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: « П р о с т е й ш и е с х е м ы у п р а в л е н и я п а м я т ь ю »

Выполнил: студент гр. ИТ-11

Иванов А.А.

Принял: преподаватель
Петров И.И.

Гомель 2021

Цель работы: изучение алгоритмов управления памятью, разработка программы менеджера памяти. Разработать программу, реализующую заданный алгоритм выделения памяти.

1. Менеджер памяти должен: По запросу процесса выделять память, согласно заданного алгоритма (таблица). На экран должна выводиться следующая информация о состоянии памяти: объем памяти, объём 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. Организовывать циклическое выделение и освобождение памяти. При этом случайным образом задается количество выделяемых блоков и их размер.

Теория

Первые ОС применяли очень простые методы управления памятью. Вначале каждый процесс пользователя должен был полностью поместиться в основной памяти, занимать непрерывную область памяти, а система принимала к обслуживанию дополнительные пользовательские процессы до тех пор, пока все они одновременно помещались в основной памяти. Затем появился "простой свопинг" (система по-прежнему размещает каждый процесс в основной памяти целиком, но иногда на основании некоторого критерия целиком сбрасывает образ некоторого процесса из основной памяти во внешнюю и заменяет его в основной памяти образом другого процесса). В настоящее время они применяются в учебных и научно-исследовательских модельных ОС, а также в ОС для встроенных (embedded) компьютеров.

Имея дело с пакетными системами, можно обходиться фиксированными разделами и не использовать ничего более сложного. В системах с разделением времени возможна ситуация, когда память не в состоянии содержать все пользовательские процессы. Приходится прибегать к свопингу (swapping) – перемещению процессов из главной памяти на диск и обратно целиком. Частичная выгрузка процессов на диск осуществляется в системах со страничной организацией (paging) и будет рассмотрена ниже.

Выгруженный процесс может быть возвращен в то же самое адресное пространство или в другое. Это ограничение диктуется методом связывания.

Для схемы связывания на этапе выполнения можно загрузить процесс в другое место памяти.

Свопинг не имеет непосредственного отношения к управлению памятью, скорее он связан с подсистемой планирования процессов. Очевидно, что свопинг увеличивает время переключения контекста. Время выгрузки может быть сокращено за счет организации специально отведенного пространства на диске (раздел для свопинга). Обмен с диском при этом осуществляется блоками большего размера, то есть быстрее, чем через стандартную файловую систему. Во многих версиях Unix свопинг начинает работать только тогда, когда возникает необходимость в снижении загрузки системы.

Свопинг. Выгружается процесс, бывший в состоянии исполнение наиболее давно.

Листинг программы:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication1
{
   class D
   {
      public static int V, V_sv, n, k_a = 0, k_success = 0;
      public static Process[] proc;
      public static Process[] sw = new Process[20];
      public static void Create()
      {
         int t;
         Console.Write("\nВведите количество разделов памяти: ");
         n = Convert.ToInt32(Console.ReadLine());
         proc = new Process[n];
         Console.Write("Введите объём раздела памяти: ");
         t = Convert.ToInt32(Console.ReadLine());
         V = n * t;
         for (int i = 0; i < n; i++)
            proc[i] = new Process(t);
         V_sv = V;
         for (int i = 0; i < sw.Length; i++)
            sw[i] = new Process(t);
      }
      public static int Swap()
```

```
        {
            Console.Clear();
            int r = -1;
            if (proc[0].data != null) r = 0;
            if (r != -1)
            {
                int g = 0, ff = 0;
                while (g < sw.Length && ff == 0)
                {
                    if (sw[g].Name_p == null)
                    {
                        sw[g].Name_p = proc[0].Name_p;
                        sw[g].Mem_p = proc[0].Mem_p;
                        sw[g].data = proc[0].data;
                        ff = 1;
                    }
                    g++;
                }
                    V_sv += proc[0].V0;
                    for (int i = 0; i < n-1; i++)
                    {
                        proc[i].Mem_p = proc[i+1].Mem_p;
                        proc[i].Name_p = proc[i+1].Name_p;
                        proc[i].data = proc[i+1].data;
                    }
                proc[n-1].Mem_p = 0;
                proc[n-1].Name_p = null;
                proc[n-1].data = null;
                Console.WriteLine("Процесс, бывший в состоянии исполнения
наиболее давно выгружен");
            }
            else { Console.WriteLine("Процессов не обнаружено");
Console.ReadKey(); }
            return r;
        }
        public static void add()
        {
            Console.Clear();
            string name;
            int vv;
            Console.Write("\nВведите имя процесса: ");
            name = Console.ReadLine();
            Console.Write("\nВведите объём памяти требуемой данному
процессу(Кб): ");
            vv = Int32.Parse(Console.ReadLine());
```

```
            k_a++;
            if (vv > V) Console.WriteLine("Ошибка: объём всей памяти меньше
объёма доступной памяти процессу");
            else
            {
               if (vv <= proc[0].V0)
               {
                  int f = 0, i = 0;
                  while (i < proc.Length && f == 0)
                  {
                     if (proc[i].Name_p == null)
                     {

                        proc[i].Name_p = name;
                        proc[i].Mem_p = vv;
                        k_success++; f++;
                        proc[i].data = new string[vv];
                        V_sv = V_sv - proc[i].V0;
                     }

                     i++;
                  }

                  if (f == 0)
                  {
                     Swap();
                     proc[n-1].Name_p = name;
                     proc[n-1].Mem_p = vv;
                     k_success++;
                     proc[n-1].data = new string[vv];
                     V_sv = V_sv - proc[n-1].V0;

                  }
               }
               else { Console.WriteLine("Ошибка: объём раздела меньше объёма
требуемой памяти"); Console.ReadKey(); }
            }

         }
         public static void Vyvod()
         {
            Console.WriteLine("\nИнформация о процессах\n");
            int f = 0;
            for (int i = 0; i < proc.Length; i++)
            {
```

```
            if (proc[i].Mem_p != 0)
            { Console.WriteLine("Процесс: {0,-9}Объём памяти(Кб): {1,-5}
Номер блока памяти: {2} ", proc[i].Name_p, proc[i].Mem_p, i + 1); f++; }

         }
         if (f == 0) Console.WriteLine("Нет процессов.");

      }
      public static void vgruzka()
      {
         Console.Clear();
         string name;
         int vv;
         name = sw[0].Name_p;
         vv = sw[0].Mem_p;
         k_a++;
         if (proc[2].Mem_p!=0) Console.WriteLine("Ошибка: невозможно
вгрузить процесс");
         else
         {
            if (vv <= proc[0].V0)
            {
               int f = 0, i = 0;
               while (i < proc.Length && f == 0)
               {
                  if (proc[i].Name_p == null)
                  {

                     proc[i].Name_p = name;
                     proc[i].Mem_p = vv;
                     k_success++; f++;
                     proc[i].data = new string[vv];
                     V_sv = V_sv - proc[i].V0;
                  }

                  i++;
               }
                  for (i = 0; i < sw.Length - 1; i++)
                  {
                     sw[i].data = sw[i + 1].data;
                     sw[i].Mem_p = sw[i + 1].Mem_p;
                     sw[i].Name_p = sw[i + 1].Name_p;
                  }
                  sw[i].data = null;
```

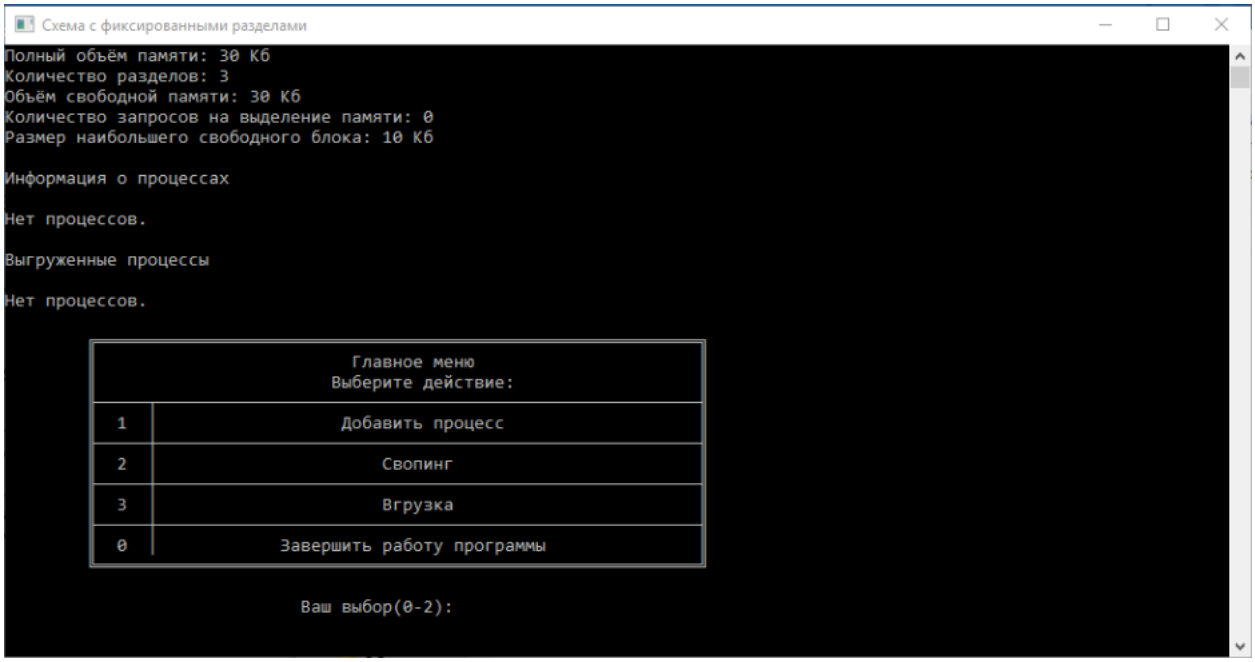
```
                    sw[i].Mem_p = 0;
                    sw[i].Name_p = null;
                if (f == 0)
                {
                    //Swap();
                    proc[n - 1].Name_p = name;
                    proc[n - 1].Mem_p = vv;
                    k_success++;
                    proc[n - 1].data = new string[vv];
                    V_sv = V_sv - sw[0].V0;

                }
            }
            else { Console.WriteLine("Ошибка: объём раздела меньше объёма
требуемой памяти"); Console.ReadKey(); }
        }

    }
    public static void SW()
    {
        Console.WriteLine("\nВыгруженные процессы\n");
        int f = 0;
        for (int i = 0; i < sw.Length; i++)
        {
            if (sw[i].Mem_p != 0)
            { Console.WriteLine("Процесс: {0,-9}Объём памяти(Кб): {1,-5} ",
sw[i].Name_p, sw[i].Mem_p); f++; }
        }
        if (f == 0) Console.WriteLine("Нет процессов.");

    }
    public static void MaxR()
    {
        int temp;
        int max = -1;
        for (int i = 0; i < proc.Length; i++)
        {
            temp = proc[i].V0 - proc[i].Mem_p;
            if (temp > max) max = temp;
        }
        if (max != -1) Console.WriteLine("Размер наибольшего свободного
блока: " + max + " Кб");
    }
    public static void KolPercent()
    {
```

```
            if (k_a != 0)
            {
                float k = 100 * (float)k_success / k_a;
                Console.WriteLine("Количество удовлетворённых запросов: {0,-
4:f1}%", k);
            }
        }
    }
    class Process
    {
        int mem_p, V_razd;
        string name_p;
        public Process(int V_r) { V_razd = V_r; }
        public string[] data;
        public int V0
        {
            get { return V_razd; }
            set { V_razd = value; }
        }
        public int Mem_p
        {
            get { return mem_p; }
            set { mem_p = value; }
        }
        public string Name_p
        {
            get { return name_p; }
            set { name_p = value; }
        }

    }

    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "Схема с фиксированными разделами ";
            int k = 12;
            D.Create();
            do
            {
                Console.Clear();
                Console.WriteLine("Полный объём памяти: " + D.V + " Кб");
                Console.WriteLine("Количество разделов: " + D.n);
                Console.WriteLine("Объём свободной памяти: " + D.V_sv + " Кб");
```

```
            Console.WriteLine("Количество запросов на выделение памяти: " +
D.k_a);
            D.KolPercent();
            D.MaxR();
            D.Vyvod();
            D.SW();

Console.WriteLine("\n\t╔═══════════════════════════════════════════════════════
═══════════════════════════════════════╗");
            Console.WriteLine("\t║                    Главное меню                    ║");
            Console.WriteLine("\t║                 Выберите действие:
                ║");

Console.WriteLine("\t╟──────┬────────────────────────────────────────────────────
──────────────────────────────────────╢");
            Console.WriteLine("\t║  1   │              Добавить процесс
                ║");

Console.WriteLine("\t╟──────┼────────────────────────────────────────────────────
──────────────────────────────────────╢");
            Console.WriteLine("\t║  2   │                 Свопинг                 ║");

Console.WriteLine("\t╟──────┼────────────────────────────────────────────────────
──────────────────────────────────────╢");
            Console.WriteLine("\t║  3   │                 Вгрузка                 ║");

Console.WriteLine("\t╟──────┼────────────────────────────────────────────────────
──────────────────────────────────────╢");
            Console.WriteLine("\t║  0   │          Завершить работу программы
                ║");

Console.WriteLine("\t╚═══════════════════════════════════════════════════════
═══════════════════════════════════════╝");
            Console.Write("\n\t\t\t     Ваш выбор(0-2): ");
            k = Convert.ToInt32(Console.ReadLine());
            switch (k)
            {
               case 1: D.add(); break;
               case 2: D.Swap(); break;
               case 3: D.vgruzka();break;
               case 0: Console.Clear(); Console.WriteLine("Выход..."); break;
               default: break;
            }
         }
         while (k != 0);
```

```
            Console.ReadKey();
        }
    }
}
```

Результат выполнения программы:



Вывод: в ходе лабораторной работе была разработана программа реализующая свопинг, который выгружает процесс бывший в исполнении наиболее давно.