Alunos: Wellington Bezerra da Silva (1413383)
Guilherme de Azevedo Pereira Marques (1220676)

Introdução

Nesse trabalho, será simulado uma memória virtual e substituição de páginas utilizando o algoritmo LFU (Least Frequently Used) - página menos referenciada é retirada da memória física.

Desenvolvimento

O trabalho é composto por 5 arquivos: semaphore.c, semaphore.h, gm.c, gm.h, vm.c, vm.h

Primeiramente, cada variável contadora de Pagefaults associada a cada um dos 4 processos serão inicializadas com 0.

Foram feitos forks() para criar processos. Um processo pai será responsável em ser o Gerenciador de Memória(GM) e os outros 4 processos filho(P1, P2, P3 eP4) ficarão responsáveis em ler de um arquivo os endereços da memória virtual e também um caracter que dirá se será feito um acesso de leitura ou um acesso de escrita. Cada processo vai ler um arquivo:

Processo 4 lerá os dados do arquivo “simulador.log”
Processo 3 lerá os dados do arquivo “compressor.log”
Processo 2 lerá os dados do arquivo “matriz.log”
Processo 1 lerá os dados do arquivo “compilador.log”

Cada linha lida do arquivo contém um endereço de 32 bits(8 caracteres) que deverá ser dividido em duas partes, pois os algarismos mais significativos representam o índice da página e os algarismos menos significativos representam o offset da página. Após o término da leitura dos endereços, o processo é terminado.

A memória física utilizada nesse trabalho tem 256 Pages Frames que estão compartilhados com 4 processos. É de responsabilidade do Gerenciador de Memória transformar cada índice para uma Page Frame da Memória Física.

O mapeamento será feito pela tabela de páginas que está presente em cada um dos processos. Para isso, será utilizada a função trans() que irá efetuar a tradução de um endereço virtual lógico para um frame físico da memória.

Foi necessário a utilização de um semáforo para gerenciar o acesso à região crítica do programa. Com o gerenciamento dessa região pelo Gerenciador de Memória, somente uma PageFault por vez seria processada. A proteção da região foi importante pois vários sinais em diferentes momentos poderiam ser recebidos indicando uma PageFault que havia ocorrido. E, caso um sinal fosse recebido e logo após um outro estivesse sido enviado, esse último seria descartado, não atendendo o processo que tivesse sinalizado um PageFault por último.

Se a entrada da PageTable indicar que um índice de página i está com um page frame, o endereço físico deverá ser mostrado, caso contrário:

- o número de PageFaults é incrementado
- os atributos da página acessada são atualizados: id do processo responsável, tipo do

acesso, numero de acessos igual a zero, endereço físico igual a -1 porque ele não está mais na memória principal.

Depois que a página é atualizada, ela é inserida na Tabela de Páginas. Assim, o processo envia um sinal SIGUSR1 para GM indicando que o índice da página i não está com uma pageFrame. Logo após o envio desse sinal, a função sighandler() será chamada. Ainda nessa função, depois de identificar o processo responsável pelo endereço, a função allocatePage() será invocada. O **tabela** e **tabela2** serão inicializados:

- tabela: área de memória do processo que chamou o allocatepage por emio do sinal 1
- tabela2**:** memória compartilhada do processo que vai perder o frame

Assim, se a memória estiver cheia, busca-se o endereço menos acessado na memória física para que ele seja retirado da mesma utilizando o algoritmo LFU. Caso contrário, o endereço físico deixa de ser -1, pois ele será calculado e colocado na memória física.

Se um processo for morto, todas as páginas que estavam em memória de um processo serão removidas.

Além disso, existe um contador que quando chega em um valor determinado, todas as métricas de desempenho são impressas e o programa é finalizado.

Conclusões:

- No início da execução, ocorre Pagefault, pois a página ainda não foi referenciada na memória principal(MP). Assim que uma página já foi referenciada na MP, seu endereço é mostrado na tela.

- Foi observado que a cada vez que o projeto é executado, os processos estão em concorrência e os endereços virtuais vão sendo transformados em endereços físicos a medida que os processos vão sendo executados. O processo selecionado é randômico.

- Foi utilizado o sinal SIGACTION para saber qual processo estava mandando o sinal

Observações:

- Para verificar o número de **pagefaults** e a quantidades de **escritas**, foi necessário definir para a memória um tamanho de 10 frames.

- Para identificar se uma página está referenciada na memória física, foi adicionado um atributo **r** na estrutura **Pagina.**

- SIGUSR2 - Sinal enviado para o processo que perdeu o frame
- SIGUSR1 - Sinal enviado de um processo para a GM quando entrada da PageTable indicar que um indice de página i não está com um page frame.
- SIGSTOP - Sinal enviado para o processo que mandou um sinal SIGUSR1 para a GM

- **<u>Alterações realizadas:</u>**

- Segue abaixo as correções feitas nessa nova versão do trabalho para que o

mesmo pudesse atender todas as exigências:

- **1) Indicação dos pagefaults corretamente**
- **2) Descrever as estruturas de dados**
- **3) Simulação do Swap**
- **4) Métricas de performance**

1) Indicação dos pagefaults corretamente

Anteriormente, o número do frame e o offset não eram indicados. Nessa versão, isso foi corrigido e os valores podem ser vistos assim que o programa é executado.

Foi consertado um erro que acontecia ao longo da execução pois eram mostradas as páginas 0,1,2,3 e depois 1,2,3,4.

2) Descrever as estruturas de dados

2.1) Página

A página é uma estrutura que será colocada e retirada da memória principal. Ela contém os seguintes elementos:

- r_w // identifica se a página é para ser escrita ou lida
- id_processo // identificador do processo
- contador_acesso //contador de acesso a página
- r // byte referenciado
- m // byte modificado
- p // vê se a página está presente na memória principal
- end_fisico // endereço físico

2.2) Tabela de Páginas

A função **criaTabelaPáginas()** cria, para cada processo, um **vetor(memória compartilhada)** que representará a Tabela de Páginas. No início da execução do programa, todas as posições desse vetor estarão devidamente inicializadas.

2.3) Memória Principal

Simulada por um vetor de inteiros onde cada elemento representa uma página presente na memória**.(memoria_principal)**

Existe também um outro vetor para auxiliar qual é o processo filho que se refere ao elemento que ta na memória. **(memorial_principal_proc)**

3) Simulação do Swap

A simulação do Swap já estava sendo realizada na função **alocaPag(),** a partir da

l**inha 133** doa arquivo **gm.c**

4) Métricas de performance

Para obter um valor satisfatório, 10 execuções foram realizadas tirando-se a média para cada valor de x, 3x e 4x. Além disso, foi necessário reduzir o tamanho da memória principal para 10 frames para que o teste pudesse ser feito de forma menos trabalhosa.

Para x = 5:

4.1) Acesso as páginas = 5

*** Numero de Page Faults = 108 ***
*** Numero de escritas = 23 ***
*** Tempo de execucao da simulacao = 120 s**

4.2) Acesso as páginas = 15

*** Numero de Page Faults = 101 ***
*** Numero de escritas = 37 ***
*** Tempo de execucao da simulacao = 120 s ***

4.3) Acesso as páginas = 20

*** Numero de Page Faults = 95 ***
*** Numero de escritas = 33 ***
*** Tempo de execucao da simulacao = 120 s ***

Observamos que ao aumentar a contagem de acesso as páginas, o número de escritas e o número de pagefaults aumentaram. Acreditamos que por limitarmos o tamanho do frame da memória a 10, isso pode ter influenciado um pouco nos resultados que obtivemos, pois maior o tamanho da memória, mais preciso seria o resultado do teste. O número de escritas variou pois os processos concorrem e cada um é responsável em ler um endereço virtual do arquivo e o seu tipo(escrita/leitura). Já o número de pagefaults foi reduzido como o esperado, pois como a contagem de acesso simula o tempo, logo a página que tem acesso mais frequente pode ficar mais tempo na memória principal.