Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Ульяновский государственный технический университет»

Дисциплина «Операционные системы».

**Лабораторная работа №2. Алгоритмы планирования**

**Выполнил**

Студент группы ИВТИИбд-21

Пешков Максим Алексеевич

**Проверил(а):**

Преподаватель Беляев К. С.

Ульяновск 2025

**Тема:** Алгоритмы планирования процессов  
**Цель работы:** Изучить и реализовать различные алгоритмы планирования процессов в операционных системах, сравнить их поведение и эффективность на основе различных метрик: оборотного времени, времени ожидания и справедливости распределения ресурсов.

**Используемый язык:** C (GCC, Unix/Linux среда)

## **Задание:**

## Реализовать следующие алгоритмы планирования процессов на языке **Си** с использованием предоставленного фреймворка:

1. **FCFS (First-Come, First-Served)** — первым пришёл, первым обслужен.
2. **SJF (Shortest Job First)** — сначала задача с наименьшим временем выполнения.
3. **Priority Scheduling** — по приоритету (более высокое числовое значение = более высокий приоритет).
4. **Round Robin (RR)** — циклическое планирование с квантом времени 10 мс.
5. **Priority Round Robin** — приоритетное планирование с циклическим обслуживанием задач с одинаковым приоритетом.

### Структура проекта

driver.c — основной файл, считывающий задачи и вызывающий планировщик.

cpu.c — содержит функцию run(Task\* task, int slice), имитирующую выполнение задачи.

task.h — заголовочный файл с описанием структуры Task.

schedule\_\*.c — реализация каждого алгоритма (fcfs, sjf, priority, rr, priority\_rr).

Makefile — позволяет собирать проект с нужным планировщиком.

**FCFS:**

List \*task\_list = NULL;

void add(char \*name, int priority, int burst) {

Task \*task = malloc(sizeof(Task));

task->name = name;

task->priority = priority;

task->burst = burst;

insert(&task\_list, task);

}

void schedule() {

Task \*current = task\_list;

while (current != NULL) {

run(current, current->burst);

current = current->next;

}

}

**SJF:**

List \*task\_list = NULL;

void add(char \*name, int priority, int burst) {

Task \*task = malloc(sizeof(Task));

task->name = name;

task->priority = priority;

task->burst = burst;

insert(&task\_list, task);

}

Task\* pickNextTask() {

Task \*shortest = task\_list;

Task \*current = task\_list;

while (current != NULL) {

if (current->burst < shortest->burst)

shortest = current;

current = current->next;

}

return shortest;

}

void schedule() {

while (task\_list != NULL) {

Task \*task = pickNextTask();

run(task, task->burst);

delete(&task\_list, task);

}

}

**Priority Scheduling:**

List \*task\_list = NULL;

void add(char \*name, int priority, int burst) {

Task \*task = malloc(sizeof(Task));

task->name = name;

task->priority = priority;

task->burst = burst;

insert(&task\_list, task);

}

Task\* pickNextTask() {

Task \*highest = task\_list;

Task \*current = task\_list;

while (current != NULL) {

if (current->priority > highest->priority)

highest = current;

current = current->next;

}

return highest;

}

void schedule() {

while (task\_list != NULL) {

Task \*task = pickNextTask();

run(task, task->burst);

delete(&task\_list, task);

}

}

**RR**:

#define QUANTUM 10

List \*task\_list = NULL;

void add(char \*name, int priority, int burst) {

Task \*task = malloc(sizeof(Task));

task->name = name;

task->priority = priority;

task->burst = burst;

insert(&task\_list, task);

}

void schedule() {

while (task\_list != NULL) {

Task \*current = task\_list;

while (current != NULL) {

int slice = (current->burst < QUANTUM) ? current->burst : QUANTUM;

run(current, slice);

current->burst -= slice;

Task \*next = current->next;

if (current->burst <= 0)

delete(&task\_list, current);

current = next;

}

}

}

**Priority Round Robin:**

#define QUANTUM 10

List \*task\_list = NULL;

void add(char \*name, int priority, int burst) {

Task \*task = malloc(sizeof(Task));

task->name = name;

task->priority = priority;

task->burst = burst;

insert(&task\_list, task);

}

int getHighestPriority() {

int max = -1;

Task \*current = task\_list;

while (current != NULL) {

if (current->priority > max)

max = current->priority;

current = current->next;

}

return max;

}

void schedule() {

while (task\_list != NULL) {

int max\_priority = getHighestPriority();

Task \*current = task\_list;

while (current != NULL) {

Task \*next = current->next;

if (current->priority == max\_priority) {

int slice = (current->burst < QUANTUM) ? current->burst : QUANTUM;

run(current, slice);

current->burst -= slice;

if (current->burst <= 0)

delete(&task\_list, current);

}

current = next;

}

}

}

### **Вывод**

В рамках данной лабораторной работы были успешно реализованы и протестированы пять алгоритмов планирования процессов. На основе анализа:

* **SJF** показал наименьшее среднее оборотное время.
* **RR** оказался наиболее справедливым при работе с задачами разной длины.
* **FCFS** может быть неэффективен при большом разбросе длительности задач.
* **Priority** и **Priority RR** учитывают приоритет, но нуждаются в дополнительной логике при равенстве приоритетов.

Эта работа позволила на практике понять, как ОС управляет процессами, и как различные стратегии влияют на производительность и поведение системы.