Announcements

• Project 1 is due Feb 27, 11:59pm
  – Not a whole lot of time, find a team now.
• *nix Crash Course offered: Feb 9, 8:30pm
• Project information sessions next week
  – Time: TBA
• Reading: Section 5.1 through 5.4

Overview for today

• Finish
  – Process States
• Process/Thread API Examples
• Fork/join model

Windows XP

• Thread state diagram in a industrial kernel

• Source: Dave Probert, Windows Internals – Copyright Microsoft 2003

Process States w/ Suspend

• Can be useful sometimes to suspend processes
  – By user request: "Z in Linux shell/job control
  – By OS decision: swapping out entire processes
    (Solaris does that, Linux & Windows don’t)

Process Creation

• Two common paradigms:
  – Cloning vs. spawning
• Cloning: (Unix)
  – “fork()” clones current process
  – child process then loads new program
• Spawning: (Windows, Pintos)
  – “exec()” spawns a new process with new program
• Difference is whether creation of new process also involves a change in program
Fork/Exec Model

- **Fork():**
  - Clone most state of parent, including memory
  - Inherit some state, e.g. file descriptors
  - Important optimization: copy-on-write
    - Some state is copied lazily
- **Exec():**
  - Overlays current process with new executable
  - Keeps process, changes program
- **Advantage:** simple, clean
- **Disadvantage:** does not optimize common case (fork followed by exec of child)

The fork()/join() paradigm

- After fork(), parent & child execute in parallel
- **Purpose:**
  - Launch activity that can be done in parallel & wait for its completion
  - Or simply: launch another program and wait for its completion (shell does that)
- **PinToS:**
  - Kernel threads: thread_create (no thread_join)
  - exec(), you’ll do wait() in Project 2

CreateProcess()

- See also system(3) on Unix systems
- PinToS exec() is like system()

Thread Creation APIs

- **How are threads embedded in the language?**
  - Java/C#
    - Thread start(), Thread join()
  - C#: Uses delegate
  - POSIX Threads Standard (in C)
    - pthread_create(), pthread_join()
    - Uses function pointer
  - **C++**
    - No standard as of yet
    - see [ISO C++ Strategic Plan for Multithreading](#)

Example pthread_create/join

```c
static void * test_single(void *arg)
{
    if (this function is executed by each thread, in parallel)
    {
        // Test the memory allocator with NTHREADS threads
        pthread_t threads[NTHREADS];
        int i;
        for (i = 0; i < NTHREADS; ++i)
        {
            if (pthread_create(threads + i, (const pthread_attr_t*)NULL, test_single, (void*)i) == -1)
            { printf("error creating thread
            "); exit(-1); }
        }
    }
    // Use Default Attributes -- could set stack address here
    for (i = 0; i < NTHREADS; ++i)
    {
        pthread_join(threads[i], NULL);
        // 2nd arg could receive exit status of thread
    }
}
```
Java Threads Example

```java
public class JavaThreads {
    public static void main(String[] av) throws Exception {
        Thread[] t = new Thread[5];
        for (int i = 0; i < t.length; i++) {
            final int tnum = i;
            Runnable runnable = new Runnable() {
                public void run() {
                    System.out.println("Thread "+tnum);
                }
            };
            t[i] = new Thread(runnable);
            t[i].start();
        }
        for (int i = 0; i < t.length; i++)
            t[i].join();
        System.out.println("all done");
    }
}
```

Threads implements Runnable – could have subclassed Thread & overridden run()

Why is taking C++ so long?

- Java didn’t – and got it wrong.
  - Took years to fix
- What’s the problem?
  - Compiler must know about concurrency to not reorder operations past implicit synchronization points
  - See also Pintos Tour 2.2.5 Memory Barriers

```c++
lock (&l);
flag = true;
unlock (&l);
lock (&l);
```

Summary

- Process States
- Process creation APIs
- Thread creation APIs
- Fork/join model