Operating Systems II

Review on OS I

Frédéric Haziza <daz@it.uu.se>

Department of Computer Systems

Uppsala University

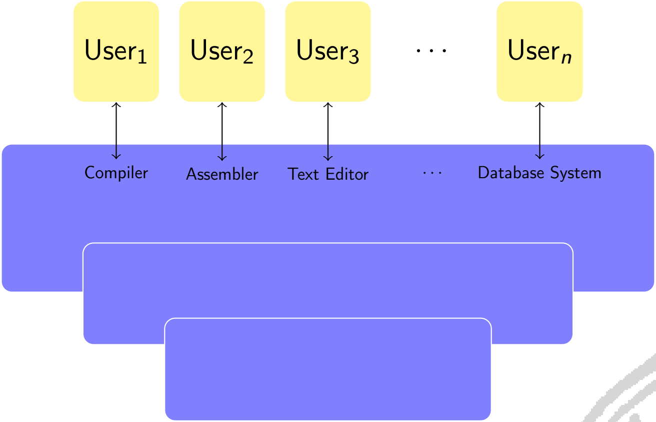
Summer 2009

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



Setting up the place

Operating System (OS)

Intermediary between the user and the machine hardware

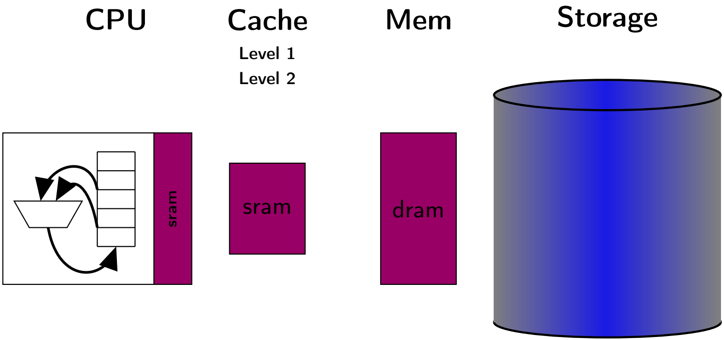
- User point of view => (ease of use)
- Hardware point of view => (resource allocation)

Definition

Operating System is everything that provides

- an environment for the programs to run
- resource management

Hardware



	CPU	Cache	Mem	Storage
2000:	1ns 3ns	10ns	150ns	5 000 000ns
1982:	200ns	200ns	200ns	10 000 000ns

Computer systems architecture

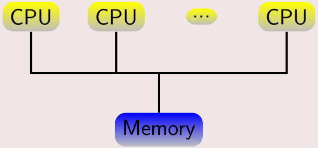
Single CPU

Multi-CPU

Clusters

Why do we need parallel computers?

Multi-CPUs



- 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-friendlyness (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.
2. ← our focus

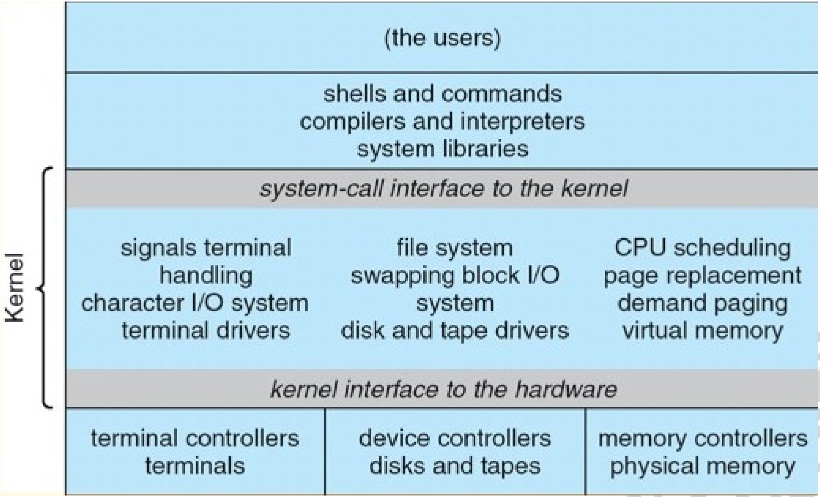
There is no standard for parallel computer systems. A great spectra of architectures exists. It is "impossible" to construct *general* efficient parallel programs.

Structuring an Operating System

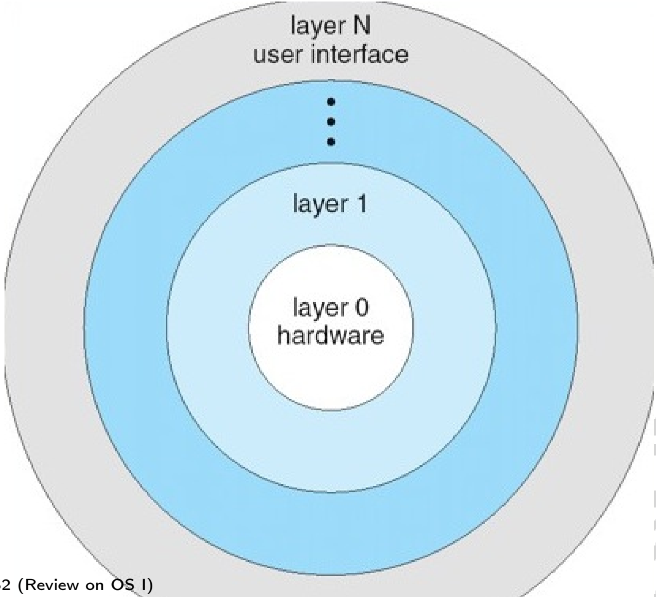
■ Monolithic	(MS-DOS, Original Unix)
■ Layered	(Unix)
■ Microkernel	(Mach)
■ Modular	(Solaris)

Not so much structure

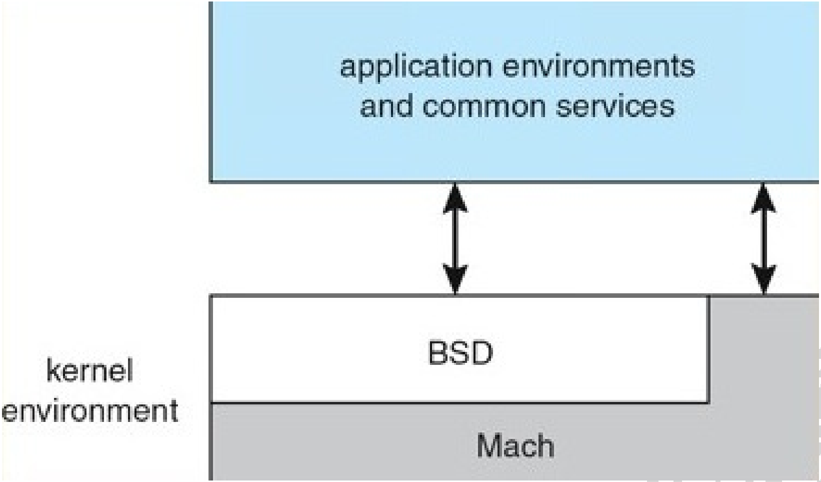
Most functionality in the least space.



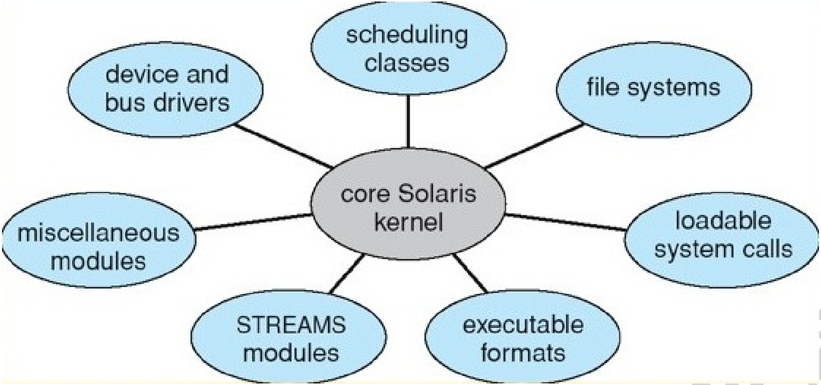
Layered Approach



Microkernel



Modular Approach



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?

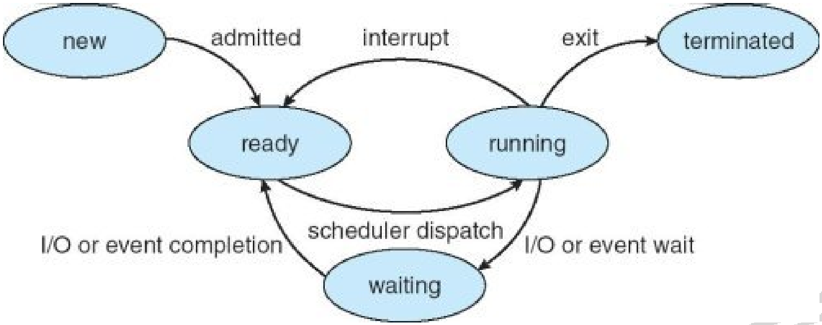
- Program in execution
- Stack (Temporary data, function parameters,...)
- Heap
- Data section (Global variables)
- CPU Registers
- Program Counter (PC)

- 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
interrupt
exit
terminated
ready
running
scheduler dispatch
I/O or event completion
I/O or event wait
waiting

& Queues

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

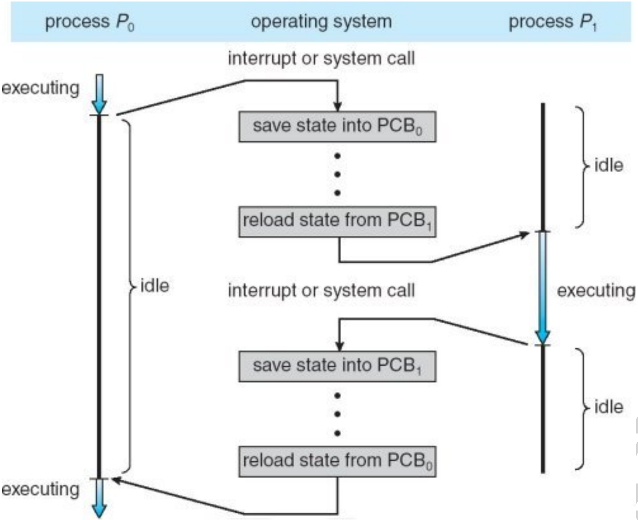
- (loads from disk)
- (dispatches from ready queue)

Who is in control?

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

- (hardware error detection)
  - Generated asynchronously by external devices and timers
  - Example: The I/O is complete or timers have expired
- (software errors, illegal instructions)
- (interface to ask the OS to perform privileged tasks

What happens at a transition?



Interprocess Communication (IPC)

2 models

Benefits

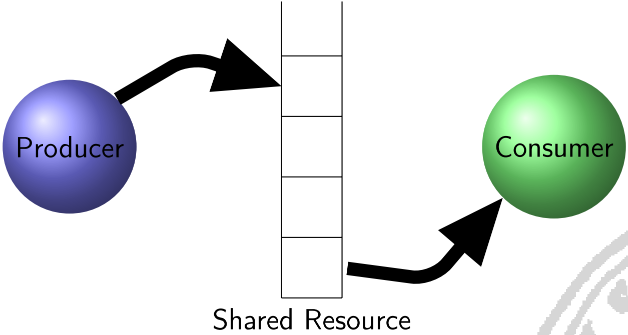
- Small amount to exchange
  => Message Passing, because no conflict to avoid
- Shared Memory
  =>Working at the speed of memory – faster

Shared Memory

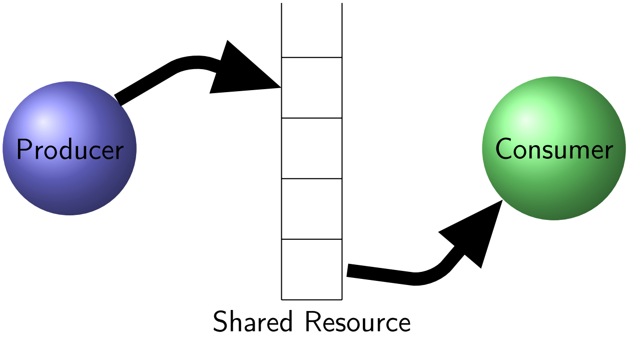
Recall that the OS prevents processes to share memory
⇒ Agreement on relaxing restriction

Example (Producer-Consumer)

Unbounded buffer and bounded buffer



Shared Memory



Requires:

- Synchronisation
  (No consumption of unproduced items)
- Waiting

Message Passing

No shared space.
Can be distributed accross 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)

From process flaws

Heavy-weight *vs* Light-weight...

Example (Web server)

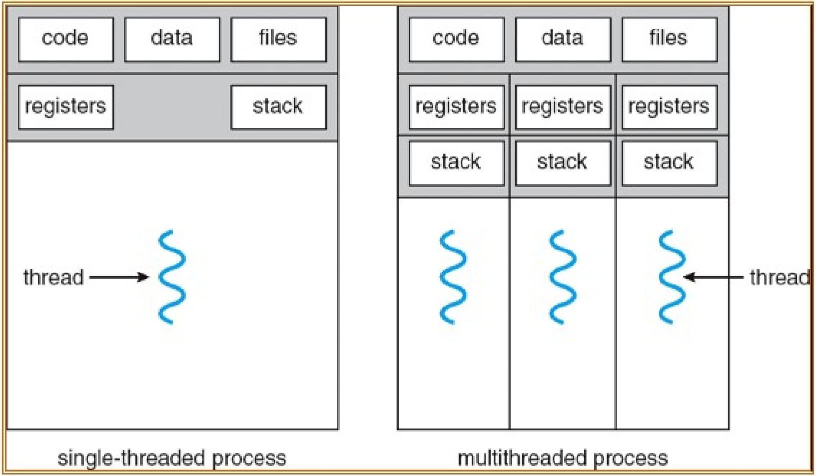
We want to serve more than one client at a time

- 1 process. If incoming request, new process created => 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

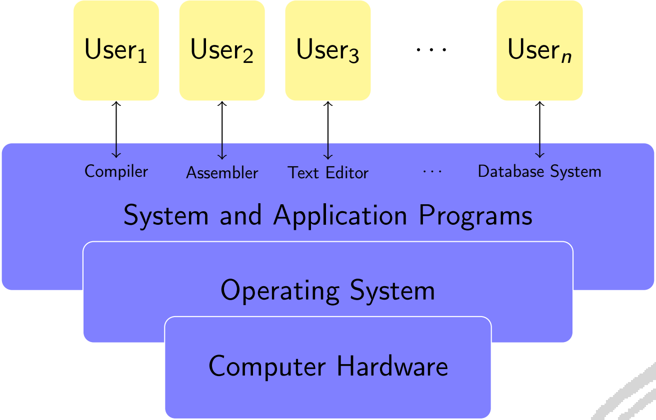
Threads


code
data
files
registers
stack
thread
single-threaded process
code
data
files
registers
registers
registers
stack
stack
stack
thread
multithreaded process

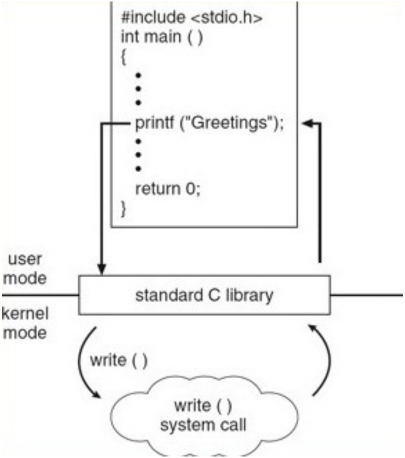
Benefits

-
- Resource sharing
- Economy
- Utilization of multiprocessor architectures

Recall – Abstract view



User vs Kernel Mode: Hardware protection



Multithread Models

Deals with correspondance between

- threads in
- threads in

One to One **Many to One** **Many to Many**

Important note

Note that...

On Operating Systems which support threads,
it is kernel-level threads – *not processes* –
that are being scheduled.

However, *process* sheduling $\approx$ *thread* scheduling.

CPU and IO Bursts

⋮

load, store,
add, store,
read from file

Wait for IO

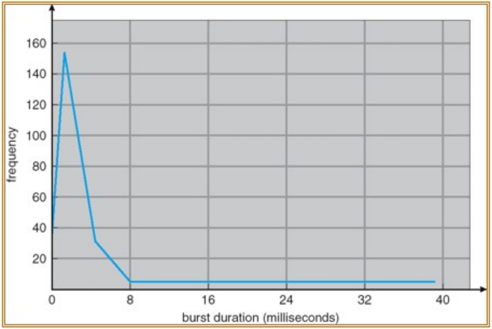
store,increment,
branch, write to file

Wait for IO

load, store,
read from file

Wait for IO

⋮



Intervals with no I/O usage

Waiting time

Sum of time waiting in queue

How do we select the next process?

- CPU as busy as possible
- Number of process that are completed per time unit
- Time between submisson and completion
- Scheduling affects only waiting time
- Time between submisson and first response

Algorithms

- : Non-preemptive, Treats ready queue as FIFO.
  - Problem: Convoy effect...
- : 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.
- : When a process arrives to RQ, sort it in and select the SJF including the running process, possibly interrupting it
- . Can be preemptive or not
  - Problem: Starvation (or Indefinite Blocking)
  - Solution: Aging
- : FCFS with Preemption. Ready Queue treated as circular queue
  - Problem: Quantum $\gg$ Context-switch
-
  - Multi-Level Feedback Queue