Федеральное агентство связи
Сибирский государственный университет телекоммуникаций и информатики

Лабораторная работа №11

Выполнил: студент группы ИП-211
Оганесян Альберт
Лацук Андрей
Проверил:
Профессор кафедры ПМиК
Малков Е. А.

Новосибирск 2024

Цель: получение навыков синхронизации с использованием мьютексов и семафоров

Задание: реализовать алгоритм “производитель-потребитель” для конечного буфера

Ход работы:

1. Конечный буфер реализуем массивом фиксированного размера *(BUFFER_SIZE)*. Для чтения и записи используем индексы *in* и *out* соответственно.

```
#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];
int in = 0, out = 0;
```

2. Определим мьютекс и семафоры. Мьютекс гарантирует, что только один поток (производитель или потребитель) имеет доступ к критической секции в любой момент времени. Семафоры отслеживают количество свободных и заполненных мест.

```
pthread_mutex_t mutex;
sem_t empty;
sem_t full;
```

3. Реализуем функцию производителя. Производитель генерирует случайное число и помещает его в буфер. После он ждёт, пока в буфере освободится место (если буфер полон).

```
void *producer(void *arg)
{
    int item;
    while (!stop_flag)
    {
        item = rand() % 100;
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);

        if (stop_flag)
        {
            pthread_mutex_unlock(&mutex);
            sem_post(&empty);
            break;
        }
```

```
        buffer[in] = item;
        printf("Производитель произвел: %d\n", item);
        in = (in + 1) % BUFFER_SIZE;
        produced_count++;

        pthread_mutex_unlock(&mutex);
        sem_post(&full);

        if (produced_count >= MAX_ITEMS)
        {
            stop_flag = 1;
            break;
        }
        sleep(rand() % 2);
    }
    return NULL;
}
```

4. Реализуем функцию потребителя. Потребитель извлекает число из буфера. После он ждёт, пока в буфере появится хотя бы один элемент (если буфер пуст).

```
void *consumer(void *arg)
{
    int item;
    while (!stop_flag || consumed_count < produced_count)
    {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);

        if (stop_flag && consumed_count >= produced_count)
        {
            pthread_mutex_unlock(&mutex);
            sem_post(&full);
            break;
        }

        item = buffer[out];
        printf("Потребитель потребил: %d\n", item);
        out = (out + 1) % BUFFER_SIZE;
        consumed_count++;

        pthread_mutex_unlock(&mutex);
        sem_post(&empty);

        sleep(rand() % 3);
    }
```

```
    return NULL;
}
```

5. Добавим флаг и сигнал завершения и лимит обработки. Если глобальная переменная set_flag установлена в 1, то производитель и потребитель завершают свои циклы. Обработчик сигналов SIGINT (Ctrl+C) позволяет корректно завершить программу по запросу пользователя. После обработки MAX_ITEMS элементов программа завершает работу.

```
#define MAX_ITEMS 8

int produced_count = 0;
int consumed_count = 0;

int stop_flag = 0;

void handle_signal(int sig)
{
    stop_flag = 1;
    printf("\nПолучен сигнал завершения. Завершаем программу...\n");
}
```

6. Соберем все в функции main

```
int main()
{
    pthread_t prod, cons;

    // Устанавливаем обработчик сигнала завершения
    signal(SIGINT, handle_signal);

    // Инициализация мьютекса и семафоров
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);

    // Создание потоков производителя и потребителя
    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);

    // Ожидание завершения потоков
    pthread_join(prod, NULL);
    pthread_join(cons, NULL);
```

```
    // Очистка ресурсов
    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);

    printf("Программа завершена. Произведено: %d, Потреблено: %d\n",
produced_count, consumed_count);
    return 0;
}
```

7. Пример вывода программы с автоматическим завершением

```
Производитель произвел: 83
Потребитель потребил: 83
Производитель произвел: 15
Потребитель потребил: 15
Производитель произвел: 86
Потребитель потребил: 86
Производитель произвел: 21
Потребитель потребил: 21
Производитель произвел: 90
Производитель произвел: 63
Потребитель потребил: 90
Потребитель потребил: 63
Производитель произвел: 26
Производитель произвел: 11
Потребитель потребил: 26
Потребитель потребил: 11
Программа завершена. Произведено: 8, Потреблено: 8
```

8. Пример вывода программы с ручным завершением

```
Производитель произвел: 83
Потребитель потребил: 83
Производитель произвел: 15
Потребитель потребил: 15
^C
Получен сигнал завершения. Завершаем программу...
Программа завершена. Произведено: 2, Потреблено: 2
```

Вывод: Алгоритм "производитель-потребитель" успешно реализован с использованием синхронизации потоков средствами мьютексов и семафоров. Программа обеспечивает корректный обмен данными между производителем и потребителем без гонок данных и взаимоблокировок.