Lecture 2:
Processes and Threads - Part 1

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com
What Is a Process?

An abstraction of a running program
The Process Model

• A process is an instance of an executing program and includes
  – Program counter
  – Registers
  – Variables
  – ...

• A process has a program, input, output, and state.
The Process Model

• A process is an instance of an executing program and includes
  – Program counter
  – Registers
  – Variables
  – …

• A process has a program, input, output, and state.

If a program is running twice, does it count as two processes? or one?
Multiprogramming

- One CPU and several processes
- CPU switches from process to process quickly
What Really Happens  What We Think It Happens
If we run the same program several times, will we get the same execution time?
Process

Termination

Creation

Implementation

State
Process Creation

• System initialization
  – At boot time
  – Foreground
  – Background (daemons)

• Execution of a process creation system call by a running process

• A user request

• A batch job
Process Termination

- Normal exit (voluntary)
- Error exit (voluntary)
- Fatal error (involuntary)
- Killed by another process (involuntary)
Process States

1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available
Process States

• Process scheduler is part of the operating system
• Scheduling: deciding which process should run and for how long

1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available
Implementation of Processes

- OS maintains a process table
- An array of structures
- One entry per process

<table>
<thead>
<tr>
<th>Process management</th>
<th>Memory management</th>
<th>File management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>Pointer to text segment info</td>
<td>Root directory</td>
</tr>
<tr>
<td>Program counter</td>
<td>Pointer to data segment info</td>
<td>Working directory</td>
</tr>
<tr>
<td>Program status word</td>
<td>Pointer to stack segment info</td>
<td>File descriptors</td>
</tr>
<tr>
<td>Stack pointer</td>
<td></td>
<td>User ID</td>
</tr>
<tr>
<td>Process state</td>
<td></td>
<td>Group ID</td>
</tr>
<tr>
<td>Priority</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheduling parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parent process</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process group</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signals</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time when process started</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU time used</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Children’s CPU time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time of next alarm</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Bit About Interrupts

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly language procedure saves registers.
4. Assembly language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler decides which process is to run next.
7. C procedure returns to the assembly code.
8. Assembly language procedure starts up new current process.
Simple Modeling of Multiprogramming

- A process spends fraction $p$ waiting for I/O
- Assume $n$ processors in memory at once
- The probability that all processes are waiting for I/O at once is $p^n$
- So -> CPU Utilization = $1 - p^n$
Multiprogramming lets processes use the CPU when it would otherwise become idle.
Threads

• Multiple threads of control within a process
• All threads of a process share the same address space
Why Threads?

• For some applications many activities can happen at once
  – With threads, programming becomes easier
  – Benefit applications with I/O and processing that can overlap

• Lighter weight than processes
  – Faster to create and restore
Example 1: A Word Processor
Example 2: Multithreaded Web Server
Processes vs Threads

- Process groups resources
- Threads are entities scheduled for execution on CPU
- No protections among threads (unlike processes) [Why?]
- Thread can be in any of several states: running, blocked, ready, and terminated
<table>
<thead>
<tr>
<th>Per process items</th>
<th>Per thread items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address space</td>
<td>Program counter</td>
</tr>
<tr>
<td>Global variables</td>
<td>Registers</td>
</tr>
<tr>
<td>Open files</td>
<td>Stack</td>
</tr>
<tr>
<td>Child processes</td>
<td>State</td>
</tr>
<tr>
<td>Pending alarms</td>
<td></td>
</tr>
<tr>
<td>Signals and signal handlers</td>
<td></td>
</tr>
<tr>
<td>Accounting information</td>
<td></td>
</tr>
</tbody>
</table>
Each thread has its own stack (Why?).
Where to Put The Thread Package?

User space                       Kernel space
Implementing Threads in User Space

- Threads are implemented by a library
- Kernel knows nothing about threads
- Each process needs its own private thread table
- Thread table is managed by the runtime system
Implementing Threads in User Space

Advantages
- Very fast thread scheduling
- Each process can have its own thread scheduling algorithm
- Scale better

Disadvantages
- Blocking system calls can block the whole process
- Page fault blocks the whole process
- No other thread of the process will ever run unless the running thread voluntarily gives up the CPU
Implementing Threads in Kernel Space

- Kernel knows about and manages the threads
- No runtime is needed in each process
- Creating/destroying/(other thread related operations) a thread involves a system call
Implementing Threads in Kernel Space

**Advantages**
- When a thread blocks (due to page fault or blocking system calls) the OS can execute another thread from the same process.

**Disadvantages**
- Cost of system call is very high.
Hybrid Implementation

Multiple user threads on a kernel thread

Kernel

Kernel thread

User space

Kernel space
Conclusions

• We covered today Chp 2 till end of section 2.2.
• Skim 2.2.7, 2.2.8, and 2.2.9
• Processes is the most central concept in OS
• Process vs Thread
• Multiprogramming vs multithreading