PUCRS - Escola Politécnica
Disciplina: Sistemas Operacionais - 2022/2 - Trabalho Prático
Prof. Fernando Luís Dotti

Gerente de Memória e Gerente de Processos - Definições

1.Gerente de Memória

Nesta fase do trabalho criaremos um gerente de memória que oferece **partições fixas**.

1.1 Valores Básicos

O gerente de memória para a VM implementa **partições fixas**, onde:
A memória tem M palavras.
O tamanho de cada Partição = P palavras (ou posições de memória).
Assim, M / P é o número de partições da memória.

O sistema deve funcionar para diferentes valores escolhidos de M e P. Por exemplo, se M=1024 e P = 64, temos 16 partições. Istto deve ser facilmente configurável e poderemos testar o sistema com tamanhos diferentes de memória de partições.

1.2 Funcionalidades do Gerente de Memória

Alocação: Dada uma demanda em número de palavras, o gerente deve responder se a alocação é possível e, caso seja, retornar a partição alocada.
No nosso sistema, para carregar um programa, deve-se alocar toda memória necessária para **código e dados.**

Desalocação: Dada a partição alocada, o gerente desloca a mesma.

Sugestão de interface - solicita-se a definição clara de uma interface para o gerente de memória.

Exemplo:

```
GM {
        Boolean   aloca(IN   int nroPalavras,   OUT partição int)
           // retorna true se consegue alocar ou falso caso negativo
        Void desaloca(IN  partição int)
           // simplesmente libera a partição
    }
```

Estruturas internas: controle de partições alocadas e disponíveis.

```
tamMemoria = M      (vide 1.1)
array mem[tamMemoria] of posicaoDeMemoria
tamPartição = P     (vide 1.1)
nroPartições = tamMemoria / tamPartição

array frameLivre[nroPartições] of boolean          // inicialmente true para todos frames
```

As partições são índices.
Cada partição com índice p inicia em (p)*tamPartição e termina em (p+1)*tamPartição -1
Exemplo para tamPartição = 64:

partição	início	fim
0	0	63
1	64	127
2	128	191
3	192	256
4	…	
…		

1.3 Carga

Nossos programas não serão alterados. Após o GM alocar a partição, deve-se proceder a carga do programa para a partição alocada.

1.4 Tradução de Endereço e Proteção de Memória

Durante a execução do programa, todo acesso à memória é **traduzido** para a posição devida, conforme o esquema de particionamento. *Lembre-se que no seu programa você utiliza endereços lógicos, considerando a abstração de que o*

programa está disposto contiguamente na memória, a partir da posição 0. E isto não será alterado. A memória física é um array de posições de memória. O endereço físico é um valor de 0 ao tamanho da memória. Ao acessar a memória física, cada endereço lógico deve ser transladado para o físico, para que então a posição específica da memória seja acessada.

Pode-se montar uma *função de tradução de endereço T* que, dado *A* endereço lógico do programa e a partição do programa, traduz A para o endereço físico que deve ser acessado na memória.

*T(A) = p * tamPartição + A*

Em um sistema real esta função é implementada em HW. *Todo* acesso à memória, seja durante *fetch*, *stores* ou *loads*, passam por esta função.

Ainda, para **proteção**, com relação ao endereço lógico A deve-se garantir que o mesmo referencia a partição.
Ou seja: *0* <= A < *tamPartição*

1.5 Demonstração de funcionamento

vide mesmo item em Gerente de Processos

2. Gerente de Processos

2.1 Funcionalidade

O GP é um módulo do SO e é responsável por

- Criar um processo, dado um programa passado como parâmetro.

  boolean criaProcesso(programa)

  verifica tamanho do programa
  pede alocação de memória ao Gerente de Memória
  se nao tem memória, retorna negativo
  Cria PCB
  Seta partição usada no pcb
  Carrega o programa
  Seta demais parâmetros do PCB (id, pc=0, etc)
  Coloca PCB na fila de prontos
  Retorna true
- Desaloca um processo

  desalocaProcesso (id)

  desaloca toda memória do processo com id
  retira de qualquer fila que esteja
  desaloca pcb

2.2 Estruturas, filas, processo rodando

Nesta fase de evolução do nosso sistema precisamos das seguintes estruturas.

PCB: Você deve criar uma estrutura de descrição do processo com as informações necessárias para a gerência dele no seu sistema. Esta estrutura é o Process Control Block. Todo processo tem um PCB próprio.

Running/rodando: No nosso sistema, apenas um processo está rodando em um determinado momento. Existe como variável do SO um ponteiro *rodando/running* que identifica o PCB do processo executando.

Ready/aptos: Da mesma forma, temos uma lista de processos *aptos/ready* que podem rodar. Trata-se de uma lista de PCBs.

2.3 Funcionamento/Testes

Agora você dispõe de um sistema que pode ter vários processos em memória. Para demonstrar o funcionamento, você deve ter um sistema iterativo: ele fica esperando comandos. A cada comando, o sistema reage e volta a esperar o próximo comando: Os comandos possíveis são:

cria <nomeDePrograma> - cria um processo com memória alocada, PCB, etc. que fica em uma lista de processos. esta chamada retorna um identificador único do processo no sistema (ex.: 1, 2, 3 …)
dump <id> - lista o conteúdo *do PCB e o conteúdo da partição de memória* do processo com id
desaloca <id> - retira o processo id do sistema, tenha ele executado ou não
dumpM <inicio, fim> - lista a memória entre posições início e fim, independente do processo
executa <id> - executa o processo com id fornecido. se não houver processo, retorna erro.
traceOn - liga modo de execução em que CPU print cada instrução executada
traceOff - desliga o modo acima
exit - sai do sistema

Voce pode criar outros nomes para os comandos. Desde que façam o descrito.