POSIX Threads and Linux Processes in C

44-550: Operating Systems

Threads

- Threads may execute in parallel or concurrently
  - Parallel: at the same time
  - Concurrently: just looks like at the same time
- Threads share global memory and source code, but not local memory

Threads in C/C++ (POSIX)

- POSIX
    - Portable Operating System Interface
    - Set of standard OS interfaces based on the Unix operating system
- A POSIX thread is also known as a `pthread`
- To understand how to use pthreads, we must investigate *function pointers*

Function Pointers

- Every operation the program performs is stored in memory
- Functions are no exception
  - The code for a function exists at some place in memory
  - When function is called, appropriate values are pushed to a memory location (parameters), and the execution pointer moves to the place in memory
- C/C++ allows us to represent function pointers as types
  - Yes, this technically means you can embed functions in C structs and make "classes"
  - No this does not mean you should embed functions in C structs and make "classes"
    - Don't listen to Dr. Hoot!
  - Take the form: `return_type (* FP_NAME)(parameter list)`
- To use a pthread, you must provide a function of the form `void * threadFunc(void* threadArg)`

A POSIX Thread Function in C

```
void * myThread (void * threadArg)
{
    // do something here, perhaps cast threadArg
    // to a pointer of another type

    // Make the thread exit
    // Note: we're returning NULL as the result
    pthread_exit(NULL);
}
```

POSIX Thread functions

- `pthread` functions can be rather complex
  - Instead of using these notes as full documentation, I will summaraize the functions here
  - Additional information can be found:
    - Using Google/Bing/Yahoo/DuckDuckGo/Ask/...
    - `man` pages
- The important functions will be introduced, then a detailed example will be worked through.

pthread_t

- `pthread_t` is the pthread type
- Stores a unique identifier to a thread
  - Thread IDs are only guaranteed to be unique within a single process
  - Never pass pointers to threads between processes

pthread_create

```
void pthread_create(pthread_t * thread,
        const pthread_attr_t *attr,
        void * (*thread_routine) (void* args),
        void * args);
```

- Takes:
  1. Address of the thread identifier variable
  2. the thread attributes
  3. the name of the function the thread will execute
  4. a pointer to the arguments
- Creates the thread, stores the identifier in the first argument, starts the thread by running `start_routine`, and passes the args variable to the function
- Go Go Gadget `man pthread_create`

pthread_exit

```
void pthread_exit(void * return_val);
```

- Takes a pointer to the value(s) to return to the parent process (whatever calls the join function)
- Terminates the thread, and returns the value specified by `return_val` to whatever thread calls `pthread_join`

pthread_join

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

- Takes a thread id and a pointer to some return data
- Waits for the thread indicated by the `pthread_t` to terminate, and stores points the `retval` pointer at whatever value gets returned

POSIX Threading Examples

- `examples/pthreads/simple-ex.c`
- `examples/pthreads/userinput.c`
- `examples/pthreads/poor-bitonic.c`
- `examples/pthreads/smarter-bitonic.c`