Alexey A. Romanenko
*arom@ccfit.nsu.ru*

Based on Sun Microsystems' presentation

- Обзор POSIX threads
- Сборка  программ
- Функции управления потоками
- И пр.

Многопоточные программы преимущественно ориентированны на исполнение на системах с общей памятью (SMP). Это такие системы, где установленно несколько процессоров или\и многоядерные процессоры и каждое ядро имеет доступ ко всей оперативной памяти компьютера. Сейчас уже редко можно встретить одноядерную систему, так что эта лекция полезна всем разработчикам ПО.

Потоки существуют внутри процесса и используют его ресурсы.

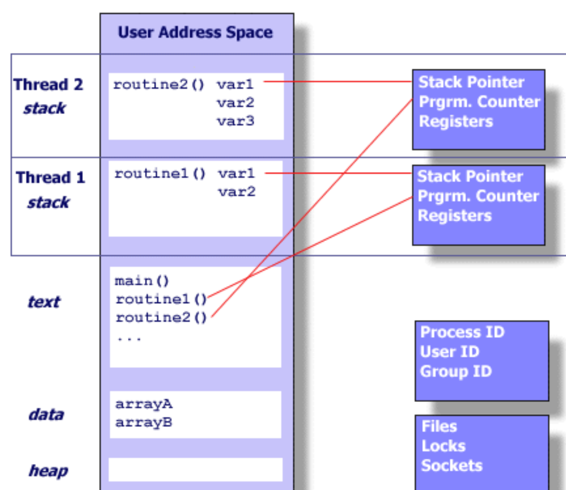
Поток (thread) – независимый поток исполнения команд в ранках процесса.

Поток не процесс!

Определяя поток часто сопоставляют поток и процесс. Процесс обладает такими свойствами, а поток такими. В русском языке поток (thread) еще называют нитью.

Нить существуют внутри процесса и используют его ресурсы.

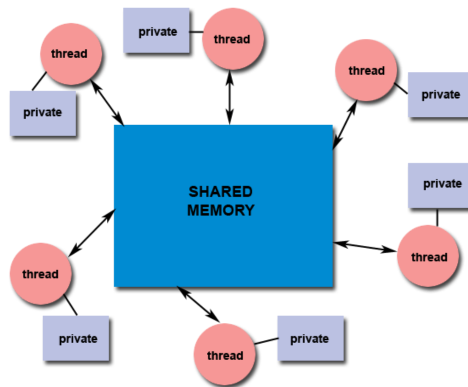
Нить (thread) – независимый поток исполнения команд в рамках процесса




Нить имеет свой стек, свои переменные, свой счетчик команд, но при этом располагается в адресном пространстве процесса и имеет общий с ним сегмент кода и пр.

# Модель разделяемой памяти

- Все потоки имеют доступ к разделяемой глоальной памяти
- Данные могкт быть как приватными так и общими
- Общие данные доступны всем потокам
- Приветные – только одному
- Требуется синхронизация для доступа к общим данным




Напомним, что в модели разделяемой памяти всю память можно разделить на общую, которая доступна всем потокам, и приватную, доступ к которой имеет только один выделеный процесс. Поскольку к переменным в общей памяти могут одновременно могут иметь доступ несколько потоков, то явно нужно обеспечивать синхронизацию доступа к ним.

- POSIX  определяет набор интерфейсов (функций заголовочных файлов) для программирования потоков. Эти рекомендации носят название POSIX threads или  Pthreads.

## Атрибуты потоков

**Общие**

- process ID
- parent process ID
- process group ID and session ID
- controlling terminal
- user and group Ids
- open file descriptors
- record locks
- file mode creation mask
- current directory and root directory

**Различные**

- thread ID (the pthread_t data type)
- signal mask (pthread_sigmask)
- the errno variable
- alternate signal stack (sigaltstack)
- real-time scheduling policy and priority

В рамках процесса различные потоки имеют свои атрибуты. Часть этих атрибутов наследуются от процесса и они общие для всех потоков, а часть индивидуальны для каждого потока.

## Thread-safe функции

- POSIX.1-2001 требует, чтобы все функции были thread-safe, за исключение следующих:
  - crypt()
  - ctime()
  - encrypt()
  - dirname()
  - localtime()
  - gethostbyname()
  - etc. see specification.

Под thread-safe функциями понимаются такие, которые безопасно можно вызывать из разных потоков. Как правило это такие функции, которые не используют глобальные ресурсы для своей работы. Стандарт требует, чтобы все функции были таковыми. Исключение делается только для некоторых из них.

## Преимущества и недостатки

- **Преимущества**:
  - Затраты на создание потоков меньше, чем на создание процессов (~ 2 ms for threads)
  - многозадачность, т.е., один процесс может обслуживать несколько клиентов
  - Переключение между потоками для ОС менее накладно, чем переключение между процессами
- **Недостатки**:
  - Многопоточное программирование тербует более аккуратного подхода к разработке поскольку
    - Отладка сложнее
    - Создание нескольких потоков на однопроцессорной машине не обязательно приведет к увеличению производительности

Естественно, что как любая модель, модель программирования потоков имеет свои преимущества и недостатки.

Основное преимущество использования потоков состоит в том, что они более легковесны, нежели процессы. Операционной системе надо меньше времени на их создание и переключение контекста между потоками по сравнению с процессами.

Недостатки, пожалуй, тоже очевидны. Так многопоточные приложения сложнее отлаживать, чем последовательные. Кроме того при некорректном использовании потоков параллельная программа может даже работать медленнее, чем последовательная

# POSIX Threads (pthreads)

- IEEE's POSIX Threads Model:
  - Модель программирования потоков для UNIX platform
  - pthreads включает международные стандарты ISO/IEC9945-1
- Программная модель pthreads определяет:
  - Создание потоков
  - Управление исполнением потоуов
  - Управление разделяемыми ресурсами процесса

Стандарт родился как модель программирования потоков для Unix систем и основывался на стандарте IEC9945-1

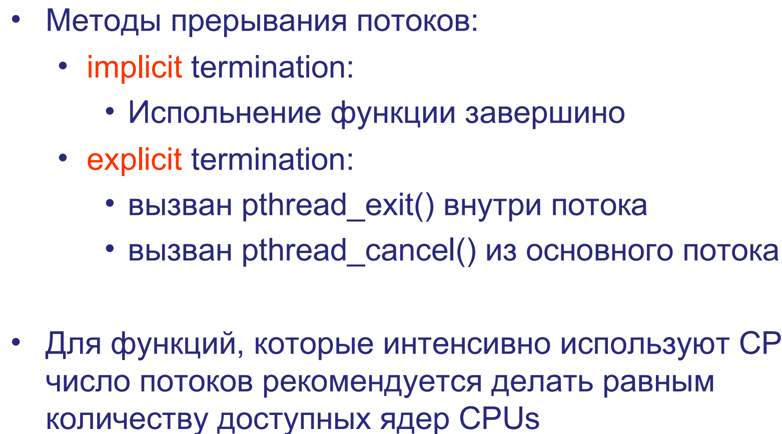
Стандарт определяет интерфейсы по созданю и управлению потоками, по механизмам разграничечинения доступа к разденяемым ресурсам процесса.

- main thread:
  - Создается когда функция main() (in C) или PROGRAM (in fortran) вызываеьтся загрузчиком процесса
  - Функция main() может создавать длчернии threads
  - Если основной поток завершает работу, процесс прерывается даже если внутри процесса существуют другие потоки, если только не предприняты специальные действия
  - Для избегания прерывания процесса можно использовать pthread_exit()

В программе всегда сужествует один поток, который создаются при загрузки функции main в Си и PROGRAM в фортране.

Далее основной поток может создать дочернии потоки. При этом если основной поток завершается, автоматически (если не предпринято специальных действий) завершаются и дочернии потоки.

Для того, чтобы завершение дочерних потоков не происходило, можно использовать функцию выхода из (основного) потока pthread_exit()




Завершение потоков может происходить по двум причинам, а именно: завершилась функция, которую поток исполняет или (что аналогично) поток вызвал pthred_exit(), поток прервали.

## Примеры Pthreads программ на C++ и Fortran 90/95

- на С++
  - необходимо включить pthread.h
  - Функции и типы имеют префикс pthread_ (за исключением семафоров)
  - Важно уметь пользоваться указателями.
- На Fortran 90/95
  - Необходимо включить модуль f_pthread
  - У функций префикс f_pthread_ (за исключением семафоров).

Далее будут представлены простые примеры программ на Си и на Фортране, в которых создаются потоки.

На Си\Си++ необходимо включить заголовочный файл, в котором описаны все протатипы функций и типы данных. В фортране необходимо подключить специальный модуль.

Все функции и типы данных на Си и на фортране имеют свой префикс. Функции работы с семафорами выбиваются из этого правила.

```
1    //*************************************************************
2    //    This is a sample threaded program in C++.  The main thread creates
3    //    4 daughter threads.  Each daughter thread simply prints out a message
4    //    before exiting.  Notice that I've set the thread attributes to joinable and
5    //    of system scope.
6    //*************************************************************
7    #include <iostream.h>
8    #include <stdio.h>
9    #include <pthread.h>
10
11   #define NUM_THREADS 4
12
13   void *thread_function( void *arg );
14
15   int main( void )
16   {
17        int i, tmp;
18        int arg[NUM_THREADS] = {0,1,2,3};
19
20        pthread_t thread[NUM_THREADS];
21        pthread_attr_t attr;
22
23        // initialize and set the thread attributes
24        pthread_attr_init( &attr );
25        pthread_attr_setdetachstate( &attr, PTHREAD_CREATE_JOINABLE );
26        pthread_attr_setscope( &attr, PTHREAD_SCOPE_SYSTEM );
27
```

Все ключевые моменты отмечеы красным цветом

```
28      // creating threads
29      for ( i=0; i<NUM_THREADS; i++ )
30      {
31          tmp = pthread_create( &thread[i], &attr, thread_function, (void *)&arg[i] );
32
33          if ( tmp != 0 )
34          {
35              cout << "Creating thread " << i << " failed!" << endl;
36              return 1;
37          }
38      }
39
40      // joining threads
41      for ( i=0; i<NUM_THREADS; i++ )
42      {
43          tmp = pthread_join( thread[i], NULL );
44          if ( tmp != 0 )
45          {
46              cout << "Joing thread " << i << " failed!" << endl;
47              return 1;
48          }
49      }
50
51      return 0;
52  }
53
```

```
54    //***********************************************************
55    //  This is the function each thread is going to run.  It simply asks
56    //  the thread to print out a message.  Notice the pointer acrobatics.
57    //***********************************************************
58    void *thread_function( void *arg )
59    {
60        int id;
61
62        id = *((int *)arg);
63
64        printf( "Hello from thread %d!\n", id );
65        pthread_exit( NULL );
66    }
```

- Как компилировать:
  - В Linux:
    > {C++ comp} -D_REENTRANT hello.cc -lpthread -o hello
  - Возможно необходимо определит _POSIX_C_SOURCE (to 199506L)
- Создание потока:

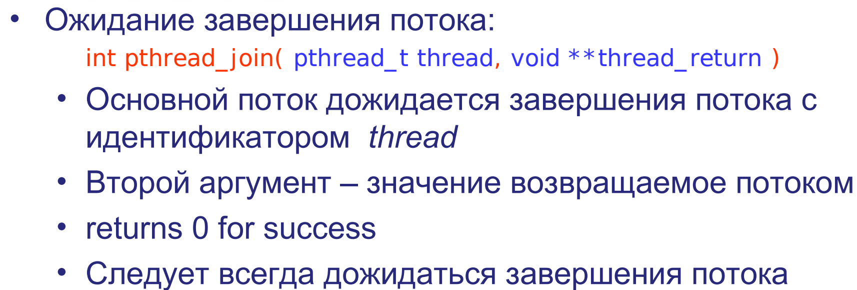
```
int pthread_create( pthread_t *thread, pthread_attr_t *attr,       void
    *(*thread_function)(void *), void *arg );
```

  - thread –указатель на идентификатор созданного потока
  - attr – атрибуты потока
  - third argument – функция, которую поток будет исполнять
  - arg – аргументы функции (обычно структура)
  - returns 0 for success

Для сборки программ необходимо минимум указать библиотеку (-lpthread). Иногда требуется определить макрос _POSIX_C_SOURCE

Создание потока производится с помощью функции pthread_craete, ее протатип на слайде.

Стоит отметить, что параметр с атрибутами не обязятелен (может быть NULL), тогда будут взяты атрибуты по-умолчанию.




Всегда следует дожидаться завершения потока (если он в состоянии joinable – по-умолчанию). Для этого используется функция pthread_join(). Последний пораметр может быть NULL.

```
1       !*****************************************************************
2       !   This is a sample threaded program in Fortran 90/95.  The main thread
3       !   creates 4 daughter threads.  Each daughter thread simply prints out
4       !   a message before exiting.  Notice that I've set the thread attributes to
5       !   be joinable and of system scope.
6       !*****************************************************************
7       PROGRAM hello
8
9       USE f_pthread
10      IMPLICIT NONE
11
12      INTEGER, PARAMETER :: num_threads = 4
13      INTEGER :: i, tmp, flag
14      INTEGER, DIMENSION(num_threads) :: arg
15      TYPE(f_pthread_t), DIMENSION(num_threads) :: thread
16      TYPE(f_pthread_attr_t) :: attr
17
18      EXTERNAL :: thread_function
19
20      DO i = 1, num_threads
21          arg(i) = i – 1
22      END DO
23
24      !initialize and set the thread attributes
25      tmp = f_pthread_attr_init( attr )
26      tmp = f_pthread_attr_setdetachstate( attr, PTHREAD_CREATE_JOINABLE )
27      tmp = f_pthread_attr_setscope( attr, PTHREAD_SCOPE_SYSTEM )
28
```

```
29      ! this is an extra variable needed in fortran (not needed in C)
30      flag = FLAG_DEFAULT
31
32      ! creating threads
33      DO i = 1, num_threads
34         tmp = f_pthread_create( thread(i), attr, flag, thread_function, arg(i) )
35         IF ( tmp /= 0 ) THEN
36               WRITE (*,*) "Creating thread", i, "failed!"
37               STOP
38         END IF
39      END DO
40
41      ! joining threads
42      DO i = 1, num_threads
43         tmp = f_pthread_join( thread(i) )
44         IF ( tmp /= 0 ) THEN
45               WRITE (*,*) "Joining thread", i, "failed!"
46               STOP
47         END IF
48      END DO
49
```

```
50   !**************************************************************
51   !    This is the subroutine each thread is going to run.  It simply asks
52   !    the thread to print out a message.  Notice that f_pthread_exit() is
53   !    a subroutine call.
54   !**************************************************************
55   SUBROUTINE thread_function( id )
56
57   IMPLICIT NONE
58
59   INTEGER :: id, tmp
60
61   WRITE (*,*) "Hello from thread", id
62   CALL f_pthread_exit()
63
64   END SUBROUTINE thread_function
```

- Как компилировать:
  - Для Unix систем:

    > (fortran compiler) –lpthread hello.f –o hello
  - Компилятор должен быть thread safe
- Концепция создания и ожидания завершения потоков аналогична C/C++.

Программа на фортране собирается аналогичным образом. Концепция создания и завершения потоков аналогично концепции в Си.

## Атрибуты потоков

- Отсоединенное состояние:

```
int pthread_attr_setdetachstate(pthread_attr_t *attr,      int
   detachstate);
```

  - detached – основной поток может не дожидаться завершения дочернего потока
  - Joinable – основной поток должен дождаться завершения дочернего потока

- Размер стека, приоритет, ...

Атрибуты потоков определяют размер стека (по-умолчанию 1МБ для 32-разрядных систем), приоритеты потоков, класс планирования и пр.

Например задать, что поток находится в отсоединенном состоянии можно с помощью функции  pthread_attr_setdetachstate(). В отсоединенном состояние на поток не надо делать pthread_join – дожидаться его заваршения.

- pipeline model – потоки исполняются друг за другом
- master-slave model – основной поток распределяет работу по дочерним потокам
- equal-worker model – все потоки эквавалентны

Распределять работу между потоками можно разными способами. Три из них наиболее распространенные представлены на слайде.

## Механизмы синхронизации потоков

- Взаимное сиключение (mutex):
  - Ограничивает доступ множества потоков к одному ресурсу в разделяемой памяти
  - обеспечивает locking/unlocking критического участка кода, где разделяемый ресурс модифицируется
  - Каждый поток ждет пока mutex будет разблокирован (потоком, который его блокировал) прежде чем войти в критическую секцию

Наличие общей разделяемой памяти приводит к необходимости синхронизовать доступ к ней в те моменты, когда один или несколько потоков пытаются поменять значение переменных.

Простейший способ – использовать взаимные блокировки (metex). Мутекс – переменная, которая может быть или заблокирована, или находиться в свободном состояние. При этом если один поток ее заблокировал, то другие, попытавшись выполнить блокировку, будут ожидать, пока заблокировавший поток не отпустит этот мутекс.

- Базовые функции:

```
int pthread_mutex_init(pthread_mutex_t *mutex, const
    pthread_mutexattr_t *mutexattr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

  - pthread_mutex_t тип данных описывающий mutex
  - mutex как ключ доступа к кретической секции. Только один поток владеет им
  - атрибуты mutex можно контролировать функцией pthread_mutex_init()
  - lock/unlock функции работают в тандеме

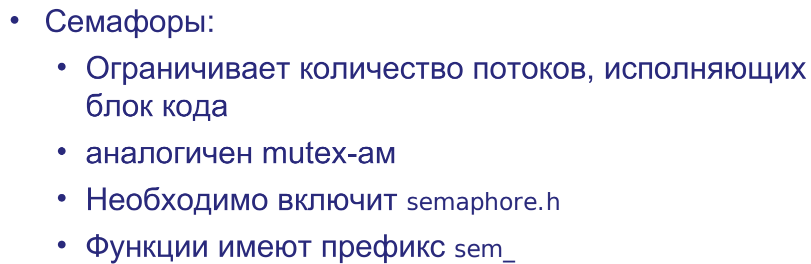
Базовые функции по созданию и использованию мутексов представлены на слайде.

Стоит еще раз сказать, что поток захвативший мутекс должен его отпусть. Никакой другой поток выполнить unlock этого мутекса не может.

```
#include <pthread.h>
...
pthread_mutex_t my_mutex;     // should be of global scope
...
int main()
{
    int tmp;
    ...
    // initialize the mutex
    tmp = pthread_mutex_init( &my_mutex, NULL );
    ...
    // create threads
    ...
    pthread_mutex_lock( &my_mutex );
        do_something_private();
    pthread_mutex_unlock( &my_mutex );
    ...
    return 0;
}
```

Пример простой программы с критической секцие  представлен на слайде.

По достежению критической секции поток перед блокировкой мутекса проверяет свободен ли мутекс и если нет, то ждет его освобождения. В противном случае поток захватывает мутекс  и после выполнения критической секции его отпускает.



- Семафоры:
  - Ограничивает количество потоков, исполняющих блок кода
  - аналогичен mutex-ам
  - Необходимо включит semaphore.h
  - Функции имеют префикс sem_

Семафор – другой примитив синхронизации. Семафоры описаны в заголовочном файле semaphore.h и функции по работе с семафорами и типы данных имеют префикс sen_

- Основные функции:
  - Создание:

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

    - Инициализирует объект sem
    - Pshared - значение 0 означает, что семафор локален для процесса
    - Начальное значение для семафора устанавливается в value
  - уничтожение:

```
int sem_destroy(sem_t *sem);
```

    - Освобождение ресурсов, связанных с sem
    - Обычно после pthread_join()
    - Результат действий с использованием уничтоженного семофора не определен

Передиспользованием семафора его надо создать, а после использования – удалить. Удаление следует выполнять после завершения всех потоков, его использующих.

- Управление семафором:

```
int sem_post(sem_t *sem);
int sem_wait(sem_t *sem);
```

  - sem_post атомарно увеличивает значение семафора на 1
  - sem_wait атомарно уменьшает значение семафора на 1; предварительно дождавшись пока значение не станет больше 0

Семофор аналогичен мутексам за одним исключением: если мутекс – бинарная переменная, то семафор – целое положительное число.

Если в критическую секцию позволительно впускать не более чем N потоков, то можно значение семафора первоначально сделать равным N и при каждом входе потока в секцию уменьшать значение семафора на 1, а при выходе увеличивать.

```
#include <pthread.h>
#include <semaphore.h>
...
void *thread_function( void *arg );
...
sem_t semaphore;        // also a global variable just like mutexes
...
int main()
{
      int tmp;
      ...
      // initialize the semaphore
      tmp = sem_init( &semaphore, 0, 0 );
      ...
      // create threads
      pthread_create( &thread[i], NULL, thread_function, NULL );
      ...
      while ( still_has_something_to_do() )
      {
          sem_post( &semaphore );
          ...
      }
      ...
      pthread_join( thread[i], NULL );
      sem_destroy( &semaphore );
      return 0;
}
```

В примере основной поток увеличивает значение семафора пока есть работа, а дочерний процес берет работу, декрементирует значение семафора.

Так продолжается пока есть данные.

```
void *thread_function( void *arg )
{
     sem_wait( &semaphore );
     perform_task_when_sem_open();
     ...
     pthread_exit( NULL );
}
```

-

- При программировании SMP систем попробуйте сначала:
  - OpenMP directives
  - SMP-enabled libraries
  - Автоматическое распараллеливание с помощью компилятора
- Применение Pthreds не гарантирует лучшую производительность
- Отладка затруднена
- Так почему же? В некоторых случаях это единственный возможный подход.

Подводя итоги стоит сказать, что многопоточное программирование с использованием потоков не самый простой способ создания параллельной программы, но иногда это единственный способ реализовать задуманное.

Перед тем как браться за программирование с использованием POSIX threads рекомендуется рассмотреть варианты реализации программы с  использованием директив OpenMP или попросит компилятор выполнить параллелизацию кода.

- *Programming with POSIX threads*, by D. Butenhof, Addison Wesley (1997).
- *Beginning Linux Programming*, by R. Stones and N. Matthew, Wrox Press Ltd (1999).
- www.opengroup.org/onlinepubs/007908799/xsh/threads.html
- www.research.ibm.com/actc/Tools/thread.htm
- www.unm.edu/cirt/introductions/aix_xlfortran/xlflr101.htm
- www.emsl.pnl.gov:2080/capabs/mset/training/ibmSP/fthreads/fthreads.html
  - *Programming with Threads*, by G. Bhanot and R. Walkup
  - *Appendix: Mixed models with Pthreads and MPI*, V. Sonnad, C. Tamirisa, and G. Bhanot


# Ссылки по теме