CS 537
Lecture 19
Threads and Cooperation

Michael Swift

Questions answered in this lecture:

- Why are threads useful?
- How does one use POSIX pthreads?

What's in a process?

- A process consists of (at least):
  - User ID
  - state flags
  - an address space
  - the code for the running program
  - the data for the running program
  - an execution stack and stack pointer (SP)
    - traces state of procedure calls made
  - the program counter (PC), indicating the next instruction
  - a set of general-purpose processor registers and their values
  - a set of OS resources
    - open files, network connections, sound channels, ...
- That's a lot of concepts bundled together!

Organizing a Process

- Scheduling / execution
  - state flags
  - an execution stack and stack pointer (SP)
  - the program counter (PC), indicating the next instruction
  - a set of general-purpose processor registers and their values
- Resource ownership / naming
  - user ID
  - an address space
  - the code for the running program
  - the data for the running program
  - a set of OS resources
    - open files, network connections, sound channels, ...

Concurrency

- Imagine a web server, which might like to handle multiple requests concurrently
  - While waiting for the credit card server to approve a purchase for one client, it could be retrieving the data requested by another client from disk, and assembling the response for a third client from cached information
- Imagine a web client (browser), which might like to initiate multiple requests concurrently
  - The CS home page has 66 "src= ..." html commands, each of which is going to involve a lot of string around! Wouldn't it be nice to be able to launch these requests concurrently?
- Imagine a parallel program running on a multiprocessor, which might like to concurrently employ multiple processors
  - For example, multiplying a large matrix – split the output matrix into k regions and compute the entries in each region concurrently using k processors

- Image a program with two independent tasks: saving (or printing) data and editing text

Why support Threads?

- Divide large task across several cooperative threads
- Multi-threaded task has many performance benefits
  - Adapt to slow devices
    One thread waits for device while other threads compute
  - Defer work
    One thread performs non-critical work in the background, when idle
  - Parallelism
    Each thread runs simultaneously on a multiprocessor
  - Modularity
    Independent tasks can be untangled

Common Programming Models

- Multi-threaded programs tend to be structured in one of three common models:
  - Manager/worker
    Single manager handles input and assigns work to the worker threads
  - Producer/consumer
    Multiple producer threads create data (or work) that is handled by one of the multiple consumer threads
  - Pipeline
    Task is divided into series of subtasks, each of which is handled in series by a different thread

What's needed?

- In each of these examples of concurrency (web server, web client, parallel program):
  - Everybody wants to run the same code
  - Everybody wants to access the same data
  - Everybody has the same privileges
  - Everybody uses the same resources (open files, network connections, etc.)
- But you'd like to have multiple hardware execution states:
  - an execution stack and stack pointer (SP)
    - traces state of procedure calls made
  - the program counter (PC), indicating the next instruction
  - a set of general-purpose processor registers and their values

How could we achieve this?

- Given the process abstraction as we know it:
  - fork several processes
  - cause each to map to the same address space to share data
    - see the `shmget()` system call for one way to do this (kind of)
- This is like making a pig fly – it's really inefficient
  - space: PCB, page tables, etc.
  - time: creating OS structures, fork and copy addr space, etc.
- Some equally bad alternatives for some of the cases:
  - Entirely separate web servers
  - Asynchronous programming in the web client (browser)

Can we do better?

- Key idea:
  - separate the concept of a process (address space, etc.)
  - from that of a minimal "thread of control" (execution state: PC, etc.)
- This execution state is usually called a thread, or sometimes, a lightweight process

Threads and processes

- Most modern OS's (Mach, Chorus, Windows XP, modern Unix (not Linux)) therefore support two entities:
  - the process, which defines the address space and general process attributes (such as open files, etc.)
  - the thread, which defines a sequential execution stream within a process
- A thread is bound to a single process
  - processes, however, can have multiple threads executing within them
  - sharing data between threads is cheap: all see same address space
- Threads become the unit of scheduling
  - processes are just containers in which threads execute

(old) Process address space