Today

- Assignment 2
- Threads
Assignment 2

• Due on Oct 28th, 11:55PM
Threads

• A thread is a lightweight process
• Multiple threads can run concurrently within the same process
Processes vs. threads
Thread resources

• Each thread has its own:
  ○ thread ID
  ○ Stack, registers, program counter, signal mask
  ○ errno

• Threads share
  ○ Process code, data, heap
  ○ Files, signals

• Threads within the same process can communicate using shared memory - *Must be done carefully!*
Pthreads

- POSIX threads
- Most widely supported threading library

- Program needs to be linked with “-lpthread”
- For example, in the makefile

  CC = gcc
  LDFLAGS = -lpthread
Thread management APIs

- Create a thread: `pthread_create()`
- Join a thread: `pthread_join()`
- Detach a thread: `pthread_detach()`
- Terminate a thread: `pthread_cancel()`
pthread_create()

- Creating a thread is like a combination of fork() and exec()
- `#include <pthread.h>`
  ```c
  int pthread_create(
      pthread_t *thread, pthread_attr_t *attr,
      void *(*function)(void *),
      void *arg);
  ```
- `tid` is the thread ID, `attr` is an attribute set, `function` is the function to be called with `arg` as the argument
Thread IDs

- Each thread has a unique ID.
- A thread can find out its ID by calling `pthread_self()`.

- Thread IDs are of type `pthread_t` which is usually an unsigned int. When debugging, it's often useful to do something like this:

```c
printf("Thread ID: %zu\n", pthread_self());
```
Thread attributes

• Attributes are a way to modify thread behaviour
  ○ Detached/joinable state
  ○ Scheduling policy
  ○ Scheduling parameters
  ○ and many more..

• You can specify NULL and get the system defaults
Thread start routine

• The new thread starts execution by invoking function()

• Both the type of return value and parameter of function() must be void *
  void * function(void *arg)

• When creating a thread, the starting address of function() is passed to pthread_create()
Thread arguments

• When function() is called, the value arg specified in the call to pthread_create() is passed as a parameter to the function.

• A function can have only 1 parameter, and it can't be larger than the size of a void *.

• Complex parameters can be passed by creating a structure and passing the address of the structure.
## pthread_join()

```c
#include <pthread.h>

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

- Makes the calling thread wait for the thread specified by `tid` to terminate (similar to `waitpid()`)
- Thread specified by `tid` must be **joinable**. If that thread has already exited, `pthread_join()` will return immediately
- If `retval` is not NULL, `pthread_join()` copies the exit status of the thread into the location pointed to by `retval`
- `pthread_join()` returns 0 on success, or an error number on failure
#include <pthread.h>

int pthread_detach(pthread_t tid)

- Marks the thread identified by tid as detached
- When the detached thread terminates, its resources are automatically released back to the system
- A detached thread cannot be joined
- pthread_detach() returns 0 on success or an error number on failure
#include <pthread.h>

int pthread_cancel(pthread_t tid);

- `pthread_cancel()` sends a cancellation request to the thread identified by `tid`
- The target thread will exit at a later point
- `pthread_cancel()` returns 0 on success or an error number on failure
Thread lifespan

• Once a thread is created, it starts executing the function `function()` specified in the call to `pthread_create()`

• If `function()` returns, the thread is terminated

• A thread can also be terminated by calling `pthread_exit()`
Exercise

• Create 2 threads, passing a different argument to each thread ("file" and "dirs")

• If the argument is "file", the thread must get a list of all files in the current working directory and write it to a file called "files.txt"

• If the argument is "dirs", the thread must get a list of all directories in the current working directory and write it to a file called "dirs.txt"
Questions?