Федеральное государственное бюджетное образовательное учреждение высшего образования

ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Физико-технический институт

Отчет по лабораторной работе №2

Дисциплина: «Операционные системы»

«Исследование принципов реализации процессов и очередей многозадачного ядра»

Выполнил: студент 4 курса

Физико-технического института

Группы 21418

Климов Артур Юрьевич

Преподаватель: кандидат физико-математических наук,

Соловьев Алексей Владимирович

Петрозаводск 2018

Оглавление

1. Техническое задание.

1.1 Назначение программы.

Программа предназначена для реализации процессов и очередей в ядре многозадачной среды.

1.2 Требования к информационной совместимости.

Программа разрабатывается на языке Pascal с помощью оболочки Turbo Pascal. Операционная система – Windows.

1.3 Требования к техническим средствам.

Программа должна запускаться на ЭВМ, технические характеристики которой позволяют работать в операционной системе на основе ядра Windows.

1.4 Требования к программной документации.

Программная документация должна содержать:
– Текст программы;
– Описание программы.

1.5 Срок сдачи программы.

Срок сдачи программы – 31.12.2018.

2. Текст программы.

2.1 Текст программы. (Приложение 1)

3. Описание программы.

3.1 Общее описание.

Данная программа предназначена для реализации процессов и очередей в ядре многозадачной среды. Программа написана на языках Pascal и Assembler. Для компиляции программы использован компилятор программной оболочки Turbo Pascal. Программа должна запускаться в системе Windows.

3.2 Блок-схема (Приложение 2).

3.3 Логическая структура программы.

3.3.1 Объекты.

Process (Процесс) состоит из 2 полей типа word для регистров ss и sp, поля Stack с типом указатель на массив из 1000 значений типа word, поля Next с типом указатель на процесс; Конструктором, деструктором, процедурами на установление и выдачи значения поля Next.

List (очередь) состоит из поля Start с типом указатель на процесс; Конструктором, деструктором, процедурами добавления и удаления объектов из очереди, функцией выдачи значения первого процесса в очереди.

Ready_List (очередь готовых) состоит из поля-указателя на текущий процесс с типом указатель на процесс; конструктором, деструктором; процедурами запуска, остановки, добавления процесса в очередь и активизации процесса в очереди.

3.3.2 Процедуры и функции.

Process.NewProcess(body : Myproc) – конструктор для процессов. Получает на вход тип Procedure. Выделяет память под стек, устанавливает значения для регистров ss и sp.

Process.DeleteProcess – деструктор, очищает память, выделенную под стэк.

Process.SetNext – процедура, устанавливает указатель на следующий процесс.

Process.GetNext : Ptr_Process – функция, выдает указатель на следующий процесс.

DisableInterrupt – процедура, запрещает прерывания.

EnableInterrupt – процедура, разрешает прерывания.

Transfer – процедура, передает управление другому процессу.

Handler – процедура, восстанавливает старый вектор прерываний, запрещает прерывания, вызывает процеду ActivateNext.

List.NewList – конструктор, инициализирует указатель на старт очереди.

List.DeleteList – деструктор, удаляет все процессы из памяти.

List.IncludeProcess – процедура, включает процесс в очередь.

List.ExcludeProcess – процедура, удаляет процесс из очереди.

List.GetStart : Ptr_Process– функция, выдает указатель на начало очереди.

Ready_List.StartReadyList – процедура, запрещает прерывания, запоминает вектор текущего процесса, устанавливает этот вектор на свободный, устанавливает в вектор $8 новый процесс, удаляет процесс из очереди, передает управление.

Ready_List.StopReadyList – процедура, запрещает прерывания, устанавливает в вектор $8 старый вектор, передает управления в главную программу.

Ready_List.AddNewProcess – процедура, добавляет новый процесс в очередь.

Readly_List.ActivateNext – процедура, активизирует новый процесс.

3.3.3 Выходные данные.

Названия процессов и счетчик их включений.

3.3.4 Вызов и загрузка.

Программа распространяет в виде исходного текста на языке Pascal и Assembler. Перед началом программы её необходимо скомпилировать.

3.3.5 Компиляция.

Происходит засчет компилятора оболочки Turbo Pascal.

Приложение 1.

Текст модуля.

```
Unit MODULE;
{$F+}
Interface
Type
   Artype      = array [0..999] of word; {array for a stack}
   Ptr_Stack   = ^Artype;  {pointer for array}
   Myproc      = procedure;
   Ptr_Process = ^Process;  {pointer for an object}
   Process = Object
         ssreg : word;
         spreg : word;
         Stack : Ptr_Stack;
         Next  : Ptr_Process;
         Constructor NewProcess (body : Myproc);
         Destructor DeleteProcess;
         Procedure SetNext (P : Ptr_Process);
         Function GetNext : Ptr_Process;
   End;
   List = Object
       Start : Ptr_Process;
       Constructor NewList;
       Destructor DeleteList;
       Procedure IncludeProcess (Current : Ptr_Process);
       Procedure ExcludeProcess (Current : Ptr_Process);
       Function GetStart : Ptr_Process;
   End;

    Ready_List = Object(List)
           CurrentProcess : Ptr_Process;
           Constructor NewReadyList;
           Destructor DeleteReadyList;
           Procedure StartReadyList;
           Procedure StopReadyList;
           Procedure AddNewProcess (P : Myproc);
           Procedure ActivateNext;
   End;
Var
     ReadyList : Ready_List;
     Main : Ptr_Process;
     Vector : pointer;
Implementation
Uses DOS;
Procedure DisableInterrupt; Assembler;
Asm
     cli
End;
Procedure EnableInterrupt; Assembler;
Asm
     sti
End;
Constructor Process.NewProcess (body : Myproc);
Begin
    New(Stack);
```

```
    ssreg := seg(Stack^);
    spreg := ofs(Stack^) + 1998 - 14;
    memw[ssreg:spreg+2] := ofs(body);
    memw[ssreg:spreg+4] := seg(body);
End;
Destructor Process.DeleteProcess;
Begin
    Dispose(Stack);
End;
Procedure Process.SetNext (P : Ptr_Process);
Begin
    Next := P;
End;
Function Process.GetNext : Ptr_Process;
Begin
    GetNext := Next;
End;
{$F+}
Procedure Transfer(OldProc, NewProc : Ptr_Process); Assembler;
Asm
  les di, OldProc
  mov es:[di], ss
  mov es:[di+2], sp
  les di, NewProc
  mov ss, es:[di]
  mov sp, es:[di+2]
  sti
  pop bp
  ret 8
End;
Procedure Handler; Interrupt;
Begin
    Asm Int 78h End;
    DisableInterrupt;
    ReadyList.ActivateNext;
End;
Procedure Idler;
Begin
    while true do
    begin
    end;
End;
Constructor List.NewList;
Begin
    Start := nil;
End;
Destructor List.DeleteList;
Var
  Current : Ptr_Process;
Begin
    while Start <> nil do
    begin
        Current := Start;
        Start := Start^.Next;
        Dispose(Current, DeleteProcess);
 end;
End;
```

```
Procedure List.IncludeProcess(Current : Ptr_Process);
Var
  P1 : Ptr_Process;
  P2 : Ptr_Process;
Begin
    P1 := GetStart;
    P2 := nil;
    while (P1 <> nil) do   {to the last process}
    begin
        P2 := P1;
        P1 := P1^.GetNext;
    end;
    if P2 = nil then    {if there is not any process}
      Start := Current
    else
      P2^.SetNext(Current); {the last setnext = this process}
    Current^.SetNext(P1);    {nil}
End;
Procedure List.ExcludeProcess (Current : Ptr_Process);
Var
  S : Ptr_Process;
Begin
    if Current = Start then    {if it is the start}
      Start := Current^.GetNext  {then we use the next process}
    else
    begin
        S := GetStart;  {memorize the start}
        while S^.GetNext <> Current do  {if the process has current as the next}
        begin
            S := S^.GetNext
        end;
        S^.SetNext(Current^.GetNext); {to pereskakivaem cherez process}
    end;
End;
Function List.GetStart : Ptr_Process;
Begin
    GetStart := Start;
End;
Constructor Ready_List.NewReadyList;
Begin
    List.NewList;
End;
Destructor Ready_List.DeleteReadyList;
Begin
    List.DeleteList;
End;
Procedure Ready_List.StartReadyList;
Begin
  DisableInterrupt;
  GetIntVec($08, Vector);  {remember vector}
  SetIntVec($78, Vector);  {to free vector}
  SetIntVec($08, Addr(Handler));
  CurrentProcess := Start;
  ExcludeProcess(CurrentProcess);
  Transfer(Main, CurrentProcess);
End;
Procedure Ready_List.StopReadyList;
```

```
Begin
  DisableInterrupt;
  SetIntVec($08, Vector);
  Transfer(CurrentProcess, Main);
End;
Procedure Ready_List.AddNewProcess(P : Myproc);
Var
   Current : Ptr_Process;
Begin
     DisableInterrupt;
     New(Current);
     Current^.NewProcess(P);
     IncludeProcess(Current);
     EnableInterrupt;
End;
Procedure Ready_List.ActivateNext;
Var
   P : Ptr_Process;
Begin
     DisableInterrupt;
     P := CurrentProcess;
     IncludeProcess(P);
     CurrentProcess := Start;
     ExcludeProcess(CurrentProcess);
     Transfer(P, CurrentProcess);
End;
Begin
     New(Main);
End.
```

Текст программы.

```
Program Lab2;

uses MODULE, CRT;

Procedure User_1; far;
Var
   i : integer;
   j : integer;
Begin
    i := 0;
    j := 0;
    while true do begin
        j := j + 1;

        if j = 10000 then
        begin
            i := i + 1;
            writeln('User_1 i =', i);
            j := 0;
            delay(100);
        end;

    end;
End;
Procedure User_2; far;
```

```
Var
          i : integer;
 j : integer;
Begin
   i := 0;
   j := 0;
   while true do begin

        j := j + 1;
        if j = 10000 then
        begin
          i := i + 1;
          writeln('User_2 i =', i);
          j := 0;
          delay(100);
      end;
   end;
End;
Procedure User_3; far;
Var
  i : integer;
  j : integer;
Begin
    i := 0;
    j := 0;
    while true do begin
         j := j + 1;
         if j = 10000 then
         begin
             i := i + 1;
             writeln('User_3 i =', i);
             j := 0;
             delay(100);
         end;
         if i = 30 then ReadyList.StopReadyList;
    end;
End;
Begin
   ReadyList.NewReadyList;
   ReadyList.AddNewProcess(User_1);
   ReadyList.AddNewProcess(User_2);
   ReadyList.AddNewProcess(User_3);
   writeln('Processes added');
   ReadyList.StartReadyList;
   ReadyList.DeleteReadyList;
   writeln('Memory is cleaned');
   readln;
End.
```

Приложение 2.