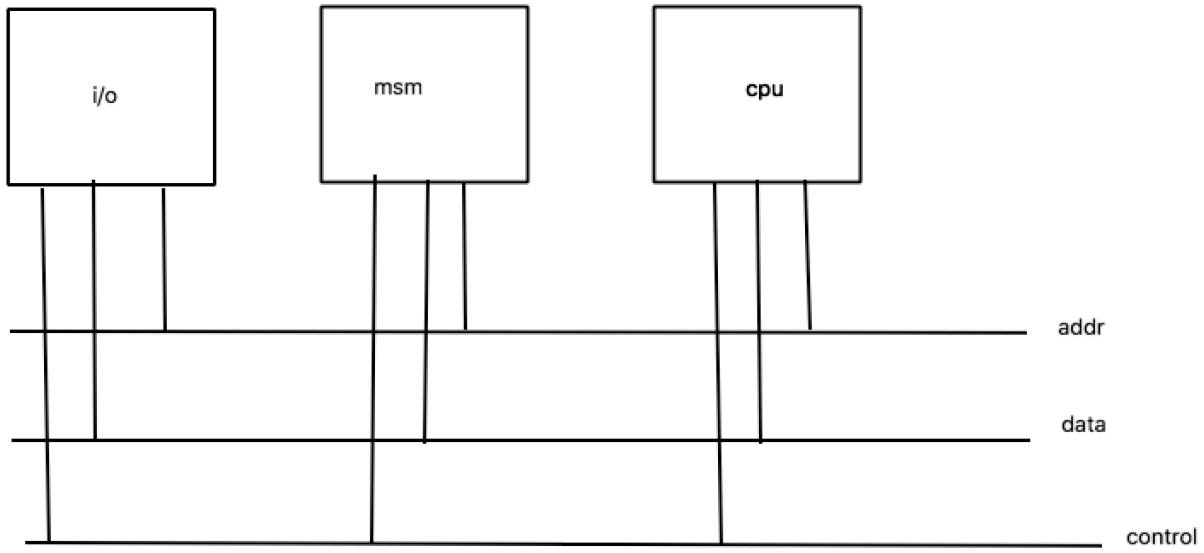
January 22, 2018

1) Google is coming
2) top 15 best midterms, they will have a tour of Google

Ch 13: I/O Systems
Beside CPU, everything is I/O

I/O Hardware

Hardware configuration:
msm: memory



How can the CPU between I/O: technically CPU has specific I/O instruction OR CPU doesn't care , copy memory to have an address.

Polling => can be simple polling OR daisy chain
Either CPU go look at I/O (polling) or I/O inform the CPU that smthing is going on (interrupt). Which method is better?

Polling

For each byte of I/O
How can we identify the source of the
Daisy Chain

Interrupts
Vectored and Non vectored.
If I/O device be recognize and CPU by catching a request, that request can recognized it's from the I/O (vectored)
Non vectored: CPU doesn't recognized which I/O sent the request.

Vectored is better, but it has a cost: more data lines and more complex design, more needed hardware but easier for software
Non vectored: more on the software, less hardware

Interrupt handler: it can be ***nonmaskable*** (cant ignored it) or ***maskable***(can ignored).
Interrupt : software (divide by 0) or hardware (printer)

Interrupt-Driven I/O Cycle
CPU initiate interrupt service routine. -> I\O has smthing to tell -> CPU receive interrupt -> interrupt handler -> CPU resumes process

In some CPU, by accepting interrupt, you clear that interrupt, then will the CPU clears the flag to see if there's other interrupts.
Interrupt mechanism used for exceptions
Syst call executes via trap to trigger kernel.

Direct Memory Access
Used to avoid ***programmed I/O*** for large data movement
Requires ***DMA*** controller
Bypasses CPU to transfer data directly between I/O device and memory

In which situation are you better with polling than interrupts?
Frequency of I/O, the mouse PS3 to mouse USB.

OS writes DMA command block into memory
Ex: you want to copy a document to USB: the cpu will send the command to DMA, DMA will take the address and copy it to USB, while you can do smthing else with the CPU.
SO DMA and CPU cannot be on the same bus line.

//revoir DMA

6 step process to perform DMA transfer

Application I/O Interface

Devices vary in many dimensions:

- Character-stream or block
  - terminal
  - disk
- sequential or random-access
  - sequential= printer
  - random-access = USB key, or CD, DVD, hard drive
- synchronous or asynchronous
  - mouse (peut etre les 2 en mm temps)
- sharable or dedicated
  - keyboard
  - tape
- speed of operation
  - diff I/O, diff speed
- read-write, read only, write only

Block and character devices

Clocks and Timers

programmable interval timer: used for timing, periodic interrupts

Nonblocking and asynchronous I/O

blocking

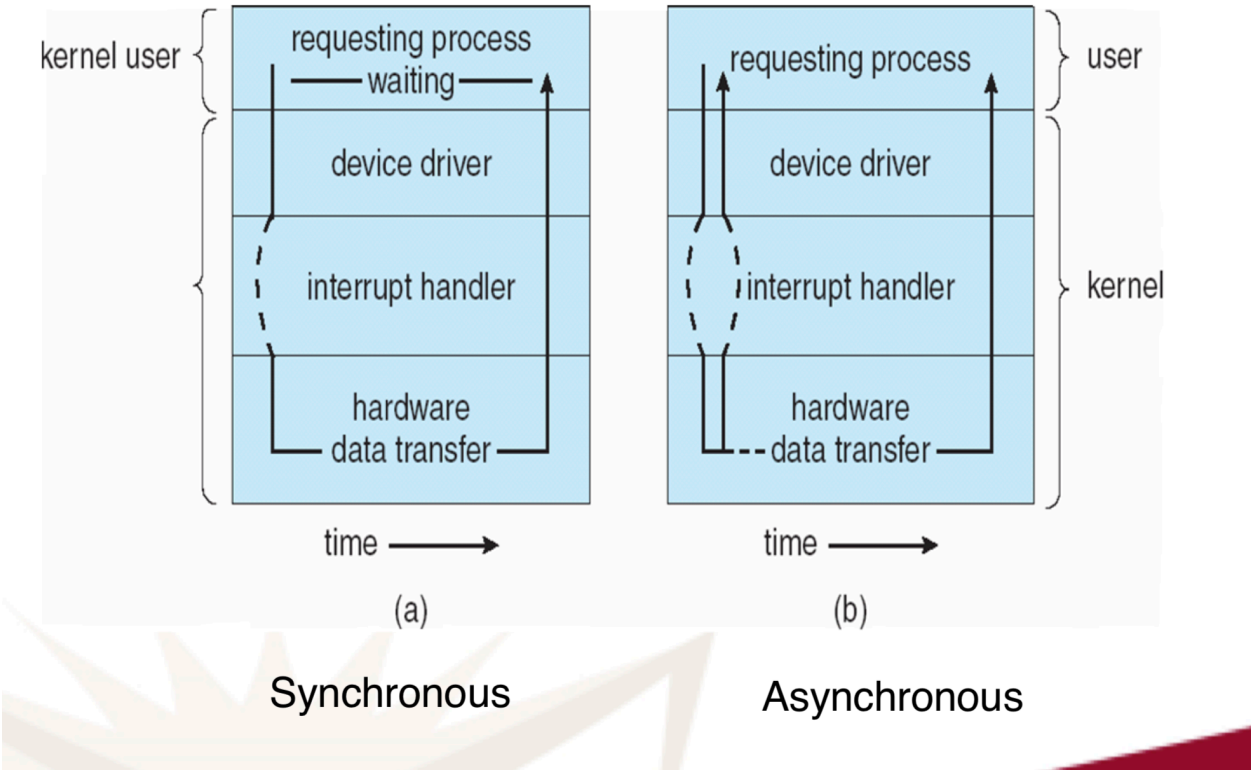
- process suspended until I/O completed

nonblocking

- I/O call returns as much as available
- user interface, data cpu
- implemented via multi-threading

asynchronous

- process runs while I/O executes
- we are talking about the clock



Asynchronous you don't have to wait; Synchronous you have to wait

Vectored I/O

Kernel I/O Subsystem

Scheduling

Buffering

- used it to store data in memory while transferring between devices
- double buffering: 2 copies of the data
  - kernel and user
  - varying sizes
  - full/being processed and not-full/being used

Catching

- faster device holding copy of data
- always just a copy
- sometimes combined with buffering

Spooling

- hold output for a device

- printing ex

Device reservation

- provides exclusive access to a device
- syst calls for allocation and de-allocation
- watch out for deadlock

I/O Protection

Which user is allow to access, which device is allow to access.
Restraining access
Memory mapped and I/O port memory must be protected too.

Kernel Data Structure

Power management

I/O requests to hardware operations

reading a file from disk for a process:

- determine device holding file
- translate name to device representation
- physically read data from disk into buffer
- make data available to requesting process
- return control to process

STREAMS Structure

putting the queue at diff level: user level to system level.

PERFORMANCE

I/O major factor in syst performance:

- Demands CPU to execute device driver, kernel I/O code
- context switches due to interrupts
- data copying
- network traffic especially stressful

What is the suitable method for fast devices?

A) vectored interrupt
B) polling
C) non vector interrupt
D) daisy chain

B because it will filter interrupts faster.

How to improve performance?

- reduce number of context switches
- reduce data copying
- reduce interrupts by using large transfers, smart controllers, polling
- using DMA
- use smarter hardware devices
- balance CPU, memory, bus, and i/O performance for highest throughput
- Move user-mode processes / daemons to kernel threads

Ch 3 : Processes

Process Concept

OS executes variety of programs;

- batch syst –**jobs**
- time-shared systs – **user programs** or **tasks**

Process: program in execution; process execution must progress in sequential fashion

Multiple parts

- the program code, also called **text section**
- current activity including **program counter**, processor registers
- **Stack** containing temporary data
  - (fct parameters, return addresses, local variables)
- **data section** containing global variables
- **Heap** containing memory dynamically allocated during run time

Program is ***passive*** entity stored on disk (executable file) process is ***active*** program process when executable file loaded into memory
Execution of program started via GUI mouse clicks command line entry of its name etc
One program can be several processes
Consider multiple users executing the same program

Process state

As a process executes it changes state

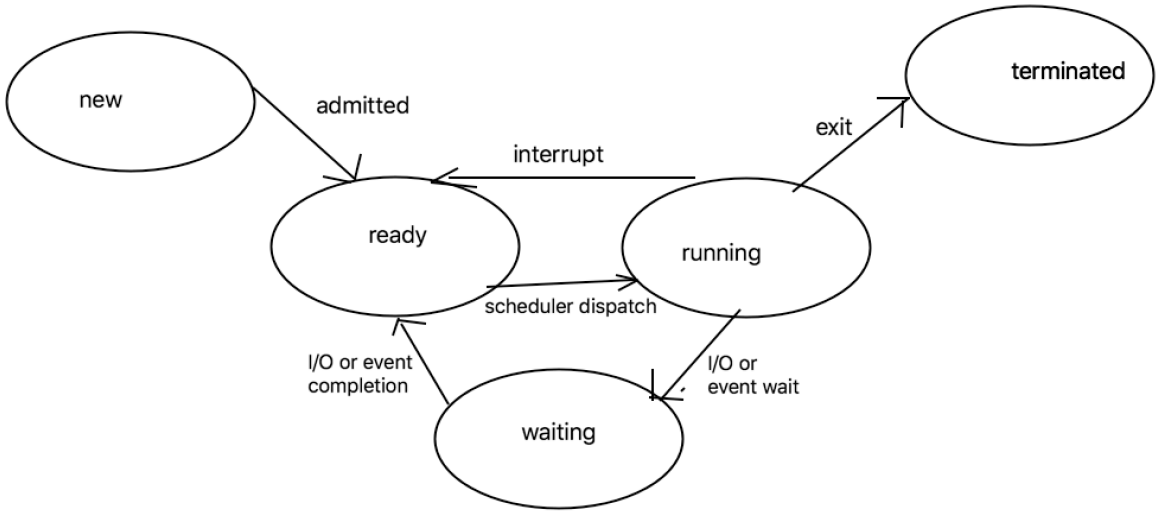
new: the process is being created

running: instructions are being executed

waiting: the process is waiting for some event to occur

ready: process is waiting to be assigned to a processor

terminated: process has finished execution



mostly in the loop ready-running-waiting, mostly waiting.

Process Control Block (PCB)

Info associate with each process also called task control block

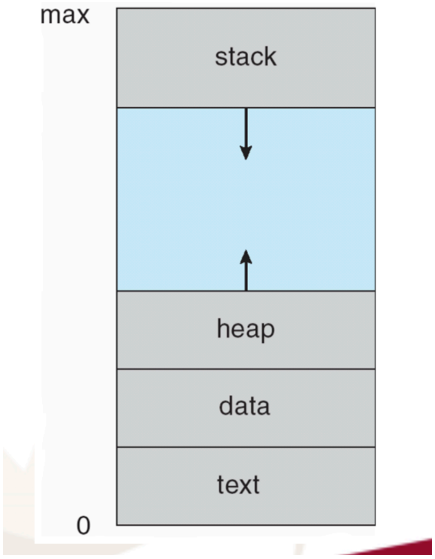
- process state – running waiting etc
- program counter – location of instruction to next execute
- CPU registers – contents of all process centric registers
- CPU scheduling info – priorities, scheduling queue pointers
- memory-management info –memory allocated to process
- accounting info – CPU used, clock time elapsed since start, time limits
- i/o status info- i/o devices allocated to process

CPU Context Switch from process to process

Should execution time, average time waiting.

February 5, 2018

Chapter 3: Process



Process Concept

OS executes many programs: Batch syst (jobs) and Time-shared systems (user programs or tasks)
Process: program in execution

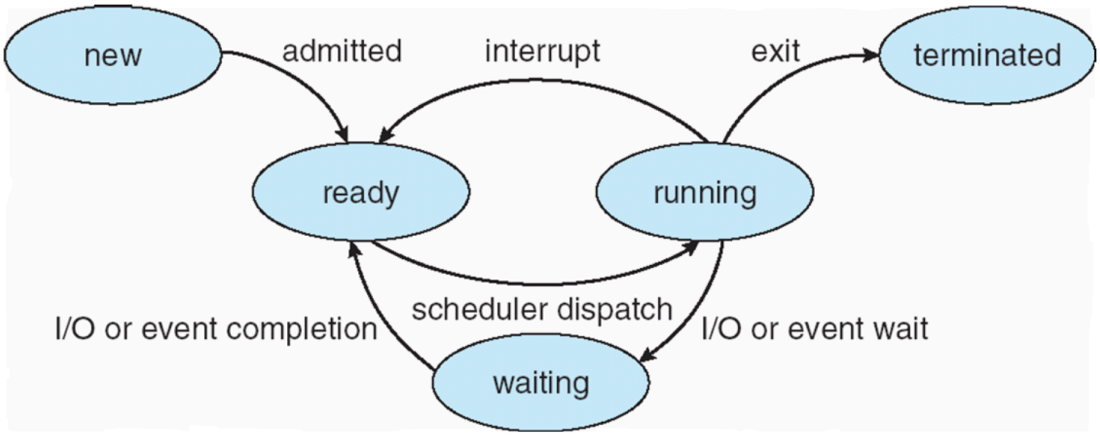
Process State

As a process executes, it changes state

- new: The process is being created
- running: Instructions are being executed
- waiting: The process is waiting for some event to occur
- ready: The process is waiting to be assigned to a processor
- terminated: The process has finished execution

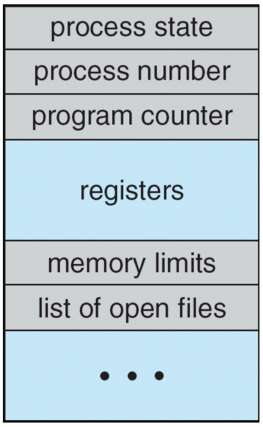
Multiparts

Program code = text section
Stack containing temporary data
Data section containing global variables
Heap containing memory dynamically allocated during run time

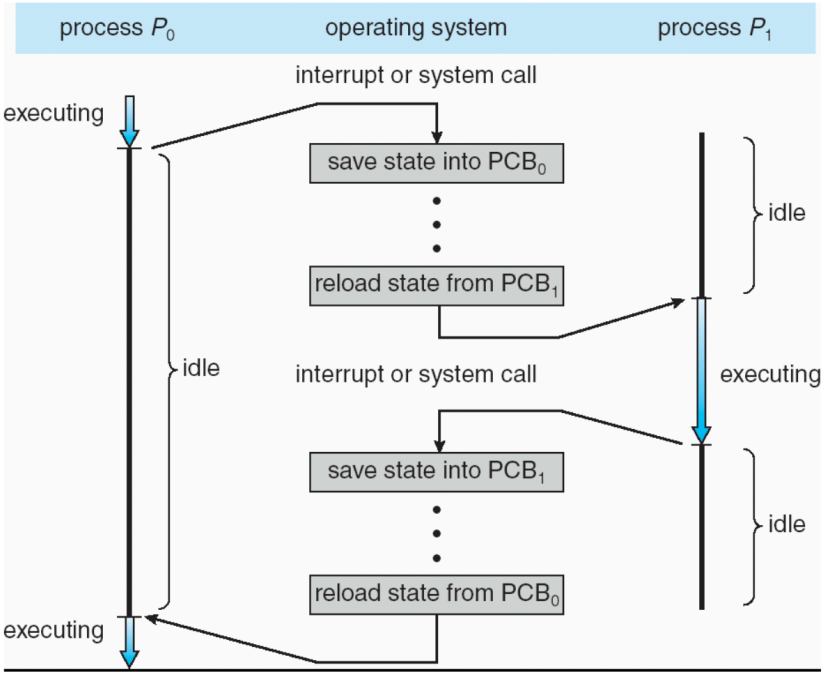


Process Control Block (PCB)

Process state – running, waiting, etc
Program counter - location of instruction to next execute
CPU registers - contents of all process centric registers
CPU scheduling information- priorities, scheduling queue pointers
Memory-management information -memory allocated to the process
Accounting information - CPU used, clock time elapsed since start, time limits
I/O status information - I/O devices allocated to process, list of open files



CPU Switch From Process to Process



How much time we spent on each process = overhead
direct cost (expenses): memory map table
indirect cost: each process has its own virtual memory space no privacy problem

Schedulers

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) ⇒ (must be fast)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) ⇒ (may be slow)
  - The long-term scheduler controls the **degree of multiprogramming**
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good ***process mix***

Concordia University

Context Switch

- When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB ➔ the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU ➔ multiple contexts loaded at once

Operations on Processes

System must provide mechanisms for:

- process creation,
- process termination,
- and so on as detailed next

Task of CPU scheduler:
does not maintain I/O queue and not decision when to generate an event that a process is waiting on

February 12, 2018

Process termination

Process executes last statement and then asks the operating system to delete it using the **exit()** system call.

- Returns status data from child to parent (via **wait()**)
- Process' resources are deallocated by operating system

Parent may terminate the execution of children processes using the **abort()** system call. Some reasons for doing so:

- Child has exceeded allocated resources
- Task assigned to child is no longer required
- The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

Some OS doesn't allow child to exists if parent has terminated. : cascading termination. & the termination is initiated by OS
Parent process may wait for termination of a child process by using wait() system call.
Process is a ***zombie*** IF no parent waiting (did not invoke wait()).
Process is a ***orphan*** IF parent terminated without invoking wait.

Amdahl's Law

Identifies performance gains from adding additional cores to an application that has both serial and parallel components
S is serial portion ; *N* processing cores

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

As *N* approaches infinity, speedup approaches 1 / *S*

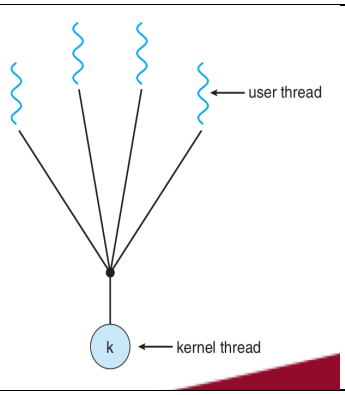
User Threads and Kernel Threads

User threads - management done by user-level threads library
Kernel threads - Supported by the Kernel

Multithreading Models

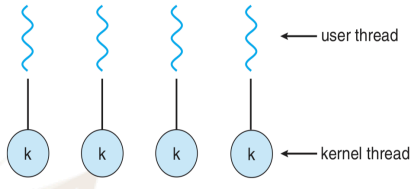
Many-to-One

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on multicore system because only one may be in kernel at a time



One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead



Many-to-Many

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads