**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

по дисциплине: «Операционные системы»

на тему: **«**Управление виртуальной памятью. Алгоритмы замещения страниц.**»**

Выполнил: студент гр. ИТИ-12

Ёвженко Ю.Д.

Принял: преподаватель-стажёр

Дашкевич Д.А.

Гомель 2022

**Цель работы**: разработать страничную модель работы с памятью.

**Задание:**

Разработать программу, реализующую заданный алгоритм замещения страниц в памяти. Менеджер памяти должен:

1. Разбивать память заданного размера на указанное количество страниц. На экран должна выводиться следующая информация о состоянии памяти: объем памяти, число страниц, число свободных страниц (%), размер страницы;

2. Размещать в памяти страницу заданного процесса, с замещением занятой по заданному алгоритму (по нажатию кнопки «ДОБАВИТЬ»). Для размещения страницы в памяти, указывается имя процесса и ее номер (вводятся отдельно). Например: *Pro* 3. После нажатия на кнопку «ДОБАВИТЬ» страница размещается в свободной странице памяти. Если задано глобальное размещение (см. вариант задания), то выбирается любая не занятая страница. При локальном размещении страница размещается только среди виртуальных страниц выделенных этому процессу. Выделение страниц в памяти выполняется при первом ее занесении процесса в память. Алгоритм замещения выполняется только при отсутствии свободных страниц под процесс.;

3. Удалять из памяти заданную страницу или все страницы заданного процесса (по нажатию кнопки «УДАЛИТЬ»). Указывается номер удаляемой страницы в памяти;

4. Организовывать циклическое обращение к страницам размещенным в памяти по нажатию на кнопку. При этом случайным образом задается количество обращений к страницам (диапазон 1..10). Для каждого обращения генерируется, случайным образом, номер страницы из диапазона [0; количество страниц памяти]. При обращении к странице в зависимости, от варианта, увеличивается ее внутренний счетчик обращений или устанавливается флаг обращения

Таблица 1 – Вариант выполняемого задания

|  |  |
| --- | --- |
| Вариант | Задание |
| 4 | Глобальное размещение. Алгоритм замещения – *LRU CLOCK*. Существует глобальный счетчик обращений к страницам. При каждом обращении к станице в ее внутренний регистр заносится значение глобального счетчика. Выгружается страница с наименьшим значением счетчика |

**Ход работы**

На рисунке 1 показан ввод кол-ва байт и страниц в менеджере и меню программы.

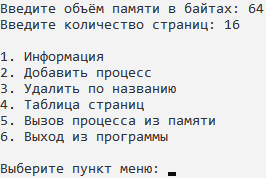


Рисунок 1 – Меню программы

На рисунке 2 показан процесс добавления объекта в память

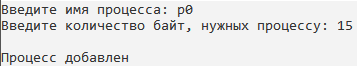


Рисунок 2 – Добавление процесса

На рисунке 3 показано освобождение памяти по названию процесса



Рисунок 3 – Удаление процесса

На рисунке 4 показано расположение процессов в памяти

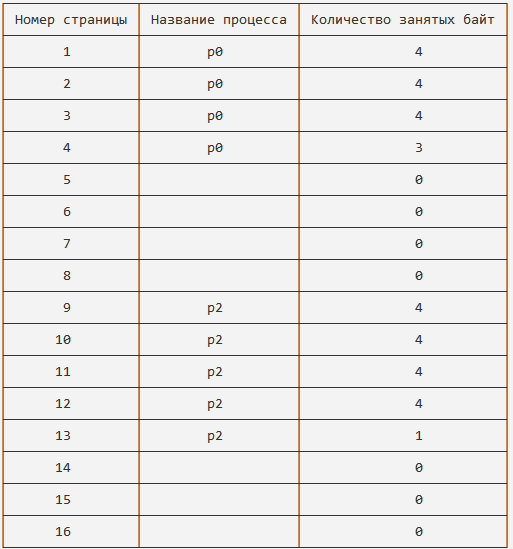


Рисунок 4 – Таблица страниц

На рисунке 5 показано состояние памяти и кол-во свободных страниц.

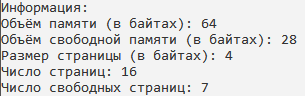


Рисунок 5 – Состояние памяти

**Вывод:** были изучены управления страничной памятью и создан менеджер памяти.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст программы**

Файл *memman.cpp*:

#include <stdio.h>

#include <conio.h>

#include <string.h>

#include <stdlib.h>

#include <iostream>

#include <locale.h>

#include <iomanip>

#include <windows.h>

char bufRus[256];

char\* Rus(const char\* text)

{

CharToOem(text, bufRus);

return bufRus;

}

using namespace std;

void add\_proc(int col\_pages, string \*processes, int \*numbers, char\* memory[], int &size1, int &free\_pages, int page\_size, string \*Vprocesses, int \*Vnumbers, char\* Vmemory[], int\* obr, int &Vcol\_pages)

{

string m;

printf(Rus("Введите имя процесса: "));

fflush(stdin);

getline(cin, m);

int i, n, min, j, n1;

printf(Rus("Введите количество байт, нужных процессу: "));

scanf("%d", &n);

n1 = n;

int pages, h;

pages = n / page\_size;

if(n % page\_size != 0)

pages++;

/\*

rand = rand();

printf("%d", rand);

dfjgmfdm

\*/

if(pages <= col\_pages) {

if(size1 >= n){

for(i = 0; i < col\_pages, n != 0; i++){

if(processes[i] == "" && numbers[i] == 0 && n >= page\_size){

processes[i] = m;

numbers[i] = page\_size;

obr[i] = 1;

memory[i] = (char\*) malloc (page\_size);

free\_pages--;

size1 = size1 - page\_size;

n = n - page\_size;

} else if(processes[i] == "" && numbers[i] == 0 && n < page\_size){

processes[i] = m;

numbers[i] = n;

obr[i] = 1;

memory[i] = (char\*) malloc (n);

free\_pages--;

size1 = size1 - page\_size;

n = 0;

}

}

puts(Rus("\nПроцесс добавлен"));

} else { // сначала занять все свободные ячейки, потом замещать наименее популярные

i = 0;

while(n != 0){

min = 500;

for(h=0; h < col\_pages; h++)

if(min > obr[h] && obr[h]>0) min = obr[h];

printf("MIN = %d\n", min);

if(processes[i] == "" && numbers[i] == 0 && size1 != 0){

processes[i] = m;

numbers[i] = page\_size;

obr[i] = 999;

memory[i] = (char\*) malloc (page\_size);

free\_pages--;

size1 = size1 - page\_size;

n = n - page\_size;

pages = pages - 1;

} else if(processes[i] != "" && numbers[i] != 0 && size1 == 0){

if(obr[i] == min){

for( h = 0; h<col\_pages; h++){

if(Vprocesses[h] == "" && Vnumbers[h] == 0) {

Vprocesses[h] = processes[i];

Vnumbers[h] = numbers[i];

Vmemory[h] = memory[i];

Vcol\_pages++;

free(memory[i]);

break;

}

}

processes[i] = m;

if(pages > 1)

{

numbers[i] = page\_size;

memory[i] = (char\*) malloc (page\_size);

n = n - page\_size;

obr[i] = 999;

pages--;

}

else

{

numbers[i] = n;

memory[i] = (char\*) malloc (n);

n = 0;

obr[i] = 999;

pages--;

printf(Rus("\nПроцесс добавлен в память с замещением старого."));

}

}

}

i++;

if(i == col\_pages) i = 0;

}

for( i = 0; i < col\_pages; i++){

if(processes[i] == m) {

obr[i] = 1;

}

}

}

} else

puts(Rus("Невозможно добавить процесс, т.к. он не помещается в памяти"));

}

void delete\_process(int col\_pages, string \*processes, int \*numbers, char\* memory[], int &size1, int &free\_pages, int page\_size, int\* obr)

{

string m;

printf(Rus("Введите имя процесса, который хотите удалить: "));

fflush(stdin);

getline(cin, m);

int d = 0, i;

for(i = 0; i < col\_pages; i++){

if(processes[i] == m && numbers[i] != 0) {

processes[i] = "";

numbers[i] = 0;

obr[i] = 0;

free(memory[i]);

free\_pages++;

size1 = size1 + page\_size;

d++;

}

}

if(d == 0)

printf(Rus("\nНазвание не действительно!"));

else

printf(Rus("\nПамять освобождена по названию процесса."));

}

void call\_process(int col\_pages, string \*processes, int \*numbers, char\* memory[], int &size1, int &free\_pages, int page\_size, int\* obr)

{

string m;

printf(Rus("Введите имя процесса, к которому хотите обратиться: "));

fflush(stdin);

getline(cin, m);

int i, b = 0;

for(i = 0; i < col\_pages; i++){

if(processes[i] == m && numbers[i] != 0) {

obr[i]++;

b++;

}

// printf("%d\n", obr[i]);

}

if(b == 0){

puts(Rus("Процесс не найден"));

}else{

puts(Rus("Процесс найден"));

}

}

void menu()

{

puts(Rus("\n1. Информация"));

puts(Rus("2. Добавить процесс"));

puts(Rus("3. Удалить по названию"));

puts(Rus("4. Таблица страниц"));

puts(Rus("5. Вызов процесса из памяти"));

puts(Rus("6. Выход из программы"));

printf(Rus("\nВыберите пункт меню: "));

}

void table(int col\_pages, string \*processes, int \*numbers) //Вывод таблицы

{

int i;

//Шапка

const char \*S;

printf("%c", 218);

for(i=0;i<16;i++) printf("%c", 196); printf("%c", 194); for(i=0;i<19;i++) printf("%c", 196); printf("%c", 194);

for(i=0;i<25;i++) printf("%c", 196); printf("%c\n", 191);

printf(Rus("%c Номер страницы %c Название процесса %c Количество занятых байт %c\n"), 179,179,179,179);

for(int j=0; j<col\_pages; j++)

{

if(j!=col\_pages-1)

{

printf("%c", 195); for(i=0;i<16;i++) printf("%c", 196); printf("%c", 197);

for(i=0;i<19;i++) printf("%c", 196); printf("%c", 197); for(i=0;i<25;i++) printf("%c", 196); printf("%c\n", 180);

S = processes[j].c\_str();

printf(Rus("%c%8d %c%10s %c%15d %c\n"), 179, j+1, 179, S, 179, numbers[j], 179);

}

else

{

printf("%c", 195); for(i=0;i<16;i++) printf("%c", 196); printf("%c", 197);

for(i=0;i<19;i++) printf("%c", 196); printf("%c", 197); for(i=0;i<25;i++) printf("%c", 196); printf("%c\n", 180);

S = processes[j].c\_str();

printf(Rus("%c%8d %c%10s %c%15d %c\n"), 179, j+1, 179, S, 179, numbers[j], 179);

printf("%c", 192); for(i=0;i<16;i++) printf("%c", 196); printf("%c", 193);

for(i=0;i<19;i++) printf("%c", 196); printf("%c", 193); for(i=0;i<25;i++) printf("%c", 196); printf("%c", 217);

}

}

}

int main()

{

int size = 0; //Размер общей памяти

int col\_pages = 0; //Количество страниц

int page\_size = 0; // Размер одной страницы

int free\_pages = 0; //Количество свободных страниц

int Vcol\_pages = 0;

printf(Rus("Введите объём памяти в байтах: "));

while(size <= 0){

scanf("%d",&size);

fflush(stdin);

if(size <= 0)

puts(Rus("Неверный объём памяти"));

}

printf(Rus("Введите количество страниц: "));

while(col\_pages <= 0){

scanf("%d",&col\_pages); //Количество страниц

fflush(stdin);

if(col\_pages <= 0)

puts(Rus("Неверное кол-во страниц"));

}

page\_size = size / col\_pages; // Размер одной страницы

free\_pages = col\_pages;

int size1 = size;

int obr[col\_pages];

string processes[col\_pages]; //Массив имен процессов

string Vprocesses[col\_pages];

int numbers[col\_pages]; //Массив количества байтов процессов

int Vnumbers[col\_pages];

for(int i = 0; i < col\_pages; i++)

{

processes[i] = "";

numbers[i] = 0;

Vprocesses[i] = "";

Vnumbers[i] = 0;

}

char\* memory[col\_pages]; //Массив памяти для процесса

char\* Vmemory[col\_pages];

int k;

while(true)

{

menu();

scanf("%d",&k);

system("cls");

switch(k)

{

case 1:

printf(Rus("Информация:\n"));

printf(Rus("Объём памяти (в байтах): %d\n"),size);

printf(Rus("Объём свободной памяти (в байтах): %d\n"),size1);

printf(Rus("Размер страницы (в байтах): %d\n"), page\_size);

printf(Rus("Число страниц: %d\n"), col\_pages);

printf(Rus("Число свободных страниц: %d\n"), free\_pages);

break;

case 2:

add\_proc(col\_pages, processes, numbers, memory, size1, free\_pages, page\_size, Vprocesses, Vnumbers, Vmemory, obr, Vcol\_pages);

break;

case 3:

delete\_process(col\_pages, processes, numbers, memory, size1, free\_pages, page\_size, obr);

break;

case 4:

printf(Rus("\n Оперативная память устройства\n"));

table(col\_pages, processes, numbers);

// printf(Rus("\n\n Внешнаяя(дисковая) память устройства\n"));

// table(Vcol\_pages, Vprocesses, Vnumbers);

break;

case 5:

call\_process(col\_pages, processes, numbers, memory, size1, free\_pages, page\_size, obr);

break;

case 6:

exit(0);

break;

default:

puts(Rus("Выбран неправильный пункт меню!"));

}

printf("\n");

}

return 0;

}