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

Troca de Gerente de Memória para Paginação (Trabalho 1B)

1.Gerente de Memória

Nesta fase do trabalho trocaremos o gerente de memória, de partições fixas para **paginação**.

1.1 Valores Básicos

O gerente de memória para a VM implementa **paginação**, onde:

- a memória tem *tamMem* palavras.
- O tamanho de Página = *tamPg* palavras (ou posições de memória).
  Assim, *tamMem / tamPg* é o número de frames da memória.

O sistema deve funcionar para diferentes valores escolhidos de *tamMem e tamPg.*

Se tamMem=1024 e tamPg=8 teremos 128 frames. Testaremos o sistema com diferentes tamanhos de memória e de página.

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 o conjunto de frames alocados : um array de inteiros com os índices dos frames. No nosso sistema, para carregar um programa, deve-se alocar toda memória necessária para **código e dados.**

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

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

```
GM {
    Boolean aloca(IN int nroPalavras, OUT tabelaPaginas []int)
        // retorna true se consegue alocar ou falso caso negativo
        // cada posição i do vetor de saída "tabelaPaginas" informa em que frame a página i deve ser hospedada

    Void desaloca(IN tabelaPaginas []int)
        // simplesmente libera os frames alocados
}
```

Estruturas internas: controle de quadros(frames) alocados e disponíveis.

Os frames terão índices.
Cada frame com índice f inicia em (f)*tamFrame e termina em (f+1)*tamFrame -1
Exemplo para tamFrame = 16 e tamMem 1024:

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

1.3 Carga

ATENÇÃO: os programas escritos para a VM não serão alterados em NADA.
Após o GM alocar frames, devolvendo a *tabelaPaginas*, deve-se proceder a carga. Cada página i do programa deve ser copiada (exatamente como tal) para o frame informado em *tabelaPaginas[i].*

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 paginação. Assim, a **tabela de páginas do processo** é utilizada durante a execução do processo. *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. Deve-se converter o endereço lógico (linear) em página e deslocamento na página, acessar a tabela de pagina, descobrir o frame a ser acessado, calcular o endereço de início do frame e adicionar o mesmo deslocamento na página (pois página e frame tem mesmo tamanho).

1.5 Chamada de Sistema para solicitar memória

Até o momento, nosso sistema operacional tem as chamadas de sistema IN e OUT como definimos. Agora vamos definir duas novas chamadas de sistema. Listamos abaixo as duas existentes para completude e melhor compreensão.

SHMALLOC(int chave); R8 = 3; R9 = chave

Aloca uma nova página lógica ao final do processo que faz a chamada. Equivale a um novo frame da memória alocado ao processo. O mapeamento de página para frame deve entrar na tabela de páginas do processo, Não havendo quadros disponíveis coloca -1 em R9 e retorna. Em caso de sucesso, associa internamente ao quadro alocado a *chave* dada como parâmetro. A partir do retorno ok desta chamada, o processo pode acessar a nova página. Supondo tamanho de página 16 e que o processo tinha 2 páginas, logicamente ele endereçava 0 a 31, agora passa a endereçar 0 a 47.
Esta página poderá ser compartilhada com outro processo. Vide a próxima chamada.

SHMREF(int chave); R8 = 4; R9 = chave

Havendo um quadro associado à *chave* dada, o processo que faz esta chamada passa a ter uma nova página logicamente no final do processo. Esta nova página, quando endereçada, acessa o mesmo frame solicitado com a chamada SHMALLOC(chave).
Vários processos podem solicitar acesso à página, bastando ter a chave para tal.

Existentes:

IN	R8=1	R9= endereço lógico do processo, para armazenar o valor lido
OUT	R8=2	R9= endereço lógico do processo, onde está o valor a ser escrito

Com estas operações poderemos fazer programas concorrentes que acessam mesmas posições de memória!

1.6 Testes e Demonstração

Você deve adaptar os comandos já definidos no trabalho 1A para trabalhar com páginas.
Os comandos possíveis são:

cria <nomeDePrograma> - cria um processo na memória. Pede ao GM para alocar frames de memória necessários. cria PCB, **seta tabela de páginas do processo no PCB**, etc. coloca processo em uma lista de processos (prontos).
Esta chamada retorna um identificador único do processo no sistema (ex.: 1, 2, 3 …)

listaProcessos - lista todos processos existentes

dump <id> - lista o conteúdo *do PCB e o conteúdo* ***das páginas*** *de memória* do processo com id

desaloca <id> - retira o processo id do sistema, tenha ele executado ou não

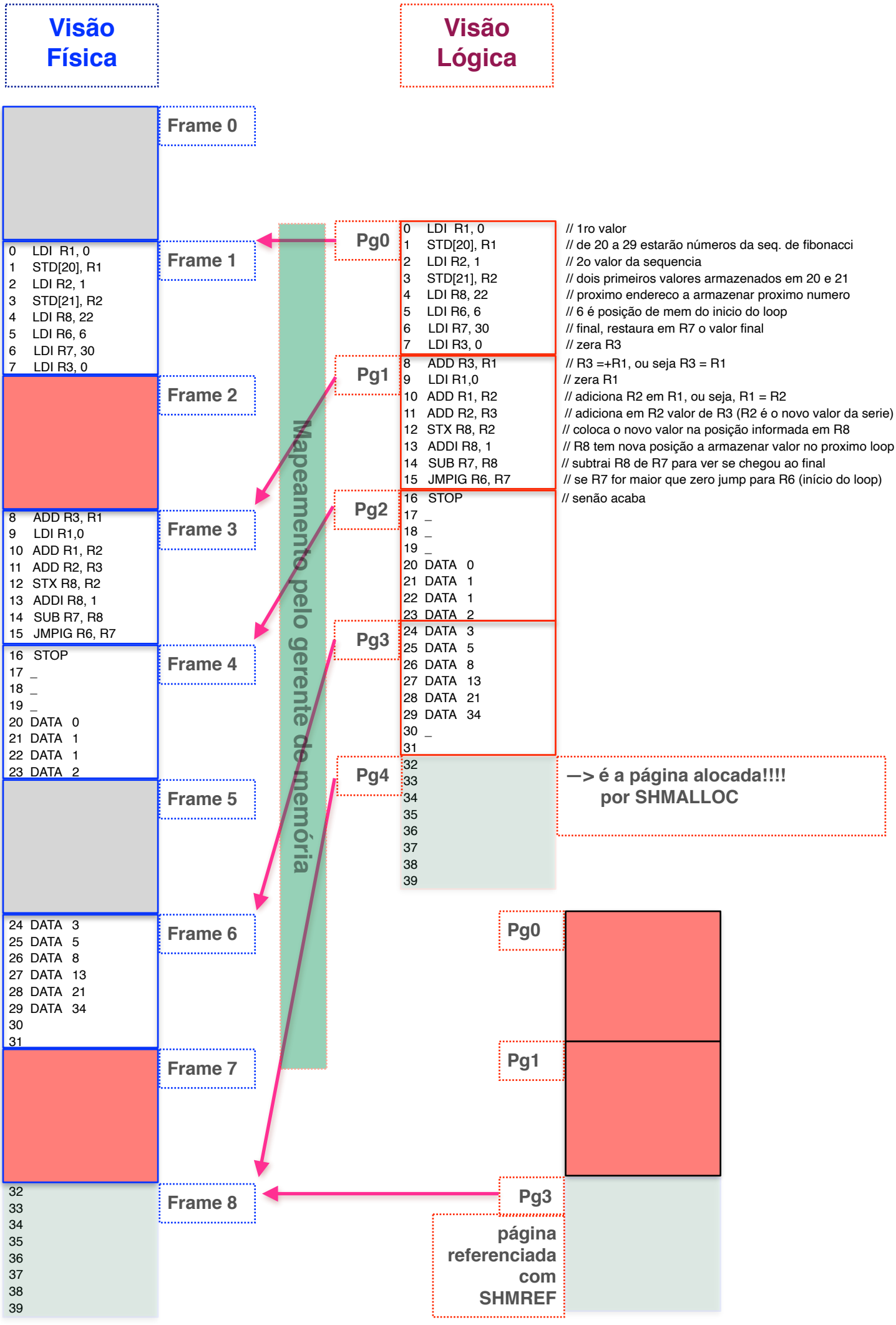
dumpM <inicio, fim> - lista a memória entre posições início e fim, independente do processo

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


Visão Física
Visão Lógica
Frame 0
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Frame 7
Frame 8
0 LDI R1, 0
1 STD[20], R1
2 LDI R2, 1
3 STD[21], R2
4 LDI R8, 22
5 LDI R6, 6
6 LDI R7, 30
7 LDI R3, 0
8 ADD R3, R1
9 LDI R1,0
10 ADD R1, R2
11 ADD R2, R3
12 STX R8, R2
13 ADDI R8, 1
14 SUB R7, R8
15 JMPIG R6, R7
16 STOP
17 _
18 _
19 _
20 DATA 0
21 DATA 1
22 DATA 1
23 DATA 2
24 DATA 3
25 DATA 5
26 DATA 8
27 DATA 13
28 DATA 21
29 DATA 34
30
31
32
33
34
35
36
37
38
39
Mapeamento pelo gerente de memória
Pg0
Pg1
Pg2
Pg3
Pg4
0 LDI R1, 0 // 1ro valor
1 STD[20], R1 // de 20 a 29 estarão números da seq. de fibonacci
2 LDI R2, 1 // 2o valor da sequencia
3 STD[21], R2 // dois primeiros valores armazenados em 20 e 21
4 LDI R8, 22 // proximo endereco a armazenar proximo numero
5 LDI R6, 6 // 6 é posição de mem do inicio do loop
6 LDI R7, 30 // final, restaura em R7 o valor final
7 LDI R3, 0 // zera R3
8 ADD R3, R1 // R3 =+R1, ou seja R3 = R1
9 LDI R1,0 // zera R1
10 ADD R1, R2 // adiciona R2 em R1, ou seja, R1 = R2
11 ADD R2, R3 // adiciona em R2 valor de R3 (R2 é o novo valor da serie)
12 STX R8, R2 // coloca o novo valor na posição informada em R8
13 ADDI R8, 1 // R8 tem nova posição a armazenar valor no proximo loop
14 SUB R7, R8 // subtrai R8 de R7 para ver se chegou ao final
15 JMPIG R6, R7 // se R7 for maior que zero jump para R6 (início do loop)
16 STOP // senão acaba
17 _
18 _
19 _
20 DATA 0
21 DATA 1
22 DATA 1
23 DATA 2
24 DATA 3
25 DATA 5
26 DATA 8
27 DATA 13
28 DATA 21
29 DATA 34
30 _
31
32
33
34
35
36
37
38
39
—> é a página alocada!!!! por SHMALLOC
Pg0
Pg1
Pg3
página referenciada com SHMREF