Software Systems

Day 23 - POSIX Threads

UNIX Pipes

- The pipe is one of the core ideas of the UNIX philosophy.
- It's not a substitute for properly-written scripts, but it is a powerful tool for quick analysis.
- https://www.youtube.com/watch?v=bKzonnwoR2I

Today: POSIX Threads

- The pthread library is a way to implement threads, semaphores, and mutexes in C.
- Some things to be aware of:
  - Flags: sometimes you have to run GCC with `-pthread`, and sometimes with `-lpthread` (in CMake, link `pthread` to the target).
  - Sometimes, you need to manage your own memory.
  - The functions are meant to be generic, so expect a lot of casting and `void*`.
- We'll also talk about the longest joke in SoftSys.

POSIX Threads: Creating Threads

- Threads are represented with the `pthread_t` type (it's typically an unsigned integer under the hood).
- Once such a variable exists, the thread is actually created with `pthread_create(thread, attr, func, arg)`.
  - `thread`: address of the `pthread_t`
  - `attr`: options to set on the thread (usually can be left `NULL`).
  - `func`: the function to run (takes `void*` and returns `void*`).
  - `arg`: pointer to the argument passed to `func`.
- `pthread_create` returns 0 if successful.

POSIX Threads: Creating Threads

- Example code:

```
#include <pthread.h>

void* foo(void* arg) {
  puts("Alan!");
  return NULL;
}

int main(void) {
  pthread_t thread = 0;
  if(pthread_create(&thread, NULL, foo, NULL)) {
    error_and_exit("Couldn't create thread");
  }
}
```

POSIX Threads: Joining Threads

- Once a thread is created, it starts running.
- To wait for a thread to finish, it needs to be *joined*.
- Threads' return values can also be collected in joining.
- In pthread, join a thread with `pthread_join(thread, ret_val)`.
  - `thread`: `pthread_t` representing the thread.
  - `ret_val`: address of `void*` to copy return value into.
- If a thread doesn't return anything, you can join it into `NULL`.
- `pthread_join` also returns 0 if successful.

POSIX Threads: Joining Threads

- Example code:

```
void* foo(void* arg) {
  int arg_num = *(int*)arg;
  int* ret_val = malloc(sizeof(int));
  *ret_val = arg_num + 1;
  return (void*)ret_val;
}
...
void* ret_val;
if (pthread_join(foo_thread, &ret_val)) {
  error_and_exit("Couldn't join thread");
}
```

POSIX Threads: Creating and Joining Threads

- Exercise:
  - In the exercises folder, you will find counters.c.
  - Fill in the main function to properly create and join two threads with the arguments `'x'` and `'y'`.
  - Don't forget to cast appropriately.

POSIX Threads: Types

- Functions that run in POSIX threads always take a single `void*` as input and return a single `void*` as output.
- Because of this, you need to cast variables back and forth when using them.
- For returning things, you typically need to malloc (and then the caller needs to free).

POSIX Threads: Types

Thread 1

```
int copy_a = x; /* a1 */
x = copy_a + 1; /* a2 */
```

Thread 2

```
int copy_b = x; /* b1 */
x = copy_b + 1; /* b2 */
```

- Exercise:
  - In the exercises folder, you will find `congress.c`, which emulates the above behavior with more threads and more increment operations.
  - Fill in the counter function to properly increment the argument passed to it.
  - Then, build and run the function. What values do you get?

POSIX Threads: Semaphores

- In the last lesson on concurrency, we saw how we can use semaphores to make guarantees on concurrent operations.
- In C, you can use the type `sem_t`, in `semaphore.h`.
- Operations:
  - `sem_init(sem_ptr, share, val)` - initialize a semaphore
  - `sem_wait(sem_ptr)` - wait on (decrement) a semaphore
  - `sem_post(sem_ptr)` - signal/post (increment) a semaphore
  - `sem_destroy(sem_ptr)` - free/clean up a semaphore
- These can be passed to threads and used.

POSIX Threads: Semaphores

- Example code:

```
void* foo(void* arg) {
  sem_post((sem_t*)arg);
  pthread_exit(NULL);
}

int main(void) {
  pthread_t thread;
  sem_t sem;
  sem_init(&sem, 0, 0);
  pthread_init(&thread, NULL, foo, &sem);
  ...
  sem_destroy(&sem);
}
```

POSIX Threads: Semaphores

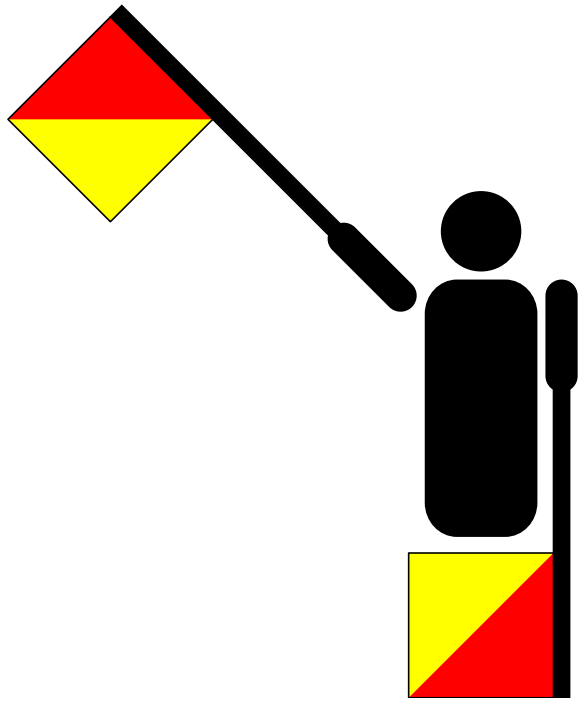
- Exercise:
  - In the exercise folder, you will find a file called dQw4w9WgXcQ.c.
  - Fill in the first, second, and main functions with appropriate semaphores and threading to make the lines print in the correct order.

Semaphores: The Longest Joke in SoftSys

- Now you've seen a bit of semaphores in C.
- So why is that the longest joke?

Semaphores: The Longest Joke in SoftSys

- This is the class Discord logo.
- Now you've seen C in semaphore.



POSIX Threads: Mutexes

- You can use semaphores to implement mutexes.
- But pthread also has `pthread_mutex_t`, which is easier to use.
- Operations:
  - `pthread_mutex_init(mutex_ptr, attr)`
  - `pthread_mutex_lock(mutex_ptr)`
  - `pthread_mutex_trylock(mutex_ptr)` - lock but don't block
  - `pthread_mutex_unlock(mutex_ptr)`
  - `pthread_mutex_destroy(mutex_ptr)`

POSIX Threads: Mutexes

- Exercise:
  - In the exercises folder, you will find progress.c.
  - Use a mutex in the thread and main function to make sure that the total count at the end is as expected.