Федеральное государственное автономное

образовательное учреждение

Высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| институт |
| Информатика |
| кафедра |

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №2**

|  |
| --- |
| Синхронизация потоков в OC GNU/Linux |
| тема |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  | А. Н. Пупков |
|  | |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ21-16/1б, 032155832 |  |  |  | Н. А. Терентьев |
|  | номер группы, зачётной книжки |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2023

**СОДЕРЖАНИЕ**

1 Цели 3

2 Задачи 3

3 Описание варианта задания 3

4 Ход выполнения 4

4.1 Программа, запускающая несколько потоков 4

4.2 Doxygen документация 6

5 Выводы 7

Список использованных источников 8

Приложение А 9

**1 Цель**

Изучение программных средств синхронизации потоков в OC.

**2 Задачи**

Для выполнения практической работы необходимо выполнить  
следующие задачи:

* ознакомиться с краткими теоретическими сведениями по синхронизации потоков в ОС GNU/Linux;
* получить у преподавателя индивидуальный вариант задания, в которой для выполнения различных частей создаются и запускаются потоки выполнения, а для синхронизации доступа к требуемым ресурсам используются соответствующие объекты ОС;
* используя изученные механизмы, разработать и отладить программу, выполняющую заданную работу;
* составить общее описание результатов, инструкции по сборке и использованию программ, а также инструкцию по получению документации, сформировать архив формата tar.gz и представить на проверку с исходными текстами программы.

**3 Описание варианта задания**

Работа выполнена в соответствии с вариантом 7.

На железнодорожной станции N путей (разделяемые ресурсы), на каждом из которых может остановиться только один поезд (поток). Когда поезд прибывает на станцию, то он занимает тот путь, который на данный момент свободен, останавливается там на определённое время, а затем отправляется дальше. Если все пути заняты, то поезд останавливается на подъезде к станции и ожидает освобождения одного из путей. Если в ожидании освобождения путей находятся несколько поездов, то они занимают освобождающиеся пути в порядке приезда на станцию. Поезда прибывают на станцию через произвольные промежутки времени. Значение N задается пользователем при старте приложения.

**4 Ход выполнения**

**4.1 Программа, запускающая несколько потоков**

В рамках практической работы необходимо разработать программу, для различных частей которой создаются и запускаются потоки выполнения, а для синхронизации доступа к требуемым ресурсам используются соответствующие объекты OC. Код функции, которую выполняет каждый поток, представлен в листинге 1.

Листинг 1 – Функция train

void\* train(void\* arg) {

int trainId = \*(int\*)arg + 1;

printf("\033[1;32mПоезд №%d прибывает на станцию.\033[0m\n",

trainId);

// Используем мьютекс для обеспечения взаимного исключения.

pthread\_mutex\_lock(&mutex);

waitingTrains++;

pthread\_mutex\_unlock(&mutex);

// Поезд ждет на семафоре station, который позволяет ему войти на

// станцию, если есть свободный путь

sem\_wait(&station);

// Оказавшись на станции, поезд уменьшает счетчик waiting\_trains и

// выводит сообщение о том, что он вошел на станцию.

pthread\_mutex\_lock(&mutex);

waitingTrains--;

printf("\033[1;34mПоезд №%d занимает путь.\033[0m\n", trainId);

pthread\_mutex\_unlock(&mutex);

ts.tv\_sec = rand() % 5 + 3;

ts.tv\_nsec = 0;

// Поезд "спит" случайное время для имитации загрузки и выгрузки

// пассажиров перед тем как покинуть станцию.

nanosleep(&ts, NULL);

printf("\033[1;31mПоезд №%d покидает станцию.\033[0m\n", trainId);

// Опубликовываем в семафоре station, указывая, что этот путь доступен.

sem\_post(&station);

return NULL;

}

Код, который запускает сами потоки, представлен в листинге 2.

Листинг 2 – Запуск потоков

for (int i = 0; i < MAX\_TRAINS; i++) {

// Присваиваем поезду уникальный идентификатор.

trainIds[i] = i;

// Создаём новый поток для поезда.

// Передаваемые аргументы: адрес элемента в массиве trains(чтобы

// сохранить идентификатор потока), NULL (для атрибутов по умолчанию),

// указатель на функцию train и адрес элемента в массиве trainIds

// (чтобы передать идентификатор поезда в функцию train).

pthread\_create(&trains[i], NULL, train,

&trainIds[i]);

ts.tv\_sec = rand() % 3;

ts.tv\_nsec = 0;

// Приостанавливаем выполнение программы на случайное время.

nanosleep(&ts, NULL);

}

// Основной поток будет ждать завершения всех потоков поездов перед тем

// как продолжить выполнение. Это важно чтобы убедиться что все ресурсы

// (например семафор station) освобождаются корректно перед завершением

// программы.

for (int i = 0; i < MAX\_TRAINS; i++) {

pthread\_join(trains[i], NULL);

}

Работа с программой представлена на рисунке 1.

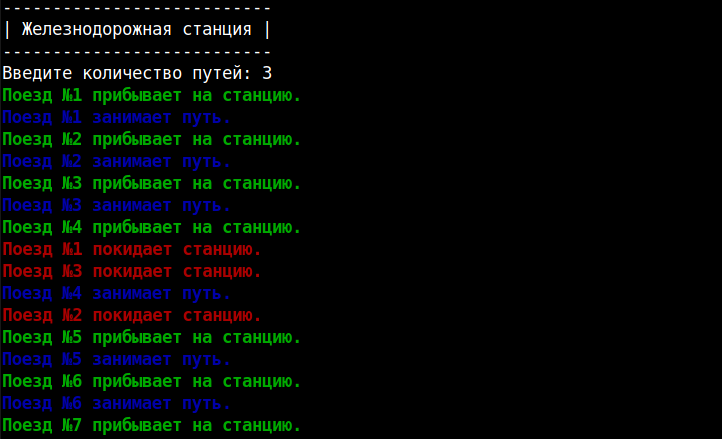


Рисунок 1 – Работа с программой

**4.2 Doxygen документация**

Все файлы содержат комментарии в стиле системы doxygen. Благодаря конфигурационному файлу была сгенерирована документация. Её внешний вид представлен на рисунке 2.

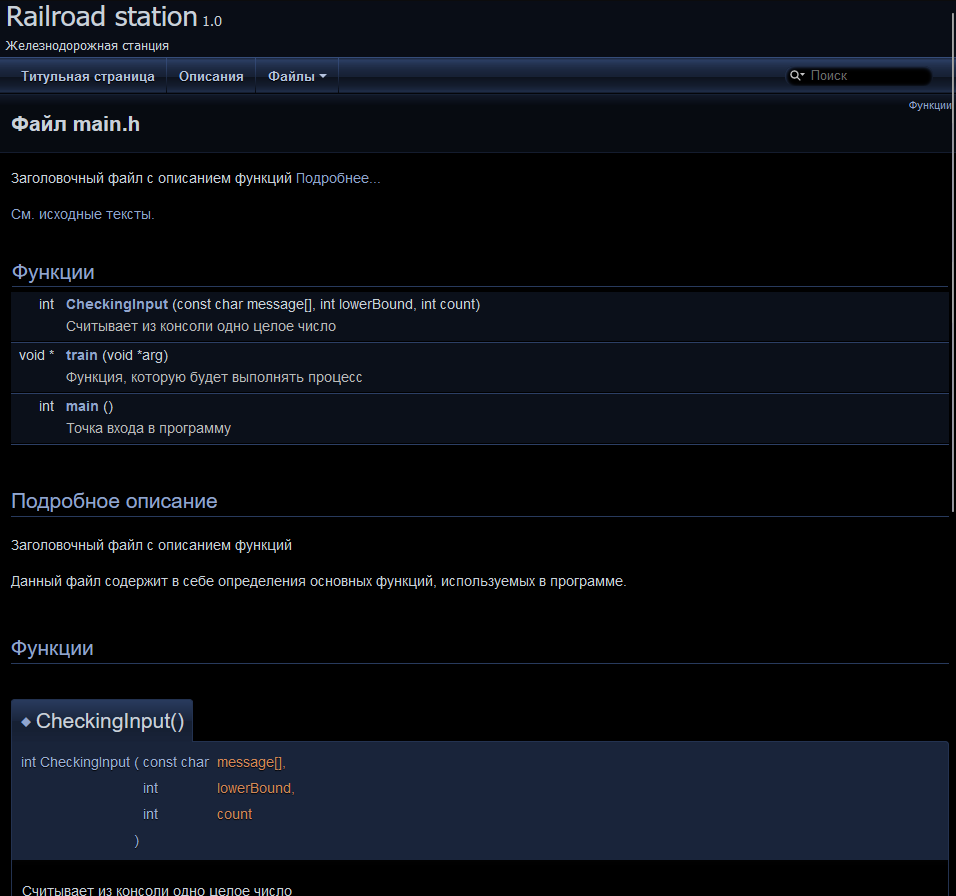


Рисунок 2 – Сгенерированная документация

**5** **Выводы**

В ходе работы были изучены программные средства синхронизации потоков в ОС GNU/Linux. Была разработана программа в соответствии с вариантом.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 СТО 4.2-07-2014 Система менеджмента качества. Общие требования  
к построению, изложении и оформлению документов учебной деятельности.  
Дата введения – 30.12.2-13.

**ПРИЛОЖЕНИЕ А**

**Исходный код файла main.c**

/\*! Код основного процесса программы \*/

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define CORRECT\_INPUT 2

#define END\_STRING '\n'

#define MAX\_TRAINS 10

struct timespec ts;

sem\_t station;

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

int waitingTrains = 0;

int CheckingInput(const char message[], int lowerBound, int count)

{

int userInput = 1;

char inputChar = '\0';

printf("%s", message);

int input = scanf("%d%c", &userInput, &inputChar);

while (input != CORRECT\_INPUT || inputChar != END\_STRING

|| userInput < lowerBound || userInput > count)

{

if (userInput >= lowerBound && userInput <= count)

{

while ((inputChar = getchar()) != '\n');

}

userInput = 1;

printf("Неверный ввод. Попробуйте снова.\n%s", message);

input = scanf("%d%c", &userInput, &inputChar);

}

return userInput;

}

void\* train(void\* arg) {

int trainId = \*(int\*)arg + 1;

printf("\033[1;32mПоезд №%d прибывает на станцию.\033[0m\n",

trainId);

// Используем мьютекс для обеспечения взаимного исключения.

pthread\_mutex\_lock(&mutex);

waitingTrains++;

pthread\_mutex\_unlock(&mutex);

// Поезд ждет на семафоре station, который позволяет ему войти на

// станцию, если есть свободный путь

sem\_wait(&station);

// Оказавшись на станции, поезд уменьшает счетчик waiting\_trains и

// выводит сообщение о том, что он вошел на станцию.

pthread\_mutex\_lock(&mutex);

waitingTrains--;

printf("\033[1;34mПоезд №%d занимает путь.\033[0m\n", trainId);

pthread\_mutex\_unlock(&mutex);

ts.tv\_sec = rand() % 5 + 3;

ts.tv\_nsec = 0;

// Поезд "спит" случайное время для имитации загрузки и выгрузки

// пассажиров перед тем как покинуть станцию.

nanosleep(&ts, NULL);

printf("\033[1;31mПоезд №%d покидает станцию.\033[0m\n", trainId);

// Опубликовываем в семафоре station, указывая, что этот путь доступен.

sem\_post(&station);

return NULL;

}

int main() {

int numberTracks, lowerBound = 1;

const char message[] = "Введите количество путей: ";

printf("---------------------------\n");

printf("| Железнодорожная станция |\n");

printf("---------------------------\n");

numberTracks = CheckingInput(message, lowerBound, MAX\_TRAINS);

sem\_init(&station, 0, numberTracks);

// Идентификаторы потоков поездов, которые будут созданы.

pthread\_t trains[MAX\_TRAINS];

// Целочисленные значения, представляющие идентификаторы каждого поезда.

int trainIds[MAX\_TRAINS];

while (!0)

{

for (int i = 0; i < MAX\_TRAINS; i++) {

// Присваиваем поезду уникальный идентификатор.

trainIds[i] = i;

// Создаём новый поток для поезда.

// Передаваемые аргументы: адрес элемента в массиве trains(чтобы

// сохранить идентификатор потока), NULL (для атрибутов по умолчанию),

// указатель на функцию train и адрес элемента в массиве trainIds

// (чтобы передать идентификатор поезда в функцию train).

pthread\_create(&trains[i], NULL, train,

&trainIds[i]);

ts.tv\_sec = rand() % 3;

ts.tv\_nsec = 0;

// Приостанавливаем выполнение программы на случайное время.

nanosleep(&ts, NULL);

}

// Основной поток будет ждать завершения всех потоков поездов перед тем

// как продолжить выполнение. Это важно чтобы убедиться что все ресурсы

// (например семафор station) освобождаются корректно перед завершением

// программы.

for (int i = 0; i < MAX\_TRAINS; i++) {

pthread\_join(trains[i], NULL);

}

// Уничтожаем семафор station.

sem\_destroy(&station);

printf("Повторить выполнение программы? (введите y "

"для этого): ");

char choice;

scanf(" %c", &choice);

if (choice != 'y' && choice != 'Y')

{

break;

}

}

return 0;

}