PUCRS - Escola Politécnica
Disciplina: Sistemas Operacionais - 2021/1 - Trabalho Prático - Fase 4
Prof. Fernando Luís Dotti

Nesta fase do trabalho iniciamos a representação de múltiplos processos no sistema. Poderemos criar e hospedar diversos processos em memória. Mas cada um executará em sua vez até o término. Este é um passo simplificador para construir o GM (gerente de memória) e a representação de processos (PCBs). [Na próxima fase adicionaremos um escalonador.]

1. Gerente de Memória

1.1 Valores Básicos

O gerente de memória para a VM implementa paginação, com:
Tamanho de Página = 16 palavras (ou posições de memória)

A memória tem 1024 palavras. Assim teremos:
Número de frames = 64

1.2 Funcionalidades do Gerente

Alocação: Dada uma demanda em número de palavras, o gerente deve responder se a alocação é possível e, caso seja, retornar o conjunto de frames alocados : um array de inteiros com os índices dos frames.

Desalocação: Dado um array de inteiros com as páginas de um processo, o gerente desloca as páginas.

Interface - solicita-se a definição clara de uma interface para o gerente de memória. Exemplo:

```
GM {
    Boolean   aloca(IN int nroPalavras,     OUT pags []int)
              // retorna true se consegue alocar, vetor tem os frames alocados
    Void desaloca(IN  pass []int)
              // simplesmente libera os frames alocados
  }
```

Estruturas internas: controle de páginas alocadas e disponíveis.

```
tamMemoria = 1024                          // fornecido anteriormente
array mem[tamMemoria] of posicaoDeMemoria
tamPag = tamFrame = 16
nroFrames = tamMemoria / tamPag            // 64 = 1024/16
array frameLivre[nroFrames] of boolean     // inicialmente true para todos frames
```

Os frames terão índices.
Cada frame com índice f inicia em (f)*tamFrame e termina em (f+1)*tamFrame -1
Exemplo:

frame	início	fim
0	0	15
1	16	31
2	32	47
3	48	63
4	64	…
	…	
62	992	1007
63	1008	1023

2. Gerência de Processos

2.1 Alocação de Processos, Carga

A instanciação de um programa para sua execução se dá na forma de um processo. Se solicitamos que um programa seja executado duas vezes, são criados dois processos distintos.

Ao criar um processo, deve-se alocar os frames necessários. Para tal, necessita-se saber o tamanho do programa. Podemos assumir que a informação de tamanho do programa é fornecida juntamente com o seu código (a sua definição). Gerente de memória aloca frames para conter o mesmo. O retorno é um array de índices de frames.

A partir disso, o programa é carregado na memória de forma a preencher os frames sequencialmente, na ordem indicada no array de alocação. Desta forma, o array de alocação de frames é também a tabela de páginas do processo. Note que o programa carregado em memória tem exatamente o mesmo formato do programa das Fases anteriores, que era alocado integralmente a partir da posição 0 de memória.

Exemplo: considere um programa de 5 páginas, que teve alocados os seguintes frames, em ordem
[2, 3, 8, 12, 10]
Chamemos agora este mesmo array de tabela de páginas do processo:
tabPaginas = [2, 3, 8, 12, 10]
assim, tabPaginas[0]=2
tabPaginas[1]=3
…
tabPaginas[4]=10
seja p uma página do processo, tabPaginas[p] é o frame onde a página p está alocada.

2.2. Estruturas de representação

Temos agora diversos processos em memória. Faz-se necessário representar cada processo existente no sistema. Assim, é necessária a criação de uma estrutura de process control block (PCB).

O PCB contém todas informações relevantes para a execução do processo. Neste instante, o PCB de um processo deve conter a tabela de páginas do mesmo, de forma a saber onde o processo se encontra. Também sugere-se que cada processo tenha um identificador (valor inteiro) para diferenciar os processos. A medida que são criados processos, este identificador é incrementado e alocado ao processo.

O sistema deve então ter uma lista de processos criados e carregados em memória.

2.3. Execução de um processo

Para executar um processo, deve-se saber onde seu código se encontra. Isto é dado pela sua tabela de páginas. Assim, a tabelas de páginas do processo é utilizada durante a execução do processo. O código do programa está escrito utilizando-se uma noção abstrata de endereçamento, chamado endereçamento lógico. Lembre-se que no seu programa você utiliza endereços relativos, considerando alocação contígua do programa, a partir da posição 0. Este é o endereçamento lógico. Ao acessar a memória de fato, cada endereço lógico deve ser transladado para o físico, para que então a posição específica da memória seja acessada.

Seja *div* a operação de divisão inteira e *mod* a operação que obtém o resto da divisão inteira, cada acesso à memória passa pela translação:

seja *A* o endereço a ser acessado, temos:
p *= A div tamPag* é o índice da página que o programa acessa
offset *= A mod tamPag* é o deslocamento dentro da página
toma-se *p* e tem-se que:
tabPaginas[p] é o frame onde a página p se encontra
*tabPaginas[p] * tamFrame* é o início do frame a ser acessado
assim, *(tabPaginas[p] * tamFrame) + offset* é o endereço a ser acessado na memória física
ou, da mesma forma, substituindo:
*(tabPaginas [(A div tamPag)] * tamFrame) + (A mod tamPag)*
Pode-se montar uma função de tradução de endereço T que, dado A endereço lógico do programa e a tabela de páginas do mesmo, Traduz A para o endereço que deve ser acessado na memória.
*T(A) = (tabPaginas [(A div tamPag)] * tamFrame) + (A mod tamPag)*

No processo de tradução deve ser verificado se a página resultante é válida, gerando erro ou permitindo o acesso. A página é válida se o programa tem o índice p da página obtida em A div tamPag = p.

3. Testes

Deve-se demonstrar que o sistema pode carregar vários processos em memória. E depois executar cada um sequencialmente, do início ao fim. Deve-se provocar que processos utilizem frames não vizinhos na memória para testar adequadamente a carga e o endereçamento dos processos.

Deve-se criar formas de ver o conteúdo da memória e os PCBs existentes.