Lecture 6: Chapter 4 - Threads
Today's class

• Will introduce threading APIs and internals

• Learning objectives
  – To have an understanding of how threads are implemented
    – To have a general understanding of the APIs
    – To familiarize with the POSIX thread API
Benefits of multithreading vs...

- Single-threaded
  - Responsiveness
    - In single-threaded, if there is a blocking I/O, then all of the process is blocked
  - Concurrency
  - Other...?

- Multi-process
  - Sharing name space of variables (ease of programming)
  - Economy
    - Sharing same memory space (memory savings)
    - Don’t have to allocate space multiple times
  - Scalability through economy
  - Other...?
Multithreading implementation models
User threads vs kernel threads

- **User threads**
  - Threads that run at **user space**
  - They are the visible threads to a programmer
  - To exploit parallelism/concurrency by users

- **Kernel threads**
  - Threads spawned by the kernel that run in **kernel space**
  - To exploit parallelism/concurrency at kernel level
  - Not visible to user processes
Many-to-one (M:1)

- All threads of a process are mapped to a single thread in the kernel
- If one thread blocks, all “tethered” threads block
- Implementations:
  - GNU threads
  - Green threads in Solaris
One-to-one (1:1)

- For each thread in user space, there is a corresponding thread in kernel space
  - Implementations
    - Linux
    - Windows
Many-to-many (M:M)

- M user threads map to “N” kernel threads.
  - Allows to limit the kernel threads
- How is the user-to-kernel thread mapping done?
  - Static mapping?
    - At its creation, a thread is statically assigned to a kernel thread
  - Dynamic mapping?
    - On each context switch, a thread may assigned to a different thread
- Implementations
- Solaris versions <
Thread Libraries
POSIX Threads (Pthreads)

- pthread_create()
- pthread_attr_init()
- pthread_exit()
- pthread_join()
- pthread_cancel()
Win32 Threads
Multi-threading Issues
Thread safety (thread-safe functions)

• Thread-safe definition
  - A function that may be safely invoked concurrently by multiple threads. Each function defined in the System Interfaces volume of POSIX.1-2008 is thread-safe unless explicitly stated otherwise. Examples are any “pure” function, a function which holds a mutex locked while it is accessing static storage, or objects shared among threads. [POSIX Std 1003.1-2008]

• How to ensure thread safety of “non-pure” functions
  - Use mutual exclusion mechanisms
  - Properly divide work to achieve separation of data
  - other...?
fork() and exec() behavior

- What happens when fork() is called?
  - Are the threads mirrored? Are the re-initialized?

Solutions
- Ad-hoc

- What happens when exec() is called?
  - What if other threads are “in the middle of doing something” (which is more than often the case)?
Thread cancellation

- **Cancellation**
  - Terminating a thread from outside the thread
  - Analogous to “process killing”

- **Asynchronous Cancellation**
  - Thread is immediately canceled irrespective of its state

- **Deferred Cancellation**
  - Thread is canceled only at points in the execution path that are safe to perform the cancellation
  - Cancellation points
Handling process signals

• Issue
  – What thread is accounted/charged for the signal handler when a signal is being delivered to a process?

• Options
  – Deliver the signal to “the obviously related” thread
  – Deliver the signal to every thread or many threads
  – Deliver the signal to a specific thread that is assigned to receive signals
Next class

- More on thread implementation and issues
- Thread behavior examples