Одесский Национальный Политехнический Университет

Кафедра «Компьютеризированные системы управления»

**КУРСОВАЯ РОБОТА**

по дисциплине «Современные технологии программирования»

на тему: «Эмуляция работы операционной системы»

Студента 3 курса, группы АТ–192

специальности «Автоматизация и компьютерно-интегрированные технологии»

Кононенко Н.В.

Руководитель: доц. Сперанский В.А.

Национальная шкала:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Количество баллов:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка: ECTS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члены комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

г. Одесса – 2021 год

Оглавление

[Введение 3](#_Toc58695182)

[Задание 5](#_Toc58695183)

[1. Структура приложения 7](#_Toc58695184)

[1.1. Модели 7](#_Toc58695185)

[1.1.1. Process 7](#_Toc58695186)

[1.1.2. Queue 8](#_Toc58695187)

[1.1.3. MemoryBlock 8](#_Toc58695188)

[1.1.4. MemoryScheduler 9](#_Toc58695189)

[1.1.5. CPU 9](#_Toc58695190)

[1.1.6. Core 9](#_Toc58695191)

[1.1.7. Timer 10](#_Toc58695192)

[1.1.8. Scheduler 10](#_Toc58695193)

[1.1.9. Вспомогательные классы 10](#_Toc58695194)

[1.1.9.1. Utils 10](#_Toc58695195)

[1.1.9.2. Configuration 11](#_Toc58695196)

[1.1.9.3. Data 11](#_Toc58695197)

[1.2. Представления 12](#_Toc58695198)

[1.3. Контроллеры 14](#_Toc58695199)

[1.3.1. MainController 14](#_Toc58695200)

[1.3.2. Controller 14](#_Toc58695201)

[1.3.3. resultController 15](#_Toc58695202)

[2. Инструкция 16](#_Toc58695203)

[2.1. Главное окно 16](#_Toc58695204)

[2.2. Окно эмуляции 16](#_Toc58695205)

[2.3. Окно результатов 17](#_Toc58695206)

[Выводы 18](#_Toc58695207)

[Список литературы 19](#_Toc58695208)

### Введение

***Операционная система (ОС)*** – это организованная совокупность программ и данных, которая выполняет функции посредника между пользователями и компьютером. ОС служит двум целям: во-первых, сделать компьютерную систему удобной для использования, и, во-вторых, эффективно использовать аппаратные средства компьютера.

ОС является *управляющей**программой*. Управляющая программа контролирует выполнение программ пользователей для предотвращения ошибок и неправильного использования компьютера.

ОС реализует множество различных функций, в том числе:

* определяет так называемый интерфейс пользователя;
* обеспечивает разделение аппаратных ресурсов между пользователями;
* дает возможность работать с общими данными в режиме коллективного пользования;
* планирует доступ пользователя к общим ресурсам;
* обеспечивает эффективное выполнение операций ввода-вывода;
* осуществляет восстановление информации и вычислительного процесса в случае ошибок.

Операционные системы могут различаться особенностями реализации внутренних алгоритмов управления основными ресурсами компьютера (процессорами, памятью, устройствами), особенностями использованных методов проектирования, типами аппаратных платформ, областями использования и многими другими свойствами.

В зависимости от особенностей использованного алгоритма управления процессором, операционные системы делят на многозадачные и однозадачные, многопользовательские и однопользовательские, на системы, поддерживающие многопоточную обработку и не поддерживающие ее, на многопроцессорные и однопроцессорные системы.

Важное влияние на облик операционной системы в целом, на возможности ее использования в той или иной области оказывают особенности и других подсистем управления локальными ресурсами  подсистем управления памятью, файлами, устройствами ввода-вывода.

Специфика ОС проявляется и в том, каким образом она реализует сетевые функции: распознавание и перенаправление в сеть запросов к удаленным ресурсам, передача сообщений по сети, выполнение удаленных запросов.

Многозадачные ОС подразделяются на три типа в соответствии с использованными при их разработке критериями эффективности:

* системы пакетной обработки (например, OC EC),
* системы разделения времени (UNIX, VMS),
* системы реального времени (QNX, RT/11).

Некоторые операционные системы могут совмещать в себе свойства систем разных типов, например, часть заданий может выполняться в режиме пакетной обработки, а часть  в режиме реального времени или в режиме разделения времени. В таких случаях режим пакетной обработки часто называют фоновым режимом.

Описать операционную систему можно только путем деления ее на меньшие компоненты. Не все ОС имеют одинаковую структуру. Однако во многих современных ОС ставится следующие компоненты:

* управление процессами;
* управление основной (оперативной) памятью;
* управление вторичной (внешней) памятью;
* управление вводом-выводом;
* управление файлами;
* защита системы;
* сетевое обслуживание;

Оптимизация работы ОС с процессами напрямую влияет на быстродействие обработки данных.

### Задание

Для модели вычислительной системы (ВС) с N-ядерным процессором и мультипрограммным режимом выполнения поступающих заданий требуется разработать программную систему для имитации процесса обслуживания заданий в вычислительных системах.

При построении модели функционирования вычислительной системы должны учитываться следующие основные моменты обслуживания заданий:

* генерация нового задания;
* постановка задания в очередь для ожидания момента освобождения процессора;
* выборка задания из очереди при освобождении процессора после обслуживания очередного задания.

**Генерация задания**:

Считается, что в распоряжении вычислительной системы имеется N ГБ оперативной памяти для размещения рабочей области процесса и M (3<=m<=5) ресурсов R1, R2,…, Rm, обращение к которым переводит процесс в состояние ожидания.

Генерация нового задания (процесса) может происходить:

* в интерактивном режиме по запросу пользователя
* автоматически системой как случайное событие

Каждый процесс характеризуется:

* именем;
* длиной рабочей области;
* интервалом непрерывного выполнения;
* причиной прекращения непрерывной работы (обращение к ресурсу или завершение работы);
* приоритетом, если он требуется используемым методом планирования процессора.

Перед постановкой задания в очередь имитируется размещения рабочей области процесса в оперативной памяти. В случае невозможности размещения процесс отвергается, в противном случае ему выделяется память и процесс помещается в очередь готовых заданий.

Размещение в ОП происходит одним из трёх методов:

1. первого подходящего;
2. наиболее подходящего;
3. наименее подходящего;

Выборка задания из очереди готовых процессов происходит в момент, когда текущий процесс исчерпал интервал непрерывной работы и освободил CPU.

В случае обращения к ресурсу процесс помещается в очередь к нему, причем время использования ресурса генерируется случайным образом.

В случае завершения процесс удаляется из очереди готовых процессов.

Все очереди к ресурсам обслуживаются алгоритмом FCFS (в порядке поступления). Считается, что в каждый момент времени процесс может обратиться только к одному ресурсу. По окончании работы с ресурсом процесс вновь помещается в очередь готовых заданий, причем генерируется новые интервал непрерывной работы и причина ее прекращения.

Таблица 1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n/n |  | Стратегия планирования | Наличие вытеснения | Способ организации очереди | | | | Динамическое повышение приоритета | Критерий вытеснения для SJF | |
|  | упорядоченный  список | не упорядочен.  список | список частично упорядочивается  через t тактов | каждому  приоритету своя очередь |
|  | По интервалу непрерывного выполнения | По оставшемуся времени |
| метод планирования памяти |
| 6 | 2 | SJF | - |  |  |  | + |  |  |  |

### Структура приложения

### Модели

### Process

Поля:

* name – имя процесса, генерируется автоматически („P” + ID)
* id – идентификатор процесса, передаётся из Queue
* priority – приоритет процесса, генерируется автоматически от 1 до n (где n = Configuration.processQuantity)
* memory – объём оперативный памяти, требуемый для выполнения процесса, генерируется автоматически (от Configuration.minProcessMemory до Configuration.maxProcessMemory
* time – время (в тактах), требуемой для выполнения процесса, генерируется автоматически

timeIn – время (такт) вхождения процесса в систему

* burstTime – количество времени (такты), которое процесс находится в обработке
* state – состояние процесса (При создании процесса всегда „New”)
* memoryBlock – блок памяти, в котором размещён процесс
* waitingTime – количество времени (такты), которое процесс ожидает в очереди.
* byLessTime – компаратор, для сортировки очереди и реализации стратегии планирования SJF

### Queue

Поля:

* allowedQueue – очередь процессов, получивших место в оперативной памяти (Согласно варианту заданию, каждый приоритет имеет свою очередь внутри allowedQueue)
* rejectedQueue – очередь процессов, для которых не удалось выделить памяти
* lastID – количество добавленных процессов в систему

Методы:

* initializeQueue - создание очередей процессов по приоритетам внутри allowedQueue = Configuration.prioritiesQuantity.
* add/add(int processQuantity) – генерация новых процессов и добавление их в очередь
* queueCleaning – очистка очереди от обработанных процессов (Обработанные процессы – процессы с State = State.Finished)

### MemoryBlock

Поля:

* start – начало размещения блока памяти
* end – конец размещения блока памяти
* byEnd – компаратор для сортировки блоков памяти в порядке размещения их в оперативной памяти
* byTheMostSuitable – компаратор для сортировки подходящих свободных блоков памяти (для реализации метода планирования памяти №2)

### MemoryScheduler

Поля:

* memoryBlocks – список блоков занятых блоков памяти

Методы:

* findFreeBlock – поиск свободного блока памяти, наиболее подходящего для процесса
* fillMemoryBlock – добавление найденного свободного блока памяти в memoryBlocks
* releaseMemoryBlock – освобождение блока памяти
* loadingOSIntoMemory – загрузка ОС в оперативную память (добавляет блок памяти размером Configuration.OSMemoryVolume в memoryBlocks)
* releaseMemory – освобождение оперативной памяти

### CPU

Поля:

* cores – массив ядер процессора (ресурсов)

Методы:

* fillCores – наполнение массива cores объектами Core, в количестве = Configuration.coresQuantity

### Core

Поля:

* isFree – занято ли Core обработкой или свободно
* process – процесс который находится в обработке на данном ядре
* processed – количество процессов, обработанных данным ядром

### Timer

Поля:

* time – количество времени (в тактах), прошедшее со времени включения системы

Методы:

* zeroing – обнуление таймера

### Scheduler

Поля:

* CPU cpu – процессор, находящийся под управлением OС
* MemoryScheduler memoryScheduler – планировщик памяти, управляемый ОС
* Queue queue – очередь процессов

Методы:

* addProcessToCore – добавление процесса на обработку в ядро
* processServiceInCPU – проверка, выполнился ли процесс. Если выполнился меняем состояние процесса на State.Finished и освобождаем ядро
* init – инициализация ОС. Загружает ОС в оперативную память и добавляет в очередь стартовые процессы, в количестве = Configuration.processQuantity
* calculateRunningTime – считает общее время занятости ядер
* run – главный метод. Запускает проверку очередей на наличие в них процессов, добавление процессов на обработку, processServiceInCPU, calculateRunningTime, очистку очередей от выполнившихся процессов(queue.queueCleaning).

### Вспомогательные классы

### Utils

Надстройка на java.util.Random для большего удобства с ним в рамках проекта.

Поля:

* Random random – объект класса random

Методы:

* getRandomNumber – возвращает случайное число в зависимости от переданных параметров

### Configuration

Конфигурационные данные для приложения

Поля:

* memoryVolume – объём оперативной памяти
* OSMemoryVolume – размер блока памяти для ОС
* minProcessMemory, maxProcessMemory – границы размера блока памяти для процесса
* maxProcessWorkTime – максимальное время работы процесса (в тактах)
* processName – одинаковая часть имени для всех процессов(„P”)
* prioritiesQuantity – количество приоритетов
* coresQuantity – количество ядер процессора
* timerDelay – задержка между тактами генератора (меньше – быстрее)
* startingProcesses – количество процессов, генерируемых с начала работы эмуляции

### Data

Данные собранные за время работы эмуляции. Используется для вывода информации об эксперименте по окончанию эмуляции.

Поля:

* executedProcesses – количество выполненных процессов
* waitingTime – сумма времени ожидания всех процессов (В контролере делится на к-во поступивших процессов и получается среднее время ожидания)
* deletedProcesses – количество удалённых процессов и процессов ушедших в rejectedQueue
* idleTime – время простоя процессора (Простой процессора = все ядра свободны)
* processesEnteredToCores – количество процессов предоставленных для обработки ядрам
* runningTime = сумма времени обработки всех процессов, поступивших на обработку (runningTime / processesEnteredToCores = среднее время обработки)
* dataZeroing – обнуление всех данных

### Представления

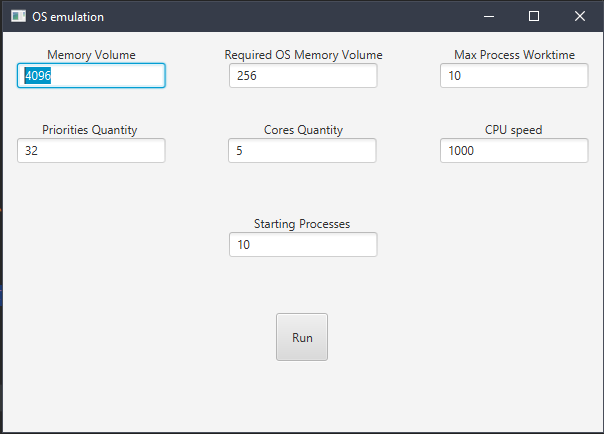


Рис.1. – Главное окно

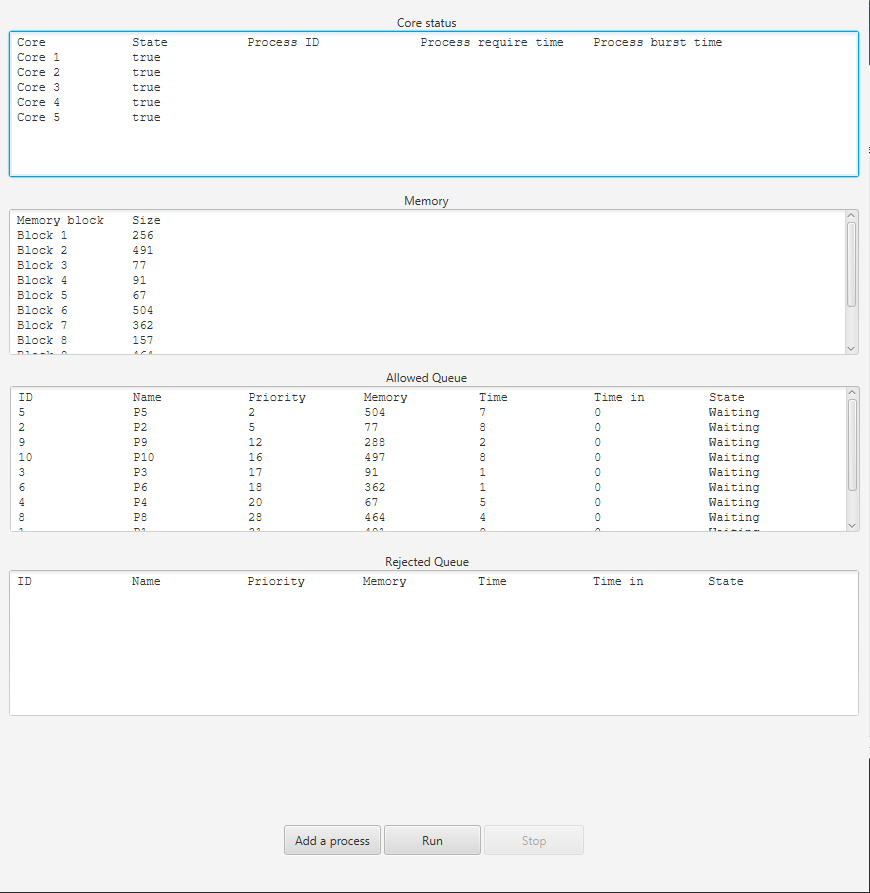


Рис.2. – Окно эмуляции

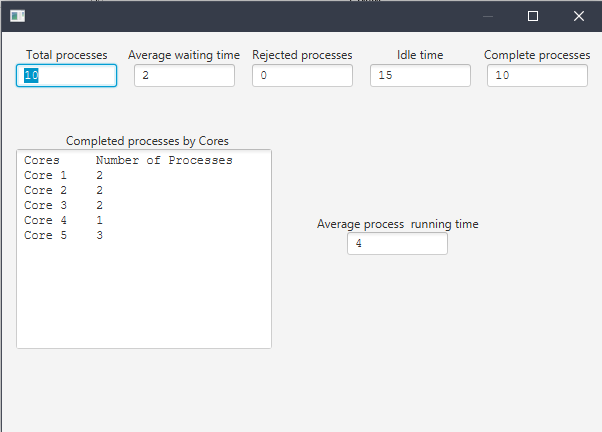


Рис.3. – Окно итогов

### Контроллеры

### MainController

Контроллер главного окна (Рис.1.)

Методы:

* сlick - срабатывает при нажатии на кнопку „Run”. Запускает метод setConfiguration, открывает окно эмуляции
* setConfiguration - Присваивает введённые значения полям в Configuration
* secondWindow – создаёт окно эмуляции

### Controller

Контроллер окна эмуляции (Рис.2.)

Поля:

* Scheduler scheduler – объект планировщик
* java.util.Timer timer – выполняет метод Task() каждые Configuration.delayTimer секунд

Методы:

* startEmulation – запуcкает эмуляцию
* print – выводит в текстовые поля информацию о текущем состоянии системы
* Task – возвращает объект TimerTask c переопределенным методом run. В методе run запускается print и scheduler.run(). Нужен для передачи TimerTask в timer
* addProcess -добавляет процесс в систему по нажатию кнопки „Add a process”
* stopTimer – останавливает работу timer
* openResultWindow – создаёт окно с результатами
* stopEmulation – принудительно завершает эмуляцию, открывает окно с результатами (Рис.3.)
* initialize – запускается при открытии окна. Обнуляет данные sample.Timer, MemoryScheduler, Data. Запускает Scheduler.init(), выводит начальное состояние системы.

### resultController

Методы:

* initialize - запускается при открытии окна. Выводит итоги работы эмуляции

Остальные методы выводят данные в текстовые поля

### Инструкция

### Главное окно

Перед началом работы необходимо ввести конфигурационные данные для системы:

* Memory Volume – устанавливает максимальный объём оперативной памяти
* Required OS Memory Volume – объём памяти, который будет занимать ОС
* Max Process WorkTime – максимальное время обработки процесса (в тактах
* Priorities Quantity – количество приоритетов
* Cores Quantity – количество ядер процессора
* CPU Speed – скорость работы процессора. Исчисляется в наносекундах (1с. = 1000). Чем меньше значение, тем меньше время между тактами процессора
* Starting Processes – количество изначально созданных процессов

Кнопка „Run” начинает эмуляцию. Перед тем как нажать на неё, необходимо удостоверится, что все поля заполнены, значения не являются строками и не равны 0.

После нажатия на кнопку открывается окно эмуляции

### Окно эмуляции

Перед началом работы эмуляции, пользователь может добавить ещё процессов в систему, нажав на кнопку „Add a process”. В текстовые поля выводится сведения о состоянии элементов системы.

Чтобы запустить эмуляцию, нажимаем кнопку „Run”.

Во время работы эмуляции можно добавить в систему ещё процессов кнопкой „Add a process”. Чтобы остановить эмуляцию и вывести сведения о ней, необходимо нажать на кнопку „Stop”.

### Окно результатов

В окне с результатами представлены сведения об итогах работы системы:

* Total processes – количество процессов поступивших в систему
* Average waiting time – время, которое процессы ждали доступа к ядру ( в тактах)
* Rejected processes – количество процессов, отправленных в очередь отказов
* Idle time – количество тактов, которое процесс работал в холостую (все ядра были свободны)
* Complete processes – количество процессов, выполненных за время эмуляции.
* Completed processes by Cores – количество процессов обработанных каждым ядром.
* Average process running time – среднее время обработки процесса

Чтобы закрыть окно, нужно нажать на крестик. После закрытия снова станет доступно главное окно, из которого можно заново запустить эмуляцию

### Выводы

В результате данной работы было создано приложение, имитирующее работу операционной системы с процессами. При разработке были выполнены следующие требования:

* Метод планирования памяти – метод наиболее подходящего
* Стратегия планирования – SJF
* Способ организации очереди – каждому приоритету своя очередь

В процессе работы над проектом были закреплены знания, полученные за курс «Современные технологии программирования», а так же улучшены навыки объектно-ориентированного программирования и проектирования приложений.

### Список литературы

1. Сетевые операционные системы / В. Г. Олифер, Н. А. Олифер. –СПб.: Питер, 2002. – 544 с.
2. Д. Цикритзис, Ф. Бернстайн. Операционные системы / пер. с англ. –М.: Мир, 1977. –336с.
3. П. Кейлингерт. Элементы операционных систем. Введение для пользователей / пер. с англ. –М.: Мир, 1985. -295с.
4. А. Шоу. Логическое проектирование операционных систем / пер. с англ. –М.: Мир, 1981. –360 с.
5. Таненбаум Э., Вудхалл А. Операционные системы. Разработка и реализация (+CD). Классика CS. 3-е изд. — СПб.: Питер, 2007. — 704 с: ил.
6. Ахо А., Хопкрофт Д., Ульман Д. – Структуры данных и алгоритмы.: Пер. с англ.: Уч. пос.- М., Издательский дом «Вильямс», 2016. – 400 с.
7. Вайсфельд М. Объектно-ориентирование мышление/ пер. с англ.: Питер, 2014. – 304 с.