**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 №8**

по дисциплине «Операционные системы»

на тему: «Тупиковые ситуации и подходы к их разрешению»

Выполнила: студентка гр. ИТП-11

Бородина Е. И.

Принял: преподаватель-стажёр

Карась О.В.

Гомель 2022

**Цель работы:** разработать программу, реализующую заданный алгоритм замещения страниц в памяти.

**Задание:**

Менеджер памяти должен:

1. По запросу процесса выделять память. На экран должна выводится следующая информация о состоянии памяти: объем памяти, объем свободной памяти, размер наибольшего свободного блока, количество запросов на выделение памяти, количество удовлетворённых запросов (%).

2. Для выделения памяти указывается имя процесса и размер блока. После нажатия на кнопку «ДОБАВИТЬ» память выделяется или выдаётся 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. Организовать циклическое выделение и освобождение памяти. При этом случайным образом задается количество выделяемых блоков и их размер.

**Вариант 4**

Страничная организация памяти. Размер страницы 2n .

**Демонстрация функций:**

****

Рисунок 1 – Начало работы программы. Вопрос – «Какое количество памяти будет использоваться?»

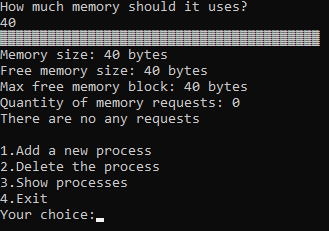
****

Рисунок 2 – Информация о состоянии памяти и меню

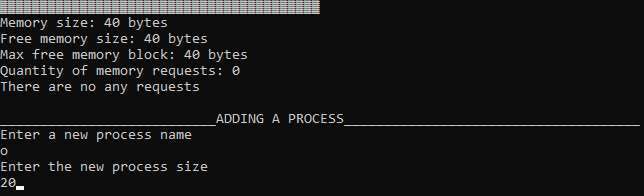
****

Рисунок 3 – Окно после выбора первого пункта меню: «Создать новый процесс»

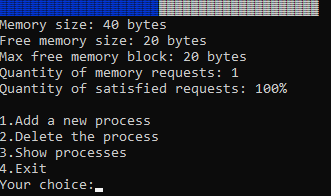
****

Рисунок 4 – Состояние памяти после создания первого процесса

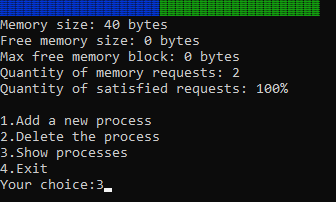


Рисунок 5 – Состояние памяти после создания второго процесса

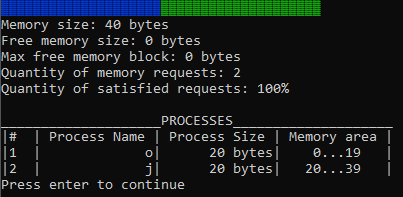


Рисунок 6 – Демонстрация состояния двух процессов

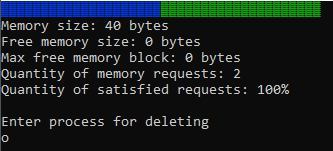


Рисунок 7 – Удаление первого процесса

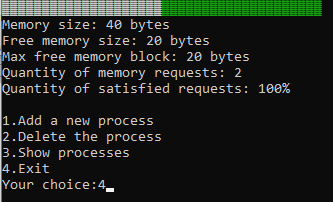


Рисунок 8 – Состояние памяти после удаления первого процесса. Выход из программы



Рисунок 9 – Командная строка после выхода из программы

**Листинг программы:**

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <windows.h>

HANDLE console = GetStdHandle(STD\_OUTPUT\_HANDLE);

struct processes

{

char name[20];//имя процесса

int begin;//первая ячейка которую процесс занимает в памяти

int size;//размер процесса

};

void showMemory(int \*memory,int mc,int \*op,int \*bp);

//memory-массив памяти,mc-количество выделяемой памяти,op-удовлетворенные запросы,bp-неудовлетворенные запросы

void addProcess(struct processes \*process,int \*procQ,int \*memory,int mc,int \*op,int \*bp);

//process-массив процессов,procQ-количество процессов

void delProcess(struct processes \*process,int \*procQ,int \*memory,int mc);

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

int main()

{

struct processes process[100];//массив процессов

int op=0,bp=0;//op-удовлетворенные запросы,bp-неудовлетворенные запросы

int mc,i,item=1;//memory cells-количество выделяемой памяти, item-пункт меню

puts("How much memory should it uses?");

scanf("%d",&mc);

if(mc<=0)

{

puts("How much memory should it uses?");

scanf("%i",&mc);

}

int memory[mc];//массив - "память"

int processQuantity=0;//количество процессов

for(i=0;i<mc;i++)memory[i]=0;//обнуление памяти

do

{

showMemory(memory,mc,&op,&bp);

puts("1.Add a new process");

puts("2.Delete the process");

puts("3.Show processes");

puts("4.Exit");

printf("Your choice:");

scanf("%i",&item);

system("cls");

showMemory(memory,mc,&op,&bp);

switch(item)

{

case 1: {addProcess(process,&processQuantity,memory,mc,&op,&bp);system("cls"); } break;

case 2: {delProcess(process,&processQuantity,memory,mc);system("cls");}break;

case 3: {showProcesses(process,processQuantity);system("cls");}break;

case 4: {system("cls");puts("PROGRAMM WAS FINISHED");puts("made by Borodzina Elizaveta ITP-11");}break;

default: {system("cls");}break;

}

}while(item!=4);

return 0;

}

void showMemory(int \*memory,int mc,int \*op,int \*bp)

{

int i,om=0;//om-occupied memory-занятая память

int maxFreeBlock=0,block=0;//максимальный свободный блок, блок памяти

for(i=0;i<mc;i++)

{

if(\*(memory+i)==0)

{

block++;

printf("\xB2");

}

else

{

SetConsoleTextAttribute(console,\*(memory+i));

block=0;

om+=1;

printf("\xB2");

SetConsoleTextAttribute(console,7);

}

if(block>maxFreeBlock)

maxFreeBlock=block;

}

printf("\nMemory size: %i bytes\n",mc);

printf("Free memory size: %i bytes\n",mc-om);

printf("Max free memory block: %i bytes\n",maxFreeBlock);

printf("Quantity of memory requests: %i\n",\*op+\*bp);

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

printf("There are no any requests\n\n");

else

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

}

void addProcess(struct processes \*process,int \*procQ,int \*memory,int mc,int \*op,int \*bp)

{

puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ADDING A PROCESS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

puts("Enter a new process name");

scanf("%s",(process+(\*procQ))->name);

puts("Enter the new process size");

scanf("%i",&((process+(\*procQ))->size));

int maxFreeBlock=0,block=0,i;

int end;//для поиска конца самого большого пустого блока

int resbeg,resend;//начало самого большого пустого блока, конец самого большого пустого блока

for(i=0;i<mc;i++)

{

if(\*(memory+i)>0)//если в этом месте массива уже есть процесс, то

{

if(block>maxFreeBlock)//мы смотрим больше ли та область массива чем наше макс значение, если да, то

{

maxFreeBlock=block;//максимальный блок равен текущему

resend=end;//конец этого блока

}

block=0;//обнуляем

}

else//если процесса нет, то

{

block++;//добавляем к блоку ещё одну ячейку

end=i;//увеличиваем значение конца блока на 1

}

}

if(block>maxFreeBlock)//если в памяти ещё нет ни одного процесса, то делаем это

{

maxFreeBlock=block;

resend=end;

}

resbeg=resend-maxFreeBlock+1;//начало блока = конец блока - его размер + 1

if(((process+(\*procQ))->size)<=maxFreeBlock)//Если размер процесса < макс блок, то

{

(process+(\*procQ))->begin=resbeg;//начало процесса равно текущему началу

for(i=resbeg;i<=resbeg-1+(process+(\*procQ))->size;i++)//элементы этого блока от начала до (начало блока-1+размер процесса)

\*(memory+i)=\*procQ+1;//значение ячейки = номер элемента массива процессов+1

(\*op)++;//количество удовлетворенных запросов увеличивается на 1

}

else//если размер <, то просто пишем что недостаточно места и начало процесса делаем = -1

{

puts("Not enough free space");

(\*bp)++;//количество неудовлетворенных запросов увеличивается на 1

(process+(\*procQ))->begin=-1;

getch();

}

\*procQ+=1;

}

void delProcess(struct processes \*process,int \*procQ,int \*memory,int mc)

{

puts("Enter process for deleting");

int number,i;

scanf("%i",&number);

for(i=number-1;i<\*procQ-1;i++)

\*(process+i)=\*(process+i+1);

(\*procQ)-=1;

for(i=0;i<mc;i++)

if(\*(memory+i)>=number)

{

if(\*(memory+i)==number)

\*(memory+i)=0;

}

}

void showProcesses(struct processes \*process,int procQ)

{

puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PROCESSES\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

int i=0;

puts("|# | Process Name | Process Size | Memory area |");

for(i=0;i<procQ;i++)

{

if((process+i)->begin!=-1)

printf("|%-3i|%14s|%8i bytes|%5i...%-5i|\n",i+1,(process+i)->name,(process+i)->size,((process+i)->begin),((process+i)->begin)-1+((process+i)->size));

else

printf("|%-3i|%14s|%8i bytes|Not in memory|\n",i+1,(process+i)->name,(process+i)->size);

}

puts("Press enter to continue");

getch();

}

**Вывод:** в результате выполнения работы была разработана программа, реализующая заданный алгоритм замещения страниц в памяти.