Федеральное агентство связи

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Сибирский государственный университет телекоммуникаций и

информатики»

(СибГУТИ)

09.03.01 Информатика и вычислительная техника

(дистанционная форма обучения)

КУРСОВАЯ РАБОТА

по дисциплине «Операционные системы»

Разработка многопроцессорной системы

Вариант № 6

Выполнил:

студент СибГУТИ,

гр.  ПБ-21 \_\_\_\_\_\_\_\_\_\_\_/ В.В. Лачков/

«23» апреля 2025 г. (подпись)

Проверил:

Преподаватель \_\_\_\_\_\_\_\_\_\_ / О.А. Бах/

«\_\_»\_\_\_\_\_\_\_\_\_ 2025 г. (подпись)

Содержание

[Введение 3](#__RefHeading___Toc9620_3279252907)

[1 Актуальность темы 4](#__RefHeading___Toc9622_3279252907)

[1.1 Цели и задачи курсовой работы 4](#__RefHeading___Toc9624_3279252907)

[1.2 Структура курсовой работы 5](#__RefHeading___Toc9626_3279252907)

[1.3 Методы и технологии реализации 6](#__RefHeading___Toc9628_3279252907)

[2 Формирование требований 8](#__RefHeading___Toc9630_3279252907)

[2.1 Спецификация требований 8](#__RefHeading___Toc9632_3279252907)

[3 Проектирование 11](#__RefHeading___Toc9634_3279252907)

[3.1.1 Выбор аппаратной платформы и ОС 11](#__RefHeading___Toc9636_3279252907)

[3.1.2 Выбор языка программирования 11](#__RefHeading___Toc9638_3279252907)

[3.1.3 Компиляция, сборка и запуск программы 12](#__RefHeading___Toc9640_3279252907)

[2.1.1 Выбор отладчика и среда разработки 14](#__RefHeading___Toc9642_3279252907)

[1. Реализация 18](#__RefHeading___Toc9644_3279252907)

[3.1 Программирование 18](#__RefHeading___Toc9646_3279252907)

[4. Тестирование 83](#__RefHeading___Toc9648_3279252907)

[Выводы 86](#__RefHeading___Toc9652_3279252907)

[Список использованной литературы 88](#__RefHeading___Toc9654_3279252907)

# 

# Введение

Современные вычислительные системы представляют собой сложные комплексы, эффективное функционирование которых обеспечивается операционными системами (ОС). Операционные системы играют ключевую роль в управлении аппаратными ресурсами компьютера, организации взаимодействия пользователя с вычислительной средой, а также в обеспечении параллельного выполнения множества задач. Изучение принципов построения и функционирования ОС является важной составляющей подготовки специалистов в области информационных технологий, поскольку позволяет не только понимать основы работы компьютера, но и разрабатывать эффективные программные решения, использующие системные механизмы.

Дисциплина «Операционные системы» охватывает широкий спектр тем, включая архитектуру ОС, управление процессами и потоками, планирование заданий, механизмы синхронизации, работу с памятью и файловыми системами. В рамках данного курса особое внимание уделяется изучению принципов многозадачности, параллельного выполнения процессов и потоков, а также взаимодействию между ними. Эти знания необходимы для разработки программного обеспечения, способного эффективно использовать вычислительные ресурсы и обеспечивать стабильную работу в многопоточной среде.

Целью данной курсовой работы является разработка программы, демонстрирующей параллельную работу нескольких процессов и потоков с возможностью их динамического управления. Программа должна обеспечивать интерактивное взаимодействие с пользователем через обработку прерываний клавиатуры, а также корректное завершение работы с освобождением всех выделенных ресурсов.

# 1 Актуальность темы

В условиях развития многопроцессорных и многоядерных систем проблема эффективного распределения ресурсов между процессами и потоками становится особенно актуальной. Современные приложения, такие как веб-серверы, базы данных, мультимедийные программы, требуют высокой степени параллелизма для обеспечения быстродействия и отзывчивости. Операционные системы предоставляют механизмы для управления потоками, синхронизации доступа к общим ресурсам и планирования выполнения задач.

Разработка программ, использующих многопоточность, сопряжена с рядом сложностей, включая проблемы гонки данных, взаимоблокировок (deadlocks) и неэффективного использования процессорного времени. Поэтому понимание принципов работы ОС, механизмов прерываний, планирования потоков и управления памятью является критически важным для создания надежного и производительного программного обеспечения.

# 1.1 Цели и задачи курсовой работы

Основной целью работы является реализация программы, моделирующей параллельное выполнение нескольких потоков с возможностью их приостановки, возобновления и динамического изменения параметров работы. В рамках курсовой работы решаются следующие задачи:

1. **Реализация потоков-производителей и потоков-потребителей**
   * Один поток считывает данные из файла в буфер, другой выводит их на экран.
   * Обеспечение синхронизации между потоками для избежания конфликтов доступа к буферу.
   * Организация корректной обработки ситуации заполнения или опустошения буфера.
2. **Динамическое управление потоками**
   * Приостановка и возобновление потоков по нажатию клавиш.
   * Обеспечение равномерного выполнения задач, не зависящего от загрузки системы.
3. **Визуализация работы потоков**
   * Разделение экрана на отдельные окна для каждого потока.
   * Отображение статусов потоков (активен, приостановлен, ожидает).
   * Вывод служебной информации (управляющие клавиши).
4. **Дополнительные функции**
   * Реализация анимации (движущегося объекта).
   * Вывод системного времени в углу экрана по заданным координатам из командной строки, при запуске программы.
   * Динамическое изменение цвета текста (для заголовка работы).

# 1.2 Структура курсовой работы

Работа состоит из следующих разделов:

1. **Введение** – обоснование актуальности, постановка целей и задач.
2. **Теоретическая часть** – обзор принципов работы ОС, механизмов потоков и синхронизации.
3. **Практическая реализация** – описание алгоритмов, структуры программы и используемых технологий.
4. **Тестирование и результаты** – демонстрация работы программы.
5. **Заключение** – выводы и перспективы дальнейшего развития проекта.

В ходе выполнения работы будут использованы материалы лекций, лабораторных занятий, а также дополнительные источники, посвященные многопоточному программированию и архитектуре операционных систем.

Таким образом, данный курсовой проект направлен на углубление понимания принципов работы операционных систем и приобретение практических навыков разработки многопоточных приложений.

# 1.3 Методы и технологии реализации

Для реализации модели процессов программа должна поддерживать таблицу(массив структур), называемую таблицей процессов, с одним элементом для каждого процесса. Элемент таблицы должен содержит информацию о состоянии процесса, счетчике команд, указателе стека, распределении памяти, а так же всю остальную информацию, которую необходимо сохранять при переключении в состояние готовности или блокировки для последующего запуска, - как если бы процесс не останавливался[2].

В разрабатываемой системе применена таблица процессов со структурой представленной на Рис. 1.

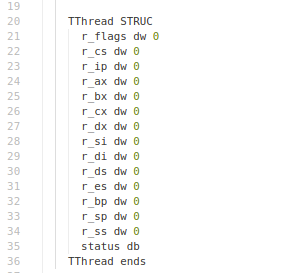


Рисунок 1 - Структура описатель потока, хранит контекст потока в таблице процессов

Поля начинающиеся с префикса «r\_» это регистры процессора. Поле «status» предназначено для хранения статуса потока: 1 — активен, 2 — приостановлен.

Иллюзия параллельности процессов на машине с одним процессором и несколькими устройствами ввода/вывода создается за счет поочередного переключения между потоками во время обработки прерывания с номером 08h (прерывание от системного таймера с частотой 18,2 раза в секунду). Содержимое счетчика команд процесса, слово состояния программы и, регистры процессора записываются в (текущий) стек аппаратными средствами. Затем происходит переход по адресу, указанному в векторе прерывания таймера и с этого момента происходит обработка прерывания процедурой-обработчиком.

Обработка прерывания начинается с сохранения регистров, в блоке управления текущим процессом в таблице процессов. Затем информация, помещенная в стек прерыванием, сохраняется в таблице процессов, и удаляется из стека. Далее из таблицы процессов извлекается контекст следующего активного процесса, и заносится в стек. После выхода из прерывания контекст нового потока (помещенного в стек) становится активным процессом системы. Такие действия, как сохранение регистров и установка указателей стека, невозможно даже выразить на языке высокого уровня(например, на C). Поэтому они выполняются небольшой программой на языке ассемблера, называемой планировщиком.

# 2 Формирование требований

# 2.1 Спецификация требований

Необходимо написать программу, реализующую параллельную работу нескольких процессов. Каждый процесс может состоять из одного или нескольких потоков. Любой из потоков, работающих в составе этих процессов, может быть приостановлен и вновь запущен некоторой определенной клавишей (можно использовать буквенные или цифровые клавиши). 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.

Программа должна работать устойчиво, без «зависаний» и непредвиденных зрительных эффектов. В случае необходимости указания параметров при запуске программы необходимо предусмотреть и обработать все возможные ошибки (с выводом диагностики на экран).

Задачи 1, 3, 6 предполагают наличие управляющего блока, который, используя прерывания таймера, случайным образом определяет очередной активный поток и выделяет ему кванты времени (возможно, в рамках большего кванта, выделенного всему процессу). Кванты времени – как для процесса в целом, так и для его потоков – изначально задавать некоторой фиксированной величины, но предусмотреть возможность её изменения во время работы путем нажатия некоторых ключевых клавиш (для каждого из потоков предусмотреть свою клавишу) – т.е. можно, например, увеличить или уменьшить квант только потока-пpоизводителя. При этом допустимы разные варианты реализации – «общий» квант может либо меняться, либо оставаться постоянным (тогда при ускорении одного потока другой автоматически замедлится, т.к. величина его кванта уменьшится).   
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, 7 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. 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».

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 №3). Имя читаемого файла задавать как параметр командной строки. После окончания файла он начинает считываться заново.
2. Вывод заголовка работы (фамилия и имя автора и название работы) постоянно меняющегося цвета. Место вывода (верхняя или нижняя строка экрана) задавать в качестве параметра командной строки.
3. Движущийся объект (в простейшем случае, например, летающий и отражающийся от границ окна шарик). Движение может осуществляться свободно или при управлении с клавиатуры.
4. В углу экрана с заданными в качестве параметров координатами поместить часы, показания которых должны соответствовать системному времени. Обновлять показания часов каждую секунду

Таблица 1 – Варианты задания

|  |  |
| --- | --- |
| Номер варианта | Номер задачи |
| 6 | 1,4,7,8 |

# 3 Проектирование

# 3.1.1 Выбор аппаратной платформы и ОС

Выбор аппаратной платформы и операционной системы в нашем случае выполнен в Спецификации требований. Для проекта выбрана архитектура (Intel x86) операционная система MS DOS.

# 3.1.2 Выбор языка программирования

Выбор языка программирования для курсового проекта обусловлен в первую очередь тем, что такие действия, как сохранение регистров и установка указателей стека, невозможно даже выразить на языке высокого уровня(например, на C). Поэтому для реализации поставленной задачи в качестве языка программирования выбран язык ассемблер — это низкоуровневый язык программирования, который был разработан в 40-х годах прошлого века. Он позволяет максимально задействовать возможности процессора, при этом применять как можно меньше команд и обращений к памяти, что уменьшает размер и повышает скорость работы программы.

Так же выбор языка программирования обусловлен такими требованиями как:

* Производительность. Благодаря своей вычислительной мощности язык обеспечивает высокую скорость исполнения кода.
* Отсутствие существенной нагрузки. Язык не утяжеляет программы, позволяет использовать их даже на старых устройствах.

При выборе языка также следует учесть и недостатки, которые ложатся на плечи разработчика, а именно:

* Сложный синтаксис. Быстро обучиться языку не получится, нужен опыт и знание теории.
* Ошибки при написании кода приводят к неполадкам при работе программ, вплоть до разрушения операционной системы
* Сложность написания кода. [3].

# 3.1.3 Компиляция, сборка и запуск программы

Для компиляции, сборки и запуска программ используется пакетный файл c.bat. BAT-файл — это последовательность команд для интерпретатора командной строки в виде текстового файла с расширением .bat. Основное предназначение пакетных файлов — автоматизация рутинных действий пользователя компьютера. Сборка модуля main.exe приведена на Рис.2.

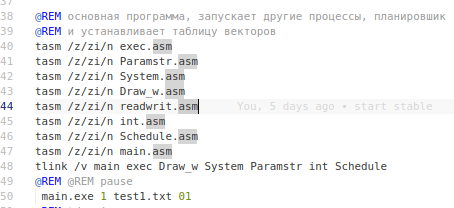


Рисунок 2 — Сборка модуля main.exe

Команда tasm — это программный пакет компании [Borland](https://ru.wikipedia.org/wiki/Borland), предназначенный для разработки программ на [языке ассемблера](https://ru.wikipedia.org/wiki/Язык_ассемблера) для архитектуры [x86](https://ru.wikipedia.org/wiki/X86) имееет следующие параметры:  
 z - Вывести на дисплей строку исходного текста с сообщением об ошибке;

zi - полная информация об идентификаторах для отладки;

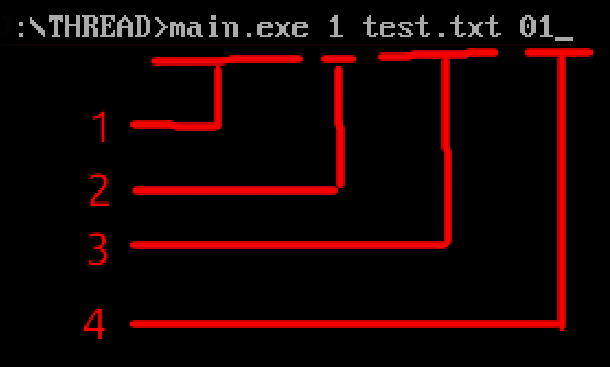
n - подавление в листингах таблицы идентификаторов;

<file name> - имя компилируемого файла.

Команда tlink — компоновщик фирмы Borland имеет следующие параметры:

v — включение полной информации для отладки.

Для запуска программы необходимо использовать параметры и последовательность приведенные на Рис. 3

 Рисунок 3 — Параметры запуска программы из командной строки

Програмный модуль main.exe имеет следующие параметры:  
 1 — название программы;

2 — параметр отвечающий за расположение заголовка работы. Значение 0 — вверху программы, значение 1 — внизу программы;

3 — название файла, из которого программа будет читать текст и выводить на экран;

4 — параметр отвечающий за положение вывода часов на экране. Этот параметр может принимать следующие значения: 00 — верхний левый угол экрана, 01 — верхний правый угол экрана, 10 — нижний левый угол экрана, 11 — правый нижний угол экрана.

При вводе некорректных параметров программа будет работать со стандартными параметрами, заложенными при разработке.

# Выбор отладчика и среда разработки

Главным инструментом для отладки программ, скомпилированных с помощью TASM, выберем Turbo Debugger v3.2 Copyright (c) 1988,92 Borland International. Это переносимый отладчик, который работает на MSDOS подобных системах [4].

В качестве основной среды разработки выберем Visual Studio Code -

текстовой редактор, разработанный Microsoft для [Windows](https://ru.wikipedia.org/wiki/Windows), Lunix и [macOS](https://ru.wikipedia.org/wiki/MacOS).

* 1. **Алгоритм работы программы**

Программная реализация проекта заключается в организации независимых программных процессов, которые запускаются ядром (файл main.exe). Ядро инициирует среду, в которой каждый процесс запущен независимо от других процессов. Вся информация о каждом процессе, дополнительная к содержимому его собственного адресного пространства, хранится в таблице операционной системы( процесса main). Эта таблица выполняет роль таблицы процессов и представляет собой массив структур (переменная thread\_array), по одной на каждый существующий в данный момент процесс[2].

Ядро программы представляет собой систему управления процессами, реализованную на ассемблере для операционной системы DOS и имеет следующие основные функции:

1. Инициализация системы:
   * + Освобождение неиспользуемой памяти (**free\_not\_used\_mem**)
     + Парсинг параметров командной строки (**make\_child\_process\_paramstr**)
     + Инициализация таблицы потоков (**init\_process\_table**)
2. Управление процессами:
   * + Запуск дочерних процессов (**init\_title\_process**,

**init\_reader\_writer\_process**, i**nit\_ball\_process, init\_clock\_process**)

* + - Ожидание завершения процессов (блок **@infinity**)
    - Корректное завершение работы системы

1. Работа с прерываниями:
   * + Перехват аппаратных прерываний (**init\_interrupts**)
     + Восстановление оригинальных обработчиков (**restore\_interrupts**)
     + Планировщик задач (**scheduler**)
2. Работа с экраном:
   * + Отрисовка окон (**draw\_window\_process**)
     + Вывод текста (**print**)
     + Очистка экрана/окон (**clrwind**)

Каждый процесс запущенный модулем main является отдельной программой, написанной на языке ассемблера. Программы загружаются в память для выполнения с помощью функции DOS Exec (Int 21h, функция 4Bh), которая играет роль системного загрузчика. Такая динамическая загрузка и запуск дочерних программ позволяет организовать иерархический программный комплекс, в котором родительский процесс инициирует дочерние процессы, а те в свою очередь могут вызывать процессы следующего уровня.

Для запуска процессов необходимо функции Exec передать блок параметров, который включает в себя адреса следующих обьектов:

* окружения (сегментный адрес, однословная ячейка);
* хвост команды (полный адрес, двухсловная ячейка);
* двух блоков управления файлами (FCB — file control block) (полные адреса, двухсловные ячейки).

Каждый запущенный модулем main процесс является TSR программой, резидентной в памяти ( Terminate and Stay Resident, TSR). Каждый из которых состоит из двух частей (секций) — инициализирущей и рабочей (резидентной). В тексте программы резидентная секция размещается вначале, инициализирущая - за ней.

При первом запуске программы она загружается в память целиком и управление передается секции инициализации, которая модифицирует векторы аперывания 2fh, настраивает программу на конкретные условия работы( исходя из параметров, переданных программе при ее вызове в хвосте командной строки) и с помощью прерывания DOS 21h с функцией 31h завершает программу, оставляя в памяти ее резидентную часть. Размер резидентной части программы вычисляется в начале программы и заносится в переменную **program\_length.**

Для взаимодействия с резидентными программами в DOS предусмотрено мультиплексное прерыввание 2fh, для этого в каждую программу включен обработчик функций этого прерывания (process\_function\_2f). При вызове соответствующей функции этого (номер функции передается в ah, а подфункции в регистре al) прерывания происходит следующие обработки подфункций:

* 0 — проверка резидента в памяти
* 0ffh — завершить процесс;

В резидентной части каждой программы имеется цикл, в котором выполняется основной процесс программы, а также проверяется флаг сигнала завершения процесса. Например в программе title.exe цикл имеет следующий вид:

@cycle:

mov ax,signal\_stop\_process

or ax,ax

jne @stop\_TSR

call process

loop @cycle

При изменении флага **signal\_stop\_process** с false на true происходит переход на метку выхода из программы, восстанавливаются измененные вектора, освобождается память выделенная программе и процесс завершается. Основной модуль для своего завершения должен дождаться завершения всех дочерних процессов.

# Реализация

# 3.1 Программирование

В листинге 1 представлен файл main.asm, это основная программа. Остальные программы и вспомогательные модули представлены ниже:

1. Листинг 1 – файл main.asm, основная программа;
2. Листинг 2 – файл readwrit.asm, программа читателя и писателя из файла
3. Листинг 3 – файл title.asm, программа читателя и писателя из файла
4. Листинг 4 – файл ball.asm, программа движущегося обьекта
5. Листинг 5 – файл clock.asm, программа вывода часов на экран
6. Листинг 6 – файл draw\_w.asm, подпрограмма отрисовки окн а программы
7. Листинг 7 – файл exec.asm, подпрограмма запуска дочернего процесса
8. Листинг 8 – файл file.asm, подпрограммы работы с файлом
9. Листинг 9 – файл int\_macr.inc, модуль вспомогательных макросов
10. Листинг 10 – файл int.asm, подпрограммы обработки прерываний клавиатуры
11. Листинг 11 – файл paramstr.asm, подпрограмма получения строки параметров
12. Листинг 12 – файл schedule.asm, подпрограммы планировщика
13. Листинг 13 – файл struc.inc, файл структур проекта
14. Листинг 14 – файл system.asm, файл вспомогательных процедур
15. Листинг 15 – файл utils.inc, файл процедуры задержки
16. Листинг 16 – файл c.bat, файл кимпиляции и сборки программ
17. Листинг 17 – файл prc\_ids.inc, файл c ID процессов

Листинг 1 – файл main.asm, основная программа

.model SMALL

include prc\_ids.inc

include struc.inc

include utils.inc

include int\_macr.inc

EXTRN exec:PROC, draw\_window:PROC, write:PROC, gotoxy:PROC, clrscr:PROC

EXTRN paramstr:PROC, paramstr\_array:BYTE

EXTRN scheduler:PROC, threadsRegistered, disableHardwareEvents, new\_09h:PROC, old\_09h, old\_08h

EXTRN is\_all

data segment PARA PUBLIC 'data'

hello db ' Esc - Вых,'

db ' F1 - зап./ост. поток чтения из файла'

db ' F2 - зап./ост. поток вывода на экран информации из файла'

db ' F3 - зап./ост. вывод заголовка работы'

db ' F4 - зап./ост. движущийся обьект'

db ' F5 - зап./ост. часы '

db 'UP/DN/LT/RT - перемещение движущегося обьекта '

db 'r - возобновить случайное движение. '

db 'Статусы: 0 - приостановлен, 1 - активен. '

hello\_length equ $-hello

text\_color equ 003h

paramstr\_err db 0ch,0dh,'Error in param string.$'

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PARAM BLOCS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FIRST PROCESS BLOCK \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

first\_param\_block\_process dw 0 ; Инициилизурем окружением основного процеесса

dd first\_process\_tail

dd 0 ; FCB1

dd 0 ; FCB2

first\_child\_process\_name db 'title.exe',0

first\_process\_tail db 0

db 0Dh

db 253 dup(0Dh)

exec\_title\_keep\_err db 'Error occured with keep title process.$'

exec\_title\_err db 'Something went wrong with the title process.$'

dw 10 dup (0)

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SECOND PROCESS BLOCK \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

second\_param\_block\_process dw 0

dd second\_process\_tail

dd 0

dd 0

second\_child\_process\_name db 'readwrit.exe',0

second\_process\_tail db 0

db 0Dh

db 253 dup(0Dh)

exec\_reader\_writer\_keep\_err db 'Error occured with keep reader-writer process.$'

exec\_reader\_writer\_err db 'Something went wrong with the reader-writer process.$'

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* THIRD PROCESS BLOCK \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

third\_param\_block\_process dw 0

dd third\_process\_tail

dd 0

dd 0

third\_child\_process\_name db 'clock.exe',0

third\_process\_tail db 0

db 0Dh

db 253 dup(0Dh)

exec\_clock\_object\_keep\_err db 'Error occured with keep clock object process.$'

exec\_clock\_object\_err db 'Something went wrong with the clock object process.$'

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FOURTH PROCESS BLOCK \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

fourth\_param\_block\_process dw 0

dd fourth\_process\_tail

dd 0

dd 0

fourth\_child\_process\_name db 'ball.exe',0

fourth\_process\_tail db 0

db 0Dh

db 253 dup(0Dh)

exec\_moved\_object\_keep\_err db 'Error occured with keep moved object process.$'

exec\_moved\_object\_err db 'Something went wrong with the moved object process.$'

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;------ screen --------

wnd TWind<16,0,22,79,17,1,21,78>

thread\_array TThread 5 dup(<>)

data ends

code segment PARA PUBLIC 'code'

assume cs:code, ds:data

main proc

mov ax, data

mov ds, ax

; Программа .EXE занимает всю свободную память

; освободим ее

call free\_not\_used\_mem

call make\_child\_process\_paramstr ; TODO :: What a fack here ???

call init\_process\_table

inc cs:threadsRegistered

@fork main\_process\_id, @infinity

call draw\_window\_process

call init\_reader\_writer\_process ; Запускаем процесс с двумя

;потоками (читатель файла/писатель на экран)

inc cs:threadsRegistered ; Увеличиваем счетчик установленных потоков

inc cs:threadsRegistered

call init\_title\_process ; Запускаем поток строки заголовка проекта

inc cs:threadsRegistered

call init\_ball\_process ; Запускаем поток движущегося обьекта

inc cs:threadsRegistered

call init\_clock\_process ; Запускаем поток часы

inc cs:threadsRegistered

call init\_interrupts ; Инициализируем прерывания

@infinity:

mov ax,cs:is\_all ; Активное ожидание - флаг

; is\_all == true - выход из программы

or ax,ax

je @infinity

mov ax,0c0ffh ; join process 1

int 2fh

mov ax,0c1ffh ; join process 2

int 2fh

mov ax,0c2ffh ; join process 3

int 2fh

mov ax,0c3ffh ; join process 4

int 2fh

@deactivate\_process main\_process\_id ; Деактивация основного потока

call restore\_interrupts ; Востанавливаем прерывания

call clrscr ; Очистим экран перед выходом

xor ax,ax ; Установим курсов вверх экрана

push ax

call gotoxy

mov ax,4c00h ; Нормальный выход, без ошибок

int 21h

main endp

; Процедура инициализирует прерывания

init\_interrupts proc

@change\_vect 9 new\_09h old\_09h

@change\_vect 08h scheduler old\_08h

ret

init\_interrupts endp

; Процедура восстанавливает прерывания

restore\_interrupts proc

@restore\_vect 9 old\_09h

@restore\_vect 08h old\_08h

ret

restore\_interrupts endp

; Процедура инициализации

; процесса заголовка проекта

init\_title\_process proc

push bp

mov bp, sp

push ax

lea ax, first\_param\_block\_process

push ax

lea ax, first\_child\_process\_name

push ax

lea ax, exec\_title\_keep\_err

push ax

lea ax, exec\_title\_err

push ax

call init\_process

pop ax

mov sp, bp

pop bp

ret

init\_title\_process endp

; Процедура инициализации читателя/писателя

init\_reader\_writer\_process proc

push bp

mov bp, sp

push ax

lea ax, second\_param\_block\_process

push ax

lea ax, second\_child\_process\_name

push ax

lea ax, exec\_reader\_writer\_keep\_err

push ax

lea ax, exec\_reader\_writer\_err

push ax

call init\_process

pop ax

mov sp, bp

pop bp

ret

init\_reader\_writer\_process endp

; Процедура инициализации часов

init\_clock\_process proc

push bp

mov bp, sp

push ax

lea ax, third\_param\_block\_process

push ax

lea ax, third\_child\_process\_name

push ax

lea ax, exec\_clock\_object\_keep\_err

push ax

lea ax, exec\_clock\_object\_err

push ax

call init\_process

pop ax

mov sp, bp

pop bp

ret

init\_clock\_process endp

; Процедура инициализации движущегося обьекта

init\_ball\_process proc

push bp

mov bp, sp

push ax

lea ax, fourth\_param\_block\_process

push ax

lea ax, fourth\_child\_process\_name

push ax

lea ax, exec\_moved\_object\_keep\_err

push ax

lea ax, exec\_moved\_object\_err

push ax

call init\_process

pop ax

mov sp, bp

pop bp

ret

init\_ball\_process endp

; Процедура инициализации процесса

; [pb+10 - param\_block\_process]

; [pb+8 - child\_process\_name]

; [pb+6 - exec\_title\_keep\_err]

; [pb+4 - exec\_title\_err]

init\_process proc

push bp

mov bp, sp

push dx

mov ax, data

push ax

mov ax,[bp+10]

push ax

mov ax, [bp+8]

push ax

call far ptr exec

mov ah, 4dh

int 21h

cmp ah,3

je @exec\_process\_exit

;lea dx, exec\_title\_keep\_err

mov dx, [bp+6]

mov ah,9

int 21h

jmp @exec\_process\_exit

cmp al, 0

je @exec\_process\_exit

; lea dx, exec\_title\_err

mov ax, [bp+4]

mov ah,9

int 21h

jmp @exec\_process\_exit

cmp al, 0

@exec\_process\_exit:

pop dx

mov sp, bp

pop bp

ret 8

init\_process endp

; Процедура получения аргументов

; командной строки для запуска дочерних процессов

make\_child\_process\_paramstr proc

push bp

mov bp,sp

push ax bx di si

call paramstr

xor ax,ax

mov al, byte ptr paramstr\_array+1

or ax,ax

je @without\_param

cmp ax, 0

je @without\_param ;Bad comman params

; First param

lea bx, paramstr\_array

inc bx

mov al, byte ptr [bx]

cmp al,1

jne @fail\_paramstr

mov al, byte ptr [bx]

mov first\_process\_tail,al

inc first\_process\_tail

lea di, first\_process\_tail

inc di

mov byte ptr [di], ' '

inc di

inc bx

mov al, byte ptr [bx]

mov byte ptr [di], al

inc di

mov byte ptr [di], 0dh

; Second param

lea si, paramstr\_array

add si, 3

inc [si]

inc [si]

lea di, second\_process\_tail

xor cx, cx

mov cl, byte ptr [si]

dec cl

dec cl

mov al,20H

movsb

stosb

cld

rep movsb

; Third param

mov cx,2

xor ax,ax

lea bx, paramstr\_array

mov al, byte ptr [bx]

cmp al, 3

jne @fail\_paramstr

inc bx

@get\_third\_param:

mov al, byte ptr [bx]

or al,al

je @without\_param

add bx,ax

inc bx

loop @get\_third\_param

xor cx,cx

dec bx

dec bx

mov cl, byte ptr [bx]

lea di, third\_process\_tail

mov si,bx

inc si

mov al,20H

stosb

rep movsb

@fail\_paramstr:

print\_str paramstr\_err

jmp @without\_param

@without\_param:

pop si di bx ax

mov sp, bp

pop bp

ret

make\_child\_process\_paramstr endp

; Процедура освобождения неиспользуемой памяти

free\_not\_used\_mem proc

push bp

mov bp, sp

push dx es bx ax

mov ax, zzz

mov dx, es

sub ax, dx

mov bx, ax

mov ah, 4ah

int 21h

pop ax bx es dx

mov sp, bp

pop bp

ret

free\_not\_used\_mem endp

; Процедура отрисовки основного окна

draw\_window\_process proc

push bp

mov bp,sp

call clrscr

mov ax, seg wnd

push ax

mov ax, offset wnd

push ax

call far ptr draw\_window

mov ax, offset wnd

push ax

mov ax, text\_color

push ax

mov ax, offset hello

push ax

mov ax, hello\_length

push ax

call print

mov sp,bp

pop bp

ret

draw\_window\_process endp

; Процедура печаты сообщения на экран

; bp+4 - str length

; bp+6 - str offset and str ended $

; bp+8 - attr

; bp+10 - TWind

print proc

local

push bp

mov bp, sp

push cx si ax bx dx di

mov di,[bp+10]

; mov dx, [bp+8]

mov dl, byte ptr [di].TWind.inner\_left\_top\_col

mov dh, byte ptr [di].TWind.inner\_left\_top\_row

push dx

call gotoxy

mov cx, [bp+4]

@std\_out\_space:

mov bx,[bp+8]

mov al, ' '

mov ah,09h

push cx

mov cx,1

int 10h

pop cx

inc dx

cmp dl, [di].TWind.right\_bottom\_col

jnz @std\_out\_space\_continue

mov dl, [di].TWind.inner\_left\_top\_col

inc dh

@std\_out\_space\_continue:

push dx

call gotoxy

loop @std\_out\_space

mov dl, byte ptr [di].TWind.inner\_left\_top\_col

mov dh, byte ptr [di].TWind.inner\_left\_top\_row

push dx

call gotoxy

mov cx, [bp+4]

@std\_out:

;\*\*\*\*\*\*\* getxy \*\*\*\*\*\*

push cx dx bx

mov bh,0

mov ah,3h

int 10h

mov ax, dx

pop bx dx cx

;\*\*\*\*\*\*\* getxy \*\*\*\*\*\*

cmp al, [di].TWind.right\_bottom\_col

jnz @std\_out\_str\_continue

mov al, [di].TWind.left\_top\_col

inc al

inc ah

push ax

call gotoxy

@std\_out\_str\_continue:

mov al, ds:[si]

call write

inc si

loop @std\_out

pop di dx bx ax si cx

mov sp, bp

pop bp

ret 10

print endp

; Процедура очистки окна

; Листать окно вверх (или очистить). Листать на 1 или более строк вверх.

; Вход:

; bp+10 (CH,CL) = строка,столбец верхнего левого угла окна (считая от 0)

; bp+8 (DH,DL) = строка,столбец нижнего правого угла окна (считая от 0)

; bp+6 (AL) = число пустых строк, вдвигаемых снизу (0=очистить все окно)

; bp+4 (BH) = видео атрибут, используемый для пустых строк

clrwind proc

push bp

mov bp, sp

push ax bx cx dx

mov cx, [bp+10]

mov dx, [bp+8]

mov al, [bp+7]

mov bh, [bp+5]

mov ah, 06h

int 10h

pop dx cx bx ax

mov sp, bp

pop bp

ret 8

clrwind endp

; Процедура инициализации таблицы потоков

init\_process\_table proc

push bp

mov bp, sp

push es di ax

xor ax,ax

mov al, bios\_process\_linked\_seg

mov es, ax

mov al, bios\_process\_linked\_offs

mov di, ax

lea ax, thread\_array

mov es:[di], ax

mov ax, seg thread\_array

mov es:[di+2], ax

pop ax di es

mov sp, bp

pop bp

ret

init\_process\_table endp

code ends

; Зарезервируем стек

stack segment STACK

dw 200h dup (0)

stack ends

; Сегмент необходим для вычисления размера программы

zzz segment

zzz ends

end

Листинг 2 – файл readwrit.asm, программа читателя и писателя из файла

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* ПРОГРАММА ЧТЕНИЯ ДАННЫХ ИЗ ФАЙЛА И ВЫВОДА НА ЭКРАН \*

; \* Использование: readwrite.exe <file\_name> \*

; \* Программа читает данные из файла и выводит данный в рабочую \*

; \* область. Структура программы Производитель-потребитель. \*

; \* Оба потока (нити) работают независимо. Для доступа к \*

; \* разделяемой памяти(кольцевому буферу), реализован алго- \*

; \* ритм Петерсона. \*

; \* \*

; \* Примечание: Если файл, название которого передано как пара- \*

; \* метр при запуске программы не найден, то читается файл с \* \*

; \* названием test.txt \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.MODEL SMALL

include prc\_ids.inc

include struc.inc

include utils.inc

include int\_macr.inc

include struc.inc

EXTRN gotoxy:PROC, draw\_window:PROC, write:PROC, \_getxy:PROC, open:PROC

EXTRN read:PROC, clrscr:PROC, paramstr\_array:BYTE, paramstr:PROC, seek\_start:PROC

data segment PARA PUBLIC 'DATA'

;---------------------- file --------------------------------

file\_name db 'text.txt',0 ; Хранит имя файла

db 100 dup(0)

handle dw 0 ; Хранит хэндл файла

file\_not\_found\_err db 'File not found!$' ; Ошибка что файл не найден

allredy\_installed\_err db 'Allredy installed!$' ; Ошибка, что резидент уже в памяти

;---------------------- file --------------------------------

;--------------------- queue ------------------------------

head dw 0 ; указатель начала кольцевого буфера

tail dw 0 ; указатель хвота кльцевого буфера

buff\_size equ 3 ; Размер очереди (количество строк)

buff\_string\_length equ 23 ; Размер строки (1 байт на длину + 22 байт на данные)

buff\_str db buff\_size dup (buff\_string\_length dup(20h)) ; кольцевой буфер

;--------------------- queue ------------------------------

;-------- for G.L.Peterson algorithm since 1981 year -------

true equ 1

false equ 0

thread\_count equ 2 ; Количество нитей

intrested db 2 dup(0)

turn db 0

data ends

code segment PARA PUBLIC 'code'

assume cs:code, ds:data, es:data, SS:stack

jmp init

stdout equ 1

cr equ 10 ; перевод каретки

; readwrite\_process\_id equ 1 ; id процесса чтения из файла

process\_write\_id equ 2 ; id процесса вывода на экран

installed equ 0ffh ; Константа для проверки уже запущен

; процеили нет, для избежания дубля

; резедента в памяти

process\_function\_2f equ 0c0h + readwrite\_process\_id -1 ; Функция процесса, прерывания 2f

;------ screen --------

wnd TWind<1,0,15,39,1,1,14,38> ; Структура для построения окна программы

program\_length dw 0 ; Хранит длину программы в параграфах

signal\_stop\_process dw 0 ; Сигнал остановки процесса

;------ Vectors -------------

old\_2fh dd 0 ; Хранит значение вектора 2f

;-----------------------------

; Обработчик прерывания 2f

new\_2fh proc ; Перепишем прерывание 2f, установим

; пользовательскую функцию, для

local ; проверки резидента в памяти и для обработки

; сигнала завершения программы

cmp ah, process\_function\_2f ; Проверяем наша функция вызвана или нет

jne @exit

cmp al,0

jne @title\_2f\_0

mov al,0ffh

jmp @exit

@title\_2f\_0:

cmp al,0ffh

jne @title\_2f\_ff

mov cs:signal\_stop\_process,1 ; Вызвана функция остановки программы

@restore\_vect 2fh cs:old\_2fh

jmp @exit

@title\_2f\_ff:

@exit:

jmp cs:old\_2fh

new\_2fh endp

; Макрос для реализации алгоритма Петерсона

@enter\_region macro process

mov si,1

sub si,process

lea di,intrested

add di,process

mov byte ptr [di], true

mov byte ptr turn, process

endm

init:

mov ax, data

mov ds,ax ; Вычисление размера программы в параграфах

mov ax,zzzz ; для оставления программы в памяти

; резедентной TSR

mov dx, es ; es = PSP

sub ax,dx ; ax = длина программы в параграфах

mov cs:program\_length,ax

xor ax,ax

mov ah,process\_function\_2f

int 2fh

cmp al, installed ; Проверим что нет дубля программы в памяти

jne @not\_installed

call do\_exit

@not\_installed: ; Устанавливаем вектор 2f

@change\_vect 2fh new\_2fh cs:old\_2fh

mov ax, seg code

mov ds, ax

call get\_file\_name\_from\_paramstr ; Получаем параметр командной строки

@without\_param:

;call clrscr ; commit it after test !!!

mov ax, seg wnd

push ax

mov ax, offset wnd

push ax

call far ptr draw\_window ; инициализация окна

mov ax,seg wnd

mov ds,ax

mov dh,byte ptr wnd.inner\_right\_bottom\_row

mov dl,1

push dx

call gotoxy

mov ax, data

mov ds, ax

@fork readwrite\_process\_id, @start\_process\_read ; установим первый поток (читает

; данные из файла)

@fork process\_write\_id, @start\_process\_write ; установим второй поток (выводит

; данные на экран)

@get\_ptr\_to\_thread\_data 2 ; т.к. процесс один, то разделим стек

; междудвумя потоками

mov ax,word ptr es:[di].TThread.r\_sp ;

sub ax,100h ;

mov word ptr es:[di].TThread.r\_sp,ax ;

outprog: ; завершаем и оставляем резидентной

mov dx,cs:program\_length

mov ax, 3100h

int 21h

@start\_process\_read: ; подпрограмма чтения из файла

call open\_file

@cycle:

@enter\_region 0

@wait\_read:

cmp byte ptr turn,0

jne @read\_turn

cmp byte ptr intrested+1, true

jmp @wait\_read

@read\_turn:

mov ax,cs:signal\_stop\_process

or ax,ax

jne @stop\_TSR

call process\_read\_from\_file

jmp @cycle

@start\_process\_write: ; подпрограмма вывода на экран

@enter\_region 1

@wait\_write:

cmp byte ptr turn,1

jne @write\_turn

cmp byte ptr intrested, true

jmp @wait\_write

@write\_turn:

mov ax,cs:signal\_stop\_process

or ax,ax

jne @stop\_TSR

call process\_stdout

jmp @start\_process\_write

@stop\_TSR:

@deactivate\_process readwrite\_process\_id ; Остановим программу и освободим память,

; восстанавливаем вектора

@rrr:

jmp @rrr ; This is STUB

do\_exit proc ; Выполняем выход из программы,

mov ax,0d23h ; с ошибкой, программа уже установлена

; в памяти

push ax ;

call gotoxy

mov ax, seg allredy\_installed\_err

mov ds,ax

lea dx, allredy\_installed\_err

mov ah,09h

int 21h

mov ax, 4Cffh

int 21h

ret

do\_exit endp

; процедура открытия файла

open\_file proc

push bp

mov bp,sp

mov ax,data

push ax

mov ds, ax

lea ax, file\_name

push ax

mov ax, 0000h

push ax ; Файл откроем на чтение

call open

jnc @file\_found

print\_str file\_not\_found\_err

@file\_found:

mov handle, ax

push ax

call seek\_start

mov sp, bp

pop bp

ret

open\_file endp

; подпрограмма вывода на экран

process\_stdout proc

push bp

mov bp,sp

push di bx cx

mov di,0

@stdout\_next\_str:

mov ax, [tail]

mov di, buff\_string\_length

mul di

xchg di, ax

lea bx, buff\_str[di]

xor cx,cx

mov cl, [bx] ; количество символов в строке

inc bx ; указатель на начало строки

call stdout\_string

mov bx, [tail]

inc bx

cmp bx, buff\_size

jl @no\_wrap\_tail

xor bx, bx ; Сброс tail, если достигнут конец буфера

@no\_wrap\_tail:

mov tail, bx

mov ax, [head]

cmp ax,bx

je @go\_out\_process\_stdout

jmp @stdout\_next\_str

@go\_out\_process\_stdout:

pop cx bx di

mov sp,bp

pop bp

ret

process\_stdout endp

; Выводит символы в цикле

; bp - increment

; cx - количество символов в строке

stdout\_string proc

push bp

mov bp, sp

@stdout\_buff\_string:

;\*\*\*\*\*\*\* getxy \*\*\*\*\*\*

push cx dx bx

mov bh,0

mov ah,3h

int 10h

mov ax, dx

pop bx dx cx

;\*\*\*\*\*\*\* getxy \*\*\*\*\*\*

cmp al,cs:wnd.inner\_right\_bottom\_col

jne @below\_row

mov al,0ah

inc cx

dec bx

jmp @do\_scroll

@below\_row:

mov al, [bx]

@do\_scroll:

cmp al,0ah

jne @stdout\_write

mov dx, offset wnd

push dx

call scroll\_window\_up

jmp @next\_char

@enter\_push:

@stdout\_write:

call write

@next\_char:

inc bx

loop @stdout\_buff\_string

mov sp, bp

pop bp

ret

stdout\_string endp

; процедура чтения данных из файла

process\_read\_from\_file proc

push bp

mov bp, sp

mov ax, [head]

mov bx, [tail]

cmp ax, buff\_size - 1

jl @process\_read

xor ax, ax ; Сброс head, если достигнут конец буфера

@process\_read:

mov di, buff\_string\_length

mul di

xchg di, ax

mov ax, data

mov ds,ax

lea dx, buff\_str[di+1]

push handle

mov ax, buff\_string\_length-1

push ax

call read

dec dx

mov bx, dx

mov byte ptr [bx],al

cmp al, buff\_string\_length-1

je @not\_eof

push word ptr handle

call seek\_start ; установим указатель на началофайла

@not\_eof:

mov ax, [head] ; Проверка на переполнение

mov bx, [tail]

inc ax

cmp ax, buff\_size

jl @no\_wrap

xor ax, ax ; Сброс head, если достигнут конец буфера

@no\_wrap:

mov head, ax

cmp ax, bx

je @queue\_full ; Если head == tail, очередь переполнена

jmp @process\_read

@queue\_full:

mov sp, bp

pop bp

ret

process\_read\_from\_file endp

; процедура прокрутки окна

;bp+4 - TWind

scroll\_window\_up proc

local

push bp

mov bp, sp

push bx cx dx ax

mov bx,[bp+4]

;\*\*\*\*\*\*\* getxy \*\*\*\*\*\*

push cx dx bx

mov bh,0

mov ah,3h

int 10h

mov ax, dx

pop bx dx cx

;\*\*\*\*\*\*\* getxy \*\*\*\*\*\*

cmp ah, cs:[bx].TWind.inner\_left\_top\_row

delay 10

mov ah, byte ptr cs:[bx].TWind.inner\_left\_top\_row

inc ah

mov al, byte ptr cs:[bx].TWind.inner\_left\_top\_col

push ax

mov ah, byte ptr cs:[bx].TWind.inner\_right\_bottom\_row

mov al, byte ptr cs:[bx].TWind.inner\_right\_bottom\_col

push ax

mov ax,1

push ax

mov ax, 0fh

push ax

call scroll\_up

mov dh, byte ptr cs:[bx].TWind.inner\_right\_bottom\_row

mov dl, byte ptr cs:[bx].TWind.inner\_left\_top\_col

push dx

call gotoxy

@done:

pop ax dx cx bx

mov sp, bp

pop bp

ret 2

scroll\_window\_up endp

; Скролинг вверх

; bp+10(сх) - координаты левого верхнего угла прямоугольной области экрана (ch - строка, cl - столбец),

; bp+8(dx) - координаты правого нижнего угла (dh ? строка, dl ? столбец),

; bp+6(al) - на сколько строк прокручивать заданное окно (при al = 0 все заданное окно очищается),

; bp+4(bh) - атрибуты для заполнения освобождающихся строк (7 - белый по черному)

scroll\_up proc

push BP

mov bp,sp

push ax bx cx

mov cx, [bp+10]

mov dx, [bp+8]

mov al, [bp+6]

mov bh, [bp+4]

mov ah, 06h

;mov bh, 0fh

int 10h

pop cx bx ax

mov sp, bp

pop bp

ret 8

scroll\_up endp

; Процедура читает имя файла из параметра запуска программы

get\_file\_name\_from\_paramstr proc

push bp

mov bp,sp

push cx bx es ds

call paramstr

mov al, byte ptr paramstr\_array

or al,al

je @@done

xor cx,cx

lea bx, [paramstr\_array+1]

mov cl, byte ptr [bx]

inc cl

inc bx

push es

push ds

pop es

mov ax,seg paramstr\_array

mov ds,ax

lea di,file\_name

mov si,bx

rep movsb

pop es

@@done:

pop es ds es bx cx

mov sp,bp

pop bp

ret

get\_file\_name\_from\_paramstr endp

code ends

; Зарезервируем стек

stack segment para stack

dw 200h dup(0)

stack ends

; Сегмент необходим для вычисления размера программы

zzzz segment

zzzz ends

end

Листинг 3 – файл title.asm, программа читателя и писателя из файла

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* ПРОГРАММА ВЫВОДА ЗАГОЛОВКА РАБОТЫ НА ЭКРАН \*

; \* Использование: title.exe <[0;1]> \*

; \* 0 - вывод заголовка вверху экрана \*

; \* 1 - вывод заголовка внизу экрана \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.model SMALL

include prc\_ids.inc

include struc.inc

include int\_macr.inc

include utils.inc

EXTRN gotoxy:PROC, clrscr:PROC, \_getxy:PROC, paramstr:PROC, paramstr\_array, atoi:PROC

; title\_process\_id equ 2 ; Номер процесса

is\_test equ 0 ; Флаг тест режим

installed equ 0ffh ; Константа для проверки уже запущен

; процеили нет, для избежания дубля резедента в памяти

function\_2f equ 0c0h + title\_process\_id -1 ; Функция процесса, прерывания 2f

data segment PARA PUBLIC 'DATA'

program\_length dw 0 ; Хранит длину программы в параграфах

; Выводимая строка заголовка

title\_str db 'Курсовой проект. Лачков Вадим Валерьевич. Группа: ПБ-21'

title\_length equ $-title\_str ; Хранит длину строки

allredy\_installed\_err db 'Allredy installed!$' ; Ошибка, что резидент уже в памяти

location dw 0 ; Хранит координаты

data ends

code segment PARA PUBLIC 'CODE'

ASSUME CS:code, DS:data, SS:stack

jmp init

;------------------- Vectors -----------------------

old\_2fh dd 0 ; Хранит значение вектора 2f

;---------------------------------------------------

color db 1 ; Хранит текущий цвет строки

signal\_stop\_process dw 0 ; Сигнал остановки процесса

maincs dw 0

;---------------------------------------------------

new\_2fh proc ; Перепишем прерывание 2f,

; установим пользовательскую функцию, для

local ; проверки резидента в памяти

; и для обработки сигнала завершения программы

cmp ah, function\_2f ; Проверяем наша функция вызвана или нет

jne @exit

cmp al,0

jne @title\_2f\_0

mov al,0ffh

jmp @exit

@title\_2f\_0:

cmp al,0ffh

jne @title\_2f\_ff ; Вызвана функция остановки программы

mov cs:signal\_stop\_process,1

@restore\_vect 2fh cs:old\_2fh

jmp @exit

@title\_2f\_ff:

@exit:

jmp cs:old\_2fh

new\_2fh endp

init:

mov ax, data

mov ds, ax

call paramstr

xor ax,ax

mov al, byte ptr paramstr\_array

or ax,ax

je @without\_param

lea ax, [paramstr\_array+1]

push ax

call atoi

cmp ax,1

jne @without\_param

inc location

@without\_param: ; Вычисление размера программы в параграфах

mov ax,zzz ; для оставления программы в памяти резедентной TSR

mov dx, es ; es = PSP

sub ax,dx

mov program\_length,ax ; ax = длина программы в параграфах

xor ax,ax

mov ah,function\_2f

int 2fh

cmp al, installed ; Проверим, что нет дубля программы в памяти

jne @not\_installed

call do\_exit

@not\_installed:

mov ax, is\_test

or ax,ax

jne @it\_is\_test

@change\_vect 2fh new\_2fh old\_2fh

jmp @main

@it\_is\_test:

call clrscr

@test\_cycle:

call process

loop @test\_cycle

@main:

mov ax, 0b800h

mov es,ax

@fork title\_process\_id, @cycle ; Установим поток в таблице потоков

mov ax, cs ; Выходим и оставляем TSR

mov cs:maincs, ax

mov dx,program\_length

mov ax, 3100h

int 21h

@cycle: ; Выводим строку, пока не получим

; сигнал на завершение

mov ax,signal\_stop\_process

or ax,ax

jne @stop\_TSR

call process

loop @cycle

@stop\_TSR: ; Останавливаем программу,

; восстанавливаем прерывание 2f

@deactivate\_process title\_process\_id

@rrr:

jmp @rrr ; This is STUB

do\_exit proc ; Выполняем выход из программы,

mov ax,0d23h ; с ошибкой, программа уже установлена в памяти

push ax

call gotoxy

mov ax, seg allredy\_installed\_err

mov ds,ax

lea dx, allredy\_installed\_err

mov ah,09h

int 21h

mov ax, 4Cffh

int 21h

ret

do\_exit endp

; Процедура вычисления координат вывода строки

get\_offset proc

mov di,79-title\_length

mov ax, 80\*2\*24

mul word ptr ds:location

add di,ax

xor si,si

ret

get\_offset endp

; Основная подпрограмма

process proc

push bp

mov bp, sp

mov ax, 0b800h

mov es,ax

mov cx,title\_length

call get\_offset

xor ax,ax

mov al,cs:color

or al,08h

mov bl,0fh

div bl

mov cs:color,ah

cld

@stdout\_title:

mov al, byte ptr ds:title\_str+[si]

stosb

mov al,cs:color

stosb

inc si

loop @stdout\_title

inc cs:color

delay 10

mov sp, bp

pop bp

ret

process endp

code ends

; Зарезервируем стек

stack segment para stack

dw 200h dup(0)

stack ends

; Сегмент необходим для вычисления размера программы

zzz segment

zzz ends

end

Листинг 4 – файл ball.asm, программа движущегося обьекта

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* ПРОГРАММА ВЫВОДА ДВИЖУЩЕГОСЯ ОБЬЕКТА НА ЭКРАНЕ \*

; \* Использование: ball.exe \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.MODEL SMALL

include struc.inc

include prc\_ids.inc

include utils.inc

include int\_macr.inc

EXTRN gotoxy:PROC, draw\_window:PROC, write:PROC, \_getxy:PROC, open:PROC

EXTRN read:PROC, clrscr:PROC, paramstr\_array:BYTE, paramstr:PROC, seek\_start:PROC

EXTRN put\_char:PROC, put\_colored\_char:PROC

public is\_random\_move

public row\_k

public col\_k

str\_length equ 2

video equ 0b800h ; Сегмент памяти видеодаптера

text\_color equ 02h ; Цвет движущегося обьекта

data segment PARA PUBLIC 'DATA'

string db 1h, text\_color ; Строка движущегося обьекта

string\_space db 20h, 0 ; Константа пробела

allredy\_installed\_err db 'Allredy installed!$' ; Ошибка, что резидент уже в памяти

row\_k db -1 ; Коэициент для строк

col\_k db -1 ; Коэфициент для колонок

is\_random\_move db 1 ; Флаг рандомное или ручное перемещение

coord dw 0 ; Текущие кординаты обьекта

data ends

code segment PARA PUBLIC 'code'

assume cs:code, ds:data, es:data, SS:stack

jmp init

; ball\_process\_id equ 3 ; id процесса в таблице процессов

installed equ 0ffh ; Константа для проверки уже запущен

; процеили нет, для избежания дубля

; резедента в памяти

process\_function\_2f equ 0c0h + ball\_process\_id -1 ; Функция процесса, прерывания 2f

;------ window --------

wnd TWind<1,40,15,79,2,41,14,78> ; Структура для построения окна программы

program\_length dw 0 ; Хранит длину программы в параграфах

signal\_stop\_process dw 0 ; Сигнал остановки процесса

;------ Vectors -------------

old\_2fh dd 0 ; Хранит значение вектора 2f

;-----------------------------

; Обработчик прерывания 2f

new\_2fh proc ; Перепишем прерывание 2f, установим

; пользовательскую функцию, для

local ; проверки резидента в памяти и для обработки

; сигнала завершения программы

cmp ah, process\_function\_2f ; Проверяем наша функция вызвана или нет

jne @exit

cmp al,0

jne @title\_2f\_0

mov al,0ffh

jmp @exit

@title\_2f\_0:

cmp al,0ffh

jne @title\_2f\_ff

mov cs:signal\_stop\_process,1 ; Вызвана функция остановки программы

@restore\_vect 2fh cs:old\_2fh

jmp @exit

@title\_2f\_ff:

@exit:

jmp cs:old\_2fh

new\_2fh endp

init:

mov ax, data

mov ds,ax ; Вычисление размера программы в параграфах

mov ax,zzzz ; для оставления программы в памяти резедентной TSR

mov dx, es ; es = PSP

sub ax,dx ; ax = длина программы в параграфах

mov cs:program\_length,ax

xor ax,ax

mov ah,process\_function\_2f

int 2fh

cmp al, installed ; Проверим что нет дубля программы в памяти

jne @not\_installed

call do\_exit

@not\_installed:

@change\_vect 2fh new\_2fh cs:old\_2fh

mov ax, seg code

mov ds, ax

@without\_param:

call init\_window ; Отрисуем окно программы

mov ax, data

mov ds, ax

@fork ball\_process\_id, @start\_process ; Установим поток в таблице потоков

call init\_process\_structure\_table ; Процедура записывает ссылку на переменные

; row\_k, col\_k

; @cycle1:

; call process

; delay 1

; @queue\_empty1:

; jmp @cycle1

outprog:

mov dx,cs:program\_length ; Выходим из программы и оставляем резедентной

mov ax, 3100h

int 21h

@start\_process: ; Цикл программы

@cycle:

mov ax,cs:signal\_stop\_process

or ax,ax

jne @stop\_TSR

call process

delay 1 ; Введем задержку при перемещении обьекта

@queue\_empty:

jmp @cycle

@stop\_TSR:

@deactivate\_process ball\_process\_id ; Остановим программу и освободим память, восстанавливаем вектора

@rrr:

jmp @rrr ; This is STUB

do\_exit proc ; Выполняем выход из программы,

mov ax,0d23h ; с ошибкой, программа уже установлена в памяти

push ax

call gotoxy

mov ax, seg allredy\_installed\_err

mov ds,ax

lea dx, allredy\_installed\_err

mov ah,09h

int 21h

mov ax, 4Cffh

int 21h

ret

do\_exit endp

; Процедура выводит на пробел

print\_space proc

push bp

mov bp,sp

push ax

lea ax,string\_space

push ax

mov ax,str\_length

push ax

call print

pop ax

mov bp,sp

pop bp

ret

print\_space endp

; Процедура выводит на экран движущийся обьект

print\_symbol proc

push bp

mov bp,sp

push ax

lea ax,string

push ax

mov ax,str\_length

push ax

call print

pop ax

mov bp,sp

pop bp

ret

print\_symbol endp

; Процедура выводит на экран символ

print proc

push BP

mov bp,sp

push es di cx

mov ax, video

mov es, ax

mov cx, coord

mov ax, cx

mov al, ah

mov ah,160

mul ah

xor ch,ch

shl cl,1

cmp cl, byte ptr [bp+4]

jbe @normalized\_col

sub cl, byte ptr [bp+4]

@normalized\_col:

add ax, cx

mov di, ax

mov si,[bp+6]

mov cx,word ptr [bp+4]

rep movsb

pop cx di es

mov sp, bp

pop bp

ret 4

print endp

; Процедура записывает ссылку на переменные row\_k, col\_k

; в межпрограмную область биос

init\_process\_structure\_table proc

push bp

mov bp, sp

push es di ax

xor ax,ax

mov al,40h

mov es, ax

mov al, 0f4h

mov di, ax

lea ax, row\_k

mov es:[di], ax

mov ax, seg row\_k

mov es:[di+2], ax

pop ax di es

mov sp, bp

pop bp

ret

init\_process\_structure\_table endp

; Основной поток

process proc

call print\_space

mov ax, coord

cmp is\_random\_move,1

jne @not\_random\_move

call step\_random

jmp @do\_move

@not\_random\_move:

call hand\_step

@do\_move:

call print\_symbol

ret

process endp

; Процедура обрабатывает нажатие на стреки

hand\_step proc

cmp ah, wnd.inner\_left\_top\_row

jne @hand\_step\_dont\_change\_row\_k\_left

cmp row\_k,-1

jne @hand\_step\_dont\_change\_row\_k\_left

mov row\_k,0

@hand\_step\_dont\_change\_row\_k\_left:

cmp ah, wnd.inner\_right\_bottom\_row

jne @hand\_step\_dont\_change\_row\_k\_right

cmp row\_k,1

jne @hand\_step\_dont\_change\_row\_k\_right

mov row\_k,0

@hand\_step\_dont\_change\_row\_k\_right:

add ah, row\_k

mov cl, wnd.inner\_left\_top\_col

inc cl

cmp al, cl

jne @hand\_step\_dont\_change\_col\_k\_left

cmp col\_k,-1

jne @hand\_step\_dont\_change\_col\_k\_left

mov col\_k,0

@hand\_step\_dont\_change\_col\_k\_left:

cmp al, wnd.inner\_right\_bottom\_col

jne @hand\_step\_dont\_change\_col\_k\_right

cmp col\_k,1

jne @hand\_step\_dont\_change\_col\_k\_right

mov col\_k,0

@hand\_step\_dont\_change\_col\_k\_right:

add al, col\_k

mov coord, ax

mov col\_k, 0

mov row\_k,0

ret

hand\_step endp

; Процедура вычисляет координаты следующего шага перемещения

step\_random proc

push bp

mov bp,sp

cmp ah, wnd.inner\_left\_top\_row

je @step\_random\_change\_row\_k

cmp ah, wnd.inner\_right\_bottom\_row

jne @step\_random\_dont\_change\_row\_k

@step\_random\_change\_row\_k:

neg row\_k

@step\_random\_dont\_change\_row\_k:

add ah, row\_k

mov cl, wnd.inner\_left\_top\_col

inc cl

cmp al, cl

je @step\_random\_change\_col\_k

cmp al, wnd.inner\_right\_bottom\_col

jne @step\_random\_dont\_change\_col\_k

@step\_random\_change\_col\_k:

neg col\_k

@step\_random\_dont\_change\_col\_k:

add al, col\_k

mov coord,ax

mov sp, bp

pop bp

ret

step\_random endp

; Процедура инициализации окна программы

init\_window proc

push bp

mov bp,sp

push ax dx

; call clrscr ; TODO:: commit it after test !!!

mov ax, seg wnd

push ax

mov ax, offset wnd

push ax

call far ptr draw\_window

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Set cursor to the center window \*\*\*\*\*\*\*\*\*\*\*\*\*

mov ax,seg wnd

mov ds,ax

push bx

xor ax,ax

mov al,byte ptr wnd.inner\_right\_bottom\_row

mov ah, byte ptr wnd.inner\_left\_top\_row

sub al, ah

mov ah,0

mov dl,2

div dl

mov ah, byte ptr wnd.inner\_left\_top\_row

add al, ah

mov bh, al

xor ax, ax

mov al,byte ptr wnd.inner\_right\_bottom\_col

mov ah,byte ptr wnd.inner\_left\_top\_col

sub al,ah

mov ah,0

div dl

mov ah,byte ptr wnd.inner\_left\_top\_col

add al, ah

mov bl, al

xchg dx, bx

pop bx

mov ax, data

mov ds,ax

mov dx,0430h

mov coord,dx

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Set cursor to the center window \*\*\*\*\*\*\*\*\*\*\*\*\*

pop dx ax

mov sp, bp

pop bp

ret

init\_window endp

code ends

; Зарезервируем стек

stack segment para stack

dw 200h dup(0)

stack ends

; Сегмент необходим для вычисления размера программы

zzzz segment

zzzz ends

end

Листинг 5 – файл clock.asm, программа вывода часов на экран

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* ПРОГРАММА ВЫВОДА ЧАСОВ НА ЭКРАН \*

; \* Использование: clock.exe <[00;01;10;11]> \*

; \* 00 - вывод в верхнем левом углу экрана \*

; \* 01 - вывод в верхнем правом углу экрана \*

; \* 10 - вывод в нижнем левом углу экрана \*

; \* 11 - вывод в нижнем правом углу экрана \*

; \* \*

; \* Примечание: Если параметр не передан или передан с ошибкой,\*

; \* то по умолчанию часы выводятся в левом верхнем углу экрана \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.MODEL SMALL

include prc\_ids.inc

include struc.inc

include utils.inc

include int\_macr.inc

EXTRN gotoxy:PROC

EXTRN clrscr:PROC, paramstr\_array, paramstr:PROC, seek\_start:PROC

text\_color equ 0ah ; Переменная хранит цвет строки часов

str\_length equ 16 ; Длина строки в байтах

video equ 0b800h ; Сегмент памяти видеодаптера

data segment PARA PUBLIC 'DATA'

allredy\_installed\_err db 'Allredy installed!$' ; Ошибка, что резидент уже в памяти

current\_time db '0',text\_color, '0',text\_color,':',text\_color,'0',text\_color

db '0',text\_color,':',text\_color,'0',text\_color,'0',text\_color

coord dw 0 ; Переменная хранит кординаты для вывода строки часов

data ends

code segment PARA PUBLIC 'code'

assume cs:code, ds:data, es:data, SS:stack

jmp init

; clock\_process\_id equ 4 ; id процесса в таблице процессов

installed equ 0ffh ; Константа для проверки уже запущен процеили нет,

; для избежания дубля резедента в памяти

process\_function\_2f equ 0c0h + clock\_process\_id - 1 ; Функция процесса, прерывания 2f

program\_length dw 0 ; Хранит длину программы в параграфах

signal\_stop\_process dw 0 ; Сигнал остановки процесса

;------ Vectors -------------

old\_2fh dd 0 ; Хранит значение вектора 2f

;-----------------------------

new\_2fh proc ; Перепишем прерывание 2f, установим пользовательскую

; функцию, для

local ; проверки резидента в памяти и для обработки сигнала

; завершения программы

cmp ah, process\_function\_2f ; Проверяем наша функция вызвана или нет

jne @exit

cmp al,0

jne @title\_2f\_0

mov al,0ffh

jmp @exit

@title\_2f\_0:

cmp al,0ffh ; Вызвана функция остановки программы

jne @title\_2f\_ff

mov cs:signal\_stop\_process,1

@restore\_vect 2fh cs:old\_2fh

jmp @exit

@title\_2f\_ff:

@exit:

jmp cs:old\_2fh

new\_2fh endp

init:

mov ax, data

mov ds,ax ; Вычисление размера программы в параграфах

mov ax,zzzz ; для оставления программы в памяти резедентной TSR

mov dx, es ; es = PSP

sub ax,dx ; ax = длина программы в параграфах

mov cs:program\_length,ax

xor ax,ax

mov ah,process\_function\_2f

int 2fh

cmp al, installed ; Проверим, что нет дубля программы в памяти

jne @not\_installed

call do\_exit

@not\_installed:

@change\_vect 2fh new\_2fh cs:old\_2fh

mov ax, seg code

mov ds, ax

call get\_coord ; Получим кординаты вывода часов

@without\_param:

; call clrscr ; TODO:: commit it after test !!!

mov ax, data

mov ds, ax

@fork clock\_process\_id, @start\_process ; Установим поток в таблице потоков

outprog:

mov dx,cs:program\_length

mov ax, 3100h

int 21h

@start\_process:

@cycle:

mov ax,cs:signal\_stop\_process

or ax,ax

jne @stop\_TSR

call process

@queue\_empty:

jmp @cycle

@stop\_TSR:

@deactivate\_process clock\_process\_id ; Остановим программу и освободим память, восстанавливаем вектора

@rrr:

jmp @rrr ; This is STUB

do\_exit proc ; Выполняем выход из программы,

mov ax,0d23h ; с ошибкой, программа уже установлена в памяти

push ax

call gotoxy

mov ax, seg allredy\_installed\_err

mov ds,ax

lea dx, allredy\_installed\_err

mov ah,09h

int 21h

mov ax, 4Cffh

int 21h

ret

do\_exit endp

; Процедура основного процесса

process proc

push BP

mov bp,sp

call get\_time

call print\_time

mov sp, bp

pop bp

ret

process endp

; Процедура вывода часов на экран

print\_time proc

push BP

mov bp,sp

push es di cx

mov ax, video

mov es, ax

mov cx, coord

mov ax, cx

mov al, ah

mov ah,160

mul ah

xor ch,ch

shl cl,1

cmp cl, str\_length

jbe @normalized\_col

sub cl, str\_length

@normalized\_col:

add ax, cx

mov di, ax

lea si,current\_time

mov cx,str\_length

rep movsb

pop cx di es

mov sp, bp

pop bp

ret

print\_time endp

; Процедура получения текущего времени

get\_time proc

push bp

mov bp,sp

push ax cx dx

mov ah,2ch

int 21h

mov ax, cx ; Получаем часы

and ax,0ff00h

shr ax,8

aam

add ax,3030h

mov byte ptr current\_time,ah

mov byte ptr [current\_time+2],al

mov ax, cx ; Получаем минуты

and ax,0ffh

aam

add ax,3030h

mov byte ptr [current\_time+6],ah

mov byte ptr [current\_time+8],al

mov ax, dx ; Получаем секунды

and ax,0ff00h

shr ax,8

aam

add ax,3030h

mov byte ptr [current\_time+12],ah

mov byte ptr [current\_time+14],al

add ax,3030h

pop dx cx ax

mov sp,bp

pop bp

ret

get\_time endp

; Процедура получения кординат вывода на экране

get\_coord proc

push bp

mov bp, sp

push cx bx es ds

call paramstr

mov al, byte ptr paramstr\_array

or al,al

je @@done

xor cx,cx

lea bx, [paramstr\_array+1]

mov cl, byte ptr [bx]

cmp cl,2

jne @@done

inc cl

inc bx

push es

push ds

pop es

mov ax,seg paramstr\_array

mov ds,ax

lea di,coord

mov si,bx

rep movsb

pop es

mov ah, byte ptr coord

mov al, byte ptr [coord+1]

sub ax, 3030h

mov byte ptr [coord+1],ah

xor ah,ah

mov cl,80

mul cl

or al,al

je @norm\_col

cmp al, 80

jbe @norm\_col

mov al,0

@norm\_col:

mov byte ptr [coord],al

mov al, byte ptr [coord+1]

mov cl,24

mul cl

cmp al, 24

jbe @norm\_row

mov al,0

@norm\_row:

mov byte ptr [coord+1], al

@@done:

pop ds es bx cx

mov sp,bp

pop bp

ret

get\_coord endp

code ends

; Зарезервируем стек

stack segment para stack

dw 200h dup(0)

stack ends

; Сегмент необходим для вычисления размера программы

zzzz segment

zzzz ends

end

Листинг 6 – файл draw\_w.asm, подпрограмма отрисовки окн а программы

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* ПОДПРОГРАММА ОТРИСОВКИ ОКНА ПРОГРАММЫ \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.model SMALL

public draw\_window

public TWind

include struc.inc

EXTRN write:PROC, gotoxy:PROC

data segment PARA PUBLIC 'data'

;------ screen symbols--------

; Символы для отрисовки рамки окна

leftTop equ 201

rightTop equ 187

horizontal equ 205

vertical equ 186

leftBottom equ 200

rightBottom equ 188

; \*\*\*\*\*\* symbols \*\*\*\*\*\*\*\*\*\*

color equ 01h ; Цвет рамки программы

video equ 0b800h ; Видео сегмент

coord dw 0 ; Адрес видео

dw 0b800h

wnd dd 0 ; Описатель окна

height dw 0 ; Хранит высоту окна

width dw 0 ; Хранит ширину окна

data ends

code segment PARA PUBLIC 'code'

assume cs:code, ds:data

; Процедура рисует рамку окна программы

; Вход BP+8 - SEG STRUCT

; bp+6 - offset struct

draw\_window proc far

push bp

mov bp,sp

push ax cx dx bx es di ds

mov ax, seg coord

mov ds, ax

mov ax, word ptr [bp+8]

mov word ptr [wnd+2],ax

mov ax,word ptr [bp+6]

mov word ptr [wnd],ax

les bx, dword ptr wnd

mov dh,es:[bx].TWind.left\_top\_row

mov dl,es:[bx].TWind.left\_top\_col

mov coord,dx

les dx,dword ptr coord

xor ax,ax

mov al,160

mul dh

xor dh,dh

shl dl,1

add ax,dx

mov di,ax

mov al, leftTop

mov ah, color

stosw

mov word ptr coord,di

xor cx,cx

les bx, dword ptr wnd

mov cl, byte ptr es:[bx].TWind.right\_bottom\_col

sub cl, byte ptr es:[bx].TWind.left\_top\_col

mov width, cx

dec cl

les di, dword ptr coord

mov al, horizontal

rep stosw

mov al, rightTop

stosw

mov word ptr coord,di

xor cx,cx

les bx, dword ptr wnd

mov cl,byte ptr es:[bx].TWind.right\_bottom\_row

sub cl, byte ptr es:[bx].TWind.left\_top\_row

mov height, cx

les di, dword ptr coord

mov al, vertical

dec cl

@left\_vertical:

add di,158

stosw

loop @left\_vertical

add di,158

mov al, rightBottom

stosw

mov cx,width

sub di,width

sub di,width

dec cx

mov al,horizontal

rep stosw

sub di,width

sub di,width

mov al,leftBottom

stosw

mov al,vertical

mov cx, height

dec cx

@right\_vertical:

sub di,162

stosw

loop @right\_vertical

pop ds

pop di

pop es

pop bx

pop dx

pop cx

pop ax

mov sp,bp

pop bp

retf 4

draw\_window endp

code ends

end

Листинг 7 – файл exec.asm, подпрограмма запуска дочернего процесса

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* ПОДПРОГРАММА ЗАПУСКА ДОЧЕРНЕГО ПРОЦЕССА \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.model SMALL

public exec

code segment PARA PUBLIC 'code'

assume cs:code

start proc

start endp

; Вход:

; [bp+6] - addr child process name

; [bp+8] - addr param block for child process

; [bp+10] - seg where data

; Выход:

; - при ошибке устанавливается флаг С

; - Для анализа кода возврата надо проанализировать al

; - вызвать 4Dh сразу после возврата из дочернего процесса

exec proc far

push bp

mov bp, sp

push ax bx dx es ds

mov ax, [bp+10]

mov es, ax

mov ds,ax

mov ah, 4bh

mov al, 0

mov bx, [bp+8]

mov dx, [bp+6]

int 21h

pop ds es dx bx ax

mov sp, bp

pop bp

ret 6

exec endp

code ends

end

Листинг 8 – файл file.asm, подпрограммы работы с файлом

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* ПОДПРОГРАММЫ ДЛЯ РАБОТЫ С ФАЙЛОМ \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.model SMALL

public open

public read

public seek\_start

code segment PARA PUBLIC 'code'

assume cs:code

; Вход: bp+4 - режим: 0 - чтение

; 1 - запись

; 2 - чтение/запись

; bp+6 - lpzStr - имя файла

; Выход: ax - handle

open proc

push bp

mov bp, sp

push dx ds

mov ax,[bp+8]

mov ds,ax

mov ah, 3dh

mov al,byte ptr [bp+4]

mov dx, [bp+6]

int 21h

pop ds dx

mov sp, bp

pop bp

ret 6

open endp

; вход: bp+4(cx) - количество байт котрые попытаемся прочитать.

; bp+6 - handle

; DS:DX - ADDR BUFFER

; выход: cx - количество прочитанных байт

read proc

push bp

mov bp, sp

push bx cx

mov cx, [bp+4]

mov ah, 3fh

mov bx, [bp+6]

xor al,al

int 21h

pop cx bx

mov sp,bp

pop bp

ret 4

read endp

; Процедура позиционирования в файле

; Вход: ah - 42h

; al - 00h - абсолютное смещение от начала файла

; 01h - смещение от текущей позиции

; 02h - смещение от конца файла

; cx - старший байт смещения

; dx - младший байт смещения

; bp+4 - handle

; Выход: AX - Код ошибки, если установлен флаг переноса CF

; Младший байт текущей позиции, если флаг переноса CF сброшен

; DX Старший байт текущей позиции

seek\_start proc

push bp

mov bp, sp

push cx dx ax bx

mov bx, [bp+4]

mov ah,42h

mov cx,0

mov dx,0

mov al,00h

int 21h

pop bx ax dx cx

mov sp,bp

pop bp

ret 2

seek\_start endp

code ends

end

Листинг 9 – файл int\_macr.inc, модуль вспомогательных макросов

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* Модуль вспомогательных макросов \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bios\_process\_linked\_seg equ 040h ; Сегмент и смещение пользовательских данных

bios\_process\_linked\_offs equ 0f0h ; предназначенных для мнэпроцессного

; взаимодействия

; es:di - The address of process in the process table

; Макрос устанавливает указатель es:di на

; первую запись в таблице потоков по номеру потока

@get\_ptr\_to\_data macro num

xor ax,ax

mov al, bios\_process\_linked\_seg

mov es,ax

mov al, bios\_process\_linked\_offs

mov bx, ax

mov di,word ptr es:[bx+4\*num]

mov ax,word ptr es:[bx+2+4\*num]

mov es,ax

endm

; Макрос устанавливает указатель es:di на

; произвольную запись в таблице потоков по номеру потока

@get\_ptr\_to\_thread\_data macro process\_num

xor ax,ax

mov al, bios\_process\_linked\_seg

mov es,ax

mov al, bios\_process\_linked\_offs

mov bx, ax

mov di,word ptr es:[bx]

mov ax,word ptr es:[bx+2]

mov es,ax

mov bx,process\_num

mov ax,size TThread

mul bx

add di,ax

endm

; Макрос инициализирует нлвую запись в ьаблице процессов

@fork macro process\_num, savearea

pushf

mov ax,cs

push ax

lea ax, savearea

push ax

xor ax,ax

push ax bx cx dx si di ds es

xor ax,ax

push ax ;bp

@get\_ptr\_to\_thread\_data process\_num

pushf

pop ax

stosw

mov ax,cs ;cs

stosw

lea ax, savearea ;ip

stosw

mov cx, 6 ;ax, bx, cx, dx, si, di

mov ax,0

rep stosw

mov ax, data ;ds

stosw

mov ax, 0 ;es

stosw

mov ax, bp

stosw ;bp

mov ax,sp

;sub ax,24

stosw ;sp

mov ax,ss

stosw ;ss

mov al, 1

stosb

xor ax,ax

mov ax,bp

endm

; Макрос устанавливает статус потока нефктивным

@deactivate\_process macro process\_num

@get\_ptr\_to\_thread\_data process\_num

mov al,0

mov byte ptr es:[di].TThread.status,al

endm

; Макрос восстанавливает старое значение вектор прерывания

@restore\_vect macro inum, savearea

push ax es dx

mov ax, 0

mov es, ax

mov ax, word ptr cs:[savearea]

mov dx, word ptr cs:[savearea+2]

cli

mov es:[inum\*4], ax

mov es:[inum\*4+2], dx

sti

pop dx es ax

endm

; Макрос сохраняет вектор прерывания и устанавливает новое значение

@change\_vect macro inum, new, savearea

push ax es dx

mov ax,0

mov es,ax

mov ax,es:[4\*inum]

mov dx,es:[4\*inum+2]

mov word ptr cs:[savearea],ax

mov word ptr cs:[savearea+2],dx

mov ax,offset cs:new

cli

mov es:[4\*inum], ax

mov es:[4\*inum+2], cs

sti

pop dx es ax

endm

;\*\*\*\*\*\*\*\*\*\*\*\*\*\* KEYBOARD \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Макрос для отработки аппаратного прерывания клавиатуры

@perepere\_int09 macro

;------ следующий код необходим для отработки аппаратного прерывания

in al,kb\_ctrl ;взять значениe порта управления клавиатурой

mov ah,al ; сохранить его

or al,80h ;установить бит разрешения для клавиатуры

out kb\_ctrl,al ; и вывести его в управляющий порт

xchg ah,al ;извлечь исходное значение порта

out kb\_ctrl,al ; и записать его обратно

mov al,eoi ;послать сигнал "конец прерывания"

out int\_ctrl,al ; контроллеру прерываний 8259

endm

;\*\*\*\*\*\*\*\*\*\*\*\*\*\* KEYBOARD \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Листинг 10 – файл int.asm, подпрограммы обработки прерываний клавиатуры

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* ПОДПРОГРАММЫ ДЛЯ ОБРАБОТКИ ПРЕРЫВАНИЙ КЛАВИАТУРЫ \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.model SMALL

include int\_macr.inc

include struc.inc

public new\_09h

public old\_09h

public is\_all

EXTRN dump:PROC

code SEGMENT PARA PUBLIC 'code'

assume cs:code, ds:data

jmp init

is\_all dw 0

int\_ctrl equ 20H ; Порт кправления прерыванием

eoi equ 20h ; Сбросить команду управления прерыванием

kb\_data equ 60h ; Порт данных клавиатуры

kb\_ctrl equ 61h ; Порт управления клавиатуры

kb\_f1\_msg db 'F',03h,'1',03h

kb\_f2\_msg db 'F',04h,'2',04h

kb\_right\_arrow\_msg db 'R',09h,'T',09h

kb\_left\_arrow\_msg db 'L',02h,'T',02h

kb\_up\_arrow\_msg db 'U',03h,'P',03h

kb\_down\_arrow\_msg db 'D',04h,'N',04h

kb\_esc equ 01h

kb\_F1 equ 3Bh ;зап./ост. поток чтения из файла

kb\_F2 equ 3Ch ;зап./ост. поток вывода на экран информации из файла

kb\_F3 equ 3Dh ;зап./ост. вывод заголовка

kb\_F4 equ 3Eh ;зап./ост. движущийся обьект

kb\_F5 equ 3Fh ;зап./ост. часы

kb\_F6 equ 40h

kb\_F7 equ 41h

kb\_F8 equ 42h

kb\_F9 equ 43h

kb\_right\_arrow equ 4dh ; Стрелка вверх

kb\_left\_arrow equ 4bh ; Стрелка влево

kb\_down\_arrow equ 50h ; Стрелка вниз

kb\_up\_arrow equ 48h ; Стрелка вверх

kb\_r equ 13h

old\_09h dd 0 ; Старый вектор 09

@kb\_handler macro sting

@perepere\_int09

; обработчики нажатия клавиши

push cs

lea ax, sting

push ax

xor ax,ax

mov ax, 4

push ax

call far ptr dump

endm

@kb\_handler\_start\_stop\_process macro process\_number

; @perepere\_int09

; обработчики нажатия клавиши

@get\_ptr\_to\_thread\_data process\_number

mov ax,0

mov al, es:[di].TThread.status

inc al

mov dl,2

div dl

mov es:[di].TThread.status, ah

jmp @to\_exit

endm

new\_09h proc far

cli

; pushf

; call cs:old\_09h

push ax bx cx dx si di ds es bp

in al,kb\_data ;читать ключ

cmp al,kb\_F1

jne @dont\_F1

@kb\_handler\_start\_stop\_process 1

@dont\_F1:

cmp al,kb\_F2

jne @dont\_F2

@kb\_handler\_start\_stop\_process 2

@dont\_F2:

cmp al,kb\_F3

jne @dont\_F3

@kb\_handler\_start\_stop\_process 3

@dont\_F3:

cmp al,kb\_F4

jne @dont\_F4

@kb\_handler\_start\_stop\_process 4

@dont\_F4:

cmp al,kb\_F5

jne @dont\_F5

@kb\_handler\_start\_stop\_process 5

@dont\_F5:

cmp al, kb\_right\_arrow

jne @dont\_kb\_right\_arrow

@perepere\_int09

@kb\_handler kb\_right\_arrow\_msg

@get\_ptr\_to\_data 1

mov byte ptr es:[di+2], 0

mov byte ptr es:[di+1], 1

@dont\_kb\_right\_arrow:

cmp al, kb\_left\_arrow

jne @dont\_kb\_left\_arrow

@perepere\_int09

@kb\_handler kb\_left\_arrow\_msg

@get\_ptr\_to\_data 1

mov byte ptr es:[di+2], 0

mov byte ptr es:[di+1], -1

@dont\_kb\_left\_arrow:

cmp al, kb\_down\_arrow

jne @dont\_kb\_down\_arrow

@perepere\_int09

@kb\_handler kb\_down\_arrow\_msg

@get\_ptr\_to\_data 1

mov byte ptr es:[di+2], 0

mov byte ptr es:[di], 1

@dont\_kb\_down\_arrow:

cmp al, kb\_up\_arrow

jne @dont\_kb\_up\_arrow

@perepere\_int09

@kb\_handler kb\_up\_arrow\_msg

@get\_ptr\_to\_data 1

mov byte ptr es:[di+2], 0

mov byte ptr es:[di], -1

@dont\_kb\_up\_arrow:

cmp al, kb\_r

jne @dont\_kb\_r

@perepere\_int09

@get\_ptr\_to\_data 1

mov byte ptr es:[di+2], 1

mov byte ptr es:[di+1], 1

mov byte ptr es:[di], 1

@dont\_kb\_r:

cmp al,kb\_esc ;

jne @dont\_esc ;

call do\_esc

@dont\_esc:

; нет, уйти на исходный обработчик

pop bp es ds di si dx cx bx ax

jmp cs:[old\_09h] ;переход на первоначальный обработчик

do\_esc proc

mov is\_all,1

ret

do\_esc endp

restore\_09h:

@restore\_vect 09h old\_09h

@to\_exit:

pop bp es ds di si dx cx bx ax

sti

jmp cs:[old\_09h]

new\_09h endp

resize equ $-new\_09h

init:

mov ax, data

mov ds, ax

; @change\_vect 9 new\_09h old\_09h

@test:

cmp is\_all,0

jne @its\_all

jmp @test

mov dx,(resize+10fh)/16

mov ax, 3100h

int 21h

@its\_all:

mov ax, 4c00h

int 21h

code ends

data segment PARA PUBLIC 'data'

data ends

zzz segment

zzz ends

end

Листинг 11 – файл paramstr.asm, подпрограмма получения строки параметров

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* ПОДПРОГРАММА ПОЛУЧЕНИЯ СТРОКИ ПАРАМЕТРОВ \*

; \* \*

; \* Примечание: \*

; \* Строка параметров возвращается в указателе на глобальную \*

; \* переменную paramstr\_array. Первый байт содержит количество \*

; \* строк. Каждая строка имеет следующий формат. Первый байт \*

; \* это количество символов в строке, затем сама бфйтовая строка\*

; \* \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.MODEL SMALL

public paramstr

public paramstr\_array

code segment PARA public USE16 'code'

assume cs:code, ds:data, es:data

paramstr proc

push bp

mov bp,sp

; Make PSP promt adress

mov ax, data

mov ds, ax

mov ax, es

mov word ptr [cmd\_tail\_addr+2],ax

push ds

pop es

lds si,cmd\_tail\_addr

xor cx,cx

xor bx,bx

mov cl, [si]

mov bl, [si]

inc si

; Trim the head

cld

@loads\_bytes\_head:

lodsb

cmp al,' '

je @next\_byte\_head

jmp @exit\_trim\_head

@next\_byte\_head:

loop @loads\_bytes\_head

@exit\_trim\_head:

; Get offset start

mov ax,bx

sub ax,cx

sub bx,ax

mov word ptr es:[supl\_offset\_start\_at],ax

lds si, es:[cmd\_tail\_addr]

xor cx,cx

xor bx,bx

mov cl, [si]

mov bl, [si]

; si = prompt length

; cx = prompt length - head offset start

add si,cx

sub cx,ax

std

@loads\_bytes\_tail:

lodsb

cmp al,' '

je @next\_byte\_tail

jmp @exit\_trim\_tail

@next\_byte\_tail:

loop @loads\_bytes\_head

@exit\_trim\_tail:

inc cx

mov bx, cx

mov ax, es:[supl\_offset\_start\_at]

sub bx,ax

mov cx,bx

; cx = string length without spaces

mov es:[tail\_raw\_length],cl

lds si, es:[cmd\_tail\_addr]

inc si

mov ax, es:[supl\_offset\_start\_at]

add si,ax

lea di, es:[tail\_raw]

cld

rep movsb

mov byte ptr es:[di], 0dh

push es

pop ds

lea si, tail\_raw

lea di, paramstr\_array

inc di

cld

@split\_string:

; Пропуск разделителей

@skip\_delimiters:

cmp al, 0Dh ; Если конец строки (0Dh), завершаем

je @split\_done

; Загружаем символ из строки в AL

mov al, byte ptr ds:[si]

inc si

cmp al, ' ' ; Если пробел, пропускаем

je @split\_string

cmp al, '/' ; Если '/', пропускаем

je @skip\_delimiters

cmp al, 0Dh ; Если конец строки (0Dh), завершаем

je @split\_done

; Начало новой подстроки

dec si ; Возвращаемся к первому символу подстроки

mov dx, di ; Сохраняем начало подстроки

mov bx, 0 ; Счетчик длины подстроки

inc di

; Копирование подстроки

@copy\_substring:

lodsb ; Загружаем символ из строки в AL

cmp al, 20h ; Если пробел, завершаем подстроку

je @end\_substring

cmp al, '/' ; Если '/', пропускаем

je @end\_substring

cmp al, 0Dh ; Если конец строки, завершаем подстроку

je @end\_substring

mov byte ptr es:[di], al

inc di

inc bx ; Увеличиваем счетчик длины

jmp @copy\_substring

; Завершение подстроки

@end\_substring:

mov byte ptr [si-1], 0 ; Заменяем разделитель на 0 (конец строки)

xchg di,dx

mov byte ptr [di], bl ; Сохраняем длину подстроки в первый байт

xchg di,dx

inc byte ptr paramstr\_array ; Увеличиваем счетчик подстрок

jmp @split\_string ; Переходим к следующей подстроке

@split\_done:

; Завершение программы

@out\_prog:

mov sp,bp

pop bp

ret

paramstr endp

code ends

data segment PARA PUBLIC 'data'

cmd\_tail\_addr dd 00000080h ; Адрес строки параметров в PSP

tail\_raw\_length db 0 ; Переменная хранит длину сырой строки

tail\_raw db 254 dup(0) ; Переменная хранит сырую строку

supl\_offset\_start\_at dw 0 ; Вспомогательная переменная

paramstr\_array db 256 dup (0) ; Массив строк параметров

data ends

end

Листинг 12 – файл schedule.asm, подпрограммы планировщика

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* ПОДПРОГРАММА ПЛАНИРОВЩИКА \*

; \* \*

; \* Примечание: программа написана из программы по \*

; \* Лабораторной работе №3 \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.model SMALL

include int\_macr.inc

include struc.inc

public old\_08h

public scheduler

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* variables \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public disableHardwareEvents

public threadsRegistered

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* variables \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

code segment PARA PUBLIC 'code'

assume cs:code, ds:code

disableHardwareEvents dw 0 ; Флаг вкл/отк перерывание стоковое прерывание

threadsRegistered dw 0 ; Количество зарегистрированных потоков в таблице потоков

parallelStart dw 1 ; Первый проход

threadNumber dw 0 ; Номер текущего процесса

tsoffset\_start dw 0 ; Вспомогательная переменная, хранит смещение

NumPrev dw 0 ; Хранит предыдущий номер потока

preemptiveSwitch dw 1 ; Флаг разрешено/запрещено переключение потоков

ThreadTable dw 0 ; Смещение твблицы потоков

mainss dw 0 ; SS - основного процесса

mainsp dw 0 ; SP - основного процесса

mainds dw 0 ; DS - основного процесса

tsoffset dw 0 ; Вспомогательная переменная, хранит смещение

old\_08h dd 0 ; Адрес стокового прерывания 08

video\_addr dw 0 ; Адрес видео

dw 0b800h

str\_c equ 0bh ; Цвет строки

str\_c\_yellow equ 0eh ; Цвет строки желтый

str\_c\_white equ 0fh ; Цвет строки белый

pos\_to\_out\_state equ 80\*2\*23 ; Константа, кординаты вывода на экран

; Строка - активный поток

status\_str db 'I',str\_c,'D',str\_c, ' '

db str\_c,'а',str\_c,'к',str\_c,'т',str\_c,'и',str\_c,'в',str\_c,'н',str\_c

db 'о',str\_c,'г',str\_c,'о',str\_c,' ',str\_c,'п',str\_c,'о',str\_c

db 'т',str\_c,'о',str\_c,'к',str\_c,'а',str\_c,':',str\_c,' ',str\_c, ' ',str\_c,' ',str\_c ;46

; Строка статусы потоков

thread\_str db ' ', str\_c\_yellow

db 'I',str\_c\_yellow,'D',str\_c\_yellow,'#',str\_c\_yellow,'0',str\_c\_yellow,'-',str\_c\_yellow,' ',str\_c\_yellow,' ',str\_c\_yellow,' ',str\_c\_yellow

db 'I',str\_c\_yellow,'D',str\_c\_yellow,'#',str\_c\_yellow,'1',str\_c\_yellow,'-',str\_c\_yellow,' ',str\_c\_yellow,' ',str\_c\_yellow,' ',str\_c\_yellow

db 'I',str\_c\_yellow,'D',str\_c\_yellow,'#',str\_c\_yellow,'2',str\_c\_yellow,'-',str\_c\_yellow,' ',str\_c\_yellow,' ',str\_c\_yellow,' ',str\_c\_yellow

db 'I',str\_c\_yellow,'D',str\_c\_yellow,'#',str\_c\_yellow,'3',str\_c\_yellow,'-',str\_c\_yellow,' ',str\_c\_yellow,' ',str\_c\_yellow,' ',str\_c\_yellow

db 'I',str\_c\_yellow,'D',str\_c\_yellow,'#',str\_c\_yellow,'4',str\_c\_yellow,'-',str\_c\_yellow,' ',str\_c\_yellow,' ',str\_c\_yellow,' ',str\_c\_yellow

db 'I',str\_c\_yellow,'D',str\_c\_yellow,'#',str\_c\_yellow,'5',str\_c\_yellow,'-',str\_c\_yellow,' ',str\_c\_yellow,' ',str\_c\_yellow,' ',str\_c\_yellow ;80

status\_threads\_table db 10 dup(-1)

scheduler proc far

push ax bx cx dx si di ds es

push bp

mov bp,sp

; \*\*\*\*\*\*\*\* call harware handler old\_08h \*\*\*\*\*\*\*\*\*\*\*\*\*

mov ax, cs:disableHardwareEvents

cmp ax,1

je @disableHardwareEvents

pushf

call cs:old\_08h

@disableHardwareEvents:

mov ax, cs:preemptiveSwitch ; if preemptiveSwitch or DisableHardwareEvents then begin {wrap begin --- start}

or ax,ax

jnz @do\_subrutine

mov ax, cs:disableHardwareEvents

or ax,ax

jnz @do\_subrutine

jmp @do\_not\_subrutine

@do\_subrutine:

mov ax, cs:threadsRegistered ;if (ThreadsRegistered > 0) or (parallelStart) then begin {first begin --- start}

cmp ax, 0

jg @go\_to\_save\_current\_process

mov ax,cs:parallelStart

cmp cs:parallelStart,1

je @go\_to\_save\_current\_process

jmp @not\_change\_processes

@go\_to\_save\_current\_process:

cli

mov ax,cs:parallelStart

cmp cs:parallelStart,1 ;if ParallelStart then begin {start parallel}

jne @not\_parallel\_start

@it\_is\_first\_save\_only\_to\_main\_process:

mov ax, sp

mov cs:mainsp, ax

mov ax, ss

mov cs:mainss, ax

mov cs:mainds, ds

push ds

mov ds, ax

mov ax, bp

mov si, ax

xor ax,ax

mov al, bios\_process\_linked\_seg

mov es,ax

mov al, bios\_process\_linked\_offs

mov bx, ax

mov di,word ptr es:[bx]

mov cs:ThreadTable, di

mov ax,word ptr es:[bx+2]

mov es,ax ; es:di -> tableThreads

add di, 22

mov cx, 12

@l:

mov ax, word ptr ds:[si]

mov word ptr es:[di], ax

add si, 2

sub di, 2

loop @l

mov ax, cs:ThreadTable

add ax, 24

mov di, ax

mov ax, bp

stosw

mov ax, ss

stosw

pop ds

jmp @end\_subroutine

@not\_parallel\_start: ;if not ParallelStart then begin

; call far ptr not\_parallel\_start\_sub\_routine

mov ax, cs:threadNumber ;tsoffset := (ThreadNumber + 1) \* sizeof(TThreadStateWord) - 5; {29-7=2 <=> 29 -sizeof(word)\*3 - sizeof(byte)}

inc ax

mov bx, size TThread

mul bx

sub ax,7

mov cs:tsoffset,ax

add ax,2

mov cs:tsoffset\_start,ax ; tsoffset\_start := tsOffset + 2;

push ds

mov ax, ss

mov ds, ax

mov ax, bp

mov si, ax

xor ax,ax

mov al, bios\_process\_linked\_seg

mov es,ax

mov al, bios\_process\_linked\_offs

mov bx, ax

mov di,word ptr es:[bx]

mov cs:ThreadTable, di

mov ax,word ptr es:[bx+2]

mov es,ax

add di, cs:tsoffset

mov cx, 12

@l1:

mov ax, word ptr ds:[si]

mov word ptr es:[di], ax

add si, 2

sub di, 2

loop @l1

mov ax, cs:ThreadTable

add ax, cs:tsoffset\_start

mov di, ax

mov ax, bp

stosw

mov ax, ss

stosw

pop ds ; end {store to TS}

@end\_subroutine:

mov ax, word ptr cs:threadNumber ;NumPrev := ThreadNumber;

mov word ptr cs:NumPrev, ax

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;call make\_new\_thread\_number

@repeat:

xor dx,dx

mov ax,cs:threadNumber

inc ax

mov cx, cs:ThreadsRegistered

div cx

mov cs:threadNumber, dx

cmp dx, cs:NumPrev

je @until

mov ax, dx

mov dx, size TThread

mul dx

add ax, cs:ThreadTable

mov di,ax

push ds

mov ds, word ptr cs:mainds

mov al, [di].TThread.status

pop ds

cmp al, 1

je @until

jmp @repeat

@until: ; until (ThreadNumber = NumPrev) or TS[ThreadNumber].active;

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;if ts[ThreadNumber].active and ((ThreadNumber <> NumPrev) or parallelStart) {loading from TS}

mov ax, cs:threadNumber

mov dx, size TThread

mul dx

add ax, cs:threadTable

mov di,ax

push ds

mov ds, word ptr cs:mainds

mov al, [di].TThread.status

pop ds

cmp al, 1

jne @no\_load\_from\_TS

mov ax, cs:threadNumber

cmp ax, cs:NumPrev

jne @load\_from\_TS

mov ax, cs:parallelStart

cmp ax, 0

je @no\_load\_from\_TS ;begin

@load\_from\_TS:

mov ax, cs:threadNumber ;tsOffset := (ThreadNumber + 1) \* sizeof(TThreadStateWord) - 3; {26}

inc ax

mov dx, size TThread

mul dx

sub ax,3

mov cs:tsOffset,ax

mov ax,cs:threadNumber ;tsoffset\_start := (ThreadNumber) \* sizeof(TThreadStateWord);

mov dx, size TThread

mul dx

mov cs:tsoffset\_start,ax

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

mov dx, ds

mov ax, cs:mainds

mov ds, ax

mov si, cs:threadTable

add si, cs:tsOffset

std

lodsw

mov ss, ax

lodsw

mov bp, ax

add ax, 12 \* 2

mov sp, ax

mov si, cs:ThreadTable

add si, cs:tsoffset\_start

cld

mov cx, 12

@m1:

lodsw

push ax

loop @m1

mov ds, dx ;end {loading from TS}

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@no\_load\_from\_TS:

; else if (not ts[Threadnumber].active) and (Threadnumber = NumPrev) then

mov ax, cs:threadNumber

mov dx, size TThread

mul dx

add ax, cs:ThreadTable

mov di,ax

push ds

mov ds, word ptr cs:mainds

mov al, [di].TThread.status

pop ds

cmp al, 1

je @not\_active\_and\_equal\_prev

mov ax, cs:threadNumber

cmp ax, cs:NumPrev

jne @not\_active\_and\_equal\_prev

@restore\_vect 1ch, old\_08h ; setintvec($8, @OldTimerVec);

mov ax, cs:mainss

mov ss, ax

mov ax, cs:mainsp

mov bp, cs:mainsp

sub ax, 12 \* 2

mov sp, ax

@not\_active\_and\_equal\_prev:

mov cs:parallelstart,0

@not\_change\_processes: ;{first begin --- end}

sti

@do\_not\_subrutine:

mov cs:disableHardwareEvents, 0

sti

; \*\*\*\*\*\*\*\*\*\*\*\*\* STAUS STDOUT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; This is not needed for the scheduler. It's just stdout status.

; Печатает статус активного потока

cld

mov cx,46

les di, dword ptr cs:video\_addr

xor ax, ax

mov ax,pos\_to\_out\_state

mov di,ax

mov ax,cs

mov ds,ax

lea si, status\_str

rep movsb

sub di,1

sub di,1

mov cx,1

mov al, byte ptr cs:threadNumber

add al,'0'

mov ah,0ch

rep stosw

lea si, thread\_str

mov cx, 96

rep movsb

; Печатает статусы потоков

mov cx, threadsRegistered

lea bx, status\_threads\_table

@get\_statuses:

mov ax, threadsRegistered

sub ax, cx

mov dx, size TThread

mul dx

add ax, cs:ThreadTable

mov si,ax

push ds

mov ds, word ptr cs:mainds

mov al, [si].TThread.status

pop ds

mov byte ptr ds:[bx], al

inc bx

loop @get\_statuses

lea bx, status\_threads\_table

mov cx, threadsRegistered

sub di, 82

@out\_statuses:

mov al, byte ptr [bx]

add al, 30h

mov ah, str\_c\_white

stosw

add di,2

inc bx

add di, 12

loop @out\_statuses

; \*\*\*\*\*\*\*\*\*\*\*\*\* STAUS STDOUT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

mov sp, bp

pop bp

pop es ds di si dx cx bx ax

iret

scheduler endp

code ends

data segment PARA PUBLIC 'data'

data ends

end

Листинг 13 – файл struc.inc, файл структур проекта

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* СТРУКТУРЫ ПРОГРАММЫ \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bios\_process\_linked\_seg equ 40h

bios\_process\_linked\_offs equ 0f0h

bios\_process\_linked\_offs\_second equ 0f4h

; Структура описатель окна, хранит внешние и внутренние кординаты

TWind STRUC

left\_top\_row db 0

left\_top\_col db 0

right\_bottom\_row db 0

right\_bottom\_col db 0

inner\_left\_top\_row db 0

inner\_left\_top\_col db 0

inner\_right\_bottom\_row db 0

inner\_right\_bottom\_col db 0

TWind ends

; Структура описатель потока, хранит контекст потока в таблице процессов

TThread STRUC

r\_flags dw 0

r\_cs dw 0

r\_ip dw 0

r\_ax dw 0

r\_bx dw 0

r\_cx dw 0

r\_dx dw 0

r\_si dw 0

r\_di dw 0

r\_ds dw 0

r\_es dw 0

r\_bp dw 0

r\_sp dw 0

r\_ss dw 0

status db

TThread ends

; 14 - words + 1 byte Длина структуры

Листинг 14 – файл system.asm, файл вспомогательных процедур

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* ВСПОМОГАТЕЛЬНЫЕ ПРОЦЕДУРЫ \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.model SMALL

public write

public clrscr

public gotoxy

public \_getxy

public put\_char

public atoi

public dump

code segment PARA PUBLIC 'code'

assume cs:code

; Процедура вывода на экран

; [bp+10 - source segment]

; [bp+8 - source offset]

; [bp+6 - count of symbols]

dump proc far

local

push bp

mov bp, sp

mov ds, [bp+10]

mov ax, 0b800h

mov es,ax

mov di,0

mov si, word ptr [bp+8]

mov cx, [bp+6]

cld

rep movsb

mov bp, sp

pop bp

retf 6

dump endp

; Процедура очистки экрана

clrscr proc

push bp

mov bp, sp

push ax cx ds es si di

mov ax, 0b800h

mov ds, ax

mov es,ax

mov si,0

mov di,0

mov cx, 80\*25

cld

lodsw

mov al,0

dec si

rep stosw

pop di si es ds cx ax

mov sp, bp

pop bp

ret

clrscr endp

; Процедура вывода символа

; в al - ASCII-выводимый символ

write proc

push bp

mov bp, sp

push bx

mov ah, 0eh

mov bh, 0

int 10h

pop bx

mov sp, bp

pop bp

ret

write endp

; Процедура перехода на кординаты экрана

; bh - видео режимы 0..8

; 0 - B8000h..b8F400h

; 1 - B9000h..BFF400h

; ...

; 7 - BF000h..BF400h

; bp+5 (dh) - Y (строка 0..24)

; bp+4 (dl) - X (столбец 0..78)

gotoxy proc

local

push bp

mov bp, sp

push dx bx ax

mov dx, [bp+4]

mov ah, 02h

mov bh, 0

int 10h

pop ax bx dx

mov sp, bp

pop bp

ret 2

gotoxy endp

;Процедура получения текущих кординат экрана

;Выход:

;DH = текущая строка курсора (см. функцию 02H)

;DL = текущий столбец курсора (см. функцию 02H)

;CH = текущая начальная строка экрана, содержащая курсор (см. функцию 01H)

;CL = текущая конечная строка экрана, содержащая курсор (см. функцию 01H)

\_getxy proc

push bp

mov bp,sp

push cx dx bx

mov bh,0

mov ah,3h

int 10h

mov ax, dx

pop bx dx cx

mov sp, bp

pop bp

ret

\_getxy endp

; Писать символ и атрибут в текущей позиции курсора

; Вход:

; bh - номер видео страницы

; al - записываемый символ (ASCII код)

; cx - счетчик (сколько экземпляров символа записать)

; bl - видео атрибут (текстовый режим) или цвет (графический режим)

put\_char proc

push bp

mov bp,sp

push ax bx

mov ah, 9

mov bh, 0

; mov bl, 70h

int 10h

pop bx ax

mov sp,bp

pop bp

ret

put\_char endp

; Перевод ыстроки в число

atoi proc

local

push bp

mov bp,sp

push bx cx di dx

mov bx,[bp+4]

xor cx,cx

mov cl,[bx]

inc bx

mov di,bx

xor ax,ax

xor bx,bx

xor dx,dx

mov dl,10

@convert:

mov bl,byte ptr [di]

cmp bx, 30h

jl @error

cmp bx,39h

jg @error

sub bx,30h

mul dl

add ax,bx

inc di

loop @convert

jmp @done

@error:

mov ax,-1

@done:

pop dx di cx bx

mov sp,bp

pop bp

ret 2

atoi endp

; Перевод числа в строку

hex\_to\_ascii proc

push bp

mov bp, sp

mov al, [bp+4]

aam

or ax, 3030h

mov sp, bp

pop bp

ret 2

hex\_to\_ascii endp

; Перевод bcd числа в bin

bcd\_to\_bin proc

push bp

mov sp, bp

push cx

mov cl, al

and al, 0Fh

shr cl, 4

mov ah, 10

mul cl

add al, ah

pop cx

mov sp, bp

pop bp

ret

bcd\_to\_bin endp

code ends

end

Листинг 15 – файл utils.inc, файл процедуры задержки

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* ВСПОМОГАТЕЛЬНАЯ ПОДПРОГРАММА ЗАДЕРЖКИ \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Макрос реализует задержку

delay macro time

local

push cx

mov cx, time

@@:

push cx

mov cx, 0ffffh

@@l:

loop @@l

pop cx

loop @@

pop cx

endm

; Макрос выводит строку на экран

print\_str macro str

push dx

mov ah,09h

lea dx, str

int 21h

pop dx

endm

Листинг 16 – файл c.bat, файл кимпиляции и сборки программ

@REM bat файл компилирует и собирает программы

@REM программа вывода чтения файла и вывода на экран

tasm /z/zi/n System.asm

tasm /z/zi/n Paramstr.asm

tasm /z/zi/n File.asm

tasm /z/zi/n Draw\_w.asm

tasm /z/zi/n readwrit.asm

tlink /v readwrit System Draw\_w File Paramstr

@REM pause

@REM readwrit.exe test1.txt

@REM программа вывода заголовка

tasm /z/zi/n System.asm

tasm /z/zi/n Paramstr.asm

tasm /z/zi/n Title.asm

tlink /v Title System Paramstr

@REM @REM pause

@REM @REM td title.exe 1

@REM программа движущегося обьекта

tasm /z/zi/n System.asm

tasm /z/zi/n Paramstr.asm

tasm /z/zi/n Draw\_w.asm

tasm /z/zi/n ball.asm

tasm /z/zi/n int.asm

tlink /v ball int System Paramstr Draw\_w

@REM @REM ball.exe

@REM программа вывода часов

tasm /z/zi/n System.asm

tasm /z/zi/n Paramstr.asm

tasm /z/zi/n clock.asm

tlink /v clock System Paramstr

@REM @REM @REM pause

@REM @REM @REM td clock.exe 11

@REM основная программа, запускает другие процессы, планировшик

@REM и устанавливает таблицу векторов

tasm /z/zi/n exec.asm

tasm /z/zi/n Paramstr.asm

tasm /z/zi/n System.asm

tasm /z/zi/n Draw\_w.asm

tasm /z/zi/n readwrit.asm

tasm /z/zi/n int.asm

tasm /z/zi/n Schedule.asm

tasm /z/zi/n main.asm

tlink /v main exec Draw\_w System Paramstr int Schedule

@REM @REM pause

main.exe 1 test1.txt 01

@REM td main.exe

Листинг 17 – файл prc\_ids.inc, файл c ID процессов

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* ID ПРОЦЕССОВ \*

; \* \*

; \* Примечание: \*

; \* Вынесены в отдельный файл для удобства разработки \*

; \* Каждая программа знает свой ID в таблице потоков независимо \*

; \* от других потоков \*

; \* \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

main\_process\_id equ 0

readwrite\_process\_id equ 1

title\_process\_id equ 3

ball\_process\_id equ 4

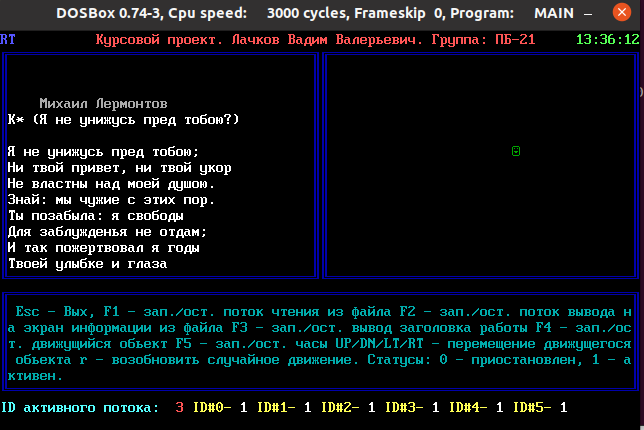
clock\_process\_id equ 5

# 4. Тестирование

Для выявления соответствия заданных в ТЗ параметров реальному результату выполним функциональное тестирование. В рамках тестирования проверим соответствующие компоненты и критерии:

* **пользовательский интерфейс;**
* **общая функциональность**
* **выполнение программы без зависаний**
* **корректное завершение программы**

**Пользовательский интерфейс соответствует спецификации и представлен на рисунке 4.**

Рисунок 4 – меню основной программы.

После запуска программы, в зависимости от входных параметров, строка проекта отображается вверху или внизу экрана. Текст в левом окне выводится без ошибок. Движущийся обьект управляется клавишами стрелок вверх, вниз, влево и вправо. Случайное движение движущегося обьекта восстанавливается после нажатия на клавишу «r». В верхнем левом углу экрана выводится сообщение с нажатой клавишей. В нижней часте экрана выводится строка с активным процессом, а также статусы процессов и их идентификаторы.

В ходе тестирования программа работает устойчиво, ошибок не обнаружено. Таким образом, по результатам тестирования делается вывод, что программа соответствует ТЗ и функциональное тестирование считается пройденным.

# Выводы

Разработанная программа служит наглядным пособием для изучения принципов работы операционных систем, в частности:

* механизмов многопоточности и синхронизации;
* планирования задач и распределения ресурсов;
* обработки прерываний и взаимодействия с пользователем.

Кроме того, проект имеет практическую ценность, так как демонстрирует подходы к созданию отзывчивых и устойчивых приложений, способных работать в условиях параллельного выполнения множества задач.

В ходе выполнения курсового проекта мной выполнен сбор информации, анализ актуальных вопросов, требующих решения с применением языка программирования ассемблера, мной были сформулированы требования и составлена спецификация к разрабатываемой программе. Выполнено проектирование: выбор аппаратной платформы и ОС, выбор языка программирования, компилятора, отладчика и среды разработки.

В части реализации выполнена разработка структуры программы, в работе представлены исходные тексты программы и модулей. Выполнено функциональное тестирования программы, и представлены результаты работы в виде снимка экрана. На основе полученных данных проанализированы и сделаны выводы о готовности программы, ее работоспособности.

В рамках курсового проекта удалось спроектировать модель многопоточной системы в операционной среде DOS. Получить навыки проектирования и реализации заданных алгоритмов по требованию ТЗ и реализовать готовый программный продукт.

# Список использованной литературы

1. Бах О.А. Курс лекций «Операционные системы». СибГУТИ.
2. Таненбфум Э., Вудхфлл А. Операционные системы: разработка и реализация (+CD). Класика CS. - Спб.: Питер, 2006. - 576с.: ил.
3. Финогенов К.Г. Самоучитель по системным функциям MS-DOS. - М.: МП «МАЛИП», 1993 — 262 с. - (Настольная книга пользователя ПК)
4. Программирование на языке ассемблера URL: https://natalia.appmat.ru/c&c++/assembler.html (Дата обращения: 23.04.2025).
5. Википедия. Свободная энциклопедия.– Режим доступа: <https://ru.wikipedia.org/wiki/GNU_Compiler_Collection> (Дата обращения: 01.04.2025).
6. Луридас, Панос. Алгоритмы для начинающих: теория и практика для разработчика /Панос Луридас:[пер. с англ. Е.М. Егоровой]. – Москва: Эксмо, 2018. – 608с.