**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.

Алгоритмы замещения страниц»

Выполнил: студент гр. ИТП-11

Король В.Н.

Принял: преподаватель

Карась О.В.

Гомель 2022

**Цель:** изучить основные алгоритмы замещение страниц и работой с виртуальной памятью.

**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 3:

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.

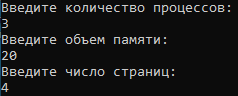


Рисунок 1 - меню выбора страниц, памяти и процессов

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.

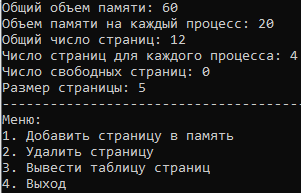


Рисунок 2 – главное меню программы

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

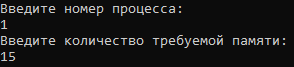


Рисунок 3 – меню для добавления страницы в памяти

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

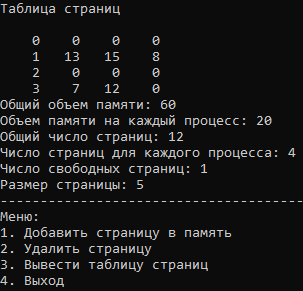


Рисунок 4 – таблица страниц

В главном меню при нажатии на кнопку 2 мы можем удалить один из страниц или все страницы одного из процессов. Пример удаления всей страницы одного из процессов указа на рисунке 5.

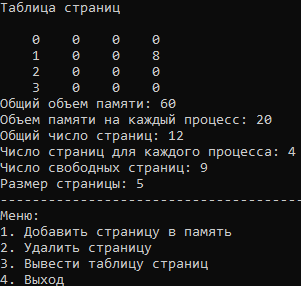
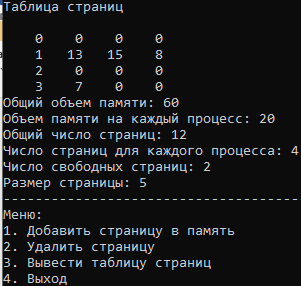


Рисунок 5 – удаление всей страницы одного процесса

По условию моего 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. Это реализуется при помощи переменное которая изначально имеет значение 0, после добавления первой страницы в память к этой переменной добавляется 1 и так далее к каждой последующей переменной добавляется единица. Далее мы просматриваем массив, где указаны эти значения и находим самое маленькое из них, следовательно страница под этим номером программа и будет замещать. Пример этого замещения указан на рисунке 6.

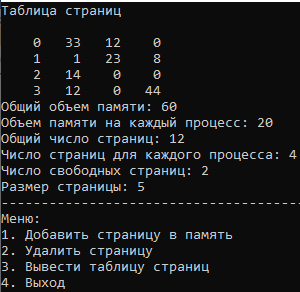
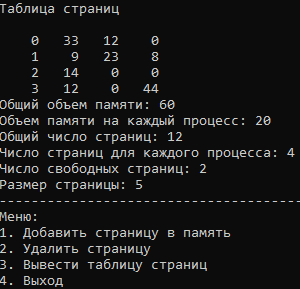


Рисунок 6 – замещения страницы, которая было добавлена раньше всех остальных в списке

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

**Листинг программы**

#include<stdio.h>

#include<string.h>

#include<conio.h>

#include<ctype.h>

#include <windows.h>

#include<math.h>

int main()

{

SetConsoleCP (1251);

SetConsoleOutputCP (1251);

int pm, kol, k = 0, del = 0, j = 0, flag = -1, v = 0, proc, f, pust = 0, i;

int obem;

int ind, ch = 0, h;

puts("Ââåäèòå êîëè÷åñòâî ïðîöåññîâ: ");

scanf("%d", &proc);

puts("Ââåäèòå îáúåì ïàìÿòè: ");

scanf("%d", &pm);

puts("Ââåäèòå ÷èñëî ñòðàíèö: ");

scanf("%d", &kol);

obem = pm/kol;

float A[proc][kol], B[2][100];

int C[proc];

while(k != 4)

{

printf("Îáùèé îáúåì ïàìÿòè: %d\n", pm\*proc);

printf("Îáúåì ïàìÿòè íà êàæäûé ïðîöåññ: %d\n", pm);

printf("Îáùèé ÷èñëî ñòðàíèö: %d\n", kol\*proc);

printf("×èñëî ñòðàíèö äëÿ êàæäîãî ïðîöåññà: %d\n", kol);

printf("×èñëî ñâîáîäíûõ ñòðàíèö: %d\n", pust);

printf("Ðàçìåð ñòðàíèöû: %d\n", obem);

puts("------------------------------------------------");

puts("Ìåíþ:");

puts("1. Äîáàâèòü ñòðàíèöó â ïàìÿòü");

puts("2. Óäàëèòü ñòðàíèöó");

puts("3. Âûâåñòè òàáëèöó ñòðàíèö");

puts("4. Âûõîä");

scanf("%d", &k);

system("cls");

switch(k)

{

case 1:

puts("Ââåäèòå íîìåð ïðîöåññà: ");

scanf("%f", &B[0][j]);

puts("Ââåäèòå êîëè÷åñòâî òðåáóåìîé ïàìÿòè: ");

scanf("%f", &B[1][j]);

j++;

break;

case 2:

puts("1. Óäàëèòü èç ïàìÿòè çàäàííóþ ñòðàíèöó");

puts("2. Óäàëèòü âñå ñòðàíèöû çàäàííîãî ïðîöåññà");

int p;

scanf("%d", &p);

if(p == 1)

{

puts("Ââåäèòå ñòðàíèöó: ");

scanf("%d", &ind);

puts("Ââåäèòå ïðîöåññ: ");

scanf("%d", &del);

A[del][ind] = 0;

}

else if(p == 2)

{

puts("Ââåäèòå ïðîöåññ: ");

scanf("%d", &del);

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

A[del][i] = 0;

C[del] = 0;

}

break;

case 3:

ch = 0;

while(ch < proc)

{

for(h = 0; h < j; h++)

{

if(ch == B[0][h])

{

flag = h;

break;

}

}

for(f = 0; f < kol; f++)

{

if(A[ch][f] == 0 && flag >= 0)

{

A[ch][f] = B[1][flag];

B[1][flag] = -1;

B[0][flag] = -1;

v = 1;

break;

}

}

for(f = 0; f < kol; f++)

{

if(v == 0 && flag >= 0)

{

A[ch][C[ch]] = B[1][flag];

B[1][flag] = -1;

B[0][flag] = -1;

C[ch]++;

if(C[ch] == kol)

C[ch] = 0;

break;

}

}

flag = -1;

ch++;

v = 0;

}

printf("Òàáëèöà ñòðàíèö\n\n");

for(f = 0; f < kol; f++)

{

printf("%5d", f);

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

printf("%5.0f", A[i][f]);

printf("\n");

}

pust = 0;

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

{

for(f = 0; f < kol; f++)

{

if(A[i][f] == 0)

pust++;

}

}

break;

case 4:

break;

default:

break;

}

}

fflush(stdin);

getchar();

return 0;

}