Министерство науки и высшего образования

Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего образования

«Новосибирский государственный технический университет»



Кафедра теоретической и прикладной информатики

### Лабораторная работа № 3 по дисциплине «Проектирование Систем Реального Времени»

### Синхронизация потоков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Факультет: | ПМИ |  |  |
| Группа: | ПМИМ-01 |  |  |
| Студенты: | Ершов П. К.  Грициенко И. Г. |  |  |
| Бригада: | 7 |  |  |
| Преподаватель: | Кобылянский В. Г. |  |  |

Новосибирск

2021

1. **Цель работы**

Целью работы является изучение механизмов синхронизации.

1. **Задание на лабораторную**
2. Скомпилировать и выполнить примеры программ. Проанализировать результаты выполнения.
3. Написать программу, реализующую задание в соответствии с вариантом:

|  |  |
| --- | --- |
| № варианта | Задание |
| 7 | На основе механизма условной переменной, разработать приложение подсчета контрольной суммы CRC16 файла. Один поток читает порциями данные из файла в общий буфер, второй поток подсчитывает контрольную сумму. |

1. **Ход работы.**
   1. Анализ работы программы-примера из пункта 2.4. работа с мутексом

Результаты выполнения кода-примера:

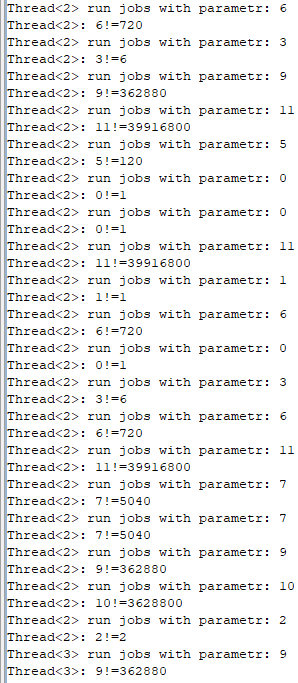


Рисунок 1. Результат выполнения программы вычисляющей факториал с применением мутексов в работе

В главном потоке сначала выполняется функция jobs\_producer(), которая формирует список задач (в данном случае, список из чисел, факториал которых нужно получить).

Затем создаются два потока с идентификаторами 2 и 3 соответственно. В этих потока выполняются функции process\_job, которые вычисляют факториал числа, удаляя его из списка.

В силу того, что функции могут обратиться к одному ресурсу одновременно, что вызовет ошибку исполнения. Для решения этой проблемы можно использовать мутекс, который позволяет потоку блокировать участок кода, защищая его от исполнения другими потоками. Это гарантирует последовательное обращения к ресурсам.

* 1. Анализ работы программы-примера из пункта 2.4. работа с семафорами

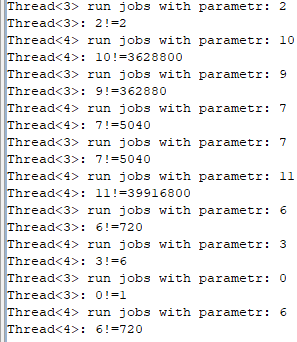


Рисунок 2. Результат выполнения программы вычисляющей факториал с применением семафоров в работе

Программа в главном потоке создаёт семафор функцией **sem\_init**(&sem, 0, 0) с изначальным значение 0. Это означает, что семафор заблокирован в начале работы программы, а с логической точки зрения, что задач нет.

Затем инициализируется поток, в котором запускается функция **jobs\_producer**(**void** \*arg), в которой создаётся число, факториал которого нужно получить. Семафор после этого увеличивается на 1 функцией **sem\_post**(&sem), что в свою очередь означает наличие задач для других потоков. При этом участок кода, связанный со списком задач блокируется с помощью блокировки мутекса. Это делается для того, чтобы сразу после разблокирования семафора не произошла критическая ошибка.

После этого инициализируются потоки, в которых выполняется функция **thread\_function**(**void** \*arg), которая с помощью функции **sem\_wait**(&sem) ждёт, когда семафор будет разблокирован. Как только семафор увеличивается на 1 потоком с функцией **jobs\_producer**, поток **thread\_function**, сразу же блокирует семафор, уменьшая его значение на 1. После этого участок кода, в котором происходит обработка списка задач блокируется мутексом, задача из массива задач удаляется, а функция получает факториал числа из задачи.

Вся программа работает 10 секунд, после чего завершается.

* 1. Анализ работы программы-примера из пункта 2.4. работа с условными переменными

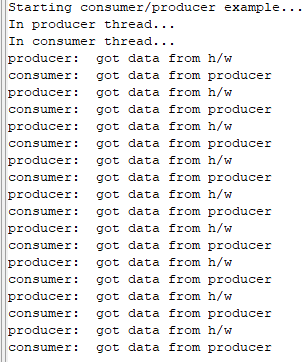


Рисунок 3. Результат выполнения программы, один поток которой производит данные, а другой потребляет

Программа создаёт в главном потоке два потока:

1. Поток выполнения функции producer, которая производит данные.
2. Поток выполнения функции consumer, которая обрабатывает (потребляет) данные.

Для блокировки потока используются мутекс. Producer создаёт данные, блокирует мутекс после чего функция **pthread\_cond\_wait()** блокирует поток исполнения producer и разблокирует мутекс и начинает ждать когда другой поток разблокирует поток выполнения producer на случайной переменной с помощью функции **pthread\_cond\_signal()**.

После того как функция **pthread\_cond\_wait()** разблокировала мутекс, функция consumer блокирует мутекс и ждёт когда появятся данные. Когда данные получены и обработаны, consumer функцией **pthread\_cond\_signal** разблокирует поток producer и разблокирует мутекс.

* 1. Программа успешно реализована

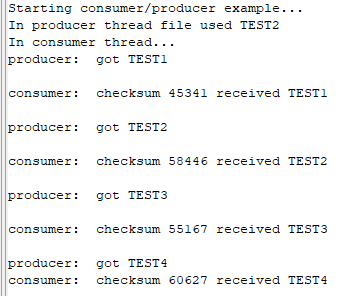


Рисунок 4. Результат выполнения программы по получению контрольной суммы CRC16

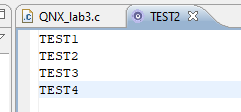


Рисунок 5. Содержимое используемого файла TEST2

1. **Код программы**

**Код программы-примера использования мутекса**

**#include** <stdlib.h>

**#include** <stdio.h>

**#include** <malloc.h>

**#include** <math.h>

**#include** <pthread.h>

**#define** N 20 // число заданий

// структура-элемент списка

**struct** job

{

**int** n; // параметр задания

**struct** job \*next; // указатель на следующее задание

};

**struct** job \*job\_queue; // список заданий

// функция выполнения задания

**void** **process\_job**(**struct** job \*the\_job)

{

**int** i, M=0;

**long** factorial=1;

**printf**("Thread<%d> run jobs with parametr: %d\n", **pthread\_self**(), the\_job->n);

// здесь проводятся вычисления

// в данном случае вычисление факториала из числа

// указанного в задании (число n).

M=the\_job->n;

**for**(i=2; i<=M; i++)

{

factorial\*=i;

}

**printf**("Thread<%d>: %d!=%ld\n", **pthread\_self**(), M, factorial);

}

// функция потока, выполняющего задания

**void** \***thread\_function**(**void** \*arg)

{

**while**(1)

{

**struct** job \*next\_job;

// захватываем мутекс

**pthread\_mutex\_lock**(&mutex);

// в критической секции монопольно работаем со списком заданий

**if** (job\_queue==NULL)

next\_job=NULL;

**else**

{

next\_job=job\_queue;

// переводим указатель на следующее задание

job\_queue=job\_queue->next;

}

**pthread\_mutex\_unlock**(&mutex);

// на выходе из критической секции

// получаем указатель на текущее задание (next\_job)

**if** (next\_job==NULL)

**break**;

process\_job(next\_job);

**free**(next\_job);

}

**return** NULL;

}

// функция-генератор заданий

**void** \***jobs\_producer**(**void** \*arg)

{

**int** i;

**struct** job \*point;

**for**(i=N; i>0; i--)

{

// создаем список из N заданий

point=(**struct** job\*)**malloc**(**sizeof**(**struct** job));

point->n=**rand**()%12;

point->next=job\_queue;

job\_queue=point;

}

**return** NULL;

}

**int** **main**(**int** argc, **char** \*argv[])

{

pthread\_t first\_tid, second\_tid;

// генерируем задания

jobs\_producer(NULL);

// и создаем два потока для их выполнения

**pthread\_create**(&first\_tid, NULL, thread\_function, NULL);

**pthread\_create**(&second\_tid, NULL, thread\_function, NULL);

// ждем завершения потоков, которые выполняют задания

**pthread\_join**(first\_tid, NULL);

**pthread\_join**(second\_tid, NULL);

**return** EXIT\_SUCCESS;

}

**Код программы-примера использования семафоров**

**#include** <stdlib.h>

**#include** <stdio.h>

**#include** <malloc.h>

**#include** <math.h>

**#include** <pthread.h>

**#include** <semaphore.h>

**struct** job

{

**int** n;

**struct** job \*next;

};

**struct** job \*job\_queue;

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

// объявляем семафор для блокирования потоков-исполнителей,

// если нет заданий в списке

sem\_t sem;

// функция выполнения задания

**void** **process\_job**(**struct** job \*the\_job)

{

**int** i, M=0;

**long** factorial=1;

**printf**("Thread<%d> run jobs with parametr: %d\n", **pthread\_self**(), the\_job->n);

// здесь проводятся вычисления

// в данном случае вычисление факториала из числа, указанного в задании (число n).

M=the\_job->n;

**for**(i=2; i<=M; i++)

{

factorial\*=i;

}

**printf**("Thread<%d>: %d!=%ld\n", **pthread\_self**(), M, factorial);

}

**void** \***thread\_function**(**void** \*arg)

{

**while**(1)

{

**struct** job \*next\_job;

// уменьшаем на единицу счетчик семафора.

// если счетчик равен нулю, то поток

// будет ждать увелечения значения счетчика

**sem\_wait**(&sem);

**pthread\_mutex\_lock**(&mutex);

// входим в критическую секцию

// и монопольно действуем со списком заданий -

// удаляем одно задание из списка

next\_job=job\_queue;

job\_queue=job\_queue->next;

**pthread\_mutex\_unlock**(&mutex);

process\_job(next\_job);

**free**(next\_job);

}

**return** NULL;

}

// функция-генератор заданий

// задания генерируется во время работы

**void** \***jobs\_producer**(**void** \*arg)

{

**struct** job \*point;

**while**(1)

{

// создаем элемент-задание

point=(**struct** job\*)**malloc**(**sizeof**(**struct** job));

point->n=**rand**()%12;

// входим в критическую секцию

// для записи задания в список

**pthread\_mutex\_lock**(&mutex);

point->next=job\_queue;

job\_queue=point;

// увеличиваем на единицу счетчик семафора,

// тем самым сообщая, что появилось задание

**sem\_post**(&sem);

**pthread\_mutex\_unlock**(&mutex);

sleep(1); // задания генерируется с задержкой

}

**return** NULL;

}

**int** **main**(**int** argc, **char** \*argv[])

{

// в начале список заданий пуст

job\_queue=NULL;

// инициализируем счетчик семафора значением ноль

**sem\_init**(&sem, 0, 0);

**pthread\_create**(NULL, NULL, jobs\_producer, NULL);

**pthread\_create**(NULL, NULL, thread\_function, NULL);

**pthread\_create**(NULL, NULL, thread\_function, NULL);

// подождем 10 секунд и завершим приложение

sleep(10);

**return** EXIT\_SUCCESS;

}

**Код программы-примера использования условных переменных**

**#include** <stdio.h>

**#include** <pthread.h>

// флаг готовности данных

**int** data\_ready=0;

//статически инициализируем мyтекс и условную переменную

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

pthread\_cond\_t condvar=PTHREAD\_COND\_INITIALIZER;

// этот поток обрабатывает данные - потребитель

**void** \***consumer**(**void** \*notused)

{

**printf**("In consumer thread...\n");

**while**(1)

{

**pthread\_mutex\_lock**(&mutex);

**while**(!data\_ready)

{

// если нет данных, то ждем, пока они появятся

**pthread\_cond\_wait**(&condvar, &mutex);

}

**printf**("consumer: got data from producer\n");

// здесь может быть обработка данных

// это имитируется функцией sleep()

sleep(1);

// устанавливаем флаг, что нет необработанных данных

data\_ready=0;

// разблокируем производителя данных

**pthread\_cond\_signal**(&condvar);

**pthread\_mutex\_unlock**(&mutex);

}

}

// этот поток получает данные - производитель

**void** \***producer**(**void** \*notused)

{

**printf**("In producer thread...\n");

**while**(1)

{

// получаем данные из устройства

// это имитируется функцией sleep()

sleep(1);

**printf**("producer: got data from h/w\n");

**pthread\_mutex\_lock**(&mutex);

**while**(data\_ready)

{

// если есть данные, то ждем, пока они обработаются

**pthread\_cond\_wait**(&condvar, &mutex);

}

// устанавливаем флаг, что есть необработанные данные

data\_ready=1;

// разблокируем потребителя данных

**pthread\_cond\_signal**(&condvar);

**pthread\_mutex\_unlock**(&mutex);

}

}

**int** **main**()

{

**printf**("Starting consumer/producer example...\n");

// создаем потоки производителя и потребителя данных

**pthread\_create**(NULL, NULL, producer, NULL);

**pthread\_create**(NULL, NULL, consumer, NULL);

// подождем немного и завершим приложение

sleep(10);

**return** 0;

}

**Код разработанной программы**

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

**#include** <pthread.h>

**#include** <sys/neutrino.h>

**#include** <errno.h>

**#define** FILENAME\_MAX 255

**#define** N 1024

// флаг готовности данных

**int** data\_ready=0;

//статически инициализируем мyтекс и условную переменную

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

pthread\_cond\_t condvar=PTHREAD\_COND\_INITIALIZER;

**char** buff[FILENAME\_MAX]; // главный массив

**unsigned** **short** **crc16**(**const** **unsigned** **char**\* data\_p, **unsigned** **short** length) // функция получения контрольной суммы

{

**unsigned** **char** x;

**unsigned** **short** crc = 0xFFFF;

**while** (length--){

x = crc >> 8 ^ \*data\_p++;

x ^= x>>4;

crc = (crc << 8) ^ ((**unsigned** **short**)(x << 12)) ^ ((**unsigned** **short**)(x <<5)) ^ ((**unsigned** **short**)x);

}

**return** crc;

}

// этот поток обрабатывает данные - потребитель

**void** \***consumer**(**void** \*notused)

{

**printf**("In consumer thread...\n");

**while**(1)

{

**pthread\_mutex\_lock**(&mutex);

**while**(!data\_ready)

{

// если нет данных, то ждем, пока они появятся

**pthread\_cond\_wait**(&condvar, &mutex);

}

**unsigned** **char** \*conbuf = (**unsigned** **char** \*)buff;

**unsigned** **short** lenght = (**unsigned** **short**)**strlen**(conbuf);

**unsigned** **short** crc16Summ = crc16(conbuf, lenght);

**printf**("consumer: checksum %i received %s\n", (**int**)crc16Summ, buff);

// здесь может быть обработка данных

sleep(1);

// устанавливаем флаг, что нет необработанных данных

data\_ready=0;

// разблокируем производителя данных

**pthread\_cond\_signal**(&condvar);

**pthread\_mutex\_unlock**(&mutex);

}

}

// этот поток получает данные - производитель

**void** \***producer**(**void** \*notused)

{

**int** i = 0;

**char** b[N];

**char** \*ch = (**char** \*) **malloc**(**sizeof** (**char**) \* N);

**char** fileName[FILENAME\_MAX];

FILE \*fl;

**strcpy**(fileName, notused);

**printf**("In producer thread file used %s\n", fileName);

**if** ((fl = **fopen**(fileName, "r")) == NULL)

{

**printf**("Невозможно открыть файл для чтения.\n");

**exit**(1);

}

**while**(!**feof**(fl))

{

sleep(1);

**fgets**(b, N, fl);

**if** (b[0] != '\n' )

**strcpy**(ch, b);

**strcpy**(buff, ch);

**printf**("producer: got %s\n", buff);

**pthread\_mutex\_lock**(&mutex);

**while**(data\_ready)

{

// если есть данные, то ждем, пока они обработаются

**pthread\_cond\_wait**(&condvar, &mutex);

}

sleep(1);

// устанавливаем флаг, что есть необработанные данные

data\_ready = 1;

// разблокируем потребителя данных

**pthread\_cond\_signal**(&condvar);

**pthread\_mutex\_unlock**(&mutex);

i++;

}

}

**int** **main**()

{

**printf**("Starting consumer/producer example...\n");

// создаем потоки производителя и потребителя данных

**char** \*file\_name = "TEST2";

**pthread\_create**(NULL, NULL, producer, (**void**\*)file\_name);

**pthread\_create**(NULL, NULL, consumer, NULL);

// подождем немного и завершим приложение

sleep(10);

**return** 0;

}