Processes & Threads

CS 475

Concurrent Programs

- Process = Address space + one thread of control
- Concurrent program = multiple threads of control
  - Multiple single-threaded processes
  - Multi-threaded process
Concurrent Systems

- Essential aspects of any concurrent system
  - Execution context - state of a concurrent entity
    - Processes: process context
    - Threads: thread context
  - Scheduling - deciding which context will run next
    - Processes: Operating System scheduler
    - Threads: Library thread scheduler (Pthreads), Java runtime
  - Synchronization - mechanisms that enable execution contexts to coordinate their use of shared resources
    - Semaphores, locks, monitors, condition variables
    - Provided at both operating system and library/language level

Processes

- Def: A process is an instance of a running program.
  - One of the most profound ideas in computer science.
  - Not the same as "program" or "processor"
- Process provides each program with two key abstractions:
  - Logical control flow
    - Each program seems to have exclusive use of the CPU.
  - Private address space
    - Each program seems to have exclusive use of main memory.
- How are these illusions maintained?
  - Process executions interleaved (multitasking)
  - Address spaces managed by virtual memory system
Traditional View of a Process

- Process = process context + code, data, and stack

<table>
<thead>
<tr>
<th>Program context:</th>
<th>Code, data, and stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data registers</td>
<td>stack</td>
</tr>
<tr>
<td>Condition codes</td>
<td>shared libraries</td>
</tr>
<tr>
<td>Stack pointer (SP)</td>
<td>run-time heap</td>
</tr>
<tr>
<td>Program counter (PC)</td>
<td>read/write data</td>
</tr>
<tr>
<td>Kernel context:</td>
<td>read-only code/data</td>
</tr>
<tr>
<td>VM structures</td>
<td></td>
</tr>
<tr>
<td>Descriptor table</td>
<td></td>
</tr>
<tr>
<td>brk pointer</td>
<td></td>
</tr>
</tbody>
</table>

Threads: Motivation

- Traditional processes created and managed by the OS kernel
- Process creation expensive - fork system call in UNIX
- Context switching expensive
- Cooperating processes - no need for memory protection (separate address spaces)
**Alternate View of a Process**

- Process = thread + code, data, and kernel context

**Thread (main thread)**
- Stack
- Thread context:
  - Data registers
  - Condition codes
  - Stack pointer (SP)
  - Program counter (PC)

**Code and Data**
- Shared libraries
- Run-time heap
- Read/write data
- Read-only code/data

**Kernel context:**
- VM structures
- Descriptor table
- Brk pointer

---

**A Process With Multiple Threads**

- Multiple threads can be associated with a process
  - Each thread has its own logical control flow (sequence of PC values)
  - Each thread shares the same code, data, and kernel context
  - Each thread has its own thread id (TID)

**Thread 1 (main thread)**
- Stack 1
- Thread 1 context:
  - Data registers
  - Condition codes
  - SP1
  - PC1

**Shared code and data**
- Shared libraries
- Run-time heap
- Read/write data
- Read-only code/data

**Thread 2 (peer thread)**
- Stack 2
- Thread 2 context:
  - Data registers
  - Condition codes
  - SP2
  - PC2

**Kernel context:**
- VM structures
- Descriptor table
- Brk pointer
Threads

- Execute in same address space
  - separate execution stack, share access to code and (global) data
- Smaller creation and context-switch time
- Can exploit fine-grain concurrency

Creating processes

- UNIX
  - fork system call
  - Used in conjunction with exec system call
fork: Creating new processes

- int fork(void)
  - creates a new process (child process) that is identical to the calling process (parent process)
  - returns 0 to the child process
  - returns child’s pid to the parent process

```c
if (fork() == 0) {
    printf("hello from child\n");
} else {
    printf("hello from parent\n");
}
```

Fork is interesting (and often confusing) because it is called once but returns twice.

Creating and Using threads

- Pthreads Multi-threading Library
  - API for
    - pthread_create, pthread_join, pthread_self, pthread_exit, pthread_detach

- Java
  - provides a Runnable interface and a Thread class as part of standard Java libraries
    - users program threads by implementing the Runnable interface or extending the Thread class
Concurrent Systems

- Essential aspects of any concurrent system
  - **Execution context** - state of a concurrent entity
    - Processes: process context
    - Threads: thread context
  - **Scheduling** - deciding which context will run next
    - Processes: Operating System scheduler
    - Threads: Library thread scheduler (Pthreads), Java runtime
  - **Synchronization** - mechanisms that enable execution contexts to coordinate their use of shared resources
    - Semaphores, locks, monitors, condition variables
    - Provided at both operating system and library/language level

Road Map

- Next two lectures: Processes & Signals in UNIX
  - Repetition of material discussed in CS 367
  - Assignment 1 (Shell Lab)
- Thread creation and management in Java and Pthreads (one lecture)
- Process & Thread synchronization mechanisms (two - three lectures)