**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 №8

по дисциплине: «Операционные системы»

на тему: **«**Простейшие схемы управления памятью**»**

Выполнил: студент гр. ИТП-11

Гаврильченко А.Е.

Принял: пр.

Карась О.В.

Гомель 2023

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

**Задание:**

Разработать программу, реализующую заданный алгоритм выделения памяти. Менеджер памяти должен:

1. По запросу процесса выделять память, согласно заданного алгоритма (таблица). На экран должна выводиться следующая информация о состоянии памяти: объем памяти, объѐм свободной памяти, размер наибольшего свободного блока, количество запросов на выделение памяти, количество удовлетворѐнных запросов (%).

2. Для выделения памяти указывается имя процесса и размер блока. После нажатия на кнопку «ДОБАВИТЬ» память выделяется или выдаѐтся сообщение о невозможности выделения.

3. Удалять из памяти заданный блок или все блоки заданного процесса (по нажатию кнопки «УДАЛИТЬ»). Указывается номер удаляемого блока и имя процесса.

4. Реализовать возможность последовательной записи/чтения информации в/из выделенную память по логическому адресу. Вывести физического адреса ячейки памяти, в которую была осуществлена запись.

5. Организовывать циклическое выделение и освобождение памяти. При этом случайным образом задается количество выделяемых блоков и их размер.

Таблица c заданием

****

**Процесс выполнения:**



Рис.1(количество байтов)

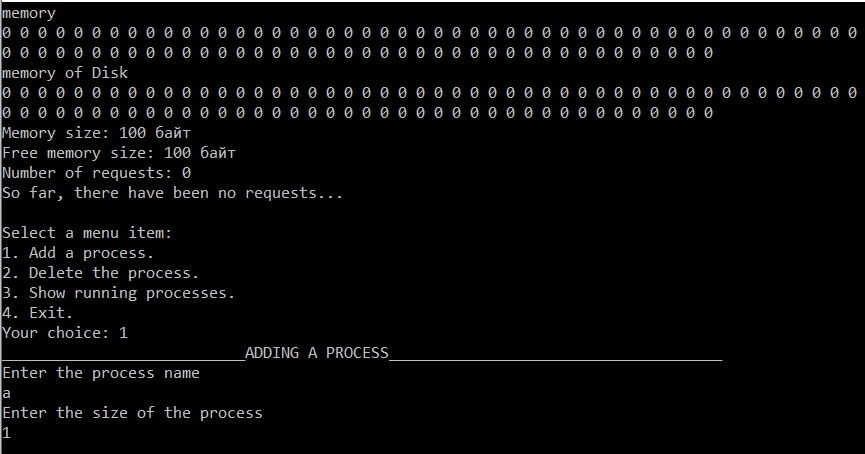


Рис.2(добавляем процесс а)

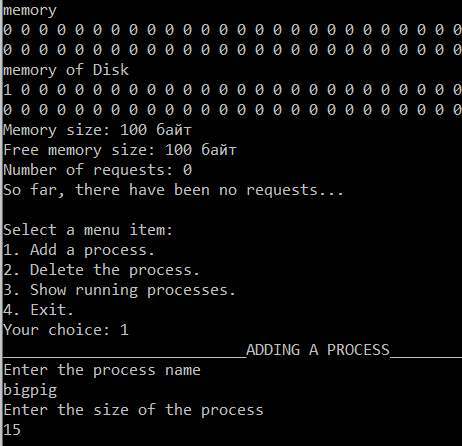


Рис.3(добавляем процесс bigpig)

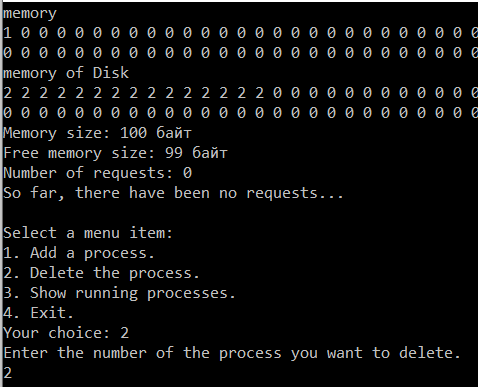


Рис.4(удаляем процесс под номером 2)

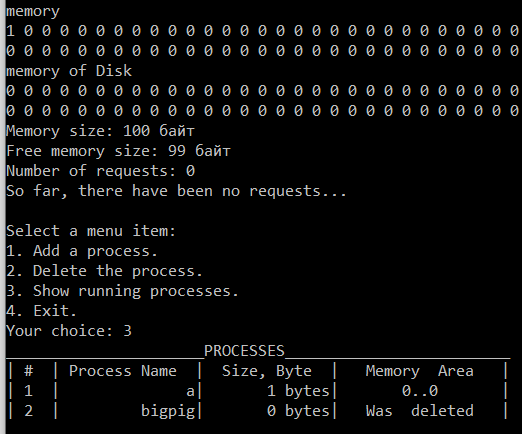


Рис.5(просмотр процессов)

**Листинг программы:**

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <windows.h>

#include <iostream>

using namespace std;

struct processes

{

char name[20];

int num;

int begin;

int size;

};

void showMemory(int \*memory, int \*memoryD, int num\_bits, int put\_bits,int \*op,int \*bp);

void addProcess(int \*memory, int \*memoryD, int num\_bits, int &pbits, struct processes \*process, int &num\_p, int &max\_size,int \*op,int \*bp);

void delProcess(struct processes \*process, int \*memory, int \*memoryD,int num\_b, int &num\_p,int \*op,int \*bp);

void showProcesses(struct processes \*process,int num\_p);

int main()

{

system("chcp 1251 && cls");

struct processes process[100];

int num\_bits, put\_bits=0;

cout << "How many bytes do I need to execute?";

cin >> num\_bits;

int memory[num\_bits] = {0};

int memoryD[num\_bits] = {0};

int max\_size=0; // for swoping

int num\_process=0; //number of processes

int op=0,bp=0; //op - zapros vipolnen, bp - zapros NE vipolnen

int menu=0;

while(menu!=4)

{

system("cls");

showMemory(memory, memoryD, num\_bits, put\_bits, &op, &bp);

cout << "Select a menu item:\n";

cout << "1. Add a process.\n";

cout << "2. Delete the process.\n";

cout << "3. Show running processes.\n";

cout << "4. Exit.\n";

cout << "Your choice: ";

cin >> menu;

switch(menu)

{

case 1:

addProcess(memory, memoryD, num\_bits, put\_bits, process, num\_process, max\_size, &op, &bp);

break;

case 2:

if(num\_process)

delProcess(process, memory, memoryD, num\_bits, num\_process, &op, &bp);

else

cout << "There are no records of processes.\n";

break;

case 3:

if(num\_process)

showProcesses(process,num\_process);

else

cout << "There are no records of processes.\n";

break;

case 4:

cout << "To finish, press any key\n";

break;

default:

cout << "There is no such item.\n";

}

getch();

}

return 0;

}

void showMemory(int \*memory, int \*memoryD, int num\_bits, int put\_bits,int \*op,int \*bp)

{

cout << "memory\n";

for(int i=0;i<num\_bits;i++)

cout << memory[i] << " ";

cout << "\nmemory of Disk\n";

for(int i=0;i<num\_bits;i++)

cout << memoryD[i] << " ";

SetConsoleOutputCP(1251);

printf("\nMemory size: %d áàéò\n",num\_bits);

printf("Free memory size: %d áàéò\n",num\_bits-put\_bits);

printf("Number of requests: %d\n",\*op+\*bp);

if(\*op+\*bp==0)

printf("So far, there have been no requests...\n\n");

else

printf("Percentage of satisfied requests: %d%%\n\n",\*op\*100/(\*op+\*bp));

}

void addProcess(int \*memory, int \*memoryD, int num\_bits, int &pbits, struct processes \*process, int &num\_p, int &max\_size,int \*op,int \*bp)

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ADDING A PROCESS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "Enter the process name\n";

fflush(stdin);

gets(process[num\_p].name);

cout << "Enter the size of the process\n";

cin >> process[num\_p].size;

if(process[memoryD[0]].size+pbits<num\_bits)

{

int i;

if(process[num\_p].size>max\_size)

{

max\_size=process[num\_p].size;

if(num\_p==0)

{

for(i=0;i<process[num\_p].size; i++)

memoryD[i]=num\_p+1;

}

else

{

for(i=pbits;i<(pbits+process[memoryD[0]-1].size);i++)

memory[i]=memoryD[0];

pbits=i;

for(i=0;i<process[num\_p].size; i++)

memoryD[i]=num\_p+1;

}

num\_p++;

}

else

{

if(num\_p==0)

process[num\_p].begin=0;

else

process[num\_p].begin=pbits;

for(i=pbits; i<(pbits+process[num\_p].size); i++)

memory[i]=num\_p+1;

pbits+=process[num\_p].size;

num\_p++;

}

\*op++;

}

else

{

cout << "not enough space\n";

\*bp++;

}

}

void delProcess(struct processes \*process, int \*memory, int \*memoryD,int num\_b, int &num\_p,int \*op,int \*bp)

{

cout << "Enter the number of the process you want to delete.\n";

int number,i;

cin >> number;

process[number-1].size=0;

bool sdvig=false;

int nziro=0;

for(i=0;i<num\_b; i++)

if(memory[i]==number)

{

memory[i]=0;

nziro++;

sdvig=true;

}

if(sdvig==true)

{

for(i=0;memory[i]!=0 && i<num\_b;i++);

i--;

for(;i<num\_b;i++)

{

if(i<(num\_b-nziro))

memory[i]=memory[i+nziro];

}

}

if(memoryD[0]==number)

for(i=0;i<num\_b; i++)

memoryD[i]=0;

\*op++;

}

void showProcesses(struct processes \*process,int num\_p)

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PROCESSES\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

int i=0;

cout << "| # | Process Name | Size, Byte | Memory Area |\n";

for(i=0;i<num\_p;i++)

{

if(process[i].size==0)

printf("| %-3i|%15s|%8i bytes| Was deleted |\n",i+1,process[i].name,process[i].size);

else

printf("| %-3i|%15s|%8i bytes|%8i..%-8i|\n",i+1,process[i].name,process[i].size,process[i].begin,process[i].begin-1+process[i].size);

}

}

**Вывод:** нами были изучены алгоритмы управления памятью с разработкой программ менеджера памяти.