# Введение в POSIX threads

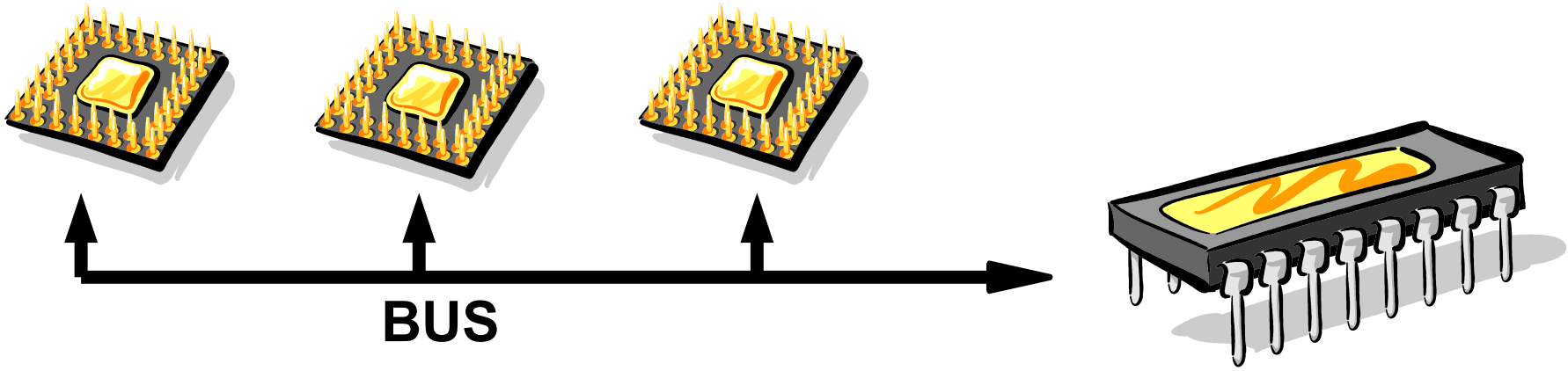
Alexey A. Romanenko
*arom@ccfit.nsu.ru*

Based on Sun Microsystems' presentation

# О чем раздел?

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

# SMP системы



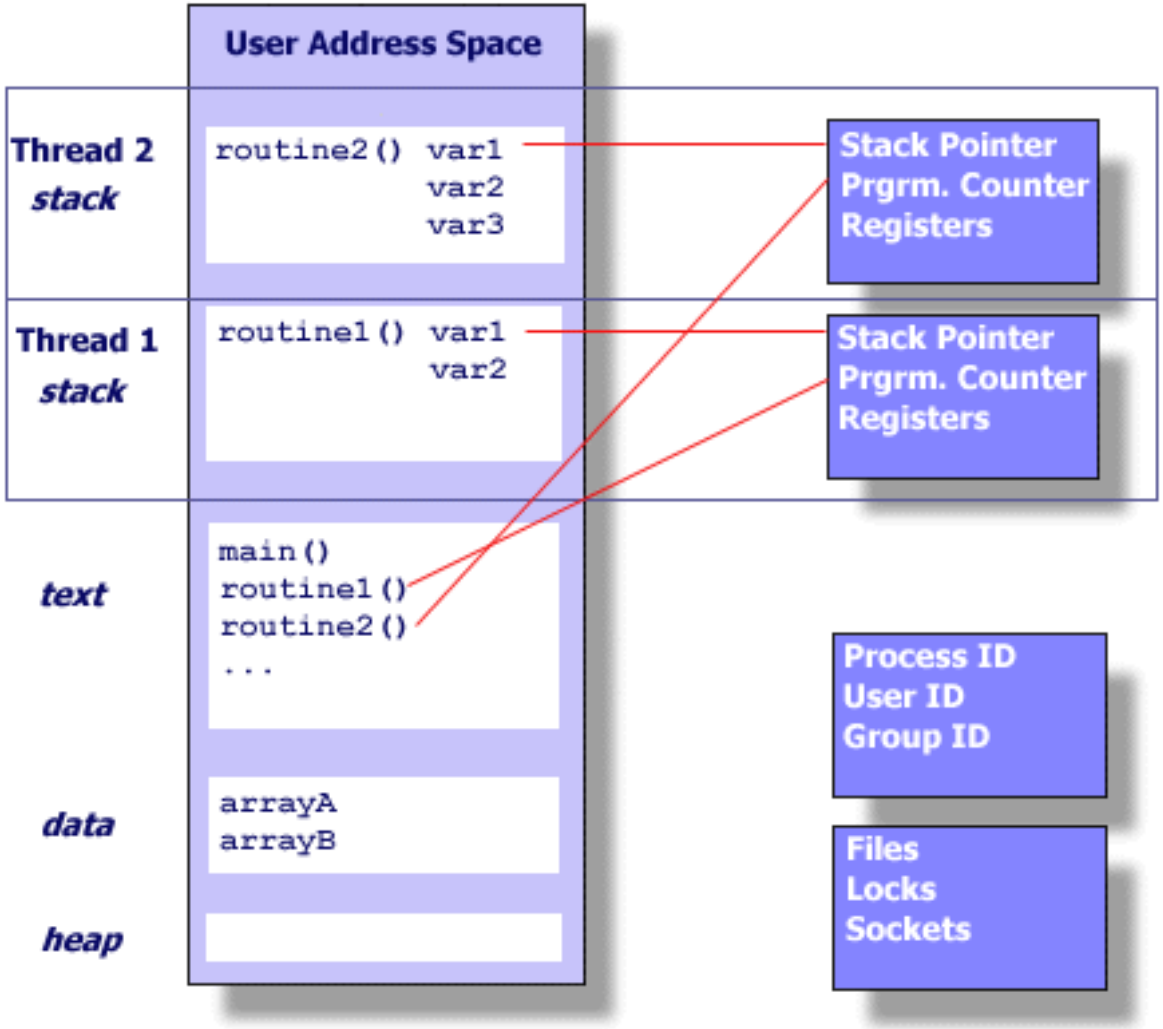
# Поток

Потоки существуют внутри процесса и используют его ресурсы.

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

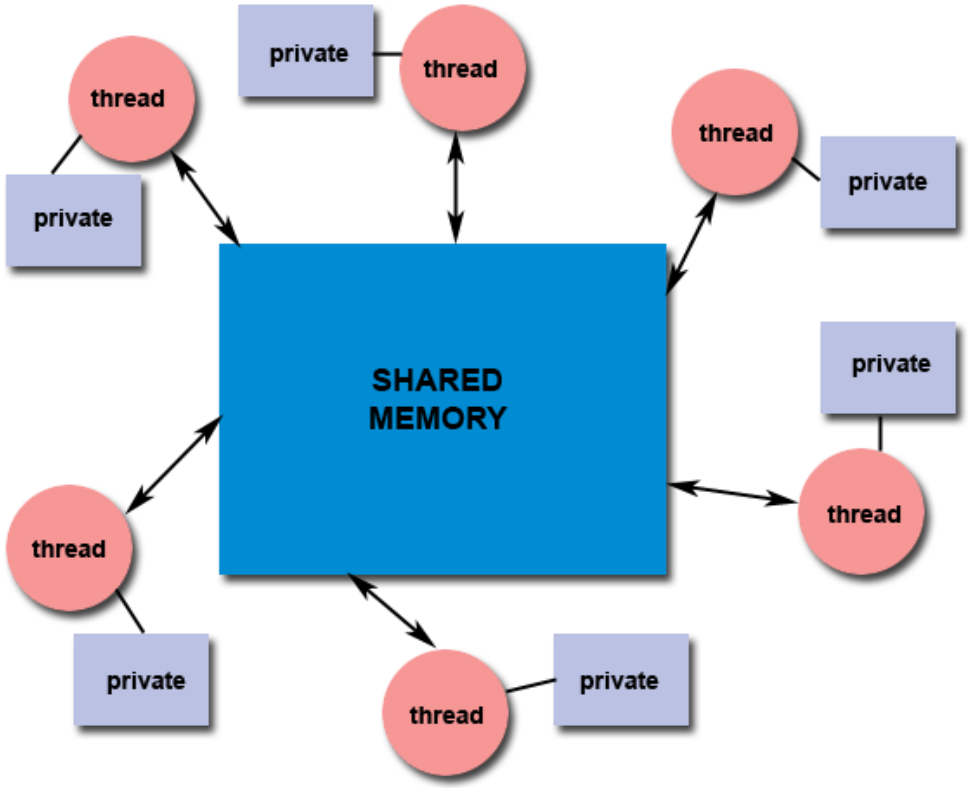
Поток не процесс!

# Thread


User Address Space
Thread 2
stack
routine2() var1
var2
var3
Thread 1
stack
routine1() var1
var2
text
main()
routine1()
routine2()
...
data
arrayA
arrayB
heap
Stack Pointer
Prgrm. Counter
Registers
Stack Pointer
Prgrm. Counter
Registers
Process ID
User ID
Group ID
Files
Locks
Sockets

# 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



# POSIX threads

- 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

# 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.

# 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

# 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

- 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()

- Методы прерывания потоков:
  - implicit termination:
    - Испольнение функции завершино
  - explicit termination:
    - вызван pthread_exit() внутри потока
    - вызван pthread_cancel() из основного потока

- Для функций, которые интенсивно используют CPU число потоков рекомендуется делать равным количеству доступных ядер CPUs

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

- на C++
  - 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).

```
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
```

```
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

- Ожидание завершения потока:

  ```
  int pthread_join( pthread_t thread, void **thread_return )
  ```

  - Основной поток дожидается завершения потока с идентификатором *thread*
  - Второй аргумент – значение возвращаемое потоком
  - returns 0 for success
  - Следует всегда дожидаться завершения потока

```
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++.

# 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, ...

# Программная модель

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

# 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

- 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

```
#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;
}
```

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

- 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

- 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

```
#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;
}
```

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

-

# Итоги

- 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.

# Литература

- *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