Process Management

Process Concepts

Frédéric Haziza <daz@it.uu.se>

Department of Computer Systems
Uppsala University

Spring 2008
Outline

1 Introduction
Recall

- Process Management
- Memory Management
- Storage Management
- Protection and Security

Not in the class: Networking, GUI, Device Drivers
Hardware

- CPU
- MEM
- Disk
Structuring an Operating System

- Monolithic (MS-DOS, Original Unix)
- Layered (Unix)
- Microkernel (Mach)
- Modular (Solaris)
Not so much structure

Most functionality in the least space.

![Diagram showing the structure of a system](image)
Layered Approach

- layer 0: hardware
- layer 1
- ... (more layers)
- layer N: user interface
Microkernel

application environments and common services

kernel environment

BSD

Mach

OSKomp’08 | Process Management (Process Concepts)
Modular Approach

- device and bus drivers
- scheduling classes
- file systems
- loadable system calls
- executable formats
- STREAMS modules
- miscellaneous modules
User vs Kernel Mode: Hardware protection
System Boot

- Small piece of code - bootstrap loader
  - locates the kernel,
  - loads it into memory,
  - and starts it

- Sometimes two-step process where boot block at fixed location loads bootstrap loader

- When power initialized on system,
  - execution starts at a fixed memory location
  - Firmware used to hold initial boot code
Outline

1. Introduction
2. Process
3. Scheduling
4. Communication
5. Threads
What characterizes a process?

- Program in execution
- Stack (Temporary data, function parameters,...)
- Heap
- Data section (Global variables)
- CPU Registers
- Program Counter (PC)

- Program code = Text section
- Program in execution = text section (executable file) *loaded in memory*
- Process in Memory: See black board
States

**New**  The process is being created

**Running**  Instructions are being executed

**Waiting**  for some event to occur (I/O completion, signal...)

**Ready**  Waiting to be assigned to a processor

**Terminated**  Finished its execution
States

- new
- admitted
- ready
- running
- waiting
- terminated

Transitions:
- new → admitted
- admitted → ready
- ready → running
- running → waiting
- waiting → (I/O or event completion) → ready
- running → (scheduler dispatch) → running
- running → (I/O or event wait) → waiting
- waiting → (I/O or event completion) → ready
- ready → (interrupt) → waiting
- waiting → (exit) → terminated
# Process Control Block (PCB)

<table>
<thead>
<tr>
<th>PCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>process state</td>
</tr>
<tr>
<td>process ID (number)</td>
</tr>
<tr>
<td>PC</td>
</tr>
<tr>
<td>Registers</td>
</tr>
<tr>
<td>memory information</td>
</tr>
<tr>
<td>open files</td>
</tr>
<tr>
<td>other resources</td>
</tr>
</tbody>
</table>
How is in control?

- **Interrupts**
  - Generated asynchronously by external devices and timers
  - Example: The I/O is complete or timers have expired
- **Traps (software errors, illegal instructions)**
- **System calls**
What happens at a transition?

process $P_0$ operating system process $P_1$

executing

interrupt or system call

save state into PCB$_0$

idle

reload state from PCB$_1$

executing

interrupt or system call

save state into PCB$_1$

idle

reload state from PCB$_0$
Scheduling Queues

Job Queue

Linked list of PCBs

- (main) job queue
- ready queue
- device queues
Scheduling

- **Job scheduler** (loads from disk)
- **CPU scheduler** (dispatches from ready queue)
Context switch

- PCB swap

- Cost?
- 10ms switch for 100ms work $\Rightarrow$ 9% wasted
Process creation and termination

See the lab 1 ...
Interprocess Communication (IPC)

2 models
- Message Passing
- Shared Memory

See black board...

Benefits
- Small amount to exchange
  => Message Passing, because no conflict to avoid
- Shared Memory
  => Working at the speed of memory – faster
Shared Memory

Recall that the OS prevents processes to share memory
⇒ Agreement on relaxing restriction

Example (Producer-Consumer)

Unbounded buffer and bounded buffer (book p98)
Shared Memory

Requires:

- Synchronisation
  (No consumption of non produced items)
- Waiting
Message Passing

No shared space.
Can be distributed across network

Example
Chat program

- send(m)
- receive(m)

Requires a communication link
- direct or indirect (mailbox/ports)
- synch. or asynch. (blocking or non-blocking)
- automatic or explicit buffering (info on the link)
Heavy-weight vs Light-weight...

Example (Web server)

We want to serve more than one client at a time

- 1 process. If incoming request, new process created => costly!
- 1 process. If same task as other one, why overhead => better to multithread

On Solaris:

- Time for creating a process = 30 x time for creating a thread
- Time for context switching = 5 x time for switching a thread
Benefits

- Responsiveness
- Resource sharing
- Economy
- Utilization of multiprocessor architectures
Multithread Models

Deals with correspondence between
- threads in user space
- threads in kernel space

One to One        Many to One        Many to Many
Issues

- `fork()` creates a copy of a process with all threads or just the one which calls the fork?
- Cancellation
- Signal handling: Read in book page 139
- Thread pool (limit in system, pre-create threads)
- Thread specific data (sharing data?)
- about the models themselves