Processes: A System View
What the OS does: 2 State Model

Processes

- not running
- running

- enter
- exit

- dispatch
- pause
What the OS does: 2 State Model

Processes

System

not running

running

enter

exit

queue

processor

enter

exit

pause

dispatch

pause
What information do we need to keep while in the queue?
What the OS stores: PCB

OS stores Process Control Block (PCB) for each process

- In-memory OS structure
- User processes cannot access it

Contents:

- Identifiers
  - pid & ppid (process ID & parent process ID)
- Processor state information
  - User-visible registers, control and status, stack
- Scheduling information
  - Process state, priority, what event the process is waiting for, ...
What the OS stores: PCB

Contents (cont’d):

• Inter-process communication
  ▪ Signals
• Privileges
  ▪ CPU instructions, memory
• Memory Management
  ▪ e.g., Page tables
• Resource ownership and utilization
Five State Process Model

“All models are wrong. Some Models are Useful”
  • George Box, statistician

2 state model
  • Too simplistic
  • What does “Not Running” mean?

7 state model
  • Considers suspending process to disk
  • See Stallings book, section 3.2

Next: 5 state model
5 State Model: States

- Running
- Not running
5 State Model: States

- Running
- Ready
- Blocked
5 State Model: States

- new
- ready
- running
- blocked
- done
5 State Model: Summary

Running
- Currently executing
- On a single processor machine, at most one process in the “running” state

Ready
- Prepared to execute

Blocked
- Waiting on some event

New
- Created, but not loaded into memory

Done
- Released from pool of executing processes
5 State Model: Transitions

Null (nothing) to New
• New process creation
5 State Model: Transitions

New to Ready
- Move to pool of executable processes
5 State Model: Transitions

Ready to Running

- Chosen to run from the pool of processes (How?)
5 State Model: Transitions

What events cause these transitions?
5 State Model: Transitions

Running to Ready
• Preempted by OS

Running to Blocked
• Request for an unavailable resource

Running to Done
• Terminated / completed
5 State Model: Transitions

Blocked to Ready

- Resource is now available
5 State Model: Transitions

Ready to Done
- Terminated by another process

Blocked to Done
- Terminated by another process
5 State Model: Transitions

- **new**
  - enter
  - created

- **ready**
  - selected to run

- **running**
  - quantum expired
  - I/O complete
  - normal or abnormal termination

- **blocked**
  - I/O request

- **done**
  - terminated
Process Queue Model

2 State Model: What is missing?

- Process exceeds time quanta
- Process makes systems call
Process Queue Model

What do we gain with multiple queues?
What do we gain with multiple queues?
Take-away questions

What would happen if user processes were allowed to disable interrupts?

In a single CPU system what is the maximum number of processes that can be in the running state?
From Processes to Threads

CS 241

February 17, 2012

Copyright © University of Illinois CS 241 Staff
Processes vs. threads

Process
- Fork is expensive (time & memory)

Thread
- A lightweight process: little memory, fast startup
- Shared memory among threads in a process
Processes vs. threads

Environment (resource)

Three processes each with one thread

One process with three threads
Processes vs. threads

Each process can include many threads

All threads of a process share:

• Process ID
• Memory (program code and global data)
• Open file/socket descriptors
• Working environment (current directory, user ID, etc.)
• Semaphores (*covered later in the course*)
• Signal handlers and signal dispositions (*covered later in the course*)
Thread usage: word processor

Working file can only be accessed by one process at a time

What would happen if this were single-threaded?
Thread usage: word processor

Working file can only be accessed by one process at a time
Thread usage: web server

What would happen if this were single-threaded?
Thread of execution

Sequential set of instructions

• Each has its own function calls & automatic (local) variables
• Need program counter and stack for each thread
Normal 1-thread function call

Calling program

processfd();

Called function

processfd() {
}

Thread of execution
Compare: Threaded function call

Creating program

Calling program

Called function

processfd();

Creating thread

processfd() {

Thread creation

Thread of execution
Thread Execution States

Events associated with a change in thread state:

- Spawn (another thread)
- Block
  - Should blocking a thread block other, or all, threads?
- Unblock
- Finish (thread)
  - De-allocate register context and stacks
Thread-Specific Resources

Each thread has its own
- Thread ID (integer)
- Stack, Registers, Program Counter

Threads in one process can communicate via shared memory
- Must be done carefully!
Processes vs. Threads

<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 executes separately

Threads in the same process share many resources

No protection among threads!! (What?)
Process vs. thread creation

<table>
<thead>
<tr>
<th>Platform</th>
<th>fork()</th>
<th>pthread_create()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>real</td>
<td>user</td>
</tr>
<tr>
<td>AMD 2.3 GHz Opteron (16 cpus)</td>
<td>12.5</td>
<td>1.0</td>
</tr>
<tr>
<td>AMD 2.4 GHz Opteron (8 cpus)</td>
<td>17.6</td>
<td>2.2</td>
</tr>
<tr>
<td>IBM 4.0 GHz POWER6 (8 cpus)</td>
<td>9.5</td>
<td>0.6</td>
</tr>
<tr>
<td>IBM 1.9 GHz POWER5 p5-575 (8 cpus)</td>
<td>64.2</td>
<td>30.7</td>
</tr>
<tr>
<td>IBM 1.5 GHz POWER4 (8 cpus)</td>
<td>104.5</td>
<td>48.6</td>
</tr>
<tr>
<td>INTEL 2.4 GHz Xeon (2 cpus)</td>
<td>54.9</td>
<td>1.5</td>
</tr>
<tr>
<td>INTEL 1.4 GHz Itanium2 (4 cpus)</td>
<td>54.5</td>
<td>1.1</td>
</tr>
</tbody>
</table>


Timings reflect 50,000 process/thread

Creations were performed with the time utility, and units are in seconds, no optimization flags.
Key points

Threads are lightweight
  • Is this good or bad?

Threads share memory and other resources
  • (Still have own stack, registers, PC, state)
  • Is this good or bad?
Next time

Monday: Using threads

Tuesday: MP3 Shell due