CS 3305
Intro to Threads
Lecture 6
Introduction

- Multiple applications run concurrently!

- This means that there are multiple processes running on a computer
Introduction

- Applications often need to perform many tasks at once

- This requires multiple threads of execution within a single process
Example: Word processor

Tasks include:
- Display graphics
- Respond to keystrokes from the user
- Perform spelling and grammar checking
Example

- Example: Web server
  - It is desirable to service requests concurrently

```plaintext
(1) request

client ─── server

(2) create new thread to service the request

thread

(3) resume listening for additional client requests
```
Introduction

- Earlier we discussed the use of forking to create a process
- For example we could
  - Word processor example: fork a process for each task
  - Web server example: fork a process for each request
- Not very efficient since a fork copies everything
Threads

- A thread is a basic unit of CPU utilization

- Threads of a process share memory but can execute independently
Thread Usage - Word Processor

A word processor program with three threads.
Thread Usage - Web Server

Web/FTP server

Process Request Client 2

Process Request Client N
Why Not Fork?

- You certainly can fork a new process
- In fact, the first implementation of Apache web servers (Apache 1.0) forked $N$ processes when the web server was started
  - “$N$” was defined in a configuration file
  - Each child process handled one connection at a time
- **Problem**: Process creation is time consuming and resource intensive
- Creating threads is not as expensive
Why Not Fork?

- Let’s look at web servers
  - Web servers have caches for read pages
  - Forking means that these caches cannot be shared
  - Using threads allows for these caches to be shared
Thread State

- Threads share
  - Code
  - Data (global variables)
  - Open files, sockets

- Threads have their own CPU context
  - Program counter (PC), Stack pointer (SP), register state
Single and Multithreaded Processes

- Single-threaded process
  - Code
  - Data
  - Files
  - Registers
  - Stack

- Multithreaded process
  - Code
  - Data
  - Files
  - Registers
  - Registers
  - Registers
  - Stack
  - Stack
  - Stack
Benefits of Threads

- **Resource Sharing**
  - Example: Word processor
    - The document is shared in memory
    - Forking would require replication

- Allocating memory and resources for process creation is costly

- **Responsiveness**
  - No replication - saves time
  - Context-switching (CPU switching from one process to another) is faster
Thread Libraries

- A thread library provides the programmer with an API for creating and managing threads.
- Three main libraries in use:
  - POSIX PThreads
    - User or kernel level
  - Win32
    - Kernel level library
    - Used in Windows OS
  - Java threads
    - JVM is running on top of a host OS
    - The Java thread API is implemented using a thread library available on the system
Pthreads: POSIX Threads

- **Pthreads** is a standard set of C library functions for multithreaded programming

- Pthread Library (60+ functions)

- Programs must include the file `pthread.h`
Thread Creation

- Thread identifiers
  - Each thread has a unique identifier (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.
pthread_create()

- Creates a new thread

```c
int pthread_create ( 
    pthread_t *thread, 
    pthread_attr_t *attr, 
    void * (*start_routine) (void *), 
    void *arg);
```

- Returns 0 to indicate success, otherwise returns error code
- **thread**: id of the new thread
- **attr**: argument that specifies the attributes of the thread to be created (NULL = default attributes)
- **start_routine**: function to use as the start of the new thread
- **arg**: argument to pass to the new thread routine
  - If the thread routine requires multiple arguments, they must be passed bundled up in an array or a structure
Let us say that you want to create a thread that simply prints “hello world...I am a thread”

```c
int main(int argc, char *argv) {

pthread_t worker_thread;

if (pthread_create(&worker_thread, NULL,
                   do_work) {
    fprintf(stderr,"Error while creating thread\n");
    exit(1);
}
...}

void *do_work() {

Printf ("\n hello world..I am a thread");

    return NULL;
}
```
Problem

- Sharing global variables is dangerous - two threads may attempt to modify the same variable at the same time.
- Use support for mutual exclusion primitives that can be used to protect against this problem.
- The general idea is to lock something before accessing global variables and to unlock as soon as you are done.
- More on this topic later in the course