

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

\_\_\_\_\_

подпись, дата

А.С. Кузнецов

инициалы, фамилия

Студент КИ18-17/16 031830504

номер группы, зачетной книжки

\_\_\_\_\_

подпись, дата

Е.В. Железкин

инициалы, фамилия

Красноярск 2019

## СОДЕРЖАНИЕ

Содержание .....	2
1 Цель работы .....	3
2 Задача работы .....	3
3 Ход работы.....	4
4 Вывод.....	11

## **1 Цель работы**

Изучение программных средств синхронизации потоков в ОС GNU/Linux.

## **2 Задача работы**

Общая постановка задачи. Требуется разработать программу в виде Linux-приложения, для выполнения различных частей которой создаются и запускаются потоки управления, а для синхронизации доступа к требуемым ресурсам используются соответствующие объекты ОС. Результат выполнения выводится на терминал/консоль. Программа должна быть устойчивой к некорректному пользовательскому вводу. Функционирование программы, если это не оговаривается особо, может быть завершено только путем принудительного снятия процесса с выполнения.

**Вариант 2. «Обедающие философы 2».** В пансионе отдыхают и предаются размышлениям 5 философов (потоки), пронумерованные от 1 до 5. В столовой расположен круглый стол, вокруг которого расставлены 5 стульев, также пронумерованные от 1 до 5. На столе находится одна большая тарелка со спагетти, которая пополняется бесконечно, также там расставлены 5 тарелок, в которые накладывается спагетти, и 5 вилок (разделяемые ресурсы), назначение которых очевидно. Для того чтобы пообедать, философ входит в столовую и садится на любой стул. При этом есть философ сможет только в том случае, если свободны две вилки – справа и слева от его тарелки. При выполнении этого условия философ поднимает одновременно обе вилки и может поглощать пищу в течение какого-то заданного времени. В противном случае, философу приходится ждать освобождения обеих вилок. Пообедав, философ кладет обе вилки на стол одновременно и уходит. Величина временного промежутка для поглощения пищи устанавливается пользователем, а появление философа в столовой является случайной величиной с равномерным законом распределения.

### 3 Ход работы

Листинг 1 – содержание файла main.c проекта SP\_2:

```
/*! \file    main.c
 * \brief    Processing queued jobs.
 *
 * \details  Задача об обедающих философах, вариант 2.
 *           Входные данные - длительность приёма пищи философами.
 *           Если ввести значение 1(длительность приёма пищи философом)
 *           то очередь будет убывать с оптимальной скоростью.
 */

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

#define COUNT_OF_FORKS 5
#define COUNT_OF_PHILOSOPHERS 5
#define MAX_LENGTH 20
#define ASCII_CONST 48
#define NS_IN_SEC 1000000000
#define NEXT_DIGIT 10

// Инициализация мьютекса и объявление семафора.
pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
sem_t freePlacesCount;

/*! \struct queue
 * \brief Структура, с помощью которой формируется очередь.
 */
struct queue
{
    /*! Ссылка на следующий элемент в очереди. */
    struct queue* next;
    /*! Номер философа данной позиции. */
    int number;
};

struct queue* currentQueue; // Очередь, в порядке которой обедают философы.

/*! \struct philosopher
 * \brief Структура, с помощью которой передаются данные
 * потокам-философам.
 */
struct philosopher
{
    /*! Длительность временного промежутка, за который обедают философы.*/
    int eatTime;
    /*! Номер философа, передающийся в поток.*/
    int number;
};

/*! \brief Функция, обрабатывающая ввод.
 * @param parameters Строка считанная с клавиатуры.
```

```

 * @return number Число, содержащееся в строке, либо ошибка.
 */
int InputChecker(char* strForCheck)
{
    int number = 0;

    for (unsigned long i = 0; i < strlen(strForCheck); i++)
    {
        if (strForCheck[i] == '0' || strForCheck[i] == '1' ||
            strForCheck[i] == '2' || strForCheck[i] == '3' ||
            strForCheck[i] == '4' || strForCheck[i] == '5' ||
            strForCheck[i] == '6' || strForCheck[i] == '7' ||
            strForCheck[i] == '8' || strForCheck[i] == '9') number =
            number * NEXT_DIGIT + (strForCheck[i] - ASCII_CONST);
        else
            return -1;
    }

    if (number == 0) return -1;

    return number;
}

 /*! \brief Поток, описывающий поведение философов.
 * @param parameters Параметр потока типа void*, в котором содержится
 * структура, содержащая номер философа и длительность приёма им пищи.
 */
void* CallPhilosopher(void* parameters)
{
    struct philosopher* currentPhilosopher =
        (struct philosopher *) parameters;

    struct timespec tw;
    struct timespec tr;
    tw.tv_sec = currentPhilosopher->eatTime;
    struct queue* temp;

    while (1)
    {
        pthread_mutex_lock(&mtx);

        if (currentQueue == NULL) // Проверяем на наличие очереди.
        {
            pthread_mutex_unlock(&mtx);
            continue;
        }

         /* Проверяем, совпадает ли номер философа(данного потока)
         с тем, что первый в очереди. */
        if (currentQueue->number == currentPhilosopher->number)
        {
            sem_wait(&freePlacesCount);  // Используя семафор для разделения
            sem_wait(&freePlacesCount);  // вилоч, отправляем философа кушать.

            temp = currentQueue;

             /* Условие, которое удаляет
             текущего философа из очереди. */
            if (currentQueue->next != NULL)

```

```

        {
            currentQueue = currentQueue->next;
            free(temp);
        } else
        {
            currentQueue = NULL;
            free(temp);
        }

        pthread_mutex_unlock(&mtx);

        printf("Философ %d кушает.\n",
            currentPhilosopher->number);

        nanosleep(&tw, &tr);

        printf("Философ %d поел.\n",
            currentPhilosopher->number);

        sem_post(&freePlacesCount); // Освобождаем вилки
        sem_post(&freePlacesCount);
    } else
        pthread_mutex_unlock(&mtx);
}

}

int main()
{
    // Инициализация семафора.
    sem_init(&freePlacesCount, 0, COUNT_OF_FORKS);
    int eatTime = -1;
    printf("Задайте временной промежуток в секундах, "
        "в течении которого философы будут обедать: ");

    //Считывание временного интервала для поедания еды.
    char input[MAX_LENGTH];
    while (1)
    {
        scanf("%s", input);
        eatTime = InputChecker(input);
        if (eatTime == -1)
            printf("Проверьте корректность ввода и повторите попытку:");
        else break;
    }

    pthread_t philosopher1Id;
    pthread_t philosopher2Id;
    pthread_t philosopher3Id;
    pthread_t philosopher4Id;
    pthread_t philosopher5Id;

    struct philosopher args1 = {eatTime, 1};
    struct philosopher args2 = {eatTime, 2};
    struct philosopher args3 = {eatTime, 3};
    struct philosopher args4 = {eatTime, 4};
    struct philosopher args5 = {eatTime, 5};

    pthread_create(&philosopher1Id, NULL, &CallPhilosopher,
        &args1);

```

```

pthread_create(&philosopher2Id, NULL, &CallPhilosopher,
               &args2);
pthread_create(&philosopher3Id, NULL, &CallPhilosopher,
               &args3);
pthread_create(&philosopher4Id, NULL, &CallPhilosopher,
               &args4);
pthread_create(&philosopher5Id, NULL, &CallPhilosopher,
               &args5);

srand(time(0));
struct timespec tw;
tw.tv_sec = 0;
struct timespec tr;

while (1) // Цикл, который формирует очередь случайным образом.
{ // Интервал добавления философов: 0-1 с.

    if (currentQueue != NULL)
    {

        pthread_mutex_lock(&mtx);
        struct queue* temp;
        struct queue* temp2 = currentQueue;
        temp = (struct queue*) malloc(sizeof (struct queue));

        temp->number = rand() % COUNT_OF_PHILOSOPHERS + 1;

        if (temp2 == NULL)
        {
            printf("Текущая очередь пуста!\n");
            pthread_mutex_unlock(&mtx);
            continue;
        }

        while (temp2->next != NULL)
            temp2 = temp2->next;
        temp2->next = temp;

        temp2 = currentQueue;

        printf("Текущая очередь(->): ");
        while (temp2->next != NULL)
        {
            printf("%d ", temp2->number);
            temp2 = temp2->next;
        }
        printf("%d\n", temp2->number); // Пометка начала очереди.
        pthread_mutex_unlock(&mtx);

        tw.tv_nsec = rand() % NS_IN_SEC; // Случайный временной
        // промежуток(0-1с), отвечающий за задержку перед появлением
        // следующего элемента в очереди.
        nanosleep(&tw, &tr);
    } else
    {
        pthread_mutex_lock(&mtx);
        currentQueue = (struct queue*) malloc(sizeof (struct queue));
        currentQueue->number = rand() % COUNT_OF_PHILOSOPHERS + 1;
        pthread_mutex_unlock(&mtx);
    }
}

```

```

    }
}

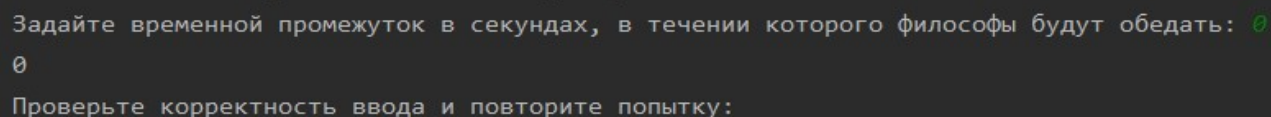
pthread_join(philosopher1Id, NULL);
pthread_join(philosopher2Id, NULL);
pthread_join(philosopher3Id, NULL);
pthread_join(philosopher4Id, NULL);
pthread_join(philosopher5Id, NULL);

struct queue* temp;
if (currentQueue != NULL)
{
    if(currentQueue->next != NULL)
    {
        while(currentQueue->next != NULL)
        {
            temp = currentQueue;
            currentQueue = currentQueue->next;
            free(temp);
        }
    }
    free(currentQueue);
}

return 0;
}

```

### 1) Примеры работы:

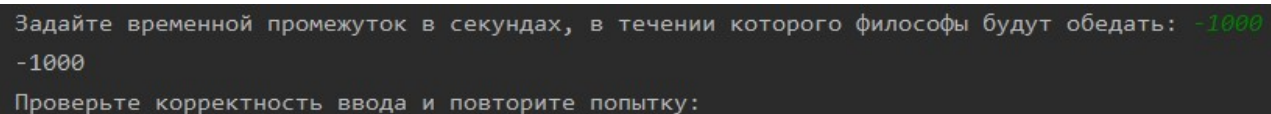


```

Задайте временной промежуток в секундах, в течении которого философы будут обедать: 0
0
Проверьте корректность ввода и повторите попытку:

```

Рисунок 1 – результат работы при вводе «0»



```

Задайте временной промежуток в секундах, в течении которого философы будут обедать: -1000
-1000
Проверьте корректность ввода и повторите попытку:

```

Рисунок 2 – результат работы при вводе «-1000»



```
Задайте временной промежуток в секундах, в течении которого философы будут обедать: 1
1
Текущая очередь(->): 1 5
Философ 1 кушает.
Философ 5 кушает.
Текущая очередь(->): 1 4
Текущая очередь(->): 1 4 5
Текущая очередь(->): 1 4 5 5
Текущая очередь(->): 1 4 5 5 4
Философ 5 поел.
Философ 1 поел.
Философ 1 кушает.
Философ 4 кушает.
Философ 4 поел.
Философ 1 поел.
Философ 5 кушает.
Текущая очередь(->): 5 4 3
Текущая очередь(->): 5 4 3 4
Текущая очередь(->): 5 4 3 4 2
Философ 5 поел.
Философ 5 кушает.
Философ 4 кушает.
Философ 4 поел.
```

Рисунок 3 - результат работы при вводе «1»

```
Задайте временной промежуток в секундах, в течении которого философы будут обедать: 2
2
Текущая очередь(->): 2 1
Философ 2 кушает.
Философ 1 кушает.
Философ 1 поел.
Философ 2 поел.
Философ 4 кушает.
Текущая очередь пуста!
Текущая очередь(->): 2 5
Философ 2 кушает.
Философ 4 поел.
Философ 5 кушает.
Текущая очередь пуста!
Текущая очередь(->): 2 2
Философ 2 поел.
Философ 2 кушает.
Текущая очередь(->): 2 4
```

Рисунок 4 – результат работы при вводе «2»

```
Задайте временной промежуток в секундах, в течении которого философы будут обедать: 3
3
Текущая очередь(->): 3 2
Философ 3 кушает.
Философ 2 кушает.
Философ 3 поел.
Философ 2 поел.
Философ 4 кушает.
Текущая очередь пуста!
Текущая очередь(->): 1 4
Философ 1 кушает.
Текущая очередь(->): 4 2
Текущая очередь(->): 4 2 5
Текущая очередь(->): 4 2 5 5
Текущая очередь(->): 4 2 5 5 1
Текущая очередь(->): 4 2 5 5 1 5
Текущая очередь(->): 4 2 5 5 1 5 5
Текущая очередь(->): 4 2 5 5 1 5 5 1
Философ 4 поел.
Философ 4 кушает.
Философ 1 поел.
Философ 2 кушает.
Философ 4 поел.
```

Рисунок 5 – результат работы при вводе «3»

#### **4 Вывод**

В ходе данной лабораторной работы были изучены азы синхронизации потоков в ОС GNU/Linux.