CSci 4061
Introduction to Operating Systems
(Thread-Basics)
Threads

• Abstraction: for an executing instruction stream
• Threads exist within a process and share its resources (i.e. memory)
• But, thread has its own stack and “PC”
• Default: always 1 thread (implicit)
Two Threads Sharing a CPU

What may cause a switch?
Why Threads?
Threads Example

A word processor with three threads

get-input

display buffer

auto-save

Keyboard

Disk

Kernel
Thread example: Web server...
Thread Example: Web server

dispatcher (...) {
  while (TRUE){
    get_next_request (&reg);
    handoff_work (&reg, &buf);
  }

  Req= url
}

worker (...) {
  wait_for_work (&buf, &reg)
  look_for_page_in_cache (&reg, &answer);
  if (page_not_in_cache (&answers) {
    read_page_from_disk (&reg, &answer);
  }
  return (&answer);

• How are these threads interacting?
  • Shared memory: threads share buffer, cache
  • Threads share globals, heap, NOT stack
Looks great

• Drawbacks?

• Alternatives?
Drawbacks: Thread Safety

```java
int counter = 0;
int increment_counter() {
    counter ++;  // counter = counter +1
    return counter;
}
```

problem?

To be thread-safe the shared variable `counter` needs to be protected by a lock
Thread Safety (cont’d)

```c
int counter = 0
lock_type counter_lock;
int increment_counter (){

    lock (counter_lock);
    counter ++;
    unlock (counter_lock);
    return counter;
}

Unix man pages will tell you if a syscall is thread-safe...or not
```
Drawbacks: per thread globals

T1
...
syscall
sets erno
...
reads erno

T2
...
syscall
sets erno
...

• In Unix, *erno* is global variable in shared library
• Options to guarantee error reporting is thread-safe
Alternatives to Threads

• Want concurrency
  • If a program (or part) cannot make progress
  • Some other part can
General Thread Options

- **Create/Fork**
  - Allocate memory for stack, perform bookkeeping
  - Parent thread creates child threads
  - Returns an id

- **Destroy/Cancel**
  - release memory (or recycle), perform bookkeeping

- **Suspend, Resume/Yield, Sleep**

- **Wait**
  - Wait for something, e.g. child finishing
Inside Threads

• A thread contains
  • pc
  • sp
  • registers
  • child threads
  • state

• What about open files?
Threads in Action

1. main
   --
   --
   creates T, T2
   --
   wait (...) 

2. T1
   T1_proc {
     read
   }

3. T2
   T2_proc {
   }

Main thread starts execution ... creates T1 and T2
Eventually main blocks or yields and scheduler picks one of the other threads, say, T1
When T1 blocks (as in read), scheduler picks another thread (e.g. T2) to run
Implementing Threads in User Space

A user-level threads package
Implementing Threads in the Kernel

A threads package managed by the kernel
User vs. Kernel Threads

• User thread advantages
  • no thread system calls! — cheaper
  • more scalable
  • more portable
  • custom control and scheduling
  • blocking is a big problem!
Kernel Threads

• Advantages
  • thread can block: OS can pick another from same process
  • can exploit multiprocessors
Thread Models

• Dispatcher-worker (master-slave)
  • a master process/thread receives request for work
  • generates/dispatches a thread to service work request
  • e.g. threaded server
  • server can create a thread on a “as needed basis” a pop-up thread
  • maybe cheaper to start a new thread than to restore an old one

• Thread Pools
  • keep a pool of pre-created threads around
  • may reduce perceived latency of creating threads to service request
Thread Models (cont’d)

• Team
  • a collection of peer threads working on some part of a problem
  • parallel program running on a shared-memory
  • N threads are created and each are given a piece of the problem or
  • threads request a task from a shared task queue

  e.g. scale element of a matrix

Or the editor example
Next Time

• Thread programming in Posix!