Projeto da Disciplina de Sistemas Operacionais (5954016)

Prof. Dr. Cléver Ricardo Guareis de Farias

1. Objetivo

Implementar em Java um sistema de gerenciamento de memória paginada. Este sistema deverá ser utilizado para simular a alocação e a liberação de espaços de memória de forma simplificada para processos virtuais.

2. Visão geral

A aplicação será formada por uma interface (gráfica ou linha de comando) com o usuário e pelo gerente de memória. O gerente de memória implementa a interface ManagementInterface (veja arquivo em anexo), a qual é utilizada pelo usuário para interagir com o gerente.

O gerente de memória controla a alocação de um conjunto de quadros de memória para que sejam utilizados por um conjunto de programas virtuais. Estes programas virtuais utilizam um esquema de endereçamento lógico de 10 bits, dos quais 5 são utilizados para a identificação da página e 5 são utilizados para o deslocamento. Assim, cada programa possui no máximo 32 páginas de memória, cada qual com 32 bytes.

O gerente de memória é parametrizado com a quantidade de quadros a serem gerenciados. Esta quantidade é definida por meio do construtor desta classe, sendo múltiplo de 32 (32, 64 ou 128 quadros).

Cada programa virtual possui três diferentes segmentos: texto, dados e pilha. O tamanho do segmento de texto é definido por um número natural maior do que 1 e menor ou igual a 960 (em bytes). O tamanho do segmento de dados é também definido por um número natural maior ou igual a 0 e menor ou igual a 928 (em bytes). O segmento de pilha possui tamanho fixo de 64 bytes. O tamanho máximo de um programa é 1024 bytes.

Uma vez definido o tamanho do segmento de texto, este não muda. Porém, o segmento de dados possui uma parte de tamanho fixo (definida na inicialização do programa na memória) e uma parte de tamanho variável (heap), a qual possui inicialmente tamanho 0, podendo aumentar ou diminuir dependendo da alocação/liberação dinâmica de memória enquanto o programa permanece em execução.

3. Estruturas para o gerenciamento da memória

O gerente de memória utiliza duas estruturas para gerenciar a memória paginada. A primeira estrutura é uma lista de tabelas de páginas, uma para cada processo. A tabela de página irá armazenar para cada entrada a informação se página é válida (bit 1), isto é, uma página logicamente endereçável pelo processo virtual, ou inválida, caso contrário (bit 0). Adicionalmente, caso trate-se de uma página válida a entrada também irá armazenar o endereço base (múltiplo de 32) do quadro de memória alocado para esta página.

A segunda estrutura é um mapa de bits, contendo um bit para cada unidade de alocação (quadro de memória). Um bit 0 indica que o quadro está livre, enquanto um bit 1 indica que o quadro está

ocupado. Um buraco é formado por um conjunto de um ou mais bits 0 contíguos. Neste sentido, o mapa de bits poderá conter múltiplos buracos os quais deverão ser utilizados para atender às necessidades de alocação de quadros de memória.

4. Linguagem para a definição dos programas simulados

Os programas deverão ser definidos em uma linguagem própria, especificada segundo a BNF apresentada na Figura 1. Todo programa contém um cabeçalho contendo a palavra-chave "program" seguido do identificador do programa (identificador deve ter o mesmo nome do arquivo contendo o programa, exceto pela extensão txt). As informações sobre o tamanho dos segmentos de texto e dados vêm a seguir. O segmento que representa a pilha não precisa ser definido explicitamente em um programa.

```
<program> ::= <program_statement> <program_segments>
<program_statement> ::= "program" <whitespace> <file_name> <line_of_end>
<program_segments> ::= <text_segment> <data_segment>
<text_segment> ::= "text" <whitespace> <text_segment_size><line_of_end>
<data_segment> ::= "data" <whitespace> <data_segment_size><line_of_end>
<line_of_end> ::= <EOL>
<whitespace> ::= " "
```

Figura 1. BNF da linguagem de representação de programas simulados

5. Dinâmica de alocação e liberação de páginas/quadros

Quando um programa é carregado para a memória, uma tabela de página para representar o processo é criada. Na sequência, devem ser alocados quadros suficientes para armazenar de forma independente os segmentos de texto e dados, bem como a pilha como múltiplos de 32 bytes (tamanho padrão do quadro). À medida em que estes quadros forem alocados, as entradas na tabela de página para aquele processo devem ser atualizadas.

A busca por um buraco grande o suficiente para atender uma dada necessidade de alocação de quadros deverá ser realizada de acordo com uma dada estratégia de busca (First-Fit, Best-Fit ou Worst-Fit), conforme atribuição definida na seção 7. Se não houver um buraco grande o suficiente para atender uma dada requisição, porém houver quadros suficientes disponíveis em múltiplos buracos, sempre procure pelo maior buraco que possa atender parcialmente a necessidade de alocação e depois repita a estratégia de busca base para alocar os quadros que ainda precisam ser alocados. Este procedimento deve ser repetido até que todos os quadros necessários sejam alocados.

A Figura 2 apresenta um exemplo de um programa definido utilizando a linguagem de definição de programas simulados, enquanto a Figura 3a ilustra um exemplo de alocação de memória para este programa, considerando-se que neste caso um processo possui no máximo 16 páginas de memória (no caso do projeto são 32 páginas). As páginas 0 a 2 são utilizadas para o armazenamento do segmento de texto, enquanto as páginas 3 a 6 são utilizadas para armazenar o segmento de dados (parte estática) com definido na especificação do programa. Finalmente, as páginas 14 e 15 são utilizadas para armazenar a pilha do programa.

```
program prog1
text 82
data 115
```

Figura 2. Exemplo que programa definido na linguagem para a definição de programas simulados.

O segmento de texto deve sempre ser alocado a partir da página 0, enquanto a pilha deve sempre ser alocada nas últimas páginas do processo. No último quadro reservado para o segmento de texto poderá haver fragmentação interna. No caso do processo *prog1*, o último quadro terá 14 bytes não utilizados. Este espaço não deve ser utilizado para qualquer outra coisa.

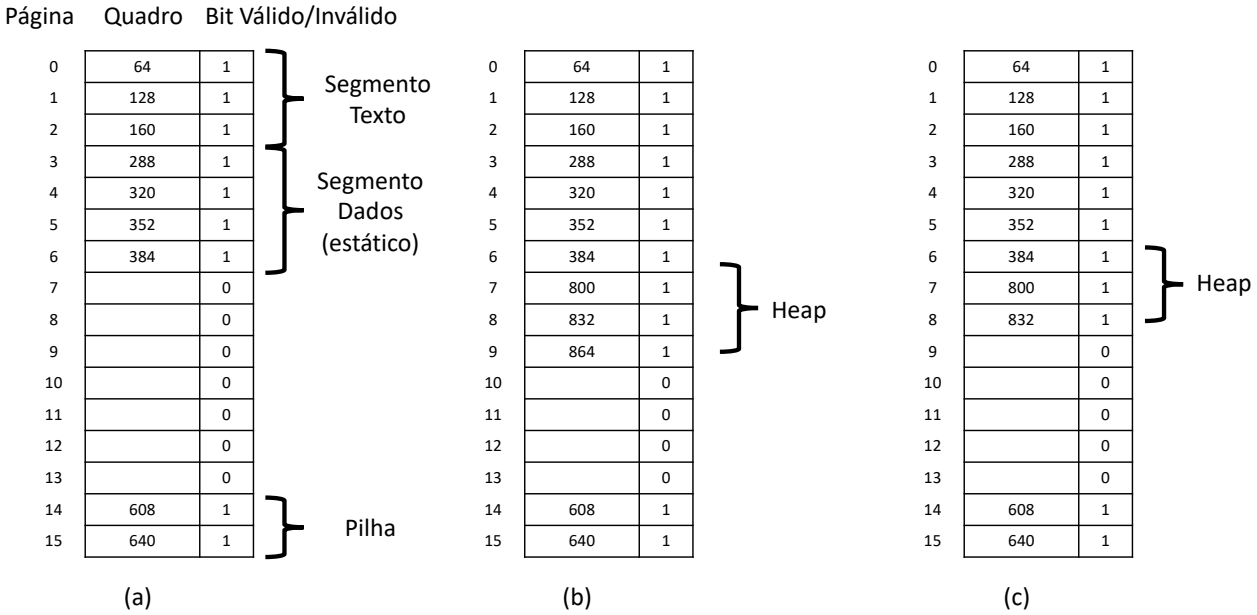

Figura 3. Exemplo alocação de memória

As páginas alocadas para o segmento de dados seguem o segmento de texto. Primeiramente, devem ser alocadas páginas para a parte estática do segmento de dados, seguidas das páginas que formam o *heap* (inicialmente vazias). Caso a parte estática do segmento de dados seja um múltiplo de 32, isto é, consuma por completo um quadro de memória, o *heap* necessariamente deverá começar em uma nova página (quadro) de memória. Porém, caso o último quadro utilizado para armazenar a parte estática do segmento de dados não seja utilizado completamente, a área de alocação dinâmica de memória inicia-se logo após o último byte alocado para a parte estática do segmento de dados. Por exemplo, considere a alocação para o processo *prog1*. Neste exemplo, foram alocadas 4 páginas (quadros) para o segmento de dados perfazendo 128 bytes, dos quais 115 bytes já foram utilizados para armazenamento de dados declarados virtualmente no programa, restam 13 bytes que serão utilizados para o *heap*.

A alocação e a liberação da memória dinâmica para um processo serão realizadas de forma simplificada segundo uma dinâmica Last In – First Out. À medida em que novas solicitações forem realizadas e eventualmente não puderem ser atendidas com o espaço de memória atualmente alocado, um novo quadro deve ser alocado e a entrada definida na tabela de página do processo

virtual. A liberação de um espaço de memória ocorre de forma simplificada sempre do final do h*eap* para o início. Quando uma página que estava sendo utilizada pelo *heap* for liberada, o quadro correspondente deve também ser liberado e a página no processo marcada como inválida.

A Figura 3b ilustra um cenário após a alocação dinâmica de 100 bytes de memória para o programa. De modo a permitir a alocação desta memória, foram alocados três quadros e atribuídos às páginas 7 a 9 do processo. Neste caso, foram utilizados 13 bytes da página 6 (sobra da parte estática), 32 bytes da página 7, 32 bytes da página 8 e 23 bytes da página 9 (13 + 32 + 32 + 23 = 100). Já a Figura 3c ilustra um cenário após liberação de 50 bytes do *heap* do processo. Com esta liberação, foram liberados os 23 bytes referentes à página 9 e 27 bytes referentes à página 8. Como toda a memória que estava sendo utilizada pela página 9 foi liberada, a entrada da própria página 9 deve ser excluída da tabela de página do processo, sendo esta entrada marcada como inválida.

6. Compartilhamento do segmento de texto

Dois ou mais processos criados a partir de um mesmo programa devem necessariamente compartilhar o segmento de texto. Dois programas serão considerados idênticos se estes possuírem o mesmo nome e os mesmos tamanhos dos segmentos de texto e dados.

A Figura 4 ilustra um exemplo do compartilhamento do segmento de texto causado pelo carregamento na memória do programa *prog1*. A Figura 4a ilustra a tabela de página gerada pelo primeiro carregamento deste programa, enquanto a Figura 4b ilustra a tabela de página gerada pelo segundo carregamento deste programa. Neste caso, há apenas o compartilhamento do segmento de texto entre esses processos. Os demais segmentos não são compartilhados.

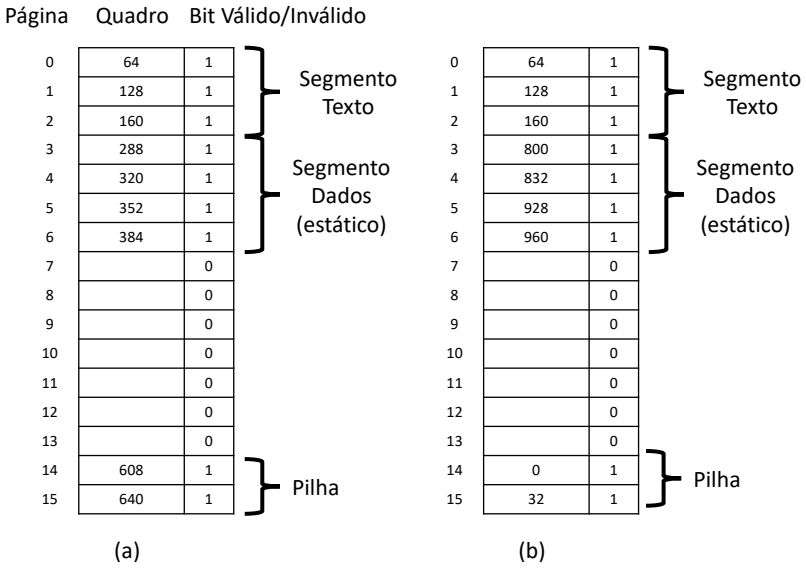

Figura 4. Compartilhamento do segmento de código por dois processos

7. Observações Finais

As seguintes estratégias de busca de quadros livres devem ser implementadas:

- FF: First-Fit

- BF: Best-Fit
- WF: Worst-Fit

A Tabela 1 apresenta as atribuições de cada grupo. Cada grupo deverá implementar uma única estratégia de busca.

Tabela 1. Definição de Grupos

Grupo	Integrantes	Estratégia de Busca
1	Guilherme Martiniano de Oliveira Gustavo Fernandes Carneiro de Castro Renan Gomes Carneiro de Freitas Tiago Costa Carvalho	FF
2	Luan Cesar Cardoso Lucas Freitas Pinto Ferreira Matheo Bellini Marumo Matheus Oliveira Ribeiro da Silva	BF
3	Gabriel Bernardes Ribeiro Luiz Felipe Triques Moraes Mateus Miquelino da Silva Yuri Schwab	WF
4	Milan Rufini de Andrade Renata Rona Garib Sabrina Kappann da Silva	FF
5	Camila Gaio	BF
6	Aleff Corrêa Matos André Bernardes Turcato Gabriel Sartoretto Gabriel Silva Sales	WF
7	Gabriela Marcelino Pereira de Souza Heitor de Paiva Boccato José Vitor Pereira Garzon Lucas de Almeida Louzada	FF
8	Matheus Carlos de Oliveira Carvalho Pietro Pugliese Longui Tiago César Miguel Kapp	BF
9	Bruno Felipe da Silva Araujo Magalhaes Bruno Pandini Cabete Jessica da Paixao Melo Marcelo Ribeiro Ramos	WF

Eventuais alterações nos grupos poderão ser realizadas até o dia 14/07, devendo estas ser encaminhadas por e-mail (farias@ffclrp.usp.br). As alterações serão confirmadas apenas após a manifestação do professor.

O projeto deverá ser entregue até o dia 29 de julho. Além dos aspectos funcionais do projeto, serão também avaliadas a estrutura, documentação e usabilidade da aplicação desenvolvida.