

**Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Ордена Трудового Красного Знамени федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский технический университет связи и информатики»**

Кафедра Системного программирования

ЛАБОРАТОРНАЯ РАБОТА №3
по дисциплине
ОПЕРАЦИОННЫЕ СИСТЕМЫ
на тему:
«Системные средства синхронизации в ОС GNU/LINUX»

Выполнил:
студент ____ Шамсутдинов Р.Ф. ____
(Ф.И.О.)
группа ____ БВТ2201 ____

Проверил:
____ Королькова Т. В. ____
(Ф.И.О., должность преподавателя)

Оценка ____

Дата ____ 18.03.2025 ____

Введение

Цель работы:

- изучение механизмов синхронизации с использованием сигналов, семафоров, мьютексов и барьеров;
- приобретение практических навыков применения средств синхронизации в многопоточных приложениях.

Задание:

1. Изучите теоретическую часть лабораторной работы.
2. Исследуйте на конкретном примере особенности 3-х по выбору из указанных методов синхронизации потоков:

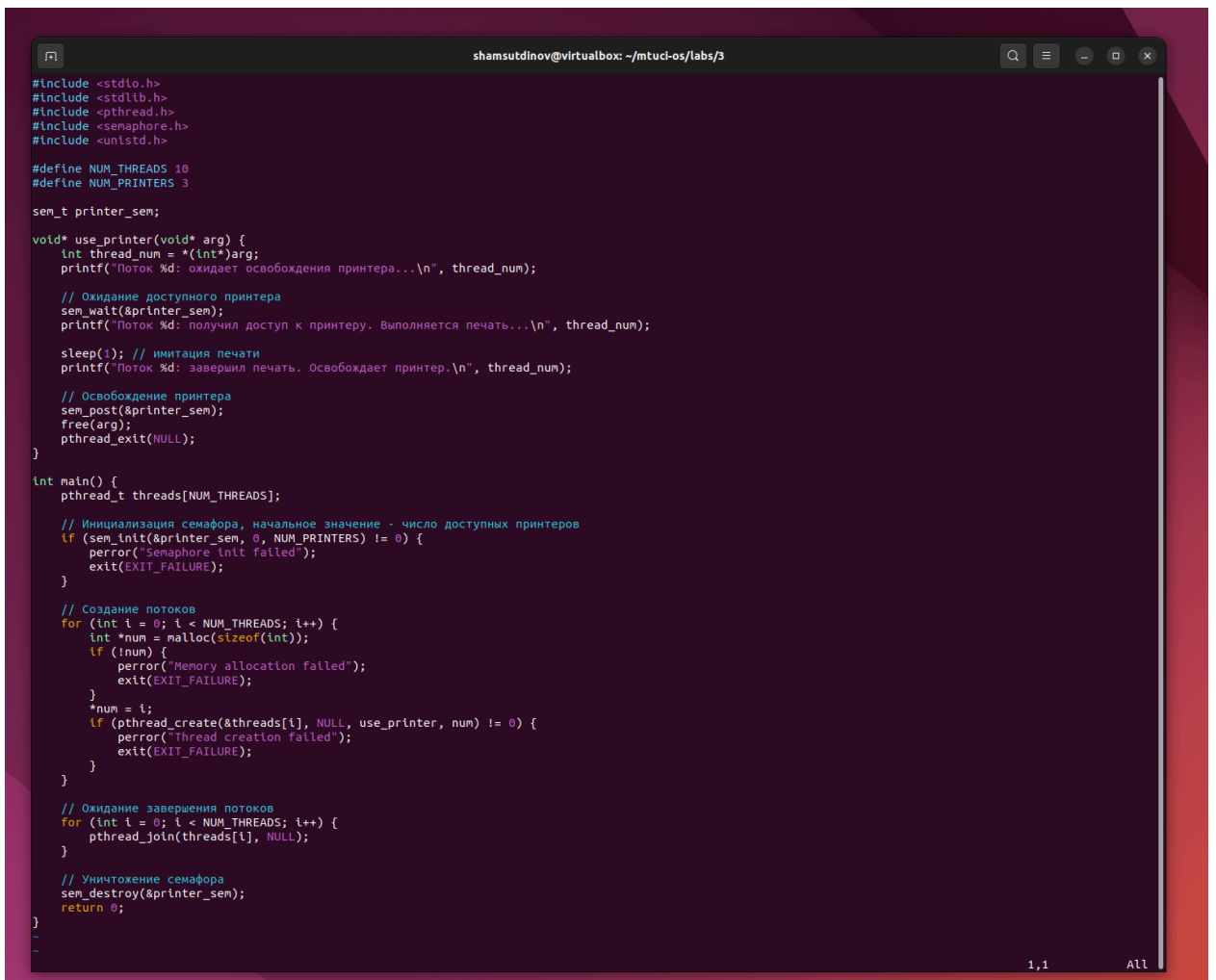
- 1) семафоры
- 2) мьютексы
- 3) сигналы
- 4) условные переменные
- 5) барьеры

Задачи для синхронизации придумайте самостоятельно, исходя из особенностей методов.

Примечание:

1. Задачи для каждого метода синхронизации должны быть различными.
2. Задачи должны наглядно демонстрировать выбранный метод синхронизации и учитывать особенности его применения.

Ход работы



```
shamsutdinov@virtualbox: ~/mtuci-os/labs/3

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define NUM_THREADS 10
#define NUM_PRINTERS 3

sem_t printer_sem;

void* use_printer(void* arg) {
    int thread_num = *(int*)arg;
    printf("Поток %d: ожидает освобождения принтера...\n", thread_num);

    // Ожидание доступного принтера
    sem_wait(&printer_sem);
    printf("Поток %d: получил доступ к принтеру. Выполняется печать...\n", thread_num);

    sleep(1); // имитация печати
    printf("Поток %d: завершил печать. Освобождает принтер.\n", thread_num);

    // Освобождение принтера
    sem_post(&printer_sem);
    free(arg);
    pthread_exit(NULL);
}

int main() {
    pthread_t threads[NUM_THREADS];

    // Инициализация семафора, начальное значение - число доступных принтеров
    if (sem_init(&printer_sem, 0, NUM_PRINTERS) != 0) {
        perror("Semaphore init failed");
        exit(EXIT_FAILURE);
    }

    // Создание потоков
    for (int i = 0; i < NUM_THREADS; i++) {
        int *num = malloc(sizeof(int));
        if (!num) {
            perror("Memory allocation failed");
            exit(EXIT_FAILURE);
        }
        *num = i;
        if (pthread_create(&threads[i], NULL, use_printer, num) != 0) {
            perror("Thread creation failed");
            exit(EXIT_FAILURE);
        }
    }

    // Ожидание завершения потоков
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    // Уничтожение семафора
    sem_destroy(&printer_sem);
    return 0;
}
```

1,1 All

Рис. 1 – первая программа (семафоры)

```
shamsutdinov@virtualbox: ~/mtuci-os/labs/3$ vim semafor.c
shamsutdinov@virtualbox:~/mtuci-os/labs/3$ gcc semafor.c -o semafor -lpthread && ./semafor
Поток 0: ожидает освобождения принтера...
Поток 0: получил доступ к принтеру. Выполняется печать...
Поток 2: ожидает освобождения принтера...
Поток 2: получил доступ к принтеру. Выполняется печать...
Поток 1: ожидает освобождения принтера...
Поток 1: получил доступ к принтеру. Выполняется печать...
Поток 3: ожидает освобождения принтера...
Поток 5: ожидает освобождения принтера...
Поток 4: ожидает освобождения принтера...
Поток 6: ожидает освобождения принтера...
Поток 7: ожидает освобождения принтера...
Поток 8: ожидает освобождения принтера...
Поток 9: ожидает освобождения принтера...
Поток 1: завершил печать. Освобождает принтер.
Поток 0: завершил печать. Освобождает принтер.
Поток 3: получил доступ к принтеру. Выполняется печать...
Поток 5: получил доступ к принтеру. Выполняется печать...
Поток 2: завершил печать. Освобождает принтер.
Поток 4: получил доступ к принтеру. Выполняется печать...
Поток 3: завершил печать. Освобождает принтер.
Поток 4: завершил печать. Освобождает принтер.
Поток 6: получил доступ к принтеру. Выполняется печать...
Поток 7: получил доступ к принтеру. Выполняется печать...
Поток 5: завершил печать. Освобождает принтер.
Поток 8: получил доступ к принтеру. Выполняется печать...
Поток 6: завершил печать. Освобождает принтер.
Поток 7: завершил печать. Освобождает принтер.
Поток 9: получил доступ к принтеру. Выполняется печать...
Поток 8: завершил печать. Освобождает принтер.
Поток 9: завершил печать. Освобождает принтер.
shamsutdinov@virtualbox:~/mtuci-os/labs/3$
```

Рис. 2 – вывод первой программы

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NUM_THREADS 5
#define NUM_ITERATIONS 1000000

int counter = 0;
pthread_mutex_t counter_mutex;

void* increment_counter(void* arg) {
    for (int i = 0; i < NUM_ITERATIONS; i++) {
        pthread_mutex_lock(&counter_mutex);
        counter++; // критическая секция
        pthread_mutex_unlock(&counter_mutex);
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t threads[NUM_THREADS];

    // Инициализация мьютекса
    if (pthread_mutex_init(&counter_mutex, NULL) != 0) {
        perror("Mutex init failed");
        exit(EXIT_FAILURE);
    }

    // Создание потоков
    for (int i = 0; i < NUM_THREADS; i++) {
        if (pthread_create(&threads[i], NULL, increment_counter, NULL) != 0) {
            perror("Thread creation failed");
            exit(EXIT_FAILURE);
        }
    }

    // Ожидание завершения потоков
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    printf("Final counter value: %d\n", counter);

    // Уничтожение мьютекса
    pthread_mutex_destroy(&counter_mutex);
    return 0;
}
```

Рис. 3 – вторая программа (мьютексы)

```
shamsutdinov@virtualbox: ~/mtuct-os/labs/3
shamsutdinov@virtualbox:~/mtuct-os/labs/3$ vim mutex.c
shamsutdinov@virtualbox:~/mtuct-os/labs/3$ gcc mutex.c -o mutex -lpthread && ./mutex
Final counter value: 5000000
shamsutdinov@virtualbox:~/mtuct-os/labs/3$
```

Рис. 4 – вывод второй программы

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define NUM_THREADS 5

pthread_barrier_t barrier;

void* thread_work(void* arg) {
    int thread_num = *(int*)arg;
    printf("Поток %d: Фаза 1 - выполняется работа...\n", thread_num);

    // Инициация работы первой фазы
    sleep(1 + thread_num % 3);
    printf("Поток %d: достиг барьера.\n", thread_num);

    // Синхронизация: ожидание всех потоков
    int rc = pthread_barrier_wait(&barrier);
    if (rc == PTHREAD_BARRIER_SERIAL_THREAD) {
        printf("Поток %d: специальный поток (serial) на барьере выполняет дополнительные действия.\n", thread_num);
    }

    printf("Поток %d: Фаза 2 - продолжается выполнение...\n", thread_num);
    free(arg);
    pthread_exit(NULL);
}

int main() {
    pthread_t threads[NUM_THREADS];

    // Инициализация барьера с числом потоков, равным NUM_THREADS
    if (pthread_barrier_init(&barrier, NULL, NUM_THREADS) != 0) {
        perror("Barrier init failed");
        exit(EXIT_FAILURE);
    }

    // Создание потоков
    for (int i = 0; i < NUM_THREADS; i++) {
        int *num = malloc(sizeof(int));
        if (!num) {
            perror("Memory allocation failed");
            exit(EXIT_FAILURE);
        }
        *num = i;
        if (pthread_create(&threads[i], NULL, thread_work, num) != 0) {
            perror("Thread creation failed");
            exit(EXIT_FAILURE);
        }
    }

    // Ожидание завершения потоков
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    // Уничтожение барьера
    pthread_barrier_destroy(&barrier);
    return 0;
}

"barrier.c" 60L, 1950B 1,1 All
```

Рис. 5 – третья программа (барьеры)

```
shamsutdinov@virtualbox: ~/mtuci-os/labs/3
shamsutdinov@virtualbox:~/mtuci-os/labs/3$ vim barrier.c
shamsutdinov@virtualbox:~/mtuci-os/labs/3$ gcc barrier.c -o barrier -lpthread && ./barrier
Поток 0: Фаза 1 - выполняется работа...
Поток 3: Фаза 1 - выполняется работа...
Поток 4: Фаза 1 - выполняется работа...
Поток 2: Фаза 1 - выполняется работа...
Поток 1: Фаза 1 - выполняется работа...
Поток 4: достиг барьера.
Поток 0: достиг барьера.
Поток 3: достиг барьера.
Поток 1: достиг барьера.
Поток 2: достиг барьера.
Поток 2: специальный поток (serial) на барьере выполняет дополнительные действия.
Поток 2: Фаза 2 - продолжается выполнение...
Поток 4: Фаза 2 - продолжается выполнение...
Поток 0: Фаза 2 - продолжается выполнение...
Поток 1: Фаза 2 - продолжается выполнение...
Поток 3: Фаза 2 - продолжается выполнение...
shamsutdinov@virtualbox:~/mtuci-os/labs/3$
```

Рис. 6 – вывод третьей программы

Заключение

Вывод: проделав работу, мы изучили механизмы синхронизации с использованием сигналов, семафоров, мьютексов и барьеров и приобрели практические навыки применения средств синхронизации в многопоточных приложениях.