Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Лабораторная работа №2. Синхронизация потоков в ОС GNU/Linux
Тема

Преподаватель _____ А. С. Кузнецов
подпись, дата — инициалы, фамилия

Студент КИ19-17/1б 031939175 _____ А. Д. Непомнящий
номер группы, зачетной книжки — подпись, дата — инициалы, фамилия

Красноярск 2021

1 Цель работы

Цель состоит в изучении программных средств синхронизации потоков в ОС.

2 Задачи

Выполнение работы сводится к следующим задачам.

1. Ознакомиться с краткими теоретическими сведениями по управлению процессами в ОС GNU/Linux.

2. Используя изученные механизмы, разработать и отладить программу, выполняющую заданную работу.

3. Написать настоящий отчет и представить его к защите с исходными текстами программы. Исходные тексты программ должны содержать комментарии в стиле системы doxygen.

Вариант 11. В аэропорту N посадочных полос (разделяемые ресурсы), на каждую из которых может сесть только один самолет (поток). Когда самолет оказывается на подлете к аэропорту, он выбирает и садится на ту полосу, которая на данный момент свободна. Определенное время полоса занята, пока самолет не доставят в зону прилета. Если на подлете все полосы заняты, то самолет кружит над аэропортом и ожидает освобождения одной из полос. Если в ожидании освобождения полос находятся несколько самолетов, то они занимают освобождающиеся полосы в порядке подлета. Самолеты прибывают в аэропорт через произвольные промежутки времени. Описанный процесс происходит бесконечно. Значение N задается пользователем при старте процесса. Величина времени подлета и его интенсивность являются случайными величинами с равномерным законом распределения.

3 Исходные тексты программ

Листинг 1 – Код в файле main.c

```
/*! \file   main.c
 *  \brief  Main file of the program. Contains main function
 */

#include <malloc.h>
#include "thread.h"
#include "input.h"
#include <stdbool.h>


/*! \brief Main function
 *
 *  \details Main function. Initialise Airfield, aircrafts numeration
 *  variable, and performs a task. Also provides minimal UI.
 *
 *  \return Doesn't return anything. According to the task, works infinitely.
 */
int main()
{
    printf("Enter number of landing strips ( [1, 1000] )\n");
    int n = checkedInputInt(numberOfStripsInputCheck);

    int aircraftNumber = FIRST_AIRCRAFT_NUMBER;

    Airfield* airfieldObject = (Airfield*) malloc(sizeof(Airfield));
    airfieldObject->landingStrips = (int*) malloc(n * sizeof(int));
    airfieldObject->numberOfStrips = n;

    for (int i = 0; i < n; i++)
    {
        airfieldObject->landingStrips[i] = EMPTY_STRIP;
    }

    sem_init(&(airfieldObject->airfieldQueueSem), 0, n);
    pthread_mutex_init(&(airfieldObject->airfieldBusyMutex), NULL);

    while (true)
    {
        Aircraft* aircraftObject = (Aircraft*) malloc(sizeof(Aircraft));
        aircraftObject->airfieldObject = airfieldObject;
        aircraftObject->number = aircraftNumber++;
        aircraftObject->strip = NULL;
        pthread_t threadId;
        pthread_create(&threadId, NULL, &aircraftThreadFunction,
                       aircraftObject);

        randomNanosleep(MIN_SLEEP, SMALL_SLEEP);

    }

}
```

Листинг 2 – Код в файле task11.h

```
/*! \file   task11.h
 *  \brief  Header file of functions essential for task 11
 */

#include <semaphore.h>
#include <stdio.h>
#include <pthread.h>
#include <time.h>
#include <stdlib.h>
#include <stdbool.h>

#ifndef TASK11_H
#define TASK11_H

/*! \enum
 *  \brief  Essential constants for task 11
 */
enum constants
{
    EMPTY_STRIP = -1,
    SMALL_SLEEP = 2,
    BIG_SLEEP = 30,
    MIN_SLEEP = 1,
    N_MIN = 1,
    N_MAX = 1000,
    FIRST_AIRCRAFT_NUMBER = 1
};


/*! \struct Airfield
 *  \brief Airfield struct
 *
 *  \details Keeps information about landing strips, semaphore, and mutex
 */
typedef struct
{
    /*!
     * Number of landing strips
     */
    int numberOfStrips;
    /*!
     * Array of 'codes' of planes at each landing strip
     */
    int* landingStrips;
    /*!
     * Semaphore object
     */
    sem_t airfieldQueueSem;
    /*!
     * Mutex object
     */
    pthread_mutex_t airfieldBusyMutex;
} Airfield;


/*! \struct Aircraft
 *  \brief Aircraft and pthread arg struct
 *
```

Окончание листинга 2

```
 *  \details Keeps information about aircraft and os also used as
 *  pthread_create arg
 */
typedef struct
{
    /*!
     * Aircraft's number
     */
    int number;
    /*!
     * Pointer to landing strip's array of parked aircraft element, where
     * this aircraft is parked
     */
    int* strip;
    /*!
     * Pointer to destination airfield
     */
    Airfield* airfieldObject;
} Aircraft;

/*! \brief Provides a procedure of landing an aircraft, by finding an empty
 * landing strip and changing necessary variables
 *
 *  \param aircraftToLand pointer to aircraft object which is up to landing
 */
void landAircraft(Aircraft* aircraftToLand);

/*! \brief Checks if number can be a number of landing strips for task 11
 *
 *  \param intToCheck number to check.
 *  \return true if number can be a number of landing strips for task 11
 *  false - otherwise
 */
bool numberOfStripsInputCheck(int intToCheck);

/*! \brief Calls nanosleep function with random amount of seconds and 0
 *  nanoseconds
 *
 *  \param minSeconds minimal number of seconds to sleep.
 *  \param maxSeconds maximal number of seconds to sleep.
 */
void randomNanosleep(int minSeconds, int maxSeconds);


#endif //TASK11_H
```

Листинг 3 – Код в файле task11.c

```
/*! \file   task11.c
 *  \brief  Implements functions of task11.h
 */

#include "task11.h"

void landAircraft(Aircraft* aircraftToLand)
{
    for (int i = 0; i < aircraftToLand->airfieldObject->numberOfStrips; i++)
```

Окончание листинга 3

```
    {
        if (aircraftToLand->airfieldObject->landingStrips[i] == -1)
        {
            aircraftToLand->airfieldObject->landingStrips[i] = aircraftToLand-
>number;
            aircraftToLand->strip = &(aircraftToLand->airfieldObject-
>landingStrips[i]);
            break;
        }
    }
}
```

Листинг 4 – Код в файле thread.h

```
/*! \file   thread.h
 *  \brief  Header file of function which is used as thread function
 */

#ifndef THREAD_H
#define THREAD_H

#include "task11.h"

/*! \brief Used as a thread function. Provides landing, waiting and parking
 * for the aircraft, implementing required thread control action
 *
 *  \param arg void* pointer. Must point to the Aircraft*
 *
 *  \return NULL
 */
void* aircraftThreadFunction(void* arg);


#endif //THREAD_H
```

Листинг 5 – Код в файле thread.c

```
/*! \file   thread.c
 *  \brief  Implements a function of thread.c
 */

#include "thread.h"


void* aircraftThreadFunction(void* arg)
{
    srand(time(NULL));


    Aircraft* aircraftObject = (Aircraft*) arg;

    printf("Aircraft %d is asking for land\n", aircraftObject->number);

    sem_wait(&(aircraftObject->airfieldObject->airfieldQueueSem));
    printf("Aircraft %d is permitted to land\n", aircraftObject->number);
```

Окончание листинга 5

```
    pthread_mutex_lock(&(aircraftObject->airfieldObject->airfieldBusyMutex));
    landAircraft(aircraftObject);
    printf("Aircraft %d has just landed\n",
           aircraftObject->number);

    pthread_mutex_unlock(
            &(aircraftObject->airfieldObject->airfieldBusyMutex));


    randomNanosleep(MIN_SLEEP, BIG_SLEEP);


    printf("Aircraft %d is getting parked\n", aircraftObject->number);
    *(aircraftObject->strip) = EMPTY_STRIP;
    aircraftObject->strip = NULL;

    sem_post(&(aircraftObject->airfieldObject->airfieldQueueSem));
    printf("Aircraft %d has just been parked\n",
           aircraftObject->number);

    free(aircraftObject);

    return NULL;
}
```

Листинг 6 – Код в файле input.h

```
/*! \file   input.h
 *  \brief  Header file of function to read integer with additional check
 */

#ifndef INPUT_H
#define INPUT_H

#include <stdbool.h>
#include <stdio.h>

/*! \enum
 *  \brief  Size of string for input
 */
enum sizes
{
    INPUT_SIZE = 200
};

/*! \brief Reads int
 *
 *  \details Reads int with additional check. Continues reading until
 *  correct value is read.
 *
 *  \param bool* Pointer to the function that checks additional condition.
 *  \return Integer read correct integer.
 */
int checkedInputInt(bool(* additionalCheck)(int));


#endif //INPUT_H
```

Листинг 7 – Код в файле input.c

```
/*! \file   input.c
 *  \brief  Implements functions of input.h
 */

#include "input.h"

int checkedInputInt(bool(* additionalCheck)(int))
{
    int result;
    char inputString[INPUT_SIZE];

    while (true)
    {
        scanf("%s", inputString);
        int flag = sscanf(inputString, "%d", &result);
        if (flag == 0 || flag == EOF)
        {
            printf("Input error!\n");
            continue;
        }
        if (!additionalCheck(result))
        {
            printf("Input error!\n");
            continue;
        }
        return result;
    }
}
```

4 Тестовые примеры работы программы

На рисунке 1 приведен пример работы программы.

```
Enter number of landing strips ( [1, 1000] )
3
3
Aircraft 1 is asking for land
Aircraft 1 is permitted to land
Aircraft 1 has just landed
Aircraft 2 is asking for land
Aircraft 2 is permitted to land
Aircraft 2 has just landed
Aircraft 3 is asking for land
Aircraft 3 is permitted to land
Aircraft 3 has just landed
Aircraft 4 is asking for land
Aircraft 5 is asking for land
Aircraft 1 is getting parked
Aircraft 1 has just been parked
Aircraft 4 is permitted to land
Aircraft 4 has just landed
Aircraft 6 is asking for land
Aircraft 7 is asking for land
Aircraft 8 is asking for land
Aircraft 9 is asking for land
Aircraft 3 is getting parked
Aircraft 3 has just been parked
Aircraft 5 is permitted to land
Aircraft 5 has just landed
Aircraft 10 is asking for land
^C
Process finished with exit code 130
```

Рисунок 1 – Пример работы программы 1