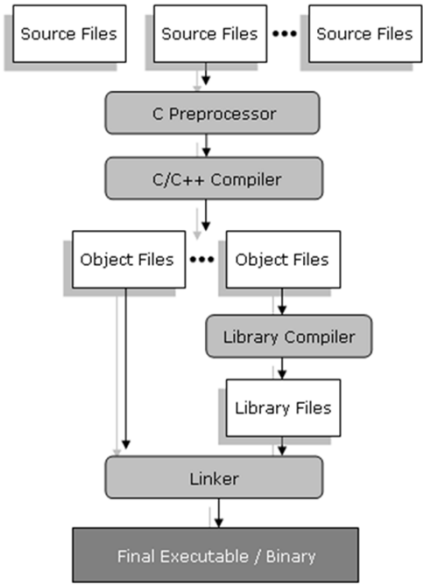
ET0023 Operating Systems

3. Programs & Processes

What is a Program?

- A sequence of instructions that the computer follows to achieve some result.
- Stored on secondary storage e.g. Hard disk or CD ROM
- Takes up hard disk space
- Can consist of one or more files.



Program Creation

- Created by software programmers
- Usually in "English-like“ syntax
- Converted to binary codes

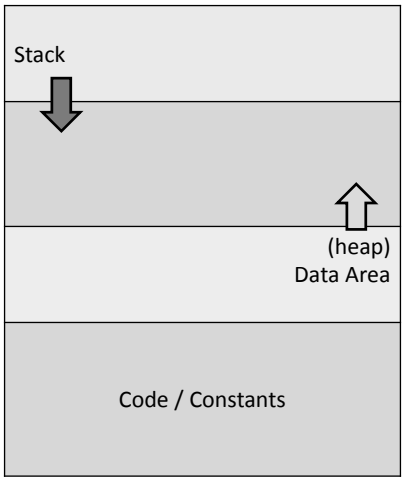
What is a Process

- An instance of a running program.
- Consists of machine instructions, data area, stack and the machine registers it is using.
- Processes are created when a user runs a program.
- A process is a program in execution.

Processes

- The OS will load the program (using a loader) into memory.
- The OS will then let the process run on the CPU
- Processes will take up memory, CPU time and other hardware resources.
- Each process will take up address space, registers and memory locations.

Processes



- Associated with the process
  - Address space
  - Memory locations
  - Set of registers
  - Other information to run the program

Programs & Processes

- Several users can run the same program at the same time (each run is a separate process)
- A program can be made up of several processes.
- On a single processor system, the processor can only execute one process at any given instant.
- The OS can be maintaining several active processes but apply the processor to each one in turn. (time-sharing or multitasking)

Role of OS

- Process Management
  - Creation of the process
  - Suspension and restart of the process
  - Establish mechanisms for process communications, process synchronization
  - Termination of the process

Process Execution

- A process is a program executing in an address space.
- During its execution, the process resides in memory, taking input, producing output.
- Has access to
  - The CPU, so that instructions can be executed
  - The registers of the CPU
  - Memory areas where process instructions, data are stored
  - The CPU Stack
  - Services provided by the OS

Process Execution

- When a process is executing, the OS will
  - Use hardware to ensure process is running in user mode
  - Ensure that process has access to memory areas
  - Limit access (read-only) to process instruction areas
- The OS will be running in supervisory (monitor) mode.

Process Table

- All information about each process is stored in an OS structure called the process table.
- The process table is stored in a data collection (e.g. array or linked list).
- Each process currently in existence has an entry.
- A suspended process consists of its address space (core image) and its process table entry.

A Process Table Entry

Process management	**Memory management**	**File management**
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

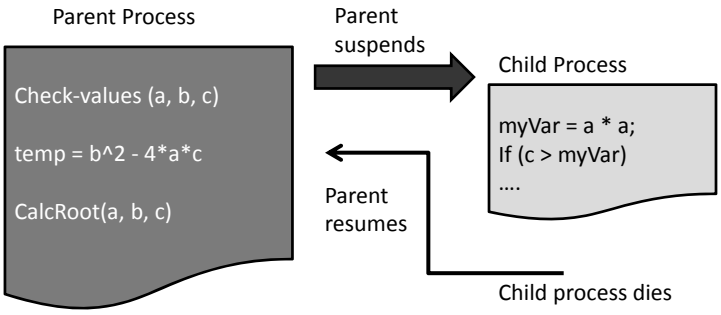
Process Control Block (PCB)

Process Control Block

- Contains information associated with the Process, usually includes
  - Process identifier
  - Process context
- Program Counter, Processor Status Word, Registers content, Stack Pointer
  - Process state and priority
  - Resources allocated
  - Accounting information

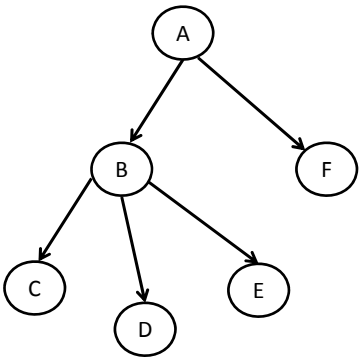
Parent-Child Processes

- A process can create another process or processes
- Parent => child processes
- Child processes inherit the resources from the parent



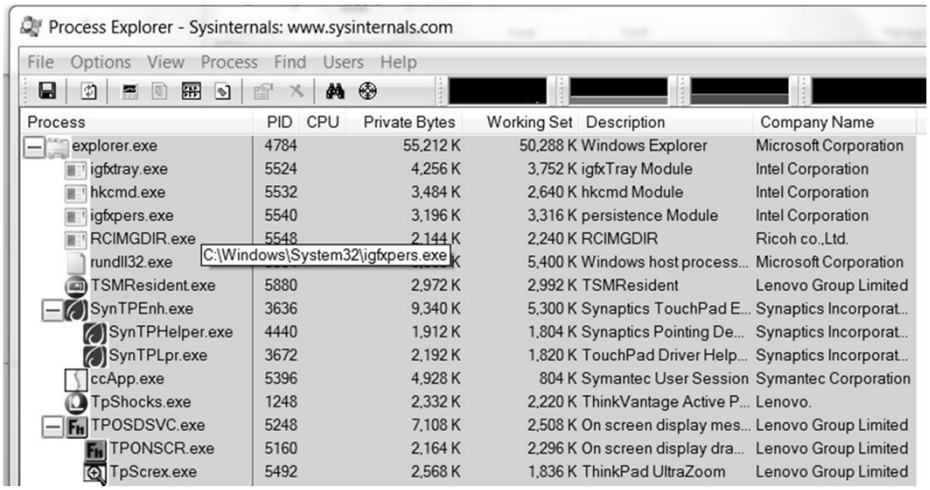
Process Tree/Hierarchy

Parents/Child processes



- A spawns 2 child processes, B and F
- B executes first and spawns 3 child processes C,D,E
- The child processes execute and complete, B completes
- F then is executed

Parent-Child Processes



Parent-Child Processes

% pstree

```
init-+-NetworkManager---{NetworkManager}
|-firefox-+-plugin-containe-+-npviewer.bin---7*[{npviewer.bin}]
|       | `-{plugin-contain}
| `-24*[{firefox}]
|-gconfd-2
|-gdm-binary-+-gdm-simple-slav-+-Xorg
|       |       |-gdm-session-wor-+-gnome-session-+-applet.py
|       |       |       |       |-bluetooth-apple---2*[{bluetooth-appl}]
|       |       |       |       |-cairo-dock---{cairo-dock}
|       |       |       |       |-compiz-+-sh---unity-window-de---{unity-window-d}
|       |       |       |       | `-{compiz}
|       |       |       |       |-evolution-alarm---{evolution-alar}
|       |       |       |       |-gdu-notificatio
|       |       |       |       |-gnome-do---gnome-do---9*[{gnome-do}]
|       |       |       |       |-gnome-panel---{gnome-panel}
|       |       |       |       |-gnome-power-man---{gnome-power-ma}
|       |       |       |       |-nautilus---3*[{nautilus}]
|       |       |       |       |-nm-applet---{nm-applet}
|       |       |       |       |-polkit-gnome-au---2*[{polkit-gnome-a}]
|       |       |       |       |-python
|       |       |       |       |-python---{python}
|       |       |       |       |-ssh-agent
|       |       |       |       |-update-notifier---{update-notifie}
|       |       |       |       |-vino-server---{vino-server}
|       |       |       |       |-zeitgeist-datah---{zeitgeist-data}
|       |       |       |       | `-2*[{gnome-session}]
|       |       |       | `-{gdm-session-wo}
|       |       | `-{gdm-simple-sla}
| `-{gdm-binary}
```

Parent-Child Processes

% ps axjf

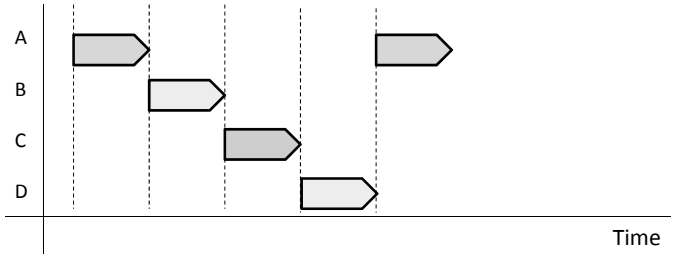
```
PPID PID PGID SID TTY TPGID STAT UID TIME COMMAND
1 962 962 962 ? -1 Ssl 0 0:10 gdm-binary
962 1114 962 962 ? -1 Sl 0 0:00 \_ /usr/lib/gdm/gdm-simple-slave --display
1114 1133 1133 1133 tty7 1133 Ss+ 0 489:41 \_ /usr/bin/X :0 -nr -verbose -auth /v
1114 1559 962 962 ? -1 Sl 0 0:00 \_ /usr/lib/gdm/gdm-session-worker
1559 1572 1572 1572 ? -1 Ssl 1000 0:08 \_ gnome-session --session=classic
1572 1648 1648 1648 ? -1 Ss 1000 0:03 \_ /usr/bin/ssh-agent /usr/bin
1572 1691 1572 1572 ? -1 Sl 1000 368:10 \_ /usr/bin/compiz
1691 1912 1912 1912 ? -1 Ss 1000 0:00 | \_ /bin/sh -c /usr/bin/com
1912 1913 1912 1912 ? -1 Sl 1000 20:01 | \_ /usr/bin/unity-wind
1572 1708 1572 1572 ? -1 Sl 1000 0:04 \_ nm-applet --sm-disable
1572 1709 1572 1572 ? -1 Sl 1000 1:03 \_ zeitgeist-datahub
1572 1719 1572 1572 ? -1 Sl 1000 0:00 \_ /usr/lib/evolution/2.32/evo
1572 1720 1572 1572 ? -1 Sl 1000 2:23 \_ gnome-panel
1572 1721 1572 1572 ? -1 SLl 1000 20:58 \_ /usr/lib/vino/vino-server -
1572 1750 1572 1572 ? -1 Sl 1000 0:18 \_ gnome-power-manager
1572 1902 1572 1572 ? -1 S 1000 0:05 \_ /usr/lib/gnome-disk-utility
1572 2051 1572 1572 ? -1 S 1000 0:23 \_ /usr/bin/python /usr/share/
1572 2201 1572 1572 ? -1 Sl 1000 0:22 \_ update-notifier
1572 11740 1572 1572 ? -1 Sl 1000 39:59 \_ nautilus
1 13080 1572 1572 ? -1 Sl 1000 0:02 gnome-terminal
13080 13084 1572 1572 ? -1 S 1000 0:00 \_ gnome-pty-helper
13080 13085 13085 13085 pts/0 13248 Ss 1000 0:00 \_ bash
13085 13248 13248 13085 pts/0 13248 R+ 1000 0:00 \_ ps axjf
```

Single Tasking

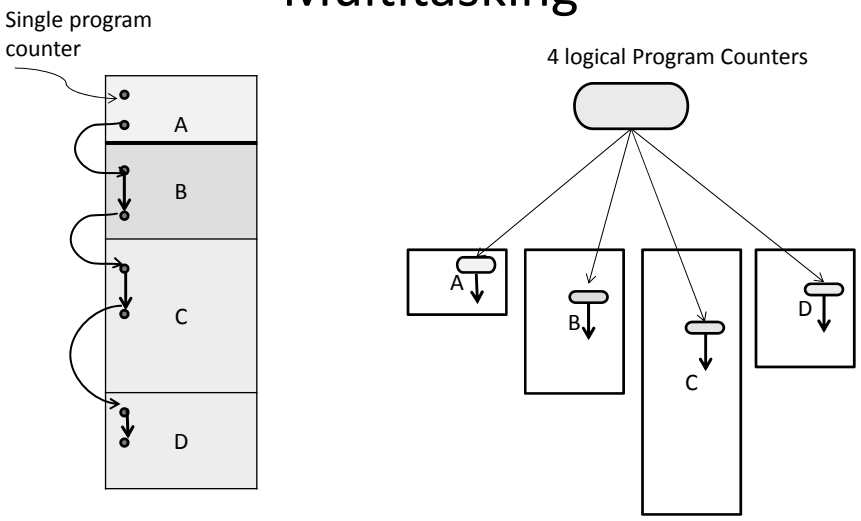
- Mono programming
- Only one process is run at any time (No switching!!!)
- E.g. MS-DOS, Embedded Software

Multitasking

- CPU switches back-and-forth from process to process (very fast)
- Appearance of running multiple programs at the same time (in fact, you're not!)
- A single processor may be shared with several processes using a scheduling algorithm



Multitasking



Process Switching

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly language procedure saves registers.
4. Assembly language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler decides which process is to run next.
7. C procedure returns to the assembly code.
8. Assembly language procedure starts up new current process.

It takes time for the CPU to switch from one process to the other!

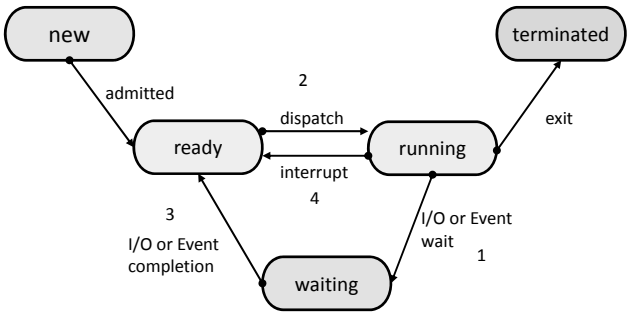
Multitasking Systems

- Cooperative Multitasking
  - Each process is given a timeslot (equal/unequal)
  - Process works through timeslot until request to release
  - Process may release or continue
- Preemptive Multitasking
  - Each process is given a timeslot (equal/unequal)
  - Process works through timeslot
  - When time is up, process switches
  - If infinite loop detected, process switches
  - If important task is required, process switches

Process States

- A process changes state in its life.
- At any one time, a process may be in one of the following states:
  - New (the process is new)
  - Running (the process is running)
  - Waiting (the process is waiting for I/O to be completed)
  - Ready (the process is ready to be executed by CPU)
  - Terminated (the process has completed, exit)

Process States

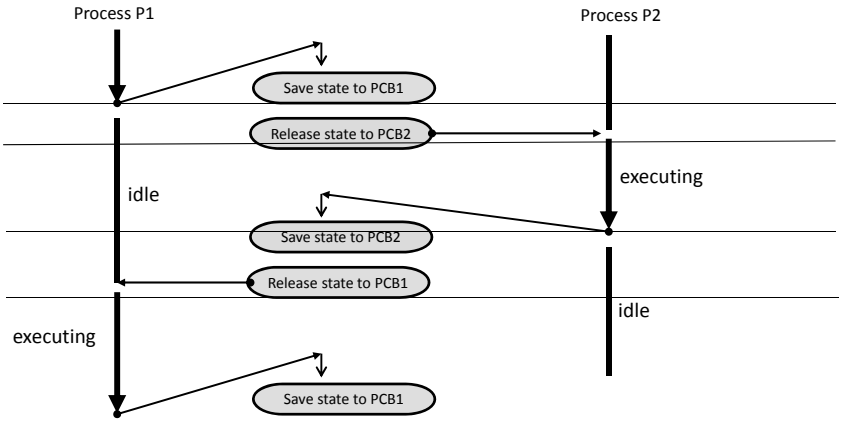


Note direction of the arrows which shows the transitions Between states

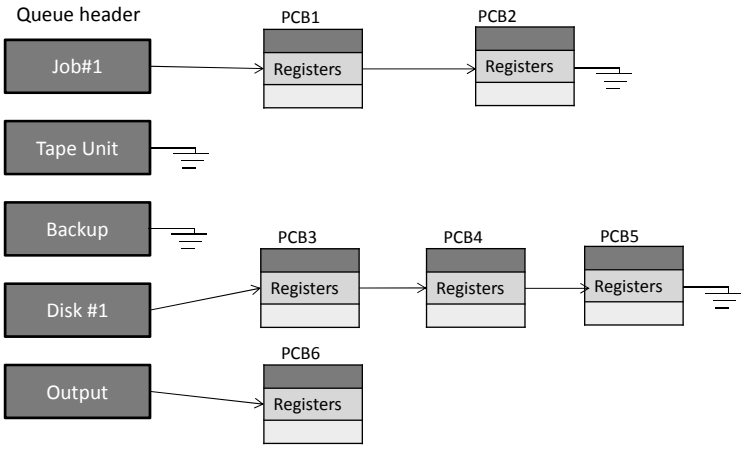
Process Scheduling

- CPU switches from Process-to-process
- Only one process can be truly active at a time.
- CPU must share its time between executions of many processes
- Scheduling – the process of sharing time
- Purpose: to maximize CPU utilization in multiprogramming environment
- Provides illusion that CPU is always with the user/task.

Scheduling 2 processes



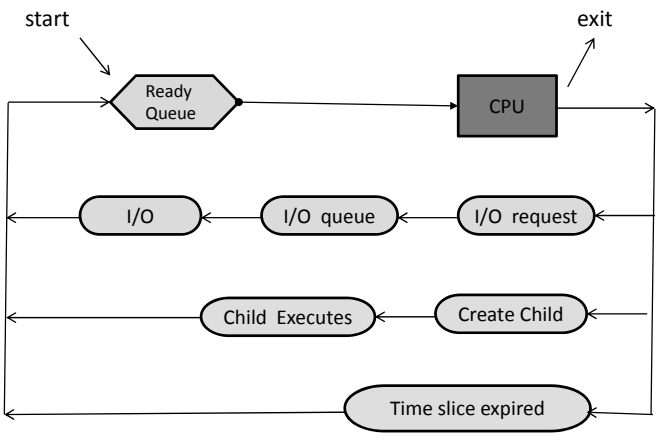
Process Queues



OS Process Queues

- Not very useful for multitasking but for scheduling devices.
- Processes are placed on the ready queue.
- Process is loaded onto ready queue
  - Process starts executing
  - I/o required, placed on I/O queue
  - Time slice expires, state is saved
  - Process keeps executing until complete Process Queues

Process Queues



Questions?