**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 №9

по дисциплине «Операционные системы»

на тему: «Простейшие схемы управления памятью»

Выполнил: студент гр. ИТП-11

Александров А. Г.

Принял: преподаватель-стажёр

Карась О. В.

Гомель 2022

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

**Ход работы**

Самым простым способом управления оперативной памятью является ее предварительное (обычно на этапе генерации или в момент загрузки системы) разбиение на несколько разделов фиксированной величины. Поступающие процессы помещаются в тот или иной раздел. При этом происходит условное разбиение физического адресного пространства. Связывание логических и физических адресов процесса происходит на этапе его загрузки в конкретный раздел, иногда – на этапе компиляции.

Каждый раздел может иметь свою очередь процессов, а может существовать и глобальная очередь для всех разделов.

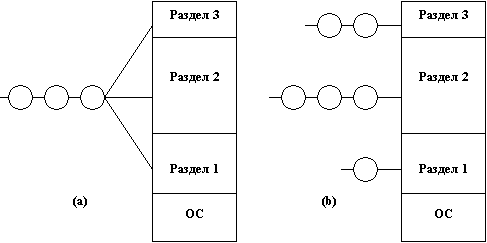


Рисунок 1 – Схема с фиксированными разделами: (a) – с общей очередью процессов, (b) – с отдельными очередями процессов

Эта схема была реализована, например, в *IBM OS/360 (MFT)* и *DEC RSX-11*.

Подсистема управления памятью оценивает размер поступившего процесса, выбирает подходящий для него раздел, осуществляет загрузку процесса в этот раздел и настройку адресов.

В какой раздел помещать процесс? Наиболее распространены три стратегии:

* первый подходящий (*First fit*). Процесс помещается в первый подходящий по размеру раздел;
* наиболее подходящий (*Best fit*). Процесс помещается в тот раздел, где после его загрузки останется меньше всего свободного места;
* наименее подходящий (*Worst fit*). При помещении в самый большой раздел в нем остается достаточно места для возможного размещения еще одного процесса;

Моделирование показало, что доля полезно используемой памяти в первых двух случаях больше, при этом первый способ несколько быстрее. Попутно заметим, что перечисленные стратегии широко применяются и другими компонентами ОС, например, для размещения файлов на диске.

Очевидный недостаток этой схемы – число одновременно выполняемых процессов ограничено числом разделов.

Другим существенным недостатком является то, что предлагаемая схема сильно страдает от внутренней фрагментации – потери части памяти, выделенной процессу, но не используемой им. Фрагментация возникает потому, что процесс не полностью занимает выделенный ему раздел или потому, что некоторые разделы слишком малы для выполняемых пользовательских программ.

**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.

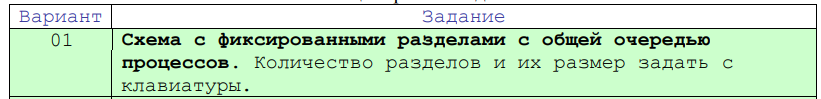


Рисунок 2 ­– Вариант задания

Ниже представлен ввод данных и общая информация о процессах:

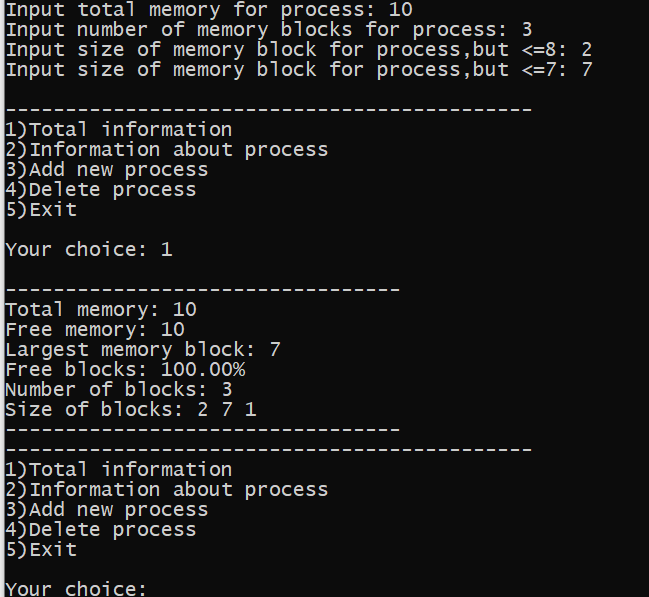


Рисунок 3 ­– Общая информация, меню выбора и ввод данных

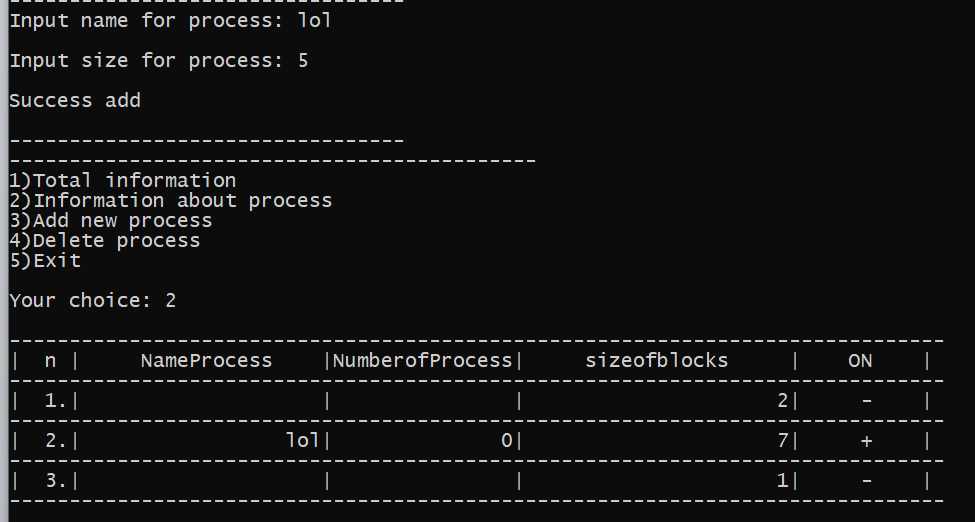


Рисунок 4 ­– Добавление процесса «lol» размера 5 в первый возможный раздел

При вызове 4 пункта мы можем удалить выбранный нами процесс, а с помощью пункта 5 закончить работу программы. Листинг представлен в приложении А.

**Вывод:** В ходе выполнения лабораторной работы были изучены алгоритмы управления памятью и разработана программа менеджера памяти.

**Приложение А**

**Листинг программы**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <windows.h>

struct condition

{

int sizeofblocks;

int condition;

int number;

}array[1000];

struct operations

{

char name[1000];

int size;

int condition;

int N;

}oper[1000];

int main()

{

int totalmemory,activtotalmemory,numberofblocks,freememory,largestmemoryblock,numberofprocess=0,success=0;

int i,n,g=0,col,flag;

float successful=100,config=100;

printf("Input total memory for process: ");

scanf("%d",&totalmemory);

activtotalmemory=totalmemory;

freememory=totalmemory;

printf("Input number of memory blocks for process: ");

scanf("%d",&numberofblocks);

while(numberofblocks>totalmemory)

{

printf("Input number of memory blocks for process <=%d: ",totalmemory);

scanf("%d",&numberofblocks);

}

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

{

array[i].sizeofblocks=1;

array[i].condition=1;

activtotalmemory-=array[i].sizeofblocks;

}

for(i=0;i<numberofblocks && activtotalmemory!=0;i++)

{

activtotalmemory+=array[i].sizeofblocks;

if(i==numberofblocks-1)

{

array[i].sizeofblocks=activtotalmemory;

}

else

{

printf("Input size of memory block for process,but <=%d: ",activtotalmemory);

scanf("%d",&n);

if(activtotalmemory-n<0)

{

activtotalmemory-=array[i].sizeofblocks;

i--;

}

else

{

array[i].sizeofblocks=n;

activtotalmemory-=n;

}

}

}

while(g!=5)

{

printf("\n--------------------------------------------");

printf("\n1)Total information\n2)Information about process\n3)Add new process\n4)Delete process\n5)Exit\n");

printf("\nYour choice: ");

scanf("%d",&g);

switch(g)

{

case 1:

{

largestmemoryblock=array[0].sizeofblocks;

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

{

if(largestmemoryblock<array[i].sizeofblocks)

{

largestmemoryblock=array[i].sizeofblocks;

}

}

printf("\n---------------------------------");

printf("\nTotal memory: %d",totalmemory);

printf("\nFree memory: %d",freememory);

printf("\nLargest memory block: %d",largestmemoryblock);

printf("\nFree blocks: %.2f%%",config);

printf("\nNumber of blocks: %d",numberofblocks);

printf("\nSize of blocks: ");

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

{

printf("%d ",array[i].sizeofblocks);

}

printf("\n---------------------------------");

break;

}

case 2:

{

printf("\n------------------------------------------------------------------------------");

printf("\n| n | NameProcess |NumberofProcess| sizeofblocks | ON |");

printf("\n------------------------------------------------------------------------------");

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

{

if(array[i].condition==1)

{

printf("\n|%3d.| | |%22d| - |",i+1,array[i].sizeofblocks);

}

else if(array[i].condition==0)

{

printf("\n|%3d.|%20s|%15d|%22d| + |",i+1,oper[array[i].number].name,oper[i].N,array[i].sizeofblocks);

}

printf("\n------------------------------------------------------------------------------");

}

printf("\n\n");

break;

}

case 3:

{

printf("\n---------------------------------");

printf("\nInput name for process: ");

scanf("%s",&oper[numberofprocess].name);

flag=1;

for(i=1;i<=numberofprocess && flag==1;i++)

{

if(oper[i].condition==1)

{

if(strcmp(oper[i].name,oper[numberofprocess].name)==0 && oper[i].condition==1)

{

flag=2;

}

while(flag==2)

{

col=1;

printf("\nThis name exist, create new name for process: ");

scanf("%s",&oper[numberofprocess].name);

for(i=0;i<numberofprocess && col==1;i++)

{

if(strcmp(oper[i].name,oper[numberofprocess].name)==0)

{

col=2;

}

}

if(col==1)

{

flag=0;

}

}

}

}

printf("\nInput size for process: ");

scanf("%d",&oper[numberofprocess].size);

largestmemoryblock=0;

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

{

if(largestmemoryblock<array[i].sizeofblocks && array[i].condition==1)

{

largestmemoryblock=array[i].sizeofblocks;

n=i;

}

}

if(largestmemoryblock>=oper[numberofprocess].size)

{

oper[numberofprocess].condition=1;

freememory-=oper[numberofprocess].size;

array[n].condition=0;

oper[numberofprocess].N=n;

array[n].number=numberofprocess;

success++;

printf("\nSuccess add\n");

}

else

{

printf("\nThere is no such memory block in which this process fits");

}

printf("\n---------------------------------");

n=0;

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

{

if(array[i].condition==1)

{

n++;

}

}

numberofprocess++;

config=n\*100/numberofblocks;

successful=success\*100/numberofprocess;

break;

}

case 4:

{

printf("\n----------------------------------------------");

printf("\nInput number of process which u want to OFF: ");

scanf("%d",&n);

if(n<numberofprocess)

{

freememory+=oper[n].size;

strcpy(oper[n].name,"");

oper[n].condition=0;

array[oper[n].N].condition=1;

oper[n].condition=0;

}

else

{

printf("Process doesn't exist");

}

n=0;

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

{

if(array[i].condition==1)

{

n++;

}

}

config=n\*100/numberofblocks;

printf("\n---------------------------------");

break;

}

}

}

return 0;

}