**1.** **Задание**

Необходимо написать программу, реализующую параллельную работу нескольких процессов. Каждый процесс может состоять из одного или нескольких потоков. В зависимости от задания, какие-то из потоков, работающих в составе этих процессов, могут быть приостановлены и вновь запущены некоторой определенной клавишей (можно использовать буквенные или цифровые клавиши). Скорость работы некоторых потоков (в зависимости от задания) тоже должна меняться нажатием клавиш. Нажатия клавиш обрабатывать с помощью прерывания от клавиатуры (по материалам лаб. работы №1).

Окончание работы программы должно происходить при приостановке всех потоков их ключевыми клавишами либо при нажатии клавиши ESC. При окончании работы необходимо выполнить корректное завершение, т.е. «дочитать» всю информацию из буфера каждого процесса (при его наличии), закрыть все открытые файлы и т.п. – по материалам лаб. работы №3.

Те потоки, которые выводят информацию на экран, должны использовать для этого каждый свое отдельное окно, обрамленное рамкой – экран визуально делится на несколько окон, в каждом из которых отображается один процесс.

Дополнительное информационное окно должно содержать описание управляющих клавиш и краткое сообщение о тех действиях, которые выполняются в программе: номер потока, его статус в текущий момент времени (активен, приостановлен, ожидает), другую информацию (указание величины выделенного потоку кванта времени, процент заполненности буфера и т.п.) в зависимости от конкретного потока.

*Особенности задач:*

Задачи 1, 3, 9 предполагают наличие управляющего блока, который, используя прерывания таймера, случайным образом определяет очередной активный поток и выделяет ему кванты времени. Кванты времени для потоков изначально задавать некоторой фиксированной величины, но предусмотреть возможность её изменения во время работы (во всех вариантах) путем нажатия некоторых ключевых клавиш (для каждого из потоков предусмотреть свою клавишу) – т.е. можно, например, увеличить или уменьшить квант только потока-производителя. Любой из потоков (тоже во всех вариантах) может быть приостановлен или вновь запущен некоторой ключевой клавишей.

Потоки этого класса задач могут иметь три статуса: «активен», «ожидает» или «приостановлен». В процессе работы может возникнуть, например, следующая ситуация. Поток-потребитель приостановлен своей ключевой клавишей, следовательно, буфер не освобождается. Поток-производитель активен, он заполнит буфер информацией и перейдёт в состояние ожидания. Из этого состояния он сможет выйти только после того, как будет возобновлена (нажатием клавиши) работа потока-потребителя, который освободит место в буфере для помещения новой информации. В случае приостановки производителя возникнет аналогичная ситуация, только с пустым буфером. Приостановка/возобновление потока возможны в любом его состоянии – как в активном, так и в состоянии ожидания.

Задачи 2, 4, 5, 6, 7, 8, 10, 11, 12 должны выполняться равномерно, независимо от степени загрузки системы. Для этого каждой из них необходимо получать управление через фиксированное количество «тиков» системного таймера, во время которого они выполнят какое-то свое элементарное действие («бегущая строка» или «летающий объект» сместится на одну позицию, сменится нота в музыке…). При такой реализации скорость каждого потока будет определяться количеством «тиков» таймера между его запусками. При соответствующем задании для уменьшения скорости такого потока достаточно после нажатия ключевой клавиши предоставлять ему управление реже, через большее число «тиков», соответственно для ускорения такого потока – опять же после нажатия ключевой клавиши – ему предоставляется управление чаще, в пределе – на каждом «тике». Так, «бегущая строка» должна двигаться равномерно с постоянной скоростью (если она не приостановлена ключевой клавишей) независимо от количества активных процессов в системе, музыка – тоже играть равномерно… Потоки этого класса задач могут иметь два статуса: «активен» и «приостановлен».

*Задания:*

1. Два потока: первый читает информацию из файла (например, стихи или текст программы) в буфер, второй эту информацию из буфера выдаёт на экран. При заполнении окна вывода до конца его содержимое не должно обновляться полностью – вывод новой информации должен осуществляться в последнюю строку, а все остальные строки смещаться вверх (по материалам лаб. работы №4). Имя читаемого файла задавать как параметр командной строки. После окончания файла он начинает считываться заново.

6. Вывод заголовка работы (фамилия и имя автора и название работы) в виде “бегущей строки” (по материалам лаб. работы №5). Место вывода (верхняя или нижняя строка экрана) задавать параметром командной строки.

Исправить на это

6. Вывод заголовка работы (фамилия и имя автора) в виде «летающей строки». Строка должна совершать свободное движение по выделенной области экрана, отражаясь от её границ согласно законам физики. Координаты стартовой точки задавать параметрами командной строки.

10. В углу экрана с заданными в качестве параметров координатами поместить часы, показания которых должны соответствовать системному времени. Обновлять показания часов каждую секунду (по материалам лаб. работы №3).

**Параметры командной строки.**

Программа принимает три параметра – положение заголовка (a – вверху, b – внизу), положение часов (1 - левый верхний угол; 2 - правый верхний угол; 3 - левый нижний угол; 4 - правый нижний угол) и имя файла с текстом.

**Описание основных алгоритмов, используемых в программе.**

В программе имеются потоки: поток-производитель-1 – для чтения информации из текстового файла и записи ее в буфер, поток-потребитель-1 – для чтения этой информации из буфера и вывода ее на экран, поток-производитель-2 – для поиска чисел, которые являются полными квадратами и записи их в буфер, поток-потребитель-2 – для чтения этих чисел из буфера и вывода их на экран.

Файл с текстом поток-производитель-1 читает посимвольно. Если файл прочитан полностью, то он открывается снова для чтения из него сначала. Полученная из символов строка заносится в буфер. А если в буфере нет свободного места, то поток переходит в состояние ожидания. Чтение из файла продолжается до тех пор, пока не истечет время, отведенное на работу данного потока, или пока буфер не будет заполнен.

Поток-потребитель-1 читает строки из буфера и выводит их на экран, обеспечивая скроллинг при заполнении окна для вывода полностью. А если буфер становится пустым, то поток переходит в состояние ожидания. Чтение из буфера продолжается до тех пор, пока не истечет время, отведенное на работу данного потока, или пока буфер не станет пустым.

Поток-производитель-2 для поиска чисел использует следующий алгоритм. Генерирует случайное число до тех пор, пока дробная часть от корня из него не будет меньше 10-4. Найденное число поток заносит в буфер. А если в буфере нет свободного места, то поток переходит в состояние ожидания. Запись в буфер поток продолжает до тех пор, пока не истечет время, отведенное на его работу, или пока буфер не будет заполнен.

Поток-потребитель-2 читает эти числа из буфера и отображает их в отдельном окне, обеспечивая скроллинг при заполнении окна для вывода полностью. А если буфер становится пустым, то поток переходит в состояние ожидания. Чтение из буфера продолжается до тех пор, пока не истечет время, отведенное на работу данного потока, или пока буфер не станет пустым.

Работа потоков-производителей и потоков-потребителей организована по принципу очереди. Числа на экране рядом с названиями потоков – это количество “тиков” таймера, отведенных каждому из этих потоков, и которое можно уменьшить или увеличить, используя ключевые клавиши. За одну секунду происходит 18,2 тиков таймера.

После каждого занесения (и извлечения) информации из буферов обновляются показания уровня заполненности буферов на экране.

Движение бегущей строки выполняется равномерно, независимо от степени загрузки системы. В процессе работы программы скорость бегущей строки можно уменьшить, используя клавишу ‘0’или увеличить, используя клавишу ‘9’. Эта скорость определяются количеством “тиков” таймера между запусками движения бегущей строки. Числа на экране – это количество “тиков” таймера, которое можно уменьшить или увеличить, используя ключевые клавиши.

Показания часов на экране обновляем с каждой секундой, увеличивая счетчик импульсов и находя наиболее близкий импульс для обновления показаний времени.

Работу любого из потоков, бегущей строки и часов можно приостановить и снова запустить, используя клавиши от F1 до F6, указанные слева от наименований потоков.

**Описание основных переменных, констант и типов.**

В программе для фиксирования состояний (статусов) процессов и потоков определён следующий тип:

type Status = (Assigned, Wait, Suspended);

*Константы.*

Title – текст заголовка;

ImpulsesPerSec – количество импульсов системных часов в секунду;

Buf1Len – размер буфера первого процесса;

Buf2Len – размер буфера второго процесса;

Delay1 – задержка для первого процесса;

Delay2 – задержка для второго процесса;

StrLen – длина строки окна вывода;

Kill – флаг завершения программы;

*Переменные.*

OldTHandler, OldKHandler – старые обработчики прерываний;

Scr – массив для прямого доступа к видеопамяти;

T – счётчик времени таймера;

Thread – состояния потоков;

FIFO1 – буфер первого процесса;

Buf2 – буфер второго процесса;

Top1, Top2 – вершины буферов;

Time – времена для потоков;

CurThread – номер текущего активного потока;

X1, Y1, X2, Y2 – координаты экрана для вывода сгенерированных чисел;

Speed - скорость движения заголовка.

x, y - координаты расположения заголовка на экране.

impulsesCounter - счетчик импульсов для вывода заголовка.

renewPauseSec - пауза между обновлениями показаний часов в секундах.

renewPause - пауза между обновлениями показаний часов в количестве импульсов таймера.

timePos - расположение часов на экране (1, 2, 3, 4).

xTime, yTime - координаты расположения часов на экране.

Hour - часы.

Min - минуты.

Sec – секунды.

sec100 - сотые доли секунды.

timeImpulsesCounter - счетчик импульсов для часов.

nextRenew - следующее количество импульсов, при котором будем обновлять показания часов .

**Описание подпрограмм.**

Print – процедура вывода строки на экран;

WriteTitle - процедура вывода заголовка на экран;

LeadingZero – эта функция переводит часы, минуты или секунды в строку с впереди стоящим нулем, если нужно;

ShowTime – процедура отображает часы на экране;

RenewTime – процедура увеличивает время на необходимое количество секунд;

THandler – обработчик прерываний таймера;

PrintStatus – вывод на экран состояния заданного потока;

ChangeStatus – изменение состояния заданного потока;

Prod1, Cons1, Prod2, Cons2 – процедуры потоков потребителей и производителей;

PrimeNumber – функция проверки числа на простоту;

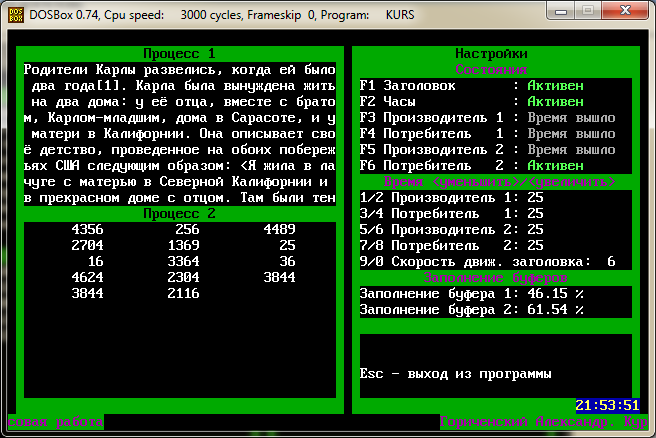
KHandler – обработчик прерываний клавиатуры;

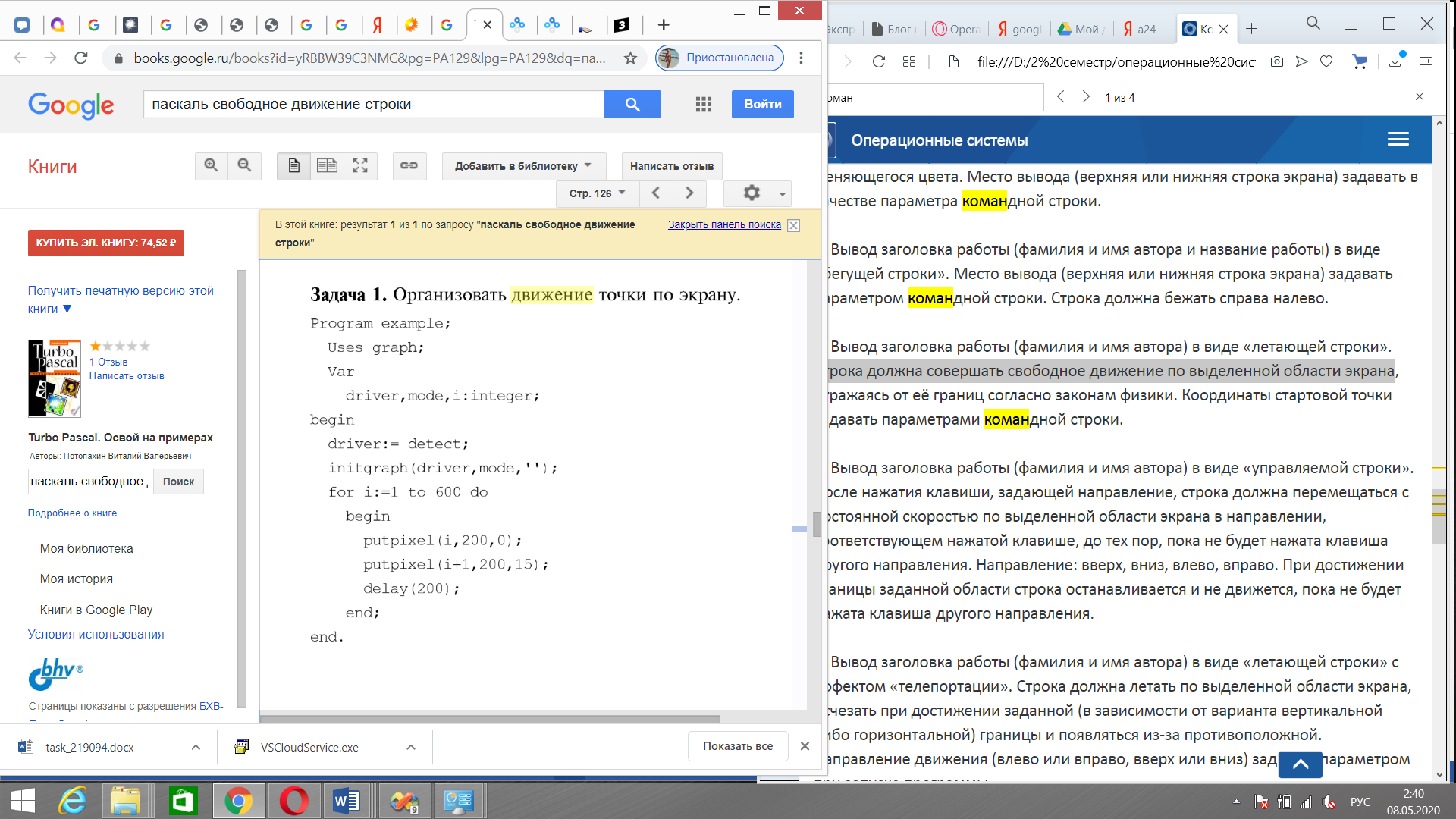
SetCurSize – процедура изменения размера курсора;

Initialize – инициализирует основные переменные;

KillProc – процедура завершения работы программы.

**Работа программы.**

**



**Исходный код программы.**

{$M $1024, 0, 0}

uses

Crt, Dos;

type

Status = (Active, Waits, Suspended); { Состояния потока }

const

{ Текст заголовка }

## Title: String = 'Юлия Кручинина. Курсовая работа';

{ количество импульсов системных часов в секунду }

impulsesPerSec = 18.2;

Buf1Len = 13; { Размер буфера первого процесса }

Buf2Len = 13; { Размер буфера второго процесса }

Delay1 = 1000; { Задержка для первого процесса }

Delay2 = 1000; { Задержка для второго процесса }

Range = 6000; { Диапазон чисел, гененируемых вторым процессом }

StrLen = 39; { Длина строки окна вывода }

Kill: Boolean = False; { Флаг завершения программы }

var

OldTHandler, OldKHandler: Procedure; { Старые обработчики прерываний }

F: Text; { Файл с текстом }

Scr: Array[1..25, 1..80] of Record { Массив для прямого доступа к видеопамяти }

Symbol: Char;

Attr: Byte;

end Absolute $B800:$0000;

T: Word; { Счётчики времени таймера }

Thread: Array[1..6] of Status; { Состояния потоков }

LP: Byte; { Номер строки вывода потока-потребителя 1 }

FIFO1: Array[1..Buf1Len] of String; { Буфер первого процесса }

Buf2: Array[1..Buf2Len] of Longint; { Буфер второго процесса }

Top1, Top2: Word; { Вершины буферов }

Time: Array[1..4] of Byte; { времена для потоков }

CurThread: Byte; { Номер текущего активного потока }

X1, Y1, X2, Y2: Byte; { Координаты экрана для вывода сгенерированных чисел }

Num: Longint;

speed: Integer; { скорость движения заголовка }

x: Integer; { координаты расположения заголовка на экране }

y: Integer;

impulsesCounter: Longint; { счетчик импульсов для вывода заголовка }

renewPauseSec: Word; { пауза между обновлениями показаний часов в секундах }

renewPause: Real; { пауза между обновлениями показаний часов в

количестве импульсов таймера }

timePos: Integer; { расположение часов на экране (1, 2, 3, 4) }

xTime: Integer; { координаты расположения часов на экране }

yTime: Integer;

hour: Word; { часы }

min: Word; { минуты }

sec: Word; { секунды }

sec100: Word; { сотые доли секунды }

timeImpulsesCounter: Longint; { счетчик импульсов для часов }

nextRenew: Real; { следующее количество импульсов, при котором будем }

{ обновлять показания часов }

{ Вывод строки S на экран начиная с i-го столбца, j-й строки цветом Color }

procedure Print(const S: String; const i, j: Byte; const Color: Byte);

var

k: Integer;

begin

if i + Length(S) - 1 > 80 then Exit;

for k := i to i + Length(S) - 1 do begin

Scr[j, k].Attr := Color;

Scr[j, k].Symbol := S[k - i + 1];

end;

end;

procedure WriteTitle; { выводит заголовок на экран }

var

newX: Integer;

i: Integer;

begin

for i := 1 to 80 do { Очистка строки, в которой движется заголовок }

Scr[y][i].Attr := 0;

newX := x;

for i := 1 to Length(Title) do { выводим заголовок по символам }

begin

Scr[y][newX].Symbol := Title[i];

Scr[y][newX].Attr := 37;

Inc(newX);

if newX > 80 then { если заголовок выходит за пределы экрана }

newX := 1;

end;

end;

function LeadingZero(w : Word) : String; { переводит часы, минуты или }

var { секунды в строку с впереди стоящим нулем, если нужно}

s : String;

begin

Str(w:0,s);

if Length(s) = 1 then

s := '0' + s;

LeadingZero := s;

end;

procedure ShowTime; { отображает часы на экране }

begin

Print(LeadingZero(hour) + ':' + LeadingZero(min) + ':' + LeadingZero(sec),

xTime, yTime, 30);

end;

procedure RenewTime; { увеличивает время на необходимое количество секунд }

var

k: Integer;

begin

Inc(sec, renewPauseSec); { увеличиваем секунды }

if sec > 59 then { проверяем, не надо ли увеличить минуты }

begin

k := sec;

sec := sec mod 60; { остаток от деления на 60 в секунды }

k := k div 60; { целая часть от деления станет добавляемыми минутами }

Inc(min, k);

if min > 59 then { проверяем, не надо ли увеличить часы }

begin

k := min;

min := min mod 60;

k := k div 60;

Inc(hour, k);

if hour > 23 then { проверяем, не наступили ли новые сутки }

hour := hour mod 24;

end;

end;

end;

{ Обработчик прерываний системного таймера }

{$F+}

procedure THandler; Interrupt;

begin

if Thread[1] = Active then begin

Inc(impulsesCounter);

if impulsesCounter mod speed = 0 then begin

Dec(x); { уменьшаем координату }

if x < 1 then { если заголовок выходит за пределы экрана }

x := 80;

WriteTitle;

impulsesCounter := 0;

end;

end;

if Thread[2] = Active then begin

Inc(timeImpulsesCounter); { увеличиваем счетчик импульсов }

{ находим наиболее близкий импульс для обновления показаний времени }

if abs(nextRenew - timeImpulsesCounter) < 0.5 then begin

RenewTime;

ShowTime;

{ определяем когда будем обновлять время в следующий раз }

nextRenew := nextRenew + renewPause;

end;

end;

Inc(T);

Inline($9C);

OldTHandler;

end;

{$F-}

{ Инициализирует время }

procedure InitializeTime;

begin

speed := 6;

WriteTitle;

impulsesCounter := 0;

renewPauseSec := 1;

{ Преобразуем паузу обновления показаний времени в секундах в

количество импульсов }

renewPause := renewPauseSec \* impulsesPerSec;

{ определяем системное время }

GetTime(hour, min, sec, sec100);

{ определяем начальное количество импульсов, чтобы не потерять

доли секунды }

timeImpulsesCounter := Round(sec100 / 100 \* impulsesPerSec);

nextRenew := renewPause; { устанавливаем когда нужно делать первое }

ShowTime; { обновление показаний часов }

end;

{ Вывод на экран состояния i-го потока }

procedure PrintStatus(const i: Byte);

var

\_Y: Byte;

begin

case Thread[i] of

Active: Print('Активен ', 66, 3 + i, LightGreen);

Waits: Print('Ожидание ', 66, 3 + i, Yellow);

Suspended: Print('Остановлен ', 66, 3 + i, LightRed);

end;

end;

{ Изменение состояния i-го потока }

procedure ChangeStatus(const i: Byte);

begin

case i of

1:

begin

if Thread[1] = Active then

Thread[1] := Suspended

else

Thread[1] := Active;

PrintStatus(1);

end;

2:

begin

if Thread[2] = Active then

Thread[2] := Suspended

else begin

Thread[2] := Active;

InitializeTime;

end;

PrintStatus(2);

end;

3:

begin

if Thread[3] <> Suspended then

Thread[3] := Suspended

else

if CurThread = 0 then Thread[3] := Active else Thread[3] := Waits;

PrintStatus(3);

end;

4:

begin

if Thread[4] <> Suspended then

Thread[4] := Suspended

else

if CurThread = 1 then Thread[4] := Active else Thread[4] := Waits;

PrintStatus(4);

end;

5:

begin

if Thread[5] <> Suspended then

Thread[5] := Suspended

else

if CurThread = 2 then Thread[5] := Active else Thread[5] := Waits;

PrintStatus(5);

end;

6:

begin

if Thread[6] <> Suspended then

Thread[6] := Suspended

else

if CurThread = 3 then Thread[6] := Active else Thread[6] := Waits;

PrintStatus(6);

end;

end;

end;

{ Поток-производитель первого процесса }

procedure Prod1;

var

i: Integer;

c: Char;

S: String;

Z: String[5];

Num: Word;

begin

if Thread[3] = Suspended then Exit;

if (Thread[3] = Waits) and (Top1 <> Buf1Len) then Thread[3] := Active;

PrintStatus(3);

T := 0;

while (T < Time[1]) and (Thread[3] <> Suspended) do begin

if Top1 = Buf1Len then begin

Thread[3] := Waits;

PrintStatus(3);

Break;

end;

i := 0;

S := '';

while not EOF(F) and (i < StrLen) do begin

Read(F, c);

if c = #13 then Break;

if c = #10 then Continue;

S := S + c;

Inc(i);

end;

while Length(S) <> StrLen do

S := S + ' ';

if EOF(F) then Reset(F);

Inc(Top1);

FIFO1[Top1] := S;

Str(Top1 / Buf1Len \* 100:2:2, Z);

Print(Z + ' % ', 66, 17, White);

Delay(Delay1);

end;

if Thread[3] = Active then begin

if Time[1] = 0 then

Delay(Delay1);

Print('Время вышло', 66, 6, LightGray);

end;

end;

{ Поток-потребитель первого процесса }

procedure Cons1;

var

i, j: Integer;

S: String;

begin

if Thread[4] = Suspended then Exit;

if (Thread[4] = Waits) and (Top1 <> 0) then Thread[4] := Active;

PrintStatus(4);

T := 0;

while (T < Time[2]) and (Thread[4] <> Suspended) do begin

if Top1 = 0 then begin

Thread[4] := Waits;

PrintStatus(4);

Break;

end;

if LP = 9 then

for i := 3 to 10 do

for j := 3 to 41 do

Scr[i, j].Symbol := Scr[i + 1, j].Symbol

else

Inc(LP);

Print(FIFO1[1], 3, 2 + LP, White);

Dec(Top1);

for i := 1 to Top1 do

FIFO1[i] := FIFO1[i + 1];

Str(Top1 / Buf1Len \* 100:2:2, S);

Print(S + ' % ', 66, 17, White);

Delay(Delay1);

end;

if Thread[4] = Active then begin

if Time[2] = 0 then

Delay(Delay1);

Print('Время вышло', 66, 7, LightGray);

end;

end;

{ Поток-производитель второго процесса }

procedure Prod2;

var

i: Integer;

c: Char;

Z: String[8];

begin

if Thread[5] = Suspended then Exit;

if (Thread[5] = Waits) and (Top2 <> Buf2Len) then Thread[5] := Active;

PrintStatus(5);

T := 0;

while (T < Time[3]) and (Thread[5] <> Suspended) do begin

if Top2 = Buf2Len then begin

Thread[5] := Waits;

PrintStatus(5);

Break;

end;

repeat

Num := 1 + Random(Range);

until Frac(Sqrt(Num)) < 1e-4;

Inc(Top2);

Buf2[Top2] := Num;

Str(Top2 / Buf2Len \* 100:2:2, Z);

Print(Z + ' % ', 66, 18, White);

Delay(Delay2);

end;

if Thread[5] = Active then begin

if Time[3] = 0 then

Delay(Delay2);

Print('Время вышло' , 66, 8, LightGray);

end;

end;

{ Поток-потребитель второго процесса }

procedure Cons2;

var

i, j: Integer;

Z: String[8];

S: String;

begin

if Thread[6] = Suspended then Exit;

if (Thread[6] = Waits) and (Top2 <> 0) then Thread[6] := Active;

PrintStatus(6);

T := 0;

while (T < Time[4]) and (Thread[6] <> Suspended) do begin

if Top2 = 0 then begin

Thread[6] := Waits;

PrintStatus(6);

Break;

end;

Str(Buf2[1]:10, S);

if X2 + Length(S) > StrLen then begin

X2 := 0;

if Y2 = 10 then begin

for i := 13 to 22 do

for j := 3 to 41 do

Scr[i, j].Symbol := Scr[i + 1, j].Symbol;

for j := 3 to 41 do

Scr[23, j].Symbol := ' ';

end

else Inc(Y2);

end;

Print(S, 3 + X2, 13 + Y2, White);

Inc(X2, Length(S) + 2);

Dec(Top2);

for i := 1 to Top2 do

Buf2[i] := Buf2[i + 1];

Str(Top2 / Buf2Len \* 100:2:2, Z);

Print(Z + ' % ', 66, 18, White);

Delay(Delay2);

end;

if Thread[6] = Active then begin

if Time[4] = 0 then

Delay(Delay2);

Print('Время вышло', 66, 9, LightGray);

end;

end;

{ Обработчик прерываний клавиатуры }

{$F+}

procedure KHandler; Interrupt;

var

s: String;

begin

case Port[$60] of

1: Kill := True; { нажата Esc }

2: if Time[1] > 0 then Dec(Time[1]); { нажата 1 }

3: if Time[1] < 50 then Inc(Time[1]); { 2 }

4: if Time[2] > 0 then Dec(Time[2]); { 3 }

5: if Time[2] < 50 then Inc(Time[2]); { 4 }

6: if Time[3] > 0 then Dec(Time[3]); { 5 }

7: if Time[3] < 50 then Inc(Time[3]); { 6 }

8: if Time[4] > 0 then Dec(Time[4]); { 7 }

9: if Time[4] < 50 then Inc(Time[4]); { 8 }

10: if speed > 1 then { 9 }

Dec(speed);

11: if speed < 10 then { 0 }

Inc(speed);

59: ChangeStatus(1); { нажата F1 }

60: ChangeStatus(2); { F2 }

61: ChangeStatus(3); { F3 }

62: ChangeStatus(4); { F4 }

63: ChangeStatus(5); { F5 }

64: ChangeStatus(6); { F6 }

end;

Str(Time[1], s);

Print(s + ' ', 66, 11, White);

Str(Time[2], s);

Print(s + ' ', 66, 12, White);

Str(Time[3], s);

Print(s + ' ', 66, 13, White);

Str(Time[4], s);

Print(s + ' ', 66, 14, White);

Str(speed, s);

Print(s + ' ', 76, 15, White);

while KeyPressed do { Очищаем буфер клавиатуры }

ReadKey;

Inline($9C);

OldKHandler;

end;

{$F-}

{ Изменение размера курсора }

procedure SetCurSize(const BegLine, EndLine: Byte);

var

Regs: Registers;

begin

with Regs do begin

AH := $01;

CH := BegLine;

CL := EndLine;

end;

Intr($10, Regs);

end;

{ Инициализация основных переменных }

procedure Initialize;

var

i, j: Integer;

s: String;

begin

InitializeTime;

Top1 := 0;

Top2 := 0;

SetCurSize($20, $00);

for i := 1 to 6 do

Thread[i] := Active;

for i := 1 to 4 do

Time[i] := 25;

PrintStatus(1);

PrintStatus(2);

Print('Не запущен', 66, 6, LightGray);

Print('Не запущен', 66, 7, LightGray);

Print('Не запущен', 66, 8, LightGray);

Print('Не запущен', 66, 9, LightGray);

Print('0.00 %', 66, 17, White);

Print('0.00 %', 66, 18, White);

Str(Time[1], s);

Print(s, 66, 11, White);

Str(Time[2], s);

Print(s, 66, 12, White);

Str(Time[3], s);

Print(s, 66, 13, White);

Str(Time[4], s);

Print(s, 66, 14, White);

Str(speed, s);

Print(s, 76, 15, White);

X1 := 0;

Y1 := 0;

X2 := 0;

Y2 := 0;

Num := 1;

LP := 0;

GetIntVec($9, @OldKHandler);

SetIntVec($9, Addr(KHandler));

GetIntVec($8, @OldTHandler);

SetIntVec($8, Addr(THandler));

end;

{ Действия при завершении работы программы }

procedure KillProc;

begin

Thread[3] := Suspended;

Time[2] := 20;

Thread[4] := Active;

Thread[5] := Suspended;

Time[4] := 20;

Thread[6] := Active;

T := High(T);

CurThread := 1;

while (Top1 <> 0) or (Top2 <> 0) do begin

T := 0;

case Random(2) of

0: Cons1;

1: Cons2;

end;

end;

SetIntVec($9, Addr(OldKHandler));

SetIntVec($8, Addr(OldTHandler));

Window(1, 1, 80, 25);

GotoXY(1, 25);

SetCurSize($06, $07);

Halt(0);

end;

var

s: String;

v: String;

code: Integer;

begin

if ParamCount <> 3 then { проверяем количество параметров команд. строки }

begin

Writeln('Ошибка! Правильный формат командной строки: Путь\kurs.exe ' +

'Положение\_заголовка(a - вверху, b - внизу) ' +

'Положение\_часов(1 - левый верхний угол; 2 - правый верхний угол; '+

'3 - левый нижний угол; 4 - правый нижний угол) '+

'Имя\_файла\_с\_текстом');

ReadKey;

Halt(1); { завершаем программу }

end;

v := ParamStr(1); { берем первый параметр командной строки }

s := Upcase(v[1]);

if (Length(v) <> 1) or ((s <> 'A') and (s <> 'B')) then

begin

Writeln('Неправильное положение заголовка!');

ReadKey;

Halt(1);

end;

x := 30;

if s = 'A' then

y := 1

else

y := 25;

s := ParamStr(2); { берем второй параметр команд. строки }

Val(s, timePos, code);

if (code <> 0) or not (timePos in [1, 2, 3, 4]) then

begin

Writeln('Неправильное положение часов!');

ReadKey;

Halt(1);

end;

case timePos of

1: begin

xTime := 2;

yTime := 2;

end;

2: begin

xTime := 72;

yTime := 2;

end;

3: begin

xTime := 2;

yTime := 24;

end;

4: begin

xTime := 72;

yTime := 24;

end;

end;

Assign(F, ParamStr(3));

{$I-}

Reset(F);

{$I+};

if IOResult <> 0 then begin

WriteLn('Ошибка при открытии файла ', ParamStr(3));

ReadLn;

Halt;

end;

TextBackground(Black);

ClrScr;

Window(2, 2, 42, 24);

TextBackground(2);

ClrScr;

TextColor(Black);

GotoXY(17, 1);

Write('Процесс 1');

Window(3, 3, 41, 23);

TextBackground(Black);

ClrScr;

Window(2, 12, 42, 12);

TextBackground(2);

ClrScr;

TextColor(Black);

GotoXY(17, 1);

Write('Процесс 2');

Window(44, 2, 79, 24);

TextBackground(2);

ClrScr;

GotoXY(14, 1);

Write('Настройки');

Window(45, 3, 78, 23);

TextBackground(Black);

ClrScr;

Window(45, 3, 78, 3);

TextBackground(2);

ClrScr;

TextColor(5);

GotoXY(13, 1);

Write('Состояния');

Window(45, 4, 78, 9);

TextBackground(Black);

ClrScr;

TextColor(White);

WriteLn('F1 Заголовок :');

WriteLn('F2 Часы :');

WriteLn('F3 Производитель 1 :');

WriteLn('F4 Потребитель 1 :');

WriteLn('F5 Производитель 2 :');

Write('F6 Потребитель 2 :');

Window(45, 10, 78, 10);

TextBackground(2);

ClrScr;

TextColor(5);

GotoXY(4, 1);

Write('Время <уменьшить>/<увеличить>');

Window(45, 11, 78, 15);

TextBackground(Black);

ClrScr;

TextColor(White);

WriteLn('1/2 Производитель 1:');

WriteLn('3/4 Потребитель 1:');

WriteLn('5/6 Производитель 2:');

WriteLn('7/8 Потребитель 2:');

Write('9/0 Скорость движ. заголовка:');

Window(45, 16, 78, 16);

TextBackground(2);

ClrScr;

TextColor(5);

GotoXY(9, 1);

Write('Заполнение буферов');

Window(45, 17, 78, 18);

TextBackground(Black);

ClrScr;

TextColor(White);

WriteLn('Заполнение буфера 1:');

Write('Заполнение буфера 2:');

Window(45, 19, 78, 19);

TextBackground(2);

ClrScr;

Window(45, 20, 78, 23);

TextBackground(Black);

ClrScr;

GotoXY(1, 3);

Write('Esc - выход из программы');

Initialize;

Randomize;

while True do begin

if Kill then KillProc;

CurThread := Random(4);

case CurThread of

0: Prod1;

1: Cons1;

2: Prod2;

3: Cons2;

end;

end;

end.