МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Чувашский государственный университет имени И.Н. Ульянова»

Факультет информатики и вычислительной техники

Кафедра вычислительной техники

Отчёт по лабораторным работам по дисциплине  
«Операционные системы»

Выполнил: студ. гр. ИВТ-41-21

Андреев Д.А.

Чебоксары 2023

Содержание

[1. Структуры данных 2](#_Toc147410991)

[2. Интерфейс 3](#_Toc147410992)

[1) Кнопки 3](#_Toc147410993)

[3) Системные параметры 3](#_Toc147410994)

[4) Параметры задач 4](#_Toc147410995)

[5) Параметры команд 4](#_Toc147410996)

[6) Данные системы 5](#_Toc147410997)

[7) Таблица задач 5](#_Toc147410998)

[3. Описание программной части 6](#_Toc147410999)

[1) Включение/выключение системы 6](#_Toc147411000)

[2) Применение всех параметров 6](#_Toc147411001)

[3) Вывод параметров на экран 6](#_Toc147411002)

[4) Создание задач 6](#_Toc147411003)

[5) Работа планировщика 7](#_Toc147411004)

[6) Определение состояния задач 9](#_Toc147411005)

[7) Выполнение задач 9](#_Toc147411006)

1. **Структуры данных**

/// <summary>

/// Определяет скорость работы программы

/// от 1000 до 0.1 Тактов в секунду

/// </summary>

public class OsAccumulator

{

public void Process()

public void Reset()

}

public class OsAccumulatorOptions(

IOsTaskManager taskManager,

double ticsPerSecond

);

public class OsCommand {

public OsCommandType Type => Options.CommandType;

public OsCommandOptions Options { get; }

public int TotalTics { get; private set; }

public bool IsComplited { get; private set; }

/// <param name="tics">Доступное количество тиков</param>

/// <returns>Оставшееся количество тиков</returns>

public int Process(int tics)

}

public class OsCommandOptions{

public OsCommandType CommandType { get; } = commandType;

public int RequiredMemory { get; } = requiredMemory;

public int RequiredTics { get; } = requiredTics;

}

public class OsCommandsFactory

{

public OsCommandOptions IOOptions { get; set; }

public OsCommandOptions ExceutableOptions { get; set; }

public OsCommand Create(OsCommandType type);

}

public interface IOsLoadBalancer

{

/// <summary>

/// Список всех задач

/// </summary>

IEnumerable<OsTask> Tasks { get; }

IEnumerable<OsTask> InterruptedTasks { get; }

void AddTask(OsTask task);

/// <summary>

/// Метод выбора исполняемой задачи, для блольшинства вариантов необходимо и достаточно прегрузить его

/// </summary>

/// <returns> Вернет текущую исполняемую задачу, ДЛЯ ВЫБРАННОГО ПРОЦЕССОРА, если та есть.</returns>

OsTask? SelectTaskOrDefault(int cpuIndex);

/// <summary>

/// Используется для синхронизации прерывания

/// (Кода таска выходит из прерваных в не прерваные, в некоторых сценариях,

/// необходимо синхронизировать количество тактов на процессорах прерывания и не прерывания)

/// Как вариант можно поставить 1 и забить

/// Либо искать среди прерваных задач,

/// ту которая имеет наименьшее количетсво тиков до завершения

/// </summary>

/// <returns></returns>

int GetMaxTicsInRound(int totalTics);

void RemoveTask(OsTask task);

}

public class OsTask

{

public int Id { get; }

public IReadOnlyCollection<OsCommand> Commands { get; }

public OsTaskState State { get; set; }

public int Priority { get; }

public int Memory { get; }

public int RequiredTics { get; }

public int ExecutionCommandIndex => \_executionIndex;

/// <returns>true if state changed</returns>

public bool OnTaskSelected();

/// <param name="tics"></param>

/// <returns>Оставшееся количество тиков, после выцполнения</returns>

public OsTaskProcessResult Process(int tics);

}

public class OsTaskFactory

{

public OsTaskFactoryOptions Options { get; set; }

public OsTask CreateTask(int totalCommandsCount);

}

public record OsTaskFactoryOptions

{

public int MaxPriority { get; }

public OsCommandsFactory CommandsFactory { get; }

}

public enum OsTaskState

{

InProcess = 0b00001, // В процессе выполнения

Interrupted = 0b00010, // Прервана, выполняется на IO

Waiting = 0b00100, // Ожидает выполнения

Paused = 0b01000, // Приостановлена

Completed = 0b10000, // Выполена

}

1. **Интерфейс**

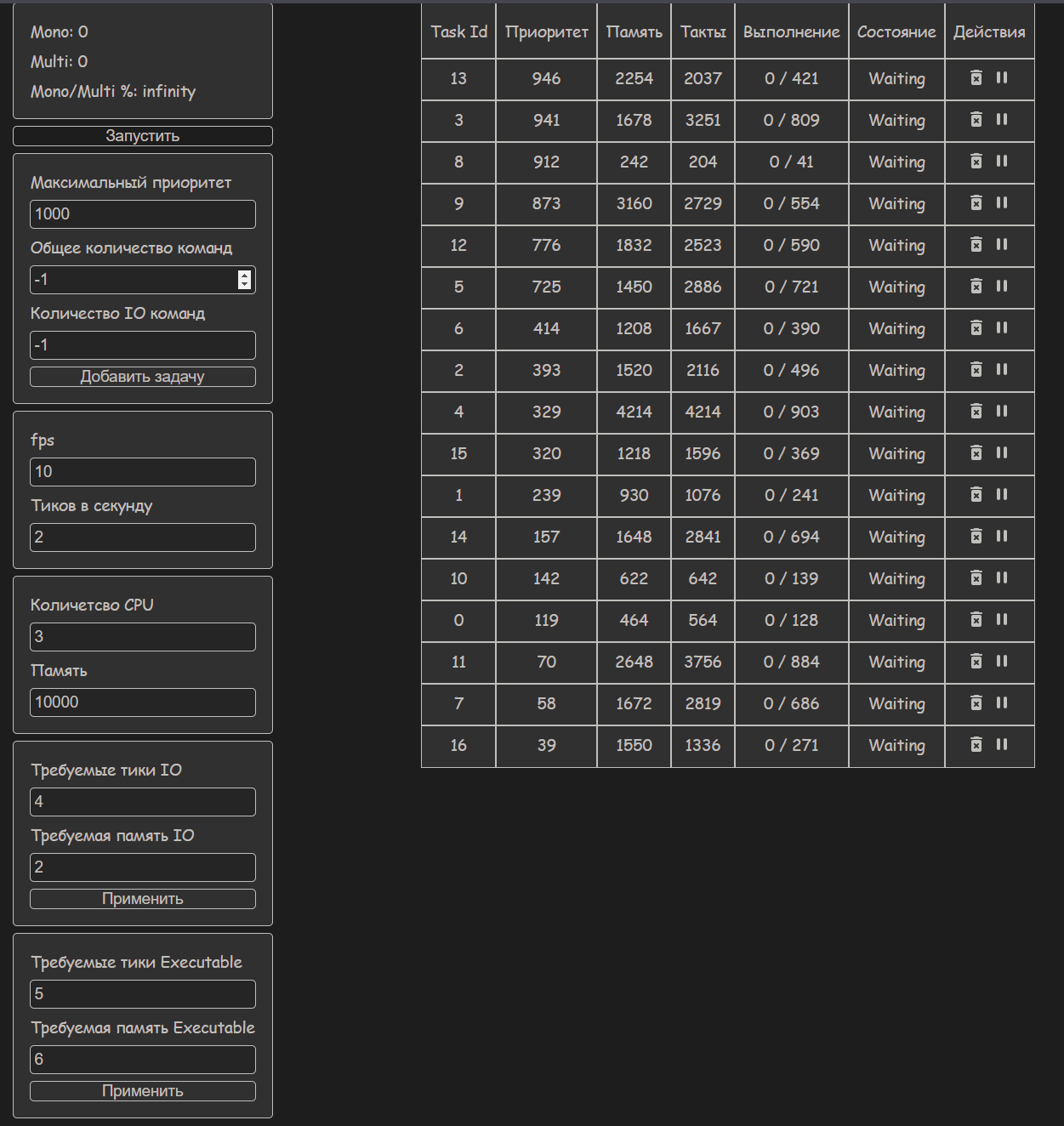


Рисунок 1 – Интерфейс программы

1. Производительность системы

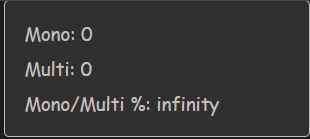


Рисунок 2 – Производительность

1. **Параметры задач**

Если указан параметр -1, то выберется рандомное значение

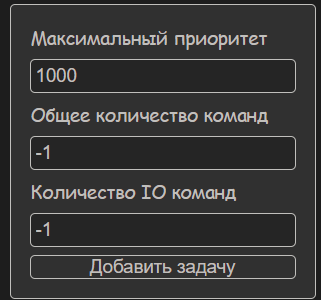


Рисунок 3 – Задание параметров задачи

1. **Системные параметры**

Fps - частота обновления UI

Тики в секунду - частота процессора

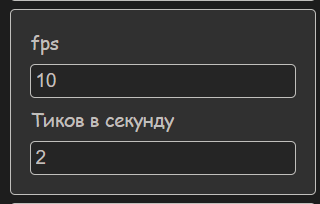


Рисунок 4 – Системные параметры

1. **Параметры менеджера задач**

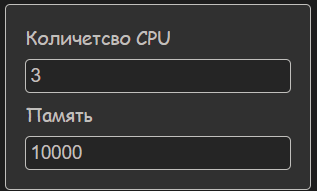


Рисунок 5 – Параметры менеджера задач

1. **Параметры команд**

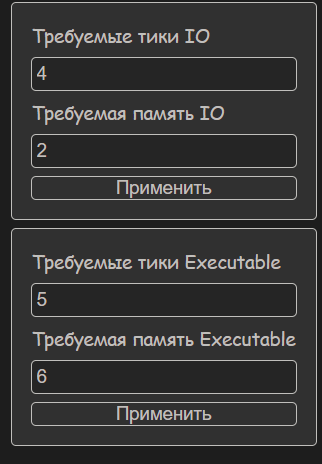


Рисунок 6 – Задание параметров команд

1. **Данные системы**

Выводятся на экран пользователя каждый раз, когда данные меняются (Рисунок 7). К данным системы относятся:

* Коэффициент производительности – коэффициент производительности многозадачной системы относительно однозадачной. Вычисляется формулой:

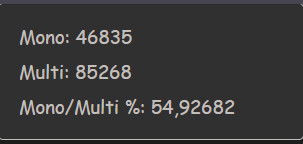


Рисунок 7 – Отображение данных системы

1. **Таблица задач**

Состоит из задач, созданной системой, которые выполняются с течением времени. В данную таблицу поступают данные из массива объектов, где каждый объект представляет собой строку в таблице со всеми значениями для колонок (Рисунок 8).

Таблица состоит из колонок:

* Task Id – это идентификационный номер задачи;
* Память – это размер памяти, занимаемой задачей в системе;
* Такты – это количество тактов, которое нужно потратить на задачу для её выполнения;
* Осталось команд – это текущее количество команд в данной задаче;
* Выполнение – это отношение завершенных команд ко всем командам в задаче;
* Состояние – это текущее состояние задачи в системе (Приостановлен, Блокирован по ВВ, Вычислит. операция, Ожидание)
* Действия, которые пользователь может совершить над задачей:
  + Приостановить / возобновить выполнение задачи;
  + Удалить задачу.



Рисунок 8 – Таблица задач

1. **Описание программной части**

// Исходники логики системы

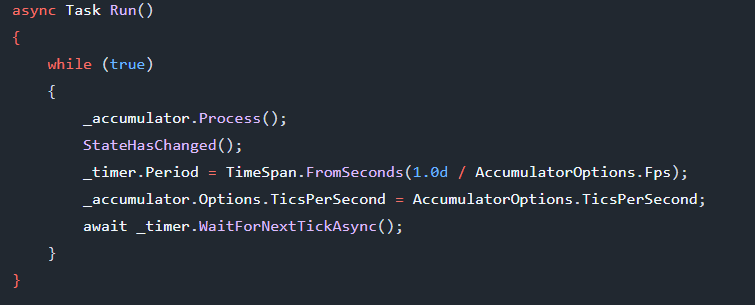
<https://github.com/Great-Ping/ShitOS/tree/main/Source/ShitOS.Core>

// Внешний вид системы

<https://github.com/Great-Ping/ShitOS/tree/main/Source/ShitOS>

1. Кнопка запуска системы начинает бесконечный цикл

Из вызова метода Accumulator.Proccess



2. Для установки параметров используется паттерн Options, все методы для установки состояний тем или иным образом меняют соответствующие поля Options



3. Для добавления задач был создан класс OsTasksFactory. Кнопка добавить задачу вызывает метод

OsTaskFactory.CreateTask, после чего созданную задачу добавляет в OsTaskManager.AddTask  


4. Работа системы начинается с вызова метода

Accumulator.Proccess

Задача Accumulator подсчитать количество тиков которые должна выполнить система.

Расчёт ведется на основе времени с момента последнего вызова Accumultaor.Proccess

В свою очередь Accumulator.Proccess вызывает QueueOsTaskManager.Proccess(int ticks)

Задача IOsTaskManager распределить тики процессоров между задачами в очереди.

Сначала выплняются задачи на Обычных процессорах, а после на процессорах прерывания.

Для равномерного распределения тиков используется IOsLoadBalancer, он распределяет задачи по процессорам и определят максимальное количество тиков за один “раунд”. В лучшем случае раундом должно считаться минимальное время для завершения первой команды у любой из выполняемых задач. Это сделано с целью обеспечения системе правдоподобности, ведь при выходе задачи из прерывания система должна перерассчитываться.

Эффективность системы подсчитывает IOsTaskManager, предполагается что в рантайме при бесконечном количестве задач mono система будет иметь ту же производительность что и все вычислительные процессоры у multi системы и прирост производительности у multi системы будет за счет процессоров ввода вывода.