Today

- Assignment 3
- Threads
Assignment 3

• Due on Mar 28
What is a Thread?

• A thread is a single sequential flow of control within a program.
Fork()
`pthread_create()`
Thread-Specific Resources

• Each thread has its own:
  – Thread ID (integer)
  – Stack, registers, program counter
  – Errno
  – arithmetic/buffer overflow

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

• We will focus on Posix Threads– most widely Supported threads programming API

• Solaris - you need to link with “-lpthread”
  – For example, in the makefile we supplied
  – CC = gcc
  – LDFLAGS = -lpthread -lrt -lm
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);
    ```
  - Thread is the thread ID, attr is an attribute set, function is the function to be called with arg
  - Compile/Link with -D_REENTRANT -lpthread
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 %u:\n",pthread_self());
```
• Thread attributes can be set using `attr`, including detached state and scheduling policy. You can specify `NULL` and get the system defaults.
Thread Start Routine

• The new thread starts execution by invoking `function()`

• The type of return value and parameter of `function()` must be `void *`

• When creating a process, 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

• 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
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()`
Detached state

• Each thread can be either joinable or detached

• **Detached:** on termination all thread resources are released by the OS. A detached thread cannot be joined
Detaching threads

- `#include <pthread.h>`
  
  `int pthread_detach (pthread_t thread);`

- A detached thread cannot be joined – it will just go away when it exits

- You cannot detach a thread that some other thread is joining it
Joinable Thread

- Joinable: on thread termination the thread ID and exit status are saved by the OS

- One thread can "join" another by calling `pthread_join` - which waits (blocks) until a specified thread exits
Joining threads

- Joining a thread is analogous to waiting for a child process.
- `#include <pthread.h>`
  ```c
  int pthread_join (pthread_t th, void **thread_return);
  ```
- `thread_return` is the exit value of the thread.
Threads Cancellation

- Cancel a thread when it is a good time to “stop”
  - Done from the “outside”, e.g. parent
  - make a cancellation request

```
#include <pthread.h>

void pthread_cancel(pthread_t thread);
```
Exercise

• Goal:
  Implement your own threads program, where two threads each open a different text file and interleave the output when printing the files to standard output:
  
  file1: line1 text here
  file2: line1 text here
  file1: line2 text here
  file2: line 2 text here
Questions?