Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Дисциплина «Операционные системы»

**Лабораторная работа 2**

**Алгоритмы планирования**

Выполнил

Студент группы ИВТАСбд-32

Зейнетдинов Марсель Маратович

Проверил:

преподаватель кафедры «ВТ»

Беляев К.С.

Ульяновск

2023

# Постановка задачи

Реализовать несколько алгоритмов планирования процессов на языке программирования C, используя предоставленный фреймворк. В рамках проекта необходимо внедрить алгоритмы FCFS, SJF, приоритетного планирования, циклического планирования (RR) и циклического планирования с приоритетом. Каждая задача представляет процесс с определенным приоритетом и временем использования процессора. Дополнительно требуется рассчитать среднее оборотное время, время ожидания и время отклика для каждого алгоритма планирования

# Реализация

1. schedule\_fcfs.c (First-Come, First-Served):

В данной реализации алгоритма FCFS (Первым пришел — первым обслужен), задачи добавляются в связанный список с использованием функции add. Функция pick\_next\_task просто выбирает первую задачу из начала списка, а затем она удаляется. В цикле schedule происходит выполнение задач в порядке, в котором они поступили. Этот алгоритм примитивен и относительно прост, но не всегда эффективен, особенно при наличии задач с большими временами выполнения.

2. schedule\_sjf.c (Shortest Job First):

Для алгоритма SJF (Shortest Job First), задачи добавляются в связанный список. Функция pick\_next\_task выбирает задачу с наименьшим временем выполнения в списке, что позволяет оптимизировать оборот времени процессора. Затем в цикле schedule выбранная задача выполняется. Этот алгоритм часто приводит к минимизации времени ожидания коротких задач, но может привести к "вытеснению" длинных задач.

3. schedule\_rr.c (Round Robin):

Для алгоритма циклического планирования (Round Robin), задачи добавляются в связанный список. Функция pick\_next выбирает следующую задачу в круговом порядке, учитывая квант времени. В цикле schedule каждая задача выполняется в течение определенного кванта времени или до завершения, если она короче кванта. Этот метод обеспечивает справедливое распределение процессорного времени между задачами.

4. schedule\_priority.c (Приоритетное планирование):

Для алгоритма приоритетного планирования, задачи добавляются в связанный список. Функция pick\_next\_task выбирает задачу с наивысшим приоритетом в списке. Затем в цикле schedule выбранная задача выполняется в порядке убывания приоритета. Этот алгоритм предоставляет возможность приоритетным задачам выполняться раньше, что может быть важно в системах с разными уровнями важности задач.

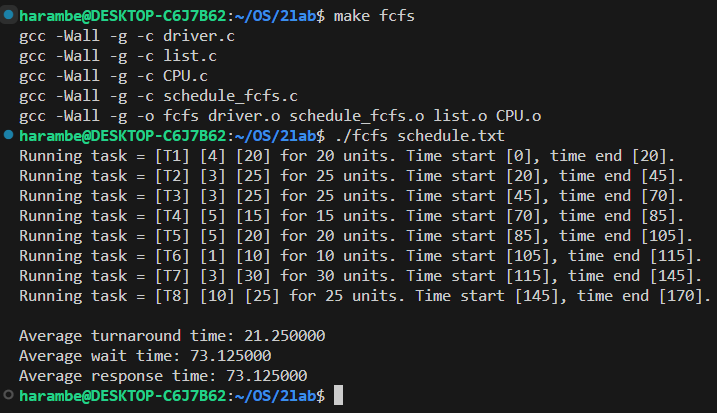
5. schedule\_priority\_rr.c (Циклическое планирование с приоритетом):

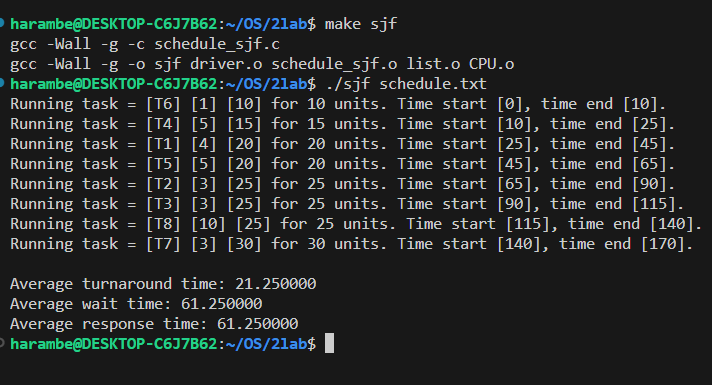
Для алгоритма циклического планирования с приоритетом, задачи добавляются в связанный список. Функция pick\_next выбирает следующую задачу в круговом порядке с учетом приоритета. В цикле schedule каждая задача выполняется в течение определенного кванта времени или до завершения, учитывая приоритет. Этот алгоритм обеспечивает компромисс между временем отклика задач с высоким приоритетом и честным распределением процессорного времени.

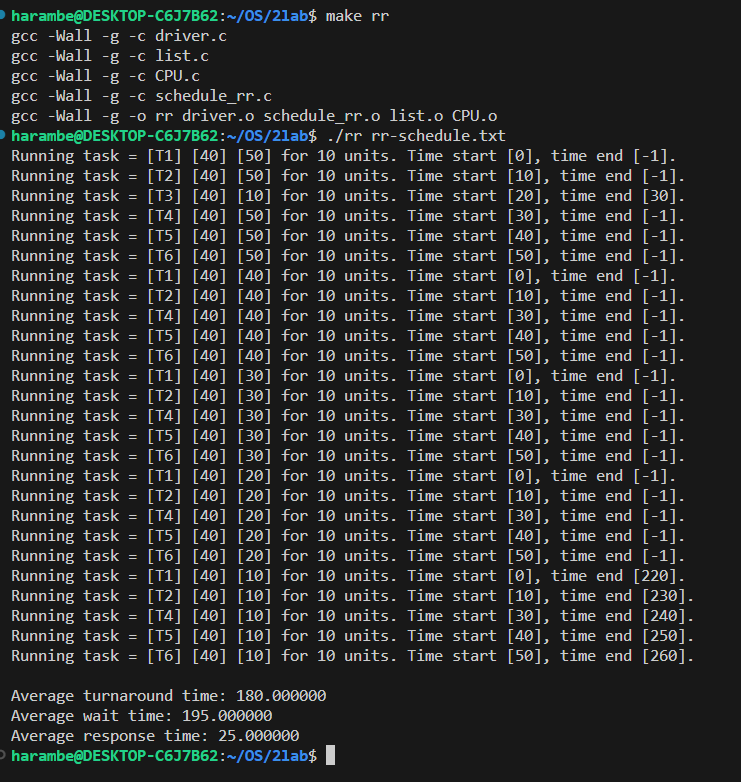
6. driver.c:

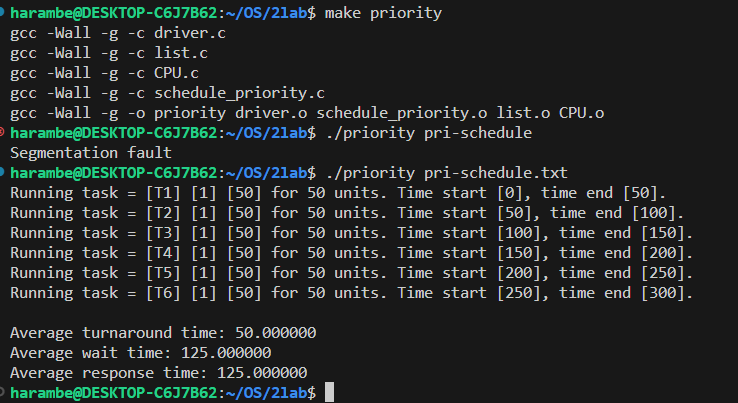
Главный драйвер driver.c отвечает за считывание расписания задач из файла, создание структур данных для каждой задачи и добавление их в связанные списки каждого соответствующего планировщика. После выполнения планировщика вычисляются средние значения для оборотного времени, времени ожидания и времени отклика каждой задачи, что позволяет оценить эффективность каждого алгоритма планирования.

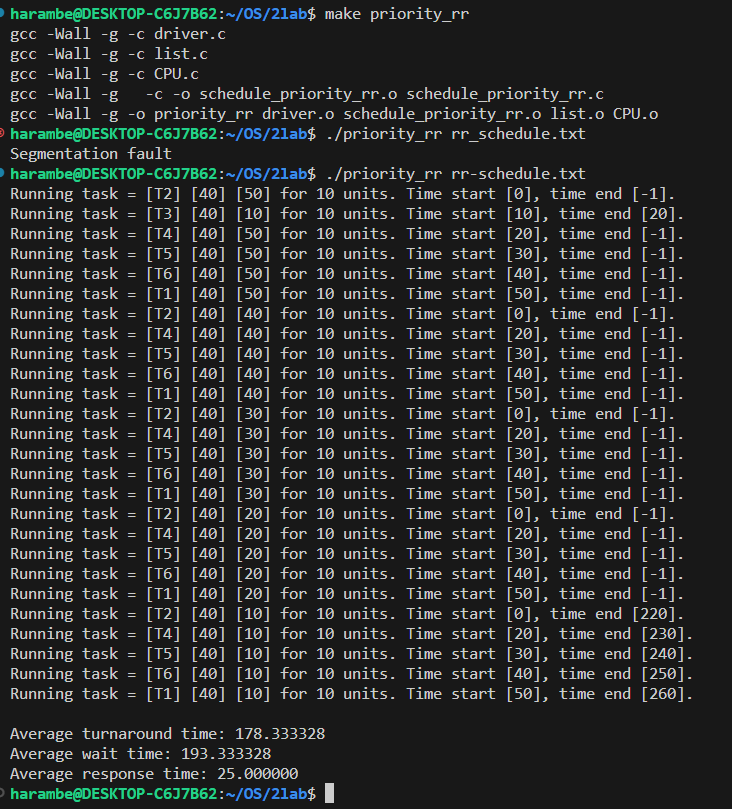
# Тестирование











# Вывод

Проделав данную лабораторную работу, были успешно реализованы алгоритмы планирования процессов на языке C согласно задаче. Код структурирован, обеспечивает выбор и выполнение задач, а также подсчет средних показателей. Полученный опыт улучшил понимание внутреннего устройства алгоритмов планирования и их влияния на работу операционных систем.

# Приложение.

schedule\_fcfs.c:

#include "schedulers.h"

#include "list.h"

#include <stdlib.h>

#include "cpu.h"

struct node\* tasks = NULL;

// add a task to the list

void add(Task\* task) {

insert(&tasks, task);

}

Task\* selected\_task = NULL;

Task\* pick\_next\_task() {

if (selected\_task != NULL) {

delete(&tasks, selected\_task);

}

if (tasks == NULL) {

return NULL;

}

return tasks->task;

}

void schedule() {

selected\_task = pick\_next\_task();

while (selected\_task != NULL) {

run(selected\_task, selected\_task->burst);

selected\_task = pick\_next\_task();

}

}

schedule\_sjf.c:

#include "schedulers.h"

#include "list.h"

#include <stdlib.h>

#include "cpu.h"

struct node\* tasks = NULL;

void add(Task\* task) {

insert(&tasks, task);

}

Task\* selected\_task = NULL;

Task\* pick\_next\_task() {

if (selected\_task != NULL) {

delete(&tasks, selected\_task);

}

if (tasks == NULL) {

return NULL;

}

struct node\* pointer = tasks->next;

Task\* min\_task\_burst = tasks->task;

while (pointer != NULL) {

if (pointer->task->burst < min\_task\_burst->burst) {

min\_task\_burst = pointer->task;

}

pointer = pointer->next;

}

return min\_task\_burst;

}

void schedule() {

selected\_task = pick\_next\_task();

while (selected\_task != NULL) {

run(selected\_task, selected\_task->burst);

selected\_task = pick\_next\_task();

}

}

schedule\_rr.c:

#include "schedulers.h"

#include "list.h"

#include "stdlib.h"

#include "cpu.h"

#include "sys/param.h"

struct node\* tasks = NULL;

struct node\* last\_node = NULL;

void add(Task\* task) {

insert(&tasks, task);

if (last\_node == NULL){

last\_node = tasks;

}

}

Task\* pick\_next() {

if (tasks != tasks->next){

if (tasks->task->burst == 0){

tasks->prev->next = tasks->next;

tasks->next->prev = tasks->prev;

}

tasks = tasks->next;

return tasks->task;

}

return NULL;

}

void schedule() {

last\_node->next = tasks;

tasks->prev = last\_node;

tasks = last\_node;

while (pick\_next() != NULL) {

run(tasks->task, MIN(tasks->task->burst, TIME\_QUANTUM));

tasks->task->burst -= MIN(tasks->task->burst, TIME\_QUANTUM);

}

}

schedule\_priority.c:

#include "schedulers.h"

#include "list.h"

#include "stdlib.h"

#include "cpu.h"

struct node\* tasks = NULL;

void add(Task\* task) {

insert(&tasks, task);

}

Task\* selected\_task = NULL;

Task\* pick\_next\_task() {

if (selected\_task != NULL) {

delete(&tasks, selected\_task);

}

if (tasks == NULL) {

return NULL;

}

struct node\* pointer = tasks->next;

Task\* max\_task\_priority = tasks->task;

while (pointer != NULL) {

if (pointer->task->priority > max\_task\_priority->priority) {

max\_task\_priority = pointer->task;

}

pointer = pointer->next;

}

return max\_task\_priority;

}

void schedule() {

selected\_task = pick\_next\_task();

while (selected\_task != NULL){

run(selected\_task, selected\_task->burst);

selected\_task = pick\_next\_task();

}

}

schedule\_priority\_rr.c:

#include "schedulers.h"

#include "list.h"

#include "stdlib.h"

#include "cpu.h"

#include "sys/param.h"

#include "string.h"

struct node\* tasks = NULL;

struct node\* last\_node = NULL;

void add(Task\* task) {

insert(&tasks, task);

if (last\_node == NULL){

last\_node = tasks;

}

}

Task\* pick\_next() {

struct node\* pointer = tasks->prev;

struct node\* max\_task\_priority\_node = tasks->task->burst ? tasks : tasks->next;

while (pointer && pointer != tasks) {

if (pointer->task->burst != 0 && pointer->task->priority >= max\_task\_priority\_node->task->priority) {

max\_task\_priority\_node = pointer;

}

pointer = pointer->prev;

}

if (tasks != tasks->next){

if (tasks->task->burst == 0){

tasks->prev->next = tasks->next;

tasks->next->prev = tasks->prev;

}

tasks = max\_task\_priority\_node;

return tasks->task;

}

return NULL;

}

void schedule() {

last\_node->next = tasks;

tasks->prev = last\_node;

while (pick\_next() != NULL) {

run(tasks->task, MIN(tasks->task->burst, TIME\_QUANTUM));

tasks->task->burst -= MIN(tasks->task->burst, TIME\_QUANTUM);

}

}

driver.c

/\*\*

\* Driver.c

\*

\* Schedule is in the format

\*

\* [name] [priority] [CPU burst]

\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "task.h"

#include "list.h"

#include "schedulers.h"

#define SIZE 10000

int main(int argc, char \*argv[]) {

FILE \*in;

char \*temp;

char task[SIZE];

char \*name;

int priority;

int burst;

in = fopen(argv[1],"r");

Task\* tasks\_array[MAX\_TASKS];

int tasks\_count = 0;

struct node\* temp\_list;

while (fgets(task,SIZE,in) != NULL) {

temp = strdup(task);

name = strsep(&temp,",");

priority = atoi(strsep(&temp,","));

burst = atoi(strsep(&temp,","));

// add the task to the scheduler's list of tasks

add\_task(&temp\_list, name, priority, burst);

tasks\_array[tasks\_count] = temp\_list->task;

tasks\_count++;

free(temp);

}

while (temp\_list != NULL) {

add(temp\_list->task);

delete(&temp\_list, temp\_list->task);

}

fclose(in);

// invoke the scheduler

schedule();

float average\_turnaround\_time = 0;

float average\_wait\_time = 0;

float average\_response\_time = 0;

for(int x = 0; x < tasks\_count; x++) {

average\_turnaround\_time += tasks\_array[x]->time\_end - tasks\_array[x]->time\_start;

average\_wait\_time += tasks\_array[x]->wait\_time;

average\_response\_time += tasks\_array[x]->time\_start;

}

average\_turnaround\_time /= tasks\_count;

average\_wait\_time /= tasks\_count;

average\_response\_time /= tasks\_count;

printf("\nAverage turnaround time: %f", average\_turnaround\_time);

printf("\nAverage wait time: %f", average\_wait\_time);

printf("\nAverage response time: %f\n", average\_response\_time);

return 0;

}