

Interrupt Service

1. Save Contents of PC
2. Load PC with address of interrupt service routine
3. Interrupt service routine(IRS) saves of the reg contents
4. IRS does whatever is needed to service device that cause intr

**Fetch**

Bring contents of mem add stored in PC and store the contents in the instruction reg. PC++

**Decode**

Interpret the contents of the IR

**Execute**

Do What instruction says

**I/O Steps and Interrupts**

1.A user’s process requests a read (or write) operation on the device –This generates an interrupt

2. The O.S. schedules the device driver process to perform that read (or write) operation when the controller is idle.

3. The driver places an “input” command into the controller’s command register. For a “write” operation, the driver first places the data in the controller’s data register and then the driver places an “output” command into the controller’s command register.

4.The driver saves bookkeeping information about this I/O operation on the device status table.

5.The driver gives control to the O.S. scheduler.

6.Other processes are scheduled by the O.S.

7.When “our” controller’s I/O is finished, it generates an interrupt.

8.The interrupt service routine for this controller (“device handler”) gets the bookkeeping information from the proper device status table.

9.The interrupt service routine stores the input data into the user process’s memory space.

10.For an output operation it merely notes that the output is completed.

11.The user process is now allowed to run again when the O.S. scheduler selects it.

**Device Controllers**

The controller is dedicated to monitor and control the device.

It relieves the O.S. from this task.

Getting data in and out of the device is extremely slow when compared to CPU speeds**.**

Busy Flag, Done Flag

**The Basic von Neumann Architecture**

Instructions and data for concurrently running processes are stored in main memory.

Instructions are brought into CPU for execution.

Data are brought into CPU for computation

CPU and device controllers execute in parallel

**Registers**

Program counter and instruction register

**ALU**

Arithmetic and logical functions

**Control**

Controls what is done

**Protecting System Resources**

CPU: No Process is allowed to monopolize the CPU

OS runs every time the time slice goes off

I/O: No process is allowed to directly control I/O devices

If a privileged instruction is attempted in user mode an interrupt is generated

Memory: No process is allowed to access a main memory address outside of its allocated space

If a memory location outside of the allocated space is accessed an interrupt is generated

**Device Driver Interface**

User Process: Uses commands to interface with device driver like “open, close, read, write”

OS – Interface to device driver: Grants use of the device, schedules the use of the CPU by the driver, allocate buffers

OS – Device Driver: Monitors the state of the controller, issues specific commands to read or write, and reads and writes the data

Controller: Waits for a command to do work, does the work, and notifies the driver (in polling) or generates an interrupt

The device driver is a part of the OS. It executes privileged instructions as it controls the device, and it reads and writes into different user memory spaces.

A user process must request the use of a device from the OS. The OS may or may not grant the use of the device. The user process waits until it is given the use of the device.

**Interrupts Caused By I/O**

After every instruction is executed CPU checks if any interrupts have been requested

**Buffering controllers**

Buffers a bit of inputs to while user process uses data1 controller is acquiring data2.

**Direct I/O vs Memory Access I/O**

- In direct I/O the CPU handles the data

- In Memory Access the controller handles it

**Process Control Block/ Process Descriptor**

A data structure where the OS stores information about each process.

The info is needed to restart the process when it gets the CPU.

1. Process ID, parent process ID, children process IDs
2. Process state – ready, waiting suspended
3. Contents of CPU registers – program counter, stack points, etc
4. Memory management info – limit registers, page table
5. CPU scheduling info – priority
6. Resources – list of resources held and requested
7. Threads – information for each thread in the process

**The OS keeps all the process descriptors in a process table**

**Programs and Processes**

- A program is an executable file on disk. **Passive**

- A process is a set of instructions being executed. **Active**

**-** One program and two processes. P1 and P2 are processes executing the same set of instructions. The program text is shared by them.

**What is a Process?**

A program running

**Status:**

Contents of PC, other CPU registers

**Resources:**

Files, printers, communication, memory, CPU, OS data structures

**Process States**

**Ready**

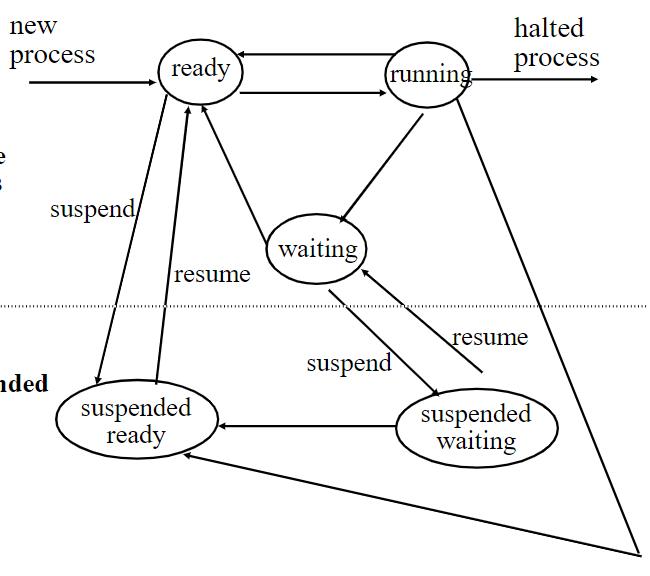
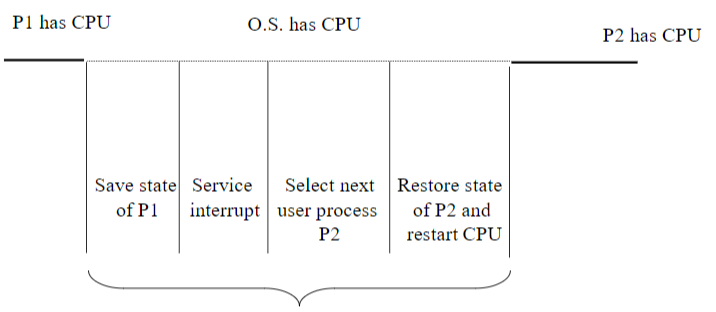
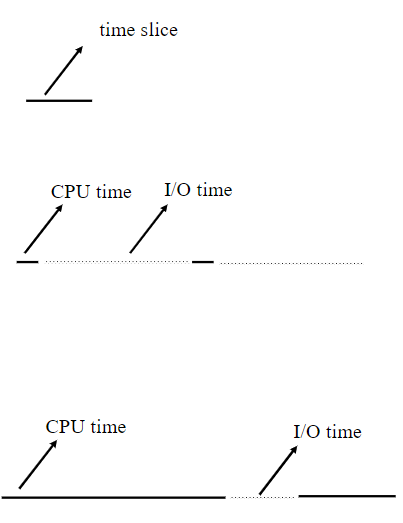
The process can be assigned a CPU if one were available

**Running:**

A CPU is executing instructions for that process

**Waiting:**

The process is waiting for some activity to be completed before it can be considered ready



**I/O and CPU Bound Processes**

- If all processes are I/O bound then the CPU is mostly idle

- If all processes are CPU bound then processes are always waiting for the CPU

- Thus the importance of having the right mix of processes for multiprogramming

\*\*\*For a multiuser system the size of the time slide should be large enough so that the large majority of interactive requests take less time than the time slice\*\*\*

CPU bound

I/O bound

**Changing Process States**

**Ready -> Running:**

When the CPU is free and can be assigned to a process

**Running -> Ready:**

When the process that is running uses its share of the CPU time

**Running -> Waiting:**

When the process that is running needs to wait for some event to be completed

**Waiting -> Ready:**

When the event that the blocked process is waiting for is completed

**OTHER STATE CHANGES ARE NOT ALLOWED**

**Context Switching**

Context Switching is the name given to saving a state of a user process and then loading the CPU with the state of a new user process.

The time take for context switching is computing time lost.

**Context Switch depends on:**

Memory Speed

Speed of saving registers

# of registers

**Multiple Registers**

Allows you to store all of your current registers on the CPU rather than moving to main mem

**Typical Time: Less than 10 micro seconds**

**Process Management**

The operating system manages the process by:

* + Creating and deleting processes
  + Suspended and restarting processes
  + Assigning resources to processes
  + Scheduling processes
  + Synchronizing processes
  + Providing communication between processes

|  |  |
| --- | --- |
| Date | Displays or set date/time |
| Who | Displays who is logged in |
| Ls | List director cont. |
| Echo | Writes argument to std output |
| Clear | Clear terminal |
| Exit | Exits from the  machine |
| Mkdir <dir\_name> | Creates directory |
| Chmod {+,-} <permissions><filename> | 700 -> 7me, 0 -> every1, 0 -> group |
| Cd <dir> | Change dir |
| Touch <filename> | Creates new file |
| Vim <filename> | Open file in vim |
| Wc -l <filename> | Finds number of lines in file |
| Head -n # <filename> | Gets first # of lines |
| Tail -n # <filename> | Gets last # of lines |
| Cat -n <filename> | Display lines of file |
| Sum=$((1+3)) | 1+2 is sum |
| Grep <word> <filename> | Print all lines of file containing word |
| Grep -v <word> <filename> | Print all lines that DO NOT contain word |
| Grep -v -n <word> <filename> | Returns the line numbers with every line (-v or not) |
| Grep -I <word> <filename> | Ignores case |
| Grep -o <word> <filename> | Print occurrence of word in the file |
| Find /usr/include/sys | Returns all files in directory |
| Find /usr | grep sem.h | Tells where the file sem.h is inside /usr |