PThreads

Gabriele Oligeri

Roberto Resoli

Università degli Studi di Trento

Dipartimento di Ingegneria e Scienza dell'Informazione,

via Sommarive 14

I - 38050 Trento - Povo, Italy

Concurrent programming

- Threads, like processes, allow a program todo more than one thing at a time
- The linux kernel schedules processes and threads asynchronously, interrupting each of them from time to time to give others a chance to execute.

Threads

- After the invocation of a program, Linux kernel creates a new process and in that process creates a single thread.
- The program is run sequentially.

- The thread can create additional threads
  - The new threads run the same program in the same process
  - Each thread may be executing a different part of the program

Process and threads

- fork() generates a child process by copying the virtual memory, file descriptors, etc.
- The child process cannot interact with the memory of the parent (and vice-versa).
  - There is no shared memory
  - Process communication is achieved by means of PIPE/FIFO

- When a process creates a new thread **nothing is copied**.
- The creating and the (new) created thread share the same memory space, file descriptors, and other system resources.
- If a thread changes the value of a variable, closes a file descriptor then other threads share the result of the operation.

Pthreads

- POSIX Threads, usually referred to as Pthreads, is a POSIX standard for threads.

- The standard, POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995), defines an API for creating and manipulating threads.

- Each thread is identified by a thread ID, all the threads have the same PID (getpid()).

- The thread ID is refered in C by the variable type: **pthread_t**

Pthread execution

- Upon execution, each thread executes a thread function.
- The function contains the code that the thread should run.
- The life of the thread begins and ends with the execution of the function.
- The function accepts a **void*** parameter and returns a **void***.

Pthread creation

```
void* func(void* arg)
{return NULL;}

int main()
{
   pthread_t thread_id;
   pthread_create (&thread_id, NULL, &func, &arg);
}
```

- The argument **arg** is passed to the function **func** by means of the **pthread_create**.
- pthread_create returns immediately and the original thread continues the execution.

Pthread joining

pthread_join (thread_id, NULL);

- The main thread might wait for another one.
- pthread_join allows the main thread to wait for the thread with thread ID thread_id.
- The second argument is constituted by the thread return value (NULL in this case).

Pthread – race conditions

- What if multiple threads change the value of the same variable ?

```
...
pthread_create(..., &pth_func, …)
...
```

```
void *pth_func()
{
        read(var);
        ...
        write(var);
}
```

- Multiple threads access pth_func
- There is no control on concurrency

  Thread 1 reads var

  Thread 2 reads var

  Thread 3 reads var and write var

  Thread 1, 2 write var
- var value is random

Pthread semaphores

- There are different ways to implement semaphores, mutex, locks, etc.
- A popular one:

  pthread_mutex_t mutex;
- Protect the shared variable with:

  pthread_mutex_lock(&mutex);

  // Change var value

  pthread_mutex_unlock(&mutex);
- Clear the mutex:

  pthread_mutex_destroy(&mutex);