Лабораторная работа №3

«Синхронизация потоков»

Цели: Изучение функций, предназначенных для синхронизации потоков в ОС Linux.

Задачи: Получение практических навыков использования мьютексов и условных переменных для решения задач доступа к ресурсам.

Срок выполнения: 4 часа

Общие сведения

При выполнении нескольких потоков довольно часто требуется их синхронизировать в таких ситуациях, как осуществление доступа к общим ресурсам. Одно из основных преимуществ использования потоков – это легкость использования средств синхронизации. В данной лабораторной работе мы рассмотрим следующие средства синхронизации, используемые в ОС Linux:

- блокировки взаимного исключения (мьютексы),
- условные переменные.

Для понимания значения синхронизации рассмотрим простой пример. Предположим, что два потока используют одну и ту же глобальную переменную.

Поток А

```
x = common_variable ;
x++ ;
common_variable = x ;
```

Поток В

```
y = common_variable ;
y-- ;
common_variable = y;
```

В данном случае результат будет зависеть от того, в какой последовательности будет происходить выполнение двух процессов. Данная ситуация называется состязанием. Состязания обычно трудно обнаружить, потому что они появляются время от времени без какой-либо закономерности.

Мьютексы

Мьютекс (mutex) – это сокращение от «взаимное исключение» (mutual exlusion). Мьютексы – это один из основных способов реализации синхронизации потоков в ОС Linux.

Мьютекс действует как блокировка, защищающая доступ к разделенной структуре данных. Основная идея работы мьютексов заключается в том, что в определенный момент времени захватить мьютекс может только один единственный поток. Никакие потоки не могут захватить мьютекс до тех пор, пока владеющий поток не освободит данный мьютекс.

Очень часто потоки, использующие мьютексы, выполняют действия над глобальными переменными. Использование мьютексов – безопасный способ изменения глобальной переменной несколькими потоками, гарантирующий, что конечное значение будет таким же, как и в случае, если бы эти действия выполнялись одним потоком.

Функции для работы с мьютексами

Функция	`int pthread_mutex_init( pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr);`
Описание	Создание и инициализация нового мьютекса `mutex` и установка его атрибутов в соответствие с атрибутами мьютекса `mutexattr`. При создании мьютекс разблокирован. Вместо указателя на атрибуты мьютекса допускается передача указателя `NULL`, что означает атрибуты по умолчанию.
Функция	`int pthread_mutex_destroy(pthread_mutex_t *mutex);`
Описание	Уничтожение мьютекса.
Функция	`int pthread_mutexattr_init(pthread_mutexattr_t *attr);` `int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);`
Описание	Создание и уничтожение атрибутов мьютекса.
Функция	`int pthread_mutex_lock(pthread_mutex_t *mutex);`
Описание	Захват мьютекса. Если данный мьютекс уже захвачен другим потоком, этот вызов заблокирует вызывающий поток до тех пор, пока мьютекс не будет освобожден.
Функция	`int pthread_mutex_trylock(pthread_mutex_t *mutex);`
Описание	Попытка захватить мьютекс. Однако если мьютекс уже захвачен другим потоком, этот вызов не будет блокировать вызывающий поток, а вернет управление немедленно. Эта функция может быть полезна для предотвращения взаимоблокировок.
Функция	`int pthread_mutex_unlock(pthread_mutex_t *mutex);`
Описание	Освобождение мьютекса, если его владелец – вызывающий поток. Вызов этой функции необходим, если поток закончил работу с защищенными данными и другие потоки ожидают освобождения мьютекса. Если мьютекс уже освобожден, или владельцем его является другой поток, то функция возвращает ошибку.

Примеры использования мьютексов

Пример 1. Программа без мьютексов.

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

int x=1;

void* compute_thread(void * argument)
{
printf("X value in thread before sleep = %d\n",x);
printf("X value in thread is incremented by 1 before sleep\n");
x++;
sleep(2);
printf("X value in thread after sleep = %d\n",x);
return;
}
```

```
main( )
{
pthread_t tid;
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_create(&tid, &attr, compute_thread,(void *)NULL);
sleep(1);
x++;
printf("Main thread increments X, after that X value is %d\n",x);
pthread_join(tid,NULL);
exit(0);
}
```

Пример 2. Предыдущий пример, но с использованием мьютекса.

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

int x=1;

/* This is the lock for thread synchronization */
pthread_mutex_t my_sync;

void* compute_thread(void * argument)
{
printf("X value in thread before sleep = %d\n",x);
printf("X value in thread is incremented by 1 before sleep\n");
pthread_mutex_lock(&my_sync);
x++;
sleep(2);
printf("X value in thread after sleep = %d\n",x);
pthread_mutex_unlock(&my_sync);
return;
}

main( )
{
pthread_t tid;
pthread_attr_t attr;
pthread_attr_init(&attr);

/* Initialize the mutex (default attributes) */

pthread_mutex_init (&my_sync,NULL);
pthread_create(&tid, &attr, compute_thread,(void *)NULL);
sleep(1);
pthread_mutex_lock(&my_sync);
x++;
printf("Main thread increments X, after that X value is %d\n",x);
```

```
pthread_mutex_unlock(&my_sync);

pthread_join(tid,NULL);

exit(0);
}
```

Условные переменные

Условные переменные предлагают еще один способ синхронизации потоков. В то время как мьютексы реализуют синхронизацию путем контроля доступа со стороны потоков к данным, условные переменные позволяют потокам синхронизироваться, исходя из значения переменной.

Без условных переменных программисту пришлось бы непрерывно опрашивать переменную (возможно в критической секции – части программы, где могут происходить состязания) до тех пор, пока ее значение не станет равным ожидаемому. Это может потребовать много ресурсов, прежде всего, процессорного времени. Условные переменные позволяют добиться той же цели без опроса. Условная переменная всегда используется в паре с мьютексом.

Типичная последовательность для использования условной переменной:

- создание и инициализация условной переменной
- создание и инициализация ассоциированного мьютекса
- определение предиката (переменной, значение которой должно быть проверено)
- захват мьютекса
- исполнение потока доходит до точки, где ему нужно ждать выполнения какого-либо события (например, установки предиката в определенное значение)
- поток ждет условную переменную, выполняя следующие действия
    - освобождение мьютекса
    - ожидание изменения предиката с использованием условной переменной
- другой поток производит вычисления, меняет значение предиката, сообщает об этом ждущему потоку, производя следующие действия
    - захват мьютекса
    - изменение предиката
    - сигнализация через условную переменную
    - освобождение мьютекса

Функции для работы с условными переменными

Функция	`int` **`pthread_cond_init`**`(pthread_cond_t *cond, pthread_condattr_t *cond_attr);`
Описание	Создание и инициализация новой условной переменной. Идентификатор созданной условной переменной возвращается через параметр cond.
Функция	`int` **`pthread_cond_destroy`**`(pthread_cond_t *cond);`
Описание	Уничтожение условной переменной cond, освобождение выделенной ранее памяти.
Функция	`int` **`pthread_cond_wait`**`(pthread_cond_t *cond, pthread_mutex_t *mutex);`
Описание	Блокировка вызывающего потока до вызова из другого потока функции

	`pthread_cond_signal` или `pthread_cond_broadcast`. Эта функция должна вызываться после захвата мьютекса; она освобождает мьютекс на все время ожидания.
Функция	`int` **`pthread_cond_signal`**`(pthread_cond_t *cond);`
Описание	Эта функция будит другой поток (неизвестно какой, если их несколько), который усыплен на условной переменной. Эта функция должна вызываться после захвата мьютекса.
Функция	`int` **`pthread_cond_broadcast`**`(pthread_cond_t *cond);`
Описание	Эта функция используется взамен `pthread_cond_signal`, если нужно разбудить сразу все усыпленные потоки.

Пример 3. Использование условной переменной.

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

void* compute_thread (void*);
pthread_mutex_t my_sync;
pthread_cond_t rx;

int thread_done = 0;
int x=1;

main( )
{
pthread_t tid;
pthread_attr_t attr;

pthread_attr_init (&attr);
pthread_mutex_init (&my_sync, NULL);
pthread_cond_init (&rx, NULL);

pthread_create(&tid, &attr, compute_thread, (void *)NULL);
pthread_mutex_lock(&my_sync);
while (!thread_done) pthread_cond_wait(&rx, &my_sync);
x++;
printf("Main thread increments X, after that X value is %d\n",x);
pthread_mutex_unlock(&my_sync);
exit(0);
}

void* compute_thread(void* dummy)
{
printf("X value in thread before sleep = %d\n",x);
printf("X value in thread is incremented by 1 before sleep\n");
pthread_mutex_lock (&my_sync);
x++;
sleep(2);
```

```
printf("X value in thread after sleep = %d\n",x);
thread_done = 1;
pthread_cond_signal (&rx);
pthread_mutex_unlock (&my_sync);
return;
}
```

Задания на лабораторную работу:

1. Выполните следующие действия. Для каждого изменения опишите, что происходит, и дайте объяснение – почему.
   1) Уберите в первом примере функцию sleep.
   2) Уберите во втором примере функцию освобождения мьютекса из порожденного потока.
   3) Уберите в третьем примере функцию pthread_cond_signal.
2. Решите классическую проблему «поставщик – потребитель» с использованием описанных в лабораторной работе средств синхронизации.

   Постановка задачи:
   Один поток производит данные, другой поток их потребляет. В промежуток времени между изготовлением и потреблением данные хранятся в буфере.

   Пример использования: Конвейер команд в Unix.

   Исходные данные:
   Данные хранятся в циклическом буфере. Циклический буфер описывается некоторой областью памяти, указателем начала данных и указателем конца данных. Поток-поставщик записывает данные в конец буфера, поток-потребитель считывает их с начала буфера. После записи или чтения соответствующим образом меняются указатели начала и конца.

   Операции чтения/записи должны быть выполнены как взаимоисключающие.
   Если операция чтения выполняется над пустым буфером (указатель начала = указатель конца), поток-потребитель должен быть заблокирован на условной переменной до тех пор, пока поток-поставщик не запишет в буфер какие-нибудь данные. Если операция записи выполняется над полным буфером, поток-поставщик должен также быть заблокирован на условной переменной до тех пор, пока поток-потребитель не считает из буфера какие-нибудь данные.

   Размер буфера – не менее 10 символов.
   Поток-поставщик и поток-потребитель работают в бесконечном цикле.
   Поток-поставщик производит по одному символу в последовательности 0,1,2...9,0,1,... и записывает его в буфер через случайный интервал времени 0,5 – 2 сек.
   Поток-потребитель считывает по одному символу через случайный интервал времени 0,5 – 2 сек из буфера и выводит их на экран в виде сообщений (например, Символ 0, Символ 1,...)

   Каждый поток совершая операцию с буфером выводит на экран информацию о текущем состоянии буфера до и после операции, тип операции, символ, состояние условной переменной.

Содержание отчета по лабораторной работе:

1. Цель и описание лабораторной работы.
2. Письменные ответы на вопросы задания 1.
3. Текст и результаты работы программы (задание 2).
4. Блок-схему работы программы.
5. Вывод.

Контрольные вопросы:

1. Какие атрибуты существуют у мьютекса?
2. Каким образом происходит блокировка мьютекса?
3. Какие атрибуты существуют у условной переменной?
4. Каким образом происходит блокировка условной переменной? Что происходит с мьютексом?
5. Как будет работать Ваша программа без условной переменной?

Рекомендуемая литература:

1. Библиотека glibc. http://www.gnu.org/software/libc
2. В.А.Костромин, "Linux для пользователя", 2002.
3. Б.В.Керниган, Р.Пайк «UNIX - универсальная среда программирования»,1992.
4. А.Робачевский, «Операционная система UNIX», 2000.
5. Руководства по операционной системе Linux с сайта **http://rus-linux.net**
6. Книги и руководства по операционной системе Linux с сайта **http://linuxland.itam.nsc.ru/book.html**