**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 10

по дисциплине «Операционные системы»

на тему: «управление виртуальной памятью»

Выполнил: студент гр. ИТП-11

ЛЕДИ ДЖОЙС

Принял: преподаватель-стажер

Карась О.В.

Гомель 2022

**Цель:** изучение алгоритмов управления виртуальной памятью, разработка программы менеджера памяти.

**Задача:**

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

1. Разбивать память заданного размера на указанное количество страниц. На экран должна выводится 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.

Вариант 2:

Глобальное размещение. Алгоритм замещения – *FIFO*. Реализуется очередь страниц в конец которой попадают страницы, размещенные в памяти, а из начала берутся для замещения.

**Задание**

*FIFO* — способ организации и манипулирования данными относительно времени и приоритетов. В соответствии с вариантом программа должна размещать новые процессы в конце страницы, а процессы, которые были первыми добавлены в память будут замещается со страницы. При запуске программы появляется меню, где пользователь может выбрать какое количество страниц, памяти и процессов будет использовать пользователь. Пример меню выбора страницы, памяти и процессов представлен на рисунке 1.

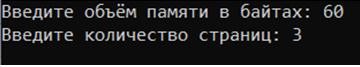


Рисунок 1 - меню выбора страниц, памяти и процессов

После этого на экран выводится меню, где показана вся информация, которую ввел пользователь, и меню, где пользователь может добавить страницу в память, удалить страницу, вывести таблицу страниц или выйти из программы. Пример главного меню программы представлен на рисунке 2.

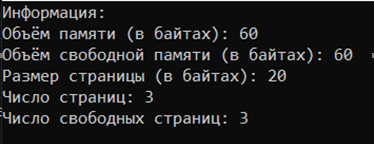


Рисунок 2 – главное меню программы

При выборе второго меню программы открывается окошко, где пользователь указывает имя процесса и размер памяти который занимает процесс. Пример этого меню указан на рисунке 3.

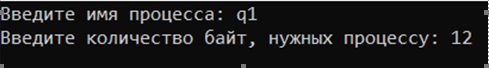


Рисунок 3 – меню для добавления страницы в памяти

После добавления нескольких страниц мы можем в главном меню выбрать четвертый пункт и вывести на экран таблицу страниц. На рисунке под номером 4 указана таблица страниц.

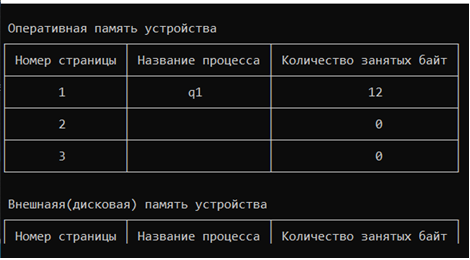


Рисунок 4 – Таблица страниц

В главном меню при нажатии на кнопку 3 мы можем удалить один из страниц или все страницы одного из процессов. Пример удаления всей страницы одного из процессов указа на рисунке 5.

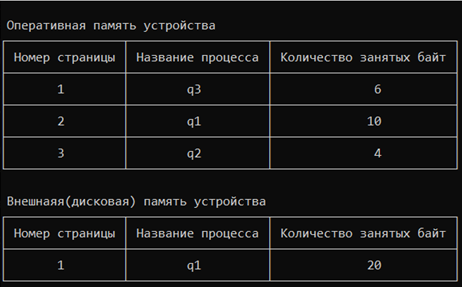


Рисунок 5 – удаление всей страницы одного процесса

По условию 4 варианта замещаться должны те процессы, которые первые попали в страницу. То есть если на рисунке 6 все страницы у первого процесса будет выгружаться тот который был первым добавлен.. Пример этого замещения указан на рисунке 6.

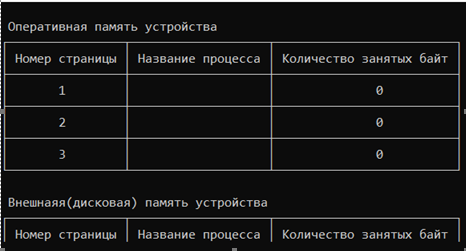


Рисунок 6 – замещения страницы, которая было добавлена раньше всех остальных в списке

**Вывод**: в ходе лабораторной работы были изучены основные алгоритмы замещение страниц и работа с виртуальной памятью.

**Листинг программы**

**#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(string \*processes, string \*Vprocesses, int \*numbers, int \*Vnumbers, char\* memory[], char\* Vmemory[], int \*Idx, int &min, int col\_pages, int &j, int &k, int &size, int page\_size, int &free\_pages)**

**{**

**string m;**

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

**fflush(stdin);**

**getline(cin, m);**

**int i,n,h;**

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

**scanf("%d", &n);**

**int pages = n / page\_size; // Количество страниц для процесса**

**if(n % page\_size != 0)**

**pages++;**

**if(size >= n)**

**{**

**i = 0;**

**while(n != 0)**

**{**

**if(processes[i] == "" && numbers[i] == 0)**

**{**

**if(free\_pages > 0) //Уменьшение количества свободных страниц на 1**

**free\_pages--;**

**processes[i] = m;**

**if(pages > 1)**

**{**

**numbers[i] = page\_size;**

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

**size = size - page\_size;**

**n = n - page\_size;**

**Idx[i] = j+1;**

**j++;**

**}**

**else**

**{**

**numbers[i] = n;**

**size = size - page\_size;**

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

**n = 0;**

**Idx[i] = j+1;**

**j++;**

**min = Idx[0];**

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

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

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

**printf(Rus("\nПроцесс успешно добавлен в память."));**

**return;**

**}**

**pages--;**

**}**

**i++;**

**}**

**}**

**else**

**{**

**i = 0;**

**while(n != 0)**

**{**

**if(processes[i] == "" && numbers[i] == 0)**

**{**

**if(free\_pages > 0) //Уменьшение количесвта свободных страниц на 1**

**free\_pages--;**

**processes[i] = m;**

**if(pages > 1)**

**{**

**numbers[i] = page\_size;**

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

**size = size - page\_size;**

**n = n - page\_size;**

**Idx[i] = j+1;**

**j++;**

**}**

**pages--;**

**}**

**else if(processes[i] != "" && numbers[i] != 0 && size == 0) //Уже size = 0!!!!!!!!**

**{**

**if(Idx[i] != -1 && Idx[i] == min) // ЗАМЕЩЕНИЕ**

**{**

**for(int g=0; g<k+1; g++)**

**{**

**if(Vprocesses[g] == "" && Vnumbers[g] == 0)**

**{**

**Vprocesses[g] = processes[i];**

**Vnumbers[g] = numbers[i];**

**Vmemory[g] = memory[i];**

**}**

**}**

**k++;**

**processes[i] = m;**

**if(pages > 1)**

**{**

**numbers[i] = page\_size;**

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

**n = n - page\_size;**

**Idx[i] = -1;**

**}**

**else**

**{**

**numbers[i] = n;**

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

**n = 0;**

**Idx[i] = -1;**

**printf(Rus("\nПроцесс успешно добавлен в память с замещением на диск старого."));**

**//return;**

**}**

**min = Idx[0];**

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

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

**pages--;**

**}**

**}**

**i++;**

**if(i>col\_pages-1) i = 0;**

**}**

**}**

**}**

**void delete\_process(char\* memory[], char\* Vmemory[], string \*processes, string \*Vprocesses, int \*numbers, int \*Vnumbers, int \*Idx, int &j, int col\_pages, int k, int &size, int page\_size, int &free\_pages)**

**{**

**string m;**

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

**fflush(stdin);**

**getline(cin, m);**

**int i,d=0,q=0;**

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

**{**

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

**{**

**processes[i] = "";**

**numbers[i] = 0;**

**free(memory[i]);**

**free\_pages++;**

**size = size + page\_size;**

**Idx[i] = 0;**

**d++;**

**}**

**}**

**for(i = 0; i < k; i++)**

**{**

**if(Vprocesses[i] == m && Vnumbers[i] != 0)**

**{**

**Vprocesses[i] = "";**

**Vnumbers[i] = 0;**

**free(Vmemory[i]);**

**}**

**}**

**if(d == 0)**

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

**else**

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

**}**

**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. 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 %d %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 j = 0, k1 = 0, min;**

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

**//cout << "Input size of memory (bites): ";**

**while(size <= 0)**

**scanf("%d",&size);**

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

**while(col\_pages <= 0)**

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

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

**free\_pages = col\_pages;**

**int size1 = size;**

**int Idx[col\_pages];**

**min = Idx[0];**

**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] = "";**

**Vprocesses[i] = "";**

**numbers[i] = 0;**

**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(processes, Vprocesses, numbers, Vnumbers, memory, Vmemory, Idx, min, col\_pages, j, k1, size1, page\_size, free\_pages);**

**break;**

**case 3:**

**delete\_process(memory, Vmemory, processes, Vprocesses, numbers, Vnumbers, Idx, j, col\_pages, k1, size1, page\_size, free\_pages);**

**break;**

**case 4:**

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

**table(col\_pages, processes, numbers);**

**//table(col\_pages, processes, Idx);**

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

**table(k1, Vprocesses, Vnumbers);**

**break;**

**case 5:**

**exit(0);**

**break;**

**default:**

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

**}**

**printf("\n");**

**}**

**return 0;**

**}**