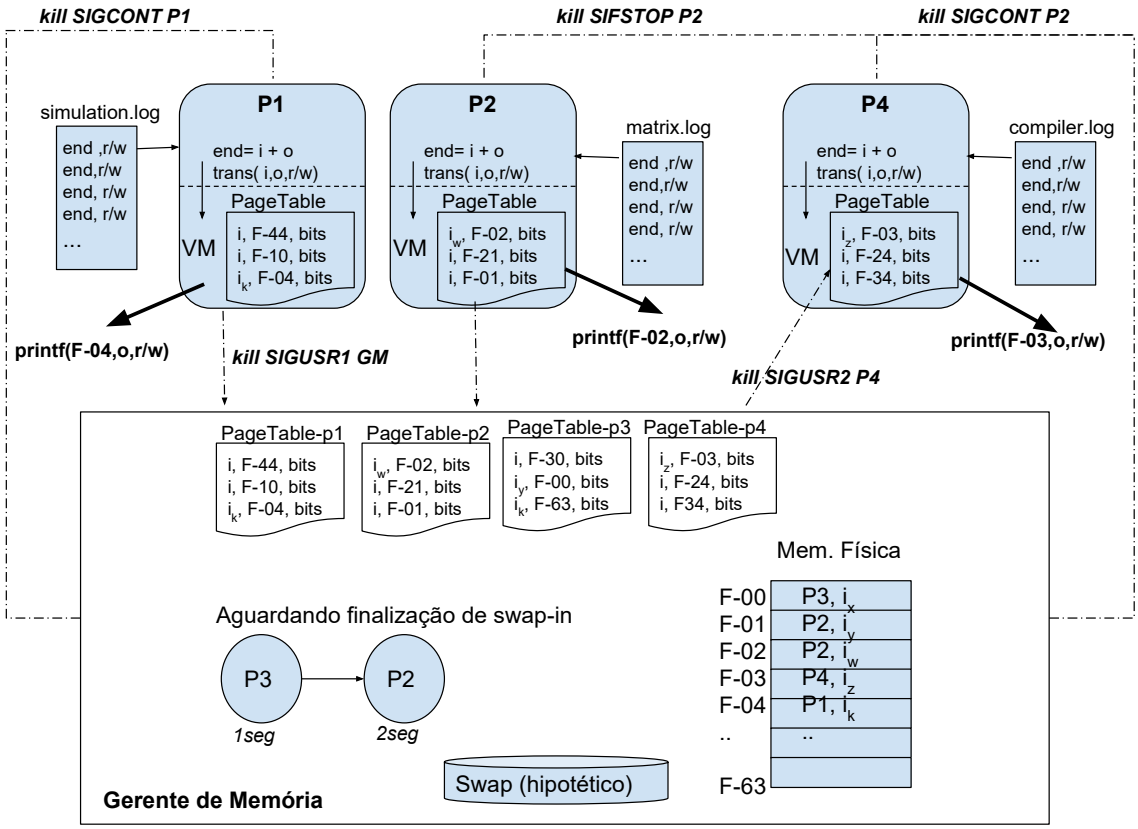

Figura 1: Interação entre processo GM e os user processes P1,..,P4

Os primeiros dois sinais permitem ao GM controlar a execução de um processo Px que tenha causado um page fault, pois esse terá que esperar 1 ou 2 segundos, para simular a espera pelo acesso ao swap em disco. E os sinais SIGUSR1 e SIGUSR2 são usados para avisar que um mapeamento (end.virtual -> end.físico) não está na Tabela de Página do processo correspondente, e que uma entrada da Tabela de página foi modificada em um processo que "perdeu a sua page frame", respectivamente.
Note que quando a página (i,o) está mapeado na memória a biblioteca VM não precisa interagir com o GM e pode imprimir o endereço físico diretamente (p.ex. `printf(p1,F-04,o,r/w)`)

4- Implementação da Substituição de páginas

O algoritmo de substituição de páginas só entra em ação quando todos os page frames estiverem ocupados (o que geralmente é o caso), e se houver um Page-fault, ou seja, uma nova página precisar ser carregada na memória física., o que também ocorre muito frequentemente, especialmente se a memória física possui apenas 256 frames (de 2^{16} bytes). Nesse caso, a página contida no quadro correspondente será: (a) sobre-escrita, se só tiver acessada para leitura, ou (b) se tiver sido escrita, precisará ser reescrita de volta a área de swap.

Nesse trabalho você deve implementar o algoritmo Least Frequently Used (LFU) global, ou seja, que permite que uma página de um processo Px seja trocada (em um page frame) por uma página de outro processo, Py. E o critério para a escolha do frame a ser substituído deverá ser o menor número de acessos à página (e ao frame) em um período recente de tempo. E em caso de números iguais de acesso, escolha o frame que não contém uma página com bit M (de modificado).

Voce deve analisar (e comparar) a eficiência das susbstituições de paginas para três variantes do algoritmo LFU com três diferentes períodos de tempo: por exemplo: acessos nos últimos 30 ms, nos últimos 90 ms, e nos 120 ms (o tempo pode ser simulado através da contagem de acessos às páginas -> x, 3x e 4x acessos). E as três métricas de desempenho serão: duração da execução da simulação, número de ocorrências de Page-faults e número de ocorrências de escrita (simulada) de uma página modificada em um frame para a área de swap.

Obs: Como alternativa ao uso da memória compartilhada e de sinais voce também pode tentar usar FIFO ou MessageQueues, mas nesse caso voce terá que manter (sincronizadas) duas cópias da Tabela de Páginas de cada processo, no processo (biblioteca VM) e no Gerente de Memória, o que é mais custoso em termos de recursos e que parece ser mais complexo. Ou seja, essa alternativa é por sua conta e risco.

Entregáveis

Você deve entregar o(s) código(s) fonte do simulador (.c e .h), e um relatório sobre o seu trabalho. Não inclua os arquivos.log no kit de entrega.

O relatório deve conter:

- Um resumo do projeto: alguns parágrafos que descrevam a estrutura geral do seu simulador e as estruturas de dados relevantes (para implementação de cada algoritmo de substituição).
- Uma análise do desempenho do algoritmo de substituição de páginas LFU para os 4 “programas” utilizados, comparando o número de page-faults e de escritas de página sujas no swap, para diferentes tempos de análise de acessos às páginas.

Esse relatório com a análise de desempenho é uma parte importante do trabalho e terá um peso significativo sobre a nota do trabalho.

Observações finais

- Os trabalhos podem ser feitos em dupla. Indique claramente os integrantes do grupo no cabeçalho do simulador e no relatório.
- Os grupos poderão ser chamados para apresentações orais/ demonstrações dos trabalhos.