Operating Systems II
Review on OS I

Frédéric Haziza <daz@it.uu.se>

Department of Computer Systems
Uppsala University

Summer 2008
Outline

1 Introduction + Quick Review on OS
   - The hardware
   - Computer systems architecture
   - OS Structure

2 Process Management
Abstract view of the components of a system

User_1 → Compiler
User_2 → Assembler
User_3 → Text Editor
... → Database System

System and Application Programs

Operating System

Computer Hardware
Setting up the place

Operating System (OS)

Intermediary between the user and the machine hardware

- User point of view => convenient (ease of use)
- Hardware point of view => efficient (resource allocation)

Definition

Operating System is everything that provides

- an environment for the programs to run
  - ensures correctness of operations
  - protects against interferences
- resource management
## Hardware

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Cache</th>
<th>Mem</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Level 1</td>
<td>Level 2</td>
<td></td>
</tr>
<tr>
<td>sram</td>
<td></td>
<td>sram</td>
<td>dram</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>CPU</th>
<th>Cache</th>
<th>Mem</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1ns</td>
<td>10ns</td>
<td>150ns</td>
<td>5 000 000ns</td>
</tr>
<tr>
<td>1982</td>
<td>200ns</td>
<td>200ns</td>
<td>200ns</td>
<td>10 000 000ns</td>
</tr>
</tbody>
</table>
Computer systems architecture

Single CPU     Multi-CPU     Clusters
Why do we need parallel computers?

Multi-CPUs

- Increased throughput
- Scaling and economy
- Reliability
- Energy

**Speed:** We want to calculate e.g. simulations faster

**Space:** We want to handle larger amounts of data

Parallel computers offer a cost effective solution, not bounded by the laws of physics
What’s more important than performance?

- Correctness
- Simplicity
- Maintainability
- Cost (programmer time)
- Stability, Robustness
- Features, Functionality
- Modularity (local changes rather than across the whole code)
- User-friendliness (HUGE growth in the 90’s)
- Security (important since year 2k)
Why is it hard to program in parallel?

Or: Why are not more of the contemporary software and hardware parallel?

Two main problems:

1. Load balance
2. Communication

Our focus

There is no standard for parallel computer systems. A great spectra of architectures exists. It is “impossible” to construct general efficient parallel programs.
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.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>system-call interface to the kernel</th>
<th>kernel interface to the hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>signals terminal handling</td>
<td>file system swapping block I/O system</td>
<td>CPU scheduling page replacement demand paging virtual memory</td>
</tr>
<tr>
<td>character I/O system terminal drivers</td>
<td>disk and tape drivers</td>
<td></td>
</tr>
<tr>
<td>terminal controllers terminals</td>
<td>device controllers disks and tapes</td>
<td>memory controllers physical memory</td>
</tr>
</tbody>
</table>
Layered Approach

- layer N
- user interface
  - ...
- layer 1
- layer 0
  - hardware
Microkernel
Modular Approach

- core Solaris kernel
- scheduling classes
- file systems
- loadable system calls
- executable formats
- STREAMS modules
- miscellaneous modules
- device and bus drivers
Outline

1 Introduction + Quick Review on OS

2 Process Management
   - Definition
   - States
   - PCB & Queues
   - Transitions
   - Communication
   - Threads
   - Scheduling
Process Management

Resources (CPU time, memory, files, I/O) are either

- given at creation or
- allocated while running.

**Definition (Process)**

Unit of work in the system. For both user and system.

= Call sequence that executes independently of others. Maintains bookkeeping and control for this activity.

- Creating / Deleting / Suspending / Resuming
- Mechanism for process synchronization
- Mechanism for process communication
- Mechanism for deadlock handling (prevention, avoidance, reparation, ...)
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
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

Actions:
- interrupt
- exit
- I/O or event completion
- scheduler dispatch
- I/O or event wait
Process Control Block & Queues

**PCB**
- process state
- process ID (number)
- PC
- Registers
- memory information
- open files
- other resources

**Job Queue**
Linked list of PCBs
- (main) job queue
- ready queue
- device queues

**Schedulers**
- Long-term/Job scheduler
  (loads from disk)
- Short-term/CPU scheduler
  (dispatches from ready queue)
Who is in control?

- To increase CPU utilization: **multitasking**
  - Job pool (in memory)
  - Interaction
  - The OS provides each user with a slice of CPU and main memory resources.

- **Interrupt** (hardware error detection)
  - Generated asynchronously by external devices and timers
  - Example: The I/O is complete or timers have expired

- **Traps** (software errors, illegal instructions)

- **System calls** (interface to ask the OS to perform privileged tasks)
What happens at a transition?
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
Recall that the OS prevents processes to share memory
⇒ Agreement on relaxing restriction

**Example (Producer-Consumer)**

Unbounded buffer and bounded buffer

![Diagram of Producer-Consumer model with shared resource]
Shared Memory

Requires:

- Synchronisation
  (No consumption of unproduced 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 async. (blocking or non-blocking)
- automatic or explicit buffering (info on the link)
From process flaws

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 $\Rightarrow$ costly!
- 1 process. If same task as other one, why overhead? $\Rightarrow$ better to multithread

On Solaris:

- Time for creating a process $= 30 \times$ time for creating a thread
- Time for context switching $= 5 \times$ time for switching a thread
Threads

single-threaded process

multithreaded process
Benefits

- Responsiveness
- Resource sharing
- Economy
- Utilization of multiprocessor architectures
Recall – Abstract view

System and Application Programs

Operating System

Computer Hardware

Compiler
Assembler
Text Editor
Database System
User vs Kernel Mode: Hardware protection

```c
#include <stdio.h>
int main ()
{
    ...
    printf ("Greetings");
    ...
    return 0;
}
```

Diagram showing the relationship between user mode, kernel mode, and standard C library with a system call.
Multithread Models

Deals with correspondence between

- threads in user space
- threads in kernel space

One to One  Many to One  Many to Many
Important note

Note that...

On Operating Systems which support threads, it is \textbf{kernel-level} threads -- \textit{not processes} -- that are being scheduled.

However, \textit{process} scheduling $\approx$ \textit{thread} scheduling.
CPU and IO Bursts

: load, store, add, store, read from file
  Wait for IO
store, increment, branch, write to file
  Wait for IO
load, store, read from file
  Wait for IO :

CPU Burst cycles
Intervals with no I/O usage

Waiting time
Sum of time waiting in ready queue
How do we select the next process?

- **CPU utilization**
  
  CPU as busy as possible

- **Throughput**
  
  Number of processes that are completed per time unit

- **Turnaround time**
  
  Time between submission and completion

- **Waiting time**
  
  Scheduling affects only waiting time

- **Response time**
  
  Time between submission and first response
Algorithms

- **FCFS**: Non-preemptive, Treats ready queue as FIFO.
  - Problem: Convoy effect...

- **SJF**: Shortest next cpu burst first
  - Problem: Difficult to know the length of the next CPU burst of each process in Ready Queue.
  - Solution: Guess/predict based on earlier bursts.

- **SJF with Preemption**: When a process arrives to RQ, sort it in and select the SJF including the running process, possibly interrupting it.

- **Priority Scheduling**: Can be preemptive or not
  - Problem: Starvation (or Indefinite Blocking)
  - Solution: Aging

- **Round-Robin**: FCFS with Preemption. Ready Queue treated as circular queue
  - Problem: Quantum $\gg$ Context-switch

- **Multilevel queues**
  - Multi-Level Feedback Queue