



CSE 333 – SECTION 7

Threads

Threads

- Sequential execution of a program.
- Contained within a process.
- Multiple threads can exist within the same process.
 - Every process starts with one thread of execution, can spawn more.
- Threads in a single process share one address space
 - Instructions (code)
 - Static (global) data
 - Dynamic (heap) data
 - Environment variables, open files, sockets, etc.

POSIX threads (Pthreads)

- The POSIX standard provides APIs for creating and manipulating threads.
- Part of the standard C/C++ libraries, declared in `pthread.h`

Core pthread functions

- `pthread_create(thread, attr, start_routine, arg)`
- `pthread_exit(status)`
- `pthread_join(thread, status)`
- `pthread_cancel (thread)`

pthread_create

```
#include <pthread.h>
int pthread_create( pthread_t *thread,
                  const pthread_attr_t *attr,
                  void *(*start_routine) (void *),
                  void *arg );
```

- `pthread_create` creates a new thread and calls `start_routine` with `arg` as its parameter.
- `pthread_create` arguments:
 - **thread:** A unique identifier for the new thread.
 - **attr:** An attribute object that may be used to set thread attributes. Use NULL for the default values.
 - **start_routine:** The C routine that the thread will execute once it is created.
 - **arg:** A single argument that may be passed to `start_routine`. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.
- Compile and link with `-lpthread`.

Terminating Threads

- There are several ways in which a thread may be terminated:
 - The thread returns normally from its starting routine; Its work is done.
 - The thread makes a call to the `pthread_exit` subroutine - whether its work is done or not.
 - The thread is canceled by another thread via the `pthread_cancel` routine.
 - The entire process is terminated due to making a call to either the `exec()` or `exit()`.
 - If `main()` finishes first, without calling `pthread_exit` explicitly itself.

pthread_exit

```
void pthread_exit(void *retval);
```

- Allows the user to terminate a thread and to specify an optional termination status parameter, *retval*.
- In functions that execute to completion normally, you can often dispense with calling pthread_exit().
- **Calling pthread_exit() from main():**
 - If main() finishes before the threads it spawned, and does not call pthread_exit() explicitly, all the threads it created will terminate.
 - To allow other threads to continue execution, the main thread should terminate by calling pthread_exit() rather than exit().

pthread_join

```
int pthread_join(pthread_t thread, void **retval);
```

- Synchronization between threads.
- `pthread_join` blocks the calling thread until the specified thread terminates and then the calling thread joins the terminated thread.
- Only threads that are created as joinable can be joined; a thread created as detached can never be joined. (Refer `pthread_create`)
- The target thread's termination return status can be obtained if it was specified in the target thread's call to `pthread_exit()`.

Demo: *pthread_demo.c*

Section exercise (not to be turned in)

- Create a program that spawns two or three different threads, each of which prints a numeric sequence.

Examples:

- First n odd numbers
 - First n factorials
 - First n primes
- Use `pthread_demo.c` for ideas, but the structure might not be the same.
 - Can you do something in the threads (maybe `sleep()`) so that different runs of the program don't always produce the same output?

Networking & Threads Exercise

- Implement a chat program in C++.
- Create two threads – Server and the Client.
- The Client thread reads from stdin, and writes anything the user types to the network.
- The Server thread reads from the network, and writes anything that it receives to stdout.
- Feel free to use any sample code from lectures or other exercises to implement the above functions.