Estrutura de Dados e Algoritmos - Turma A
Prof.: Fernando W. Cruz

1º Projeto de pesquisa

Igor Ribeiro Barbosa Duarte - Mat.: 14/0098275
Keli Cristina Borges - Mat.: 11/0126416
Vinícius Pinheiro da Silva Correa- Mat.: 14/0066543

Universidade de Brasília - UnB
Faculdade Gama - FGA
Brasília, DF - 2014

Sumário

1. Introdução

Este trabalho trata-se do primeiro projeto de pesquisa da disciplina de EDA. As próximas seções irão descrever o problema, o objetivo e como este trabalho foi realizado. A estrutura do código criado para o programa também será detalhado.

2. Descrição do Problema

Nesta seção estão descritos conceitos importantes para melhor compreensão do problema e do objetivo deste trabalho.

2.1. Gerenciamento de memória com mapas de bits

O Sistema Operacional dever gerenciar memorias que são atribuídas dinamicamente. Segundo Tanenbaum (2000), de uma maneira geral existem duas maneiras de monitorar uso de memória: mapa de bits e listas livres. Neste trabalho não será tratado sobre mapa de bits, mas sim listas livres ou listas encadeadas.

A. Listas Encadeadas

Uma forma de realizar o monitoramento da memória é manter uma lista encadeada dos segmentos de memórias alocadas e livres, onde um segmento pode ser um processo ou uma lacuna entre dois processos.

A figura a seguir representa uma lista encadeada de segmentos, onde cada entrada da lista especifica um processo (P) ou uma lacuna (H):

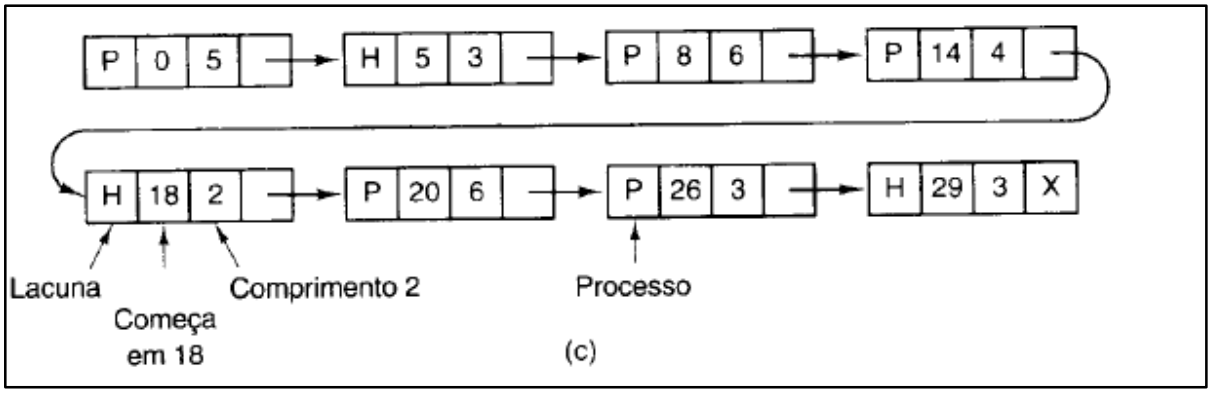


Figura 01: Parte da memória como uma lista (fonte: Tanenbaum(2000))

Tomando como base a Figura 01, temos que a lista está classificada por endereço, facilitando a atualização da lista. Um processo que termina poderá ter normalmente dois vizinhos, que podem ser processos ou lacunas. As combinações possíveis podem ser visualizadas na figura a seguir:

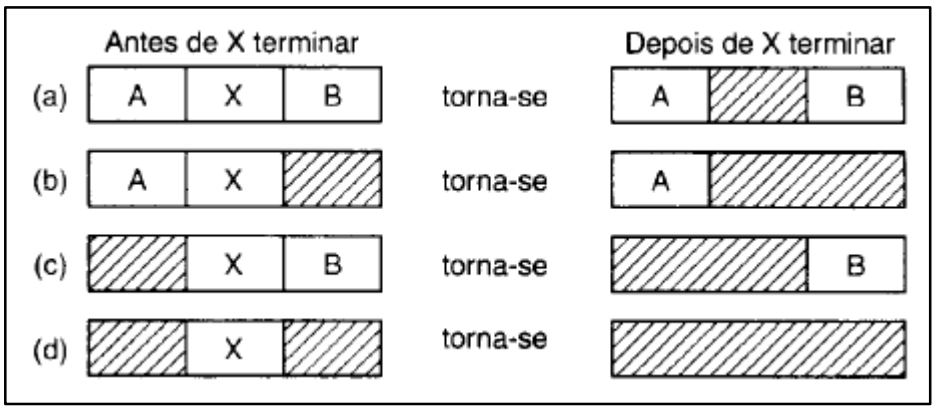


Figura 4: Combinações de vizinho para o processo terminante X (fonte: Tanenbaum (2000))

Levando em consideração que a entrada na tabela de processos referente ao processo que está terminado normalmente ira apontar para a a entrada da lista do próprio processo, Tanenbaum(2000) sugere que seja mais conveniente que se utilize uma lista duplamente encadeada, facilitando a localização da entrada anterior e verificar se uma união pode ser possível ou não.

Com uma lista por endereço é possível utilizar vários algoritmos para alocar memória para um processo recentemente criado ou trocado para a memória. O algoritmo do **primeiro ajuste** é considerado o mais simples, pois o gerenciador de memória percorre toda a lista até localizar uma lacuna que seja suficientemente grande para que possa ser dividida em duas partes: uma parte para o processo e outra para a memória não utilizada. Além deste, existem vários outros algoritmos bem conhecidos como: próximo ajuste, melhor ajuste, pior ajuste, ajuste rápido, etc. Tanenbaum(2000). Estes algoritmos podem ser utilizados para gerência de memória.

2.2. Requisitos para o projeto

A solução computacional para este projeto deve considerar:

a. O controle da memória deve ser feito com estruturas de dados dinâmicos com uso de cabeçalhos;
b. O código deve ser escrito em linguagem C em ambiente Linux/gcc;
c. O código deve ser modularizado na forma de funções devidamente documentadas para facilitar a compreensão do texo (a organização do código - na forma de módulos ou funções bem definidas - será levada em consideração para a mensão dada ao projeto)
d. O programa deve permitir que o usuário informe o tamanho total da memória e o tamanho das páginas;
e. O programa deve permitir as funções de inclusão e exclusão de processos a qualquer instante, observando-se os limites da memória;
f. A inclusão de um processo deve ser feita sempre encaixando-o no primeiro espaço livre encontrado
g. O programa deve ser capaz de atender a uma demanda de exclusão de processos da memória solicitada pelo usuário;

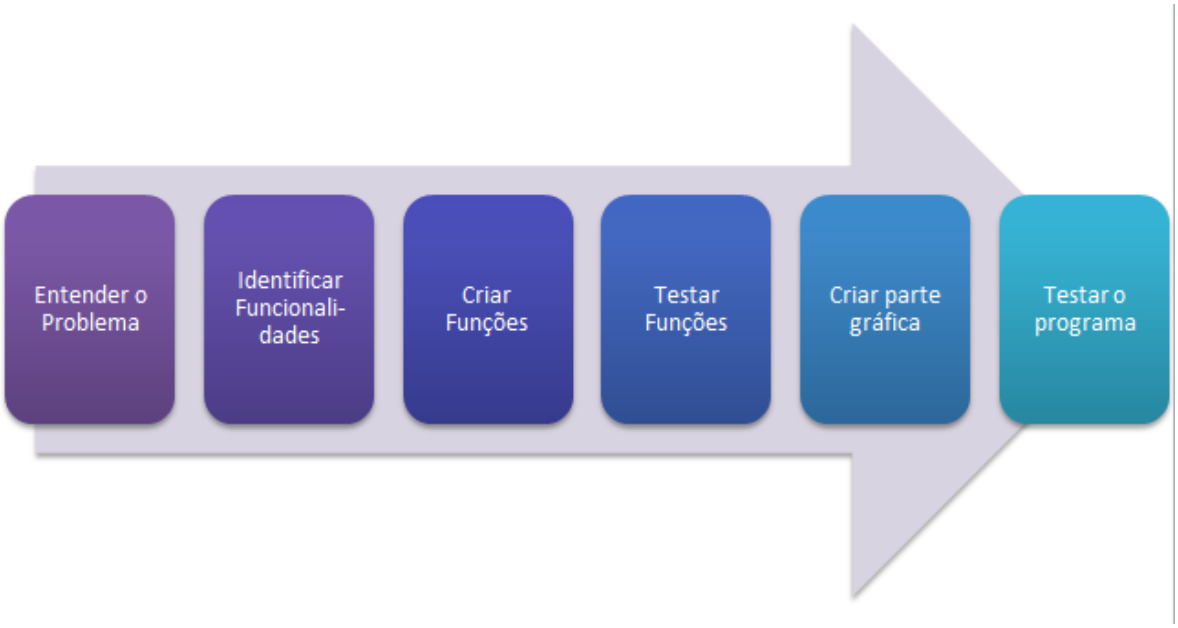
h. O programa deve ser capaz de atender à demanda de exclusão de processos na memória de modo a garantir que as áreas livres espalhadas na memória sejam colocadas em conjunto.
i. O programa deve apresentar o status da memória e da lista encadeada relacionada a qualquer momento, utilizando uma biblioteca gráfica;
j. O programa deve simular a inclusão/ exclusão de vários processos;

3. Objetivo

Este projeto tem por objetivo apresentar um programa desenvolvido em linguagem C que seja capaz de simular o gerenciamento de uma memória de modo a acomodar as demandas dos processos. O programa permite: inclusão e exclusão de processos na memória mantendo o *status* corrente da memória.

4. Metodologia

Visando um melhor aproveitamento dos horários do integrantes da equipe, buscando a participação de todos, optou-se por uma reunião presencial que ocorreu na segunda-feira (27/08/2015) para o planejamento da execução do trabalho. Elaborou-se o seguinte fluxo de trabalho:



Nesta reunião ficou acordado que os próximos encontros seriam virtuais objetivando a participação de todos.

4.1. Ferramentas de suporte

Como ferramentas de suporte foram utilizadas:
Facebook: comunição entre a equipe;

Git hub: repositório do código fonte e da versão final do relatório;
Google drive: repositório do relatório;

5. Descrição da Solução

Para a implementação da solução com base nas funcionalidades identificadas a partir dos requisitos temos o seguinte menu de opções:

1 - Inserir processo
2 - Encerrar processo
3 - Reorganizar processos
4 - Mostrar status da memória
5 - Mostrar todos os processos

Para a execução destes itens, as seguintes bibliotecas foram criadas e incluídas:

a. graph.h: Onde estão incluídas as assinaturas das variáveis e das funções usadas para criar a interface gráfica onde serão colocados os dados relacionados ao status da memória;
b. graph.c: Implementação do arquivo graph.h
c. list.h: arquivo que possui as structs Node (elemento) e List (cabeçalho). Possui as assinaturas das funções que serão executadas de acordo como o menu de opções, além da função de inicialização da lista;
d. list.c: Implementação do arquivo list.h

Todo o código do projeto pode ser acessado no repositório web https://github.com/VinyPinheiro/EDA_Trab1

6. Bibliografia

TANENBAUM, Andrew S.. WOODHULL, Albert S. Sistemas Operacionais - Projeto e Implementação - 2ª Edição - Ed. Bookman, 2000.

7. Anexo I - Descrição de funções e estruturas de dados

a. graph.h

void mountScreen(const char* process_number,const char* process_memory,const char* total_memory);

Objetivo: Cria a tela de interface visual
Entradas: Quantidade de processos, Memória ocupada, Memória Total.
Saídas: -

---

int convert(const char *x);

Objetivo: Converter uma string numérica em um inteiro
Entradas: String com valor numérico
Saídas: Inteiro.

---

b. list.h

void startList(List *li);

Objetivo: Inicializa a lista
Entradas: Ponteiro com o cabeçalho da lista
Saídas: -

---

void insert(List *li, int sizeNewProcess)

Objetivo: Insere processos na Lista
Entradas: Ponteiro com o cabeçalho da lista, Tamanho do novo processo
Saídas: -

---

void closeProcess(List *li, int page)

Objetivo: Finaliza um processo da lista
Entradas: Ponteiro com o cabeçalho da lista, local onde inicia o processo
Saídas: -

---

void showProcesses(List *li)

Objetivo: Mostra todos os processos da lista de maneira tabular
Entradas: Ponteiro com o cabeçalho da lista
Saídas: -

---

void organizeProcesses(List *li)

Objetivo: Reorganiza os dados da lista de modo a não ter nenhum fragmento de espaço vazio entre os processos

Entradas: Ponteiro com o cabeçalho da lista
Saídas: -

---

c. main.c

void menu()

Objetivo: Escrever o menu principal
Entradas: -
Saídas: -
