Processes & Process Management

A process may be defined as follows:

- A program in execution
- An instance of a program running on a computer
- The entity that can be assigned and executed on a processor

A process consists of an executable program; its associated data (e.g. variables, work space, buffers, etc.) and its execution context (e.g. copies of contents of process registers, PC, data registers, priority, state, etc.).

Processes and the OS

The Operating system:

- Interleaves the execution of several processes to maximize processor utilization while providing reasonable response time
- Allocates resources to processes
- Supports inter-process communication and user creation of processes

Elements of a Process

At any given point in time while the program is executing, the process can be uniquely characterized by a number of elements, including the following:

- **Identifier** – this is a unique identifier associated with this process, to distinguish it from all other processes**.**
- **State** - if the process is currently executing, it is in the running state.
- **Priority** - Priority level relative to other processes.

Elements of a Process (cont.)

- **Program counter - t**he address of the next instruction in the program to be executed.
- **Memory pointers - i**ncludes pointers to the program code and data associated with this process, plus any memory blocks shared with other processes.
- **Context data - t**hese are data that are present in registers in the processor while the process is executing.
- **I/O status information** – this includes outstanding I/O requests, I/O devices (e.g. disk drives) assigned to this process, a list of files in use by the process, etc.
- **Accounting** information – this may include the amount of processor time and clock time used, time limits, account numbers, etc.

Process Control Block

This list is stored in a data structure called a **process control block**

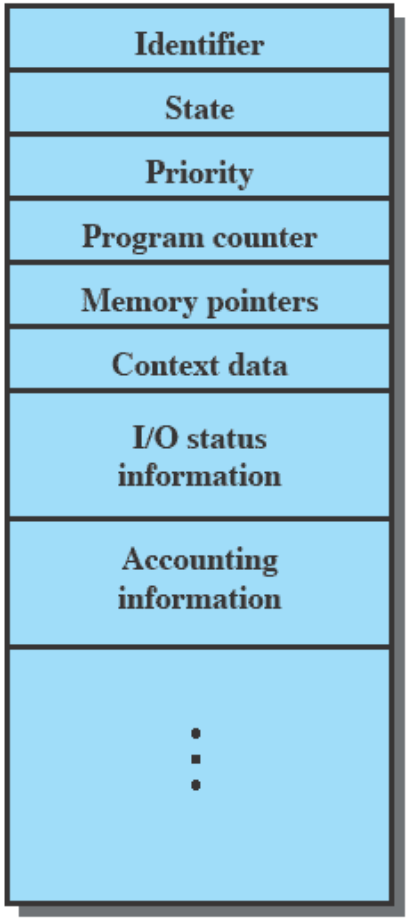
- The process control block contains sufficient information so that it is possible to interrupt a running process and later resume execution as if the interruption had not occurred.
- It is the key tool that enables the OS to support multiple processes and to provide for multiprocessing.
- When a process is interrupted, the current values of the program counter and the processor registers (context data) are saved in the appropriate fields of the corresponding process control block, and the state of the process is changed to some other value, such as *blocked or ready.*

Process Control Block (cont.)

- *The OS is* now free to put some other process in the running state. The program counter and context data for this process are loaded into the processor registers and this process now begins to execute.

- We can say that a process consists of program code and associated data plus a process control block. For a single-processor computer, at any given time, at most one process is executing and that process is in the *running state.*

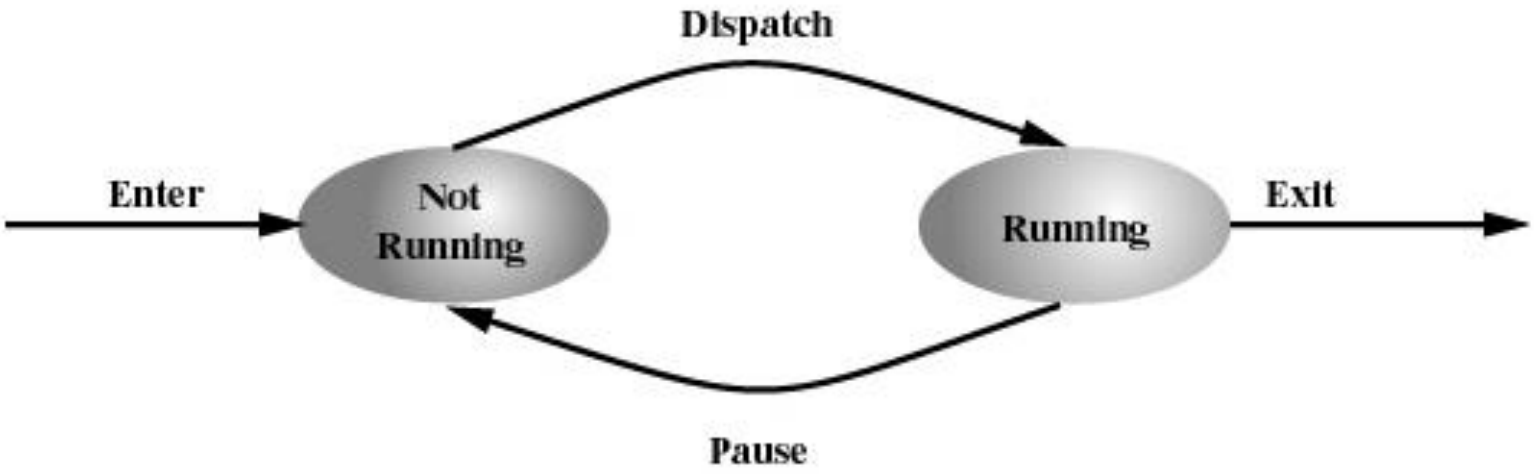
Process Control Block



Process States: 2 State Model

A simple model of process behavior is the 2 state model.

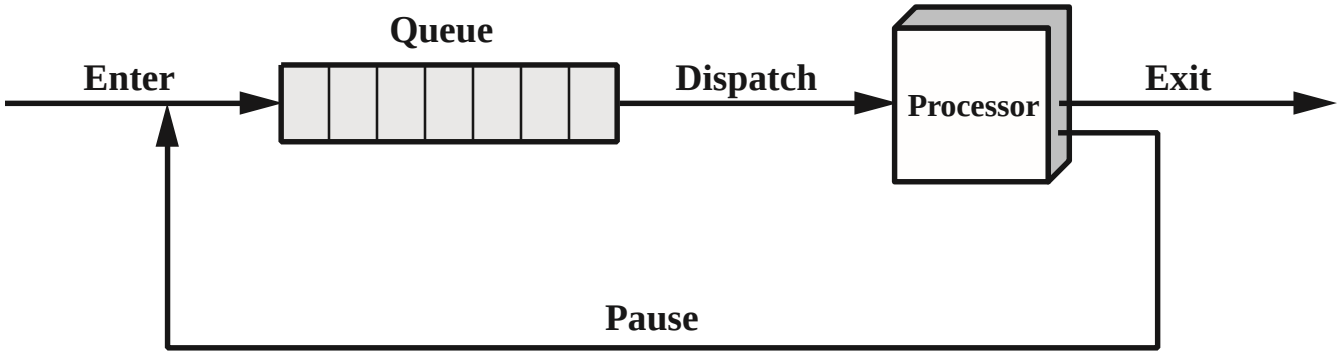
A Process may be in one of two states: Running & Not-running with two transitions Dispatch and Pause



Process States: 2 State Model

The not-running state consists of a number of processes waiting in a queue each waiting for an opportunity.

The dispatcher program switches the processor from one process to another.



(b) Queuing diagram

Creation + Termination of Jobs

Process can be created e.g. a user submits a job

Process can be terminated

Reasons for termination

- Normal completion
- Time limit exceeded
- Memory unavailable
- Bounds violation
- Protection error - example: write to read-only file
- Arithmetic error

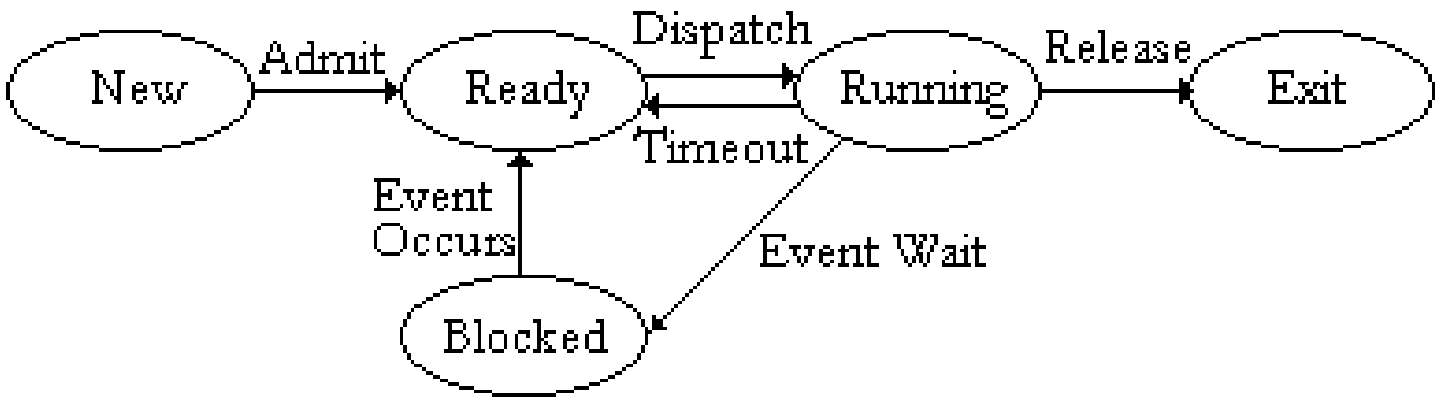
Termination of Jobs (cont.)

- Time overrun - process waited longer than a specified maximum for an event
- I/O failure
- Invalid instruction - happens when try to execute data
- Privileged instruction
- Data misuse
- Operating system intervention - such as when deadlock occurs
- Parent terminates so child processes terminate
- Parent request

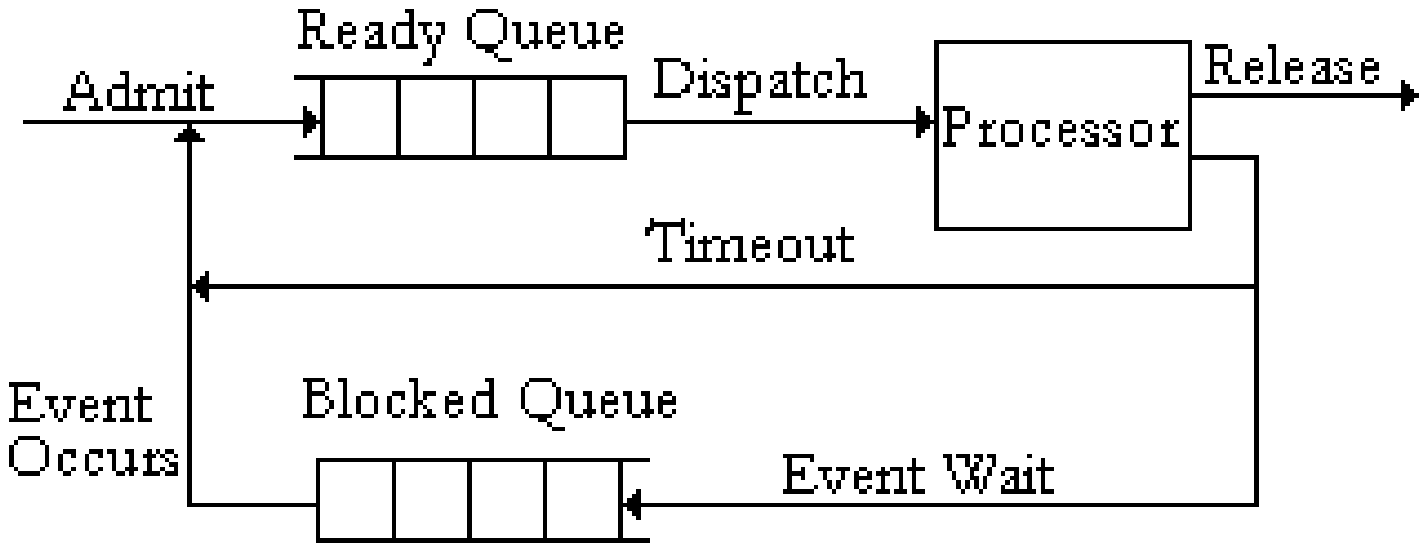
5 – State Model

A running process may be blocked (e.g. if it's waiting for input).

A more representative model of process activity is the 5 state model below.



5 – State Model



Process States

- *Running*: currently being run
- *Ready*: ready to run
- *Blocked*: waiting for an event (I/O)
- *New*: just created, not yet admitted to set of runnable processes
- *Exit*: completed/error exit

Process State Transitions

- ***Ready*** ***Running***
  When the process is chosen to use the cpu.

- ***Running*** ***Ready***
  When a process has completed its allowed cpu time slice.

- ***Running*** ***Blocked***
  A process makes a request for something that it must wait for e.g I/O request.

- ***Blocked*** ***Ready***
  The event for which the process is waiting occurs e.g. I/O completed.

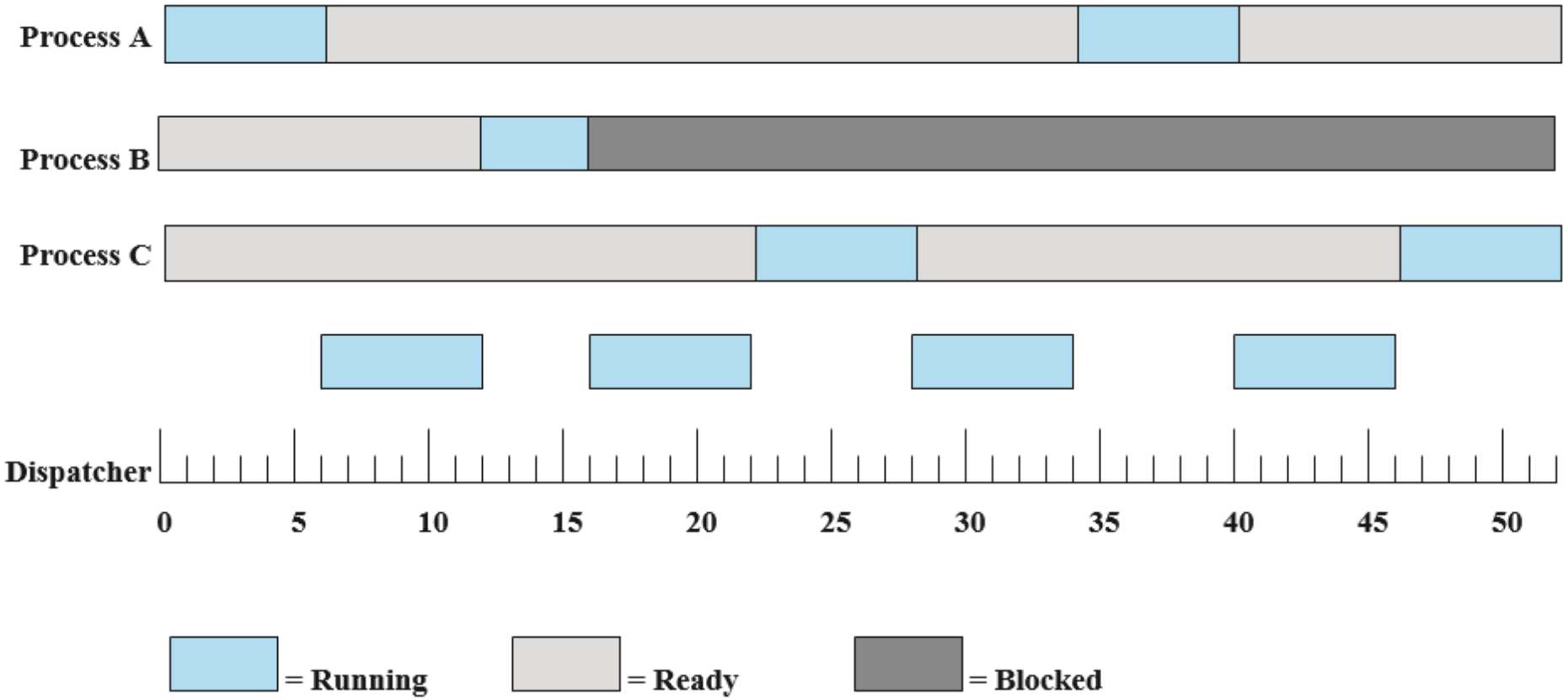
Process Transitions Example

In the following slide, three processes (& the dispatcher) are taking turns.

When process B is blocked the other processes continue to execute alternatively until it becomes unblocked.

The dispatcher which facilitates the switching between processes also requires cpu time.

Process Transitions Example



Suspended Processes

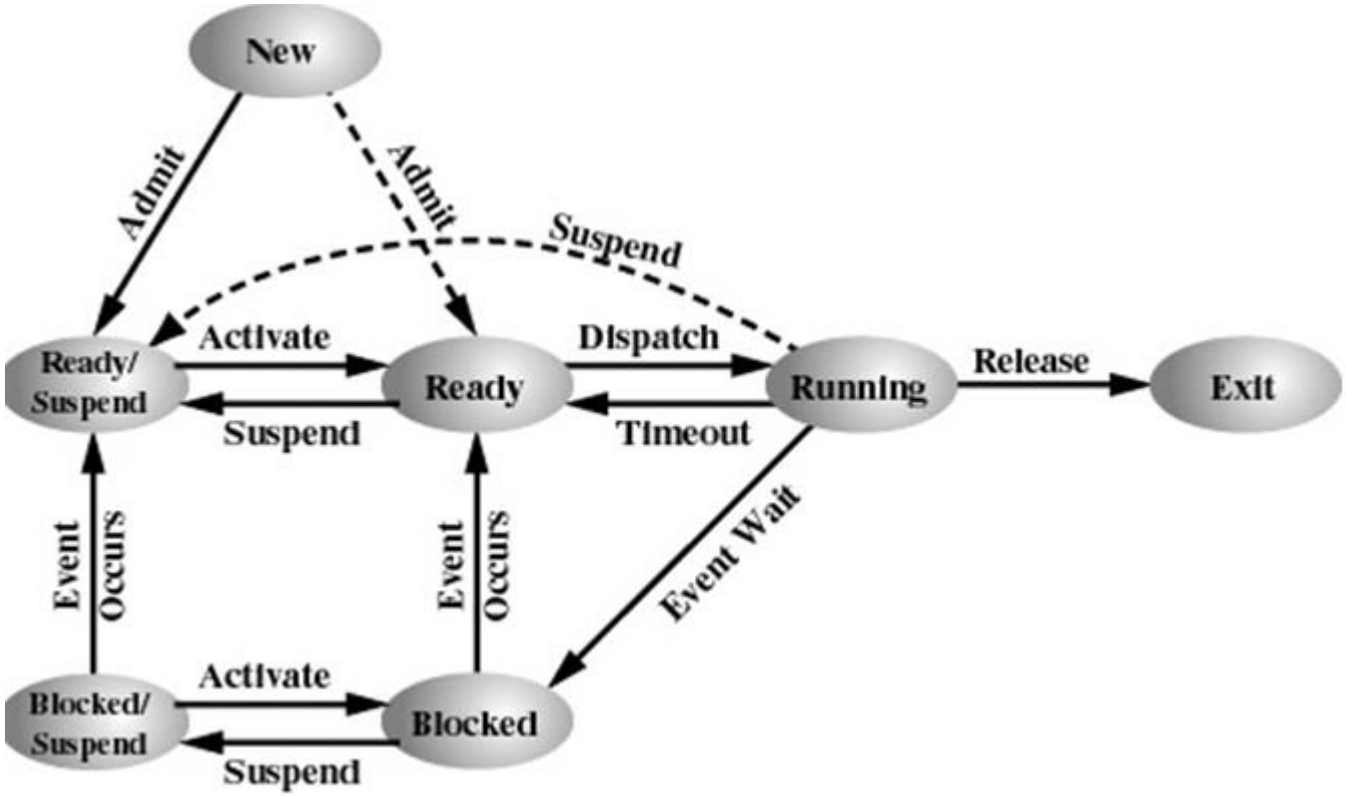
Until now, it is assumed all active processes are loaded into main memory.

- Processor is faster than I/O so all processes could be waiting for I/O
- Swap these processes to disk to free up more memory
- Blocked state becomes suspend state when swapped to disk
- Two new states:

Blocked, suspend Process is on disk and awaiting an event.

Ready, suspend Process is in disk and is available for execution (when loaded into main memory)

Suspended Processes



Suspend Transitions

- **Blocked - Blocked Suspend** – Swap out a blocked process to free memory
- **Blocked Suspend - Ready Suspend** – Event occurs (O.S. must track expected event)
- **Ready Suspend - Ready** – Activate a process (higher priority, memory available)
- **Ready - Ready Suspend** – Need to free memory for higher-priority processes
- **New Ready or Ready Suspend** – Can choose where to put new processes
- **Blocked Suspend - Blocked** – Reload process expecting event soon
- **Running Ready - Suspend** – Preempt for higher-priority process

Processes & Resources (cont.)

Each process P1, P2, etc needs access to resources such as I/O, processor, etc.

To manage these resources the operating system needs to have certain information about the current status of each process and resource.

Tables are constructed for each entity the operating system manages.

Memory and I/O Tables

Memory Tables

- Allocation of main memory to processes
- Allocation of secondary memory to processes
- Protection attributes for access to shared memory regions
- Information needed to manage virtual memory

I/O Tables

- I/O device is available or assigned.
- Status of I/O operation.
- Location in main memory being used as the source or destination of the I/O transfer.

File and Process Tables

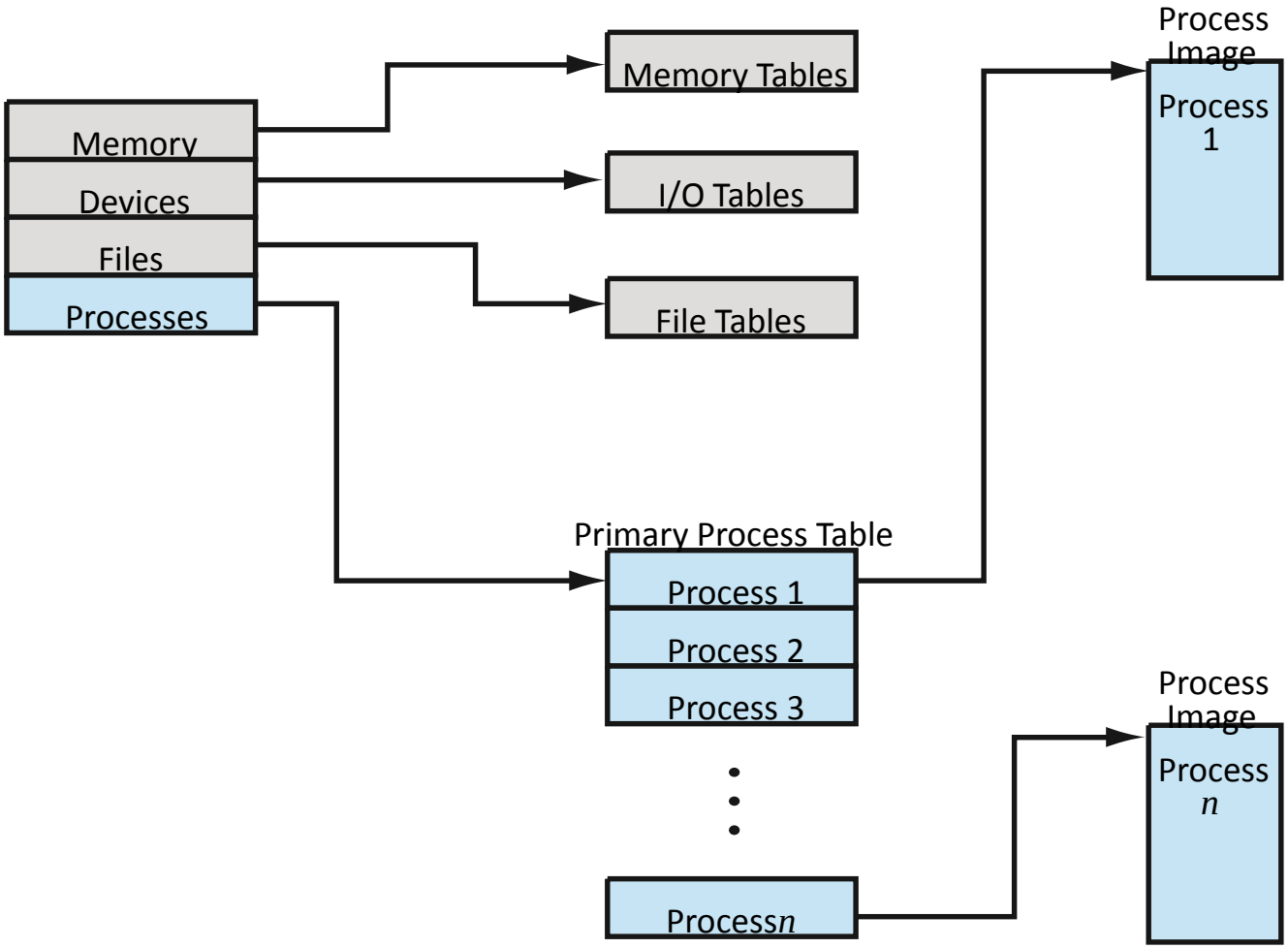
File Tables

- Existence of files
- Location on secondary memory
- Current Status
- Attributes
- Sometimes this information is maintained by a file-management system

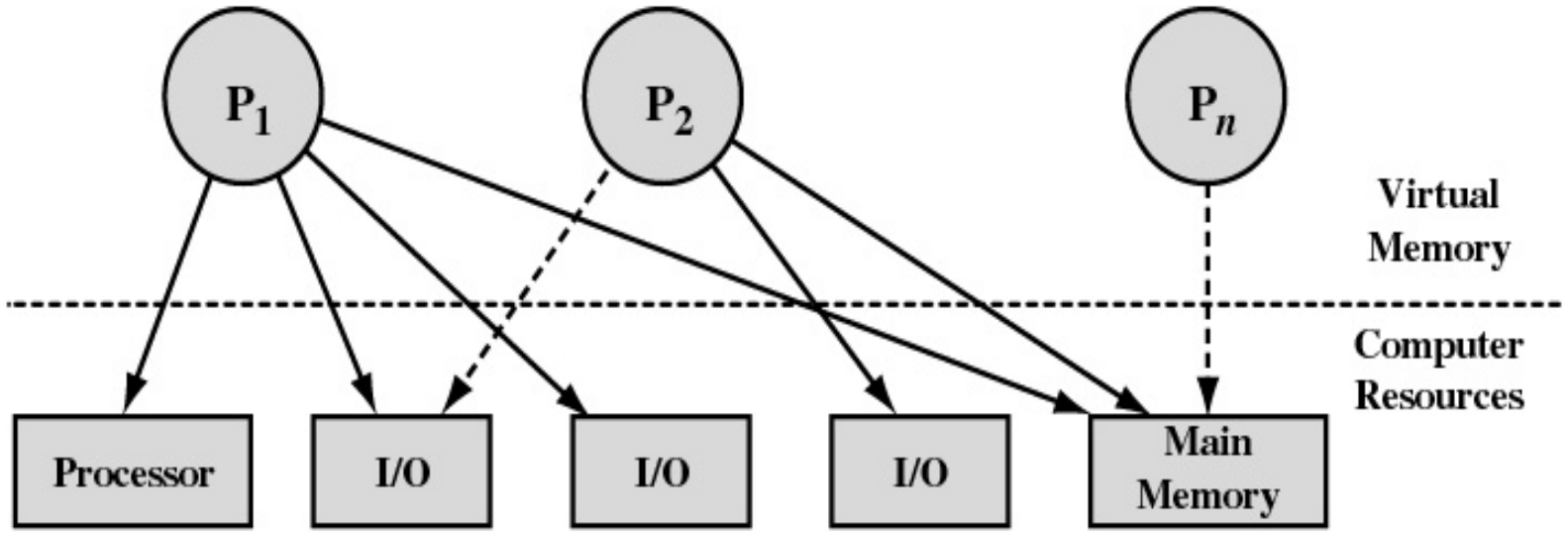
Process Table

- Where process is located
- Attributes necessary for its management
- Process ID
- Process state
- Location in memory

Process Tables

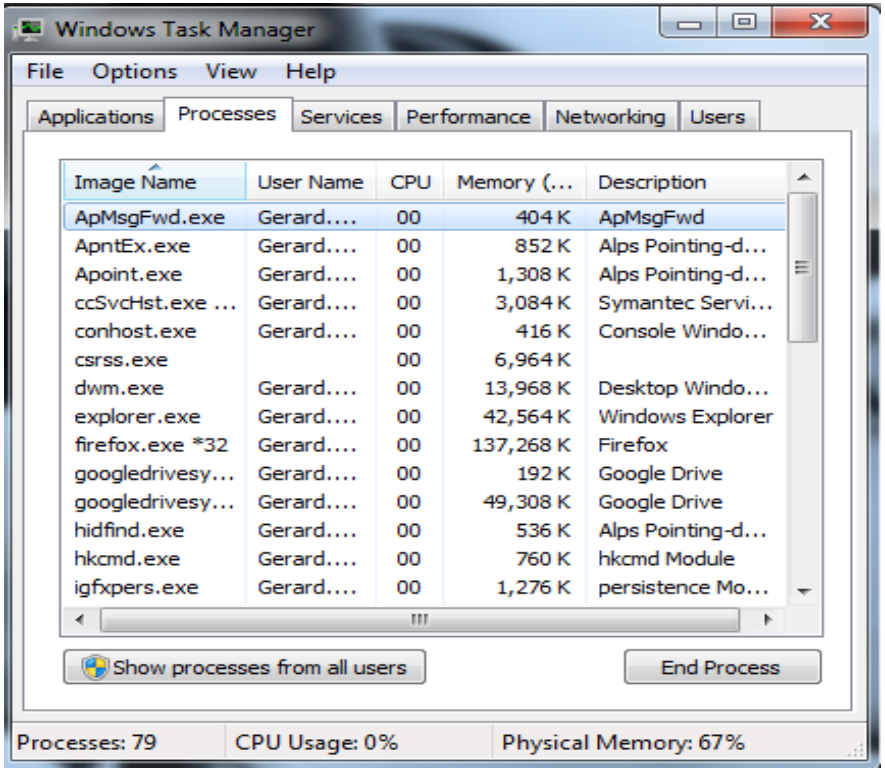


Processes & Resources



Windows Processes

In windows, you can run the task manager to see which processes are running at any time.



Windows Processes (cont.)

The task manager lists the process names, CPU usage, memory, etc.

If a system is sluggish – one should look for any process that's using a disproportionate amount of memory or CPU time.

A process can be terminated here (be very careful).

Linux Processes

Every process has a unique process identity - pid

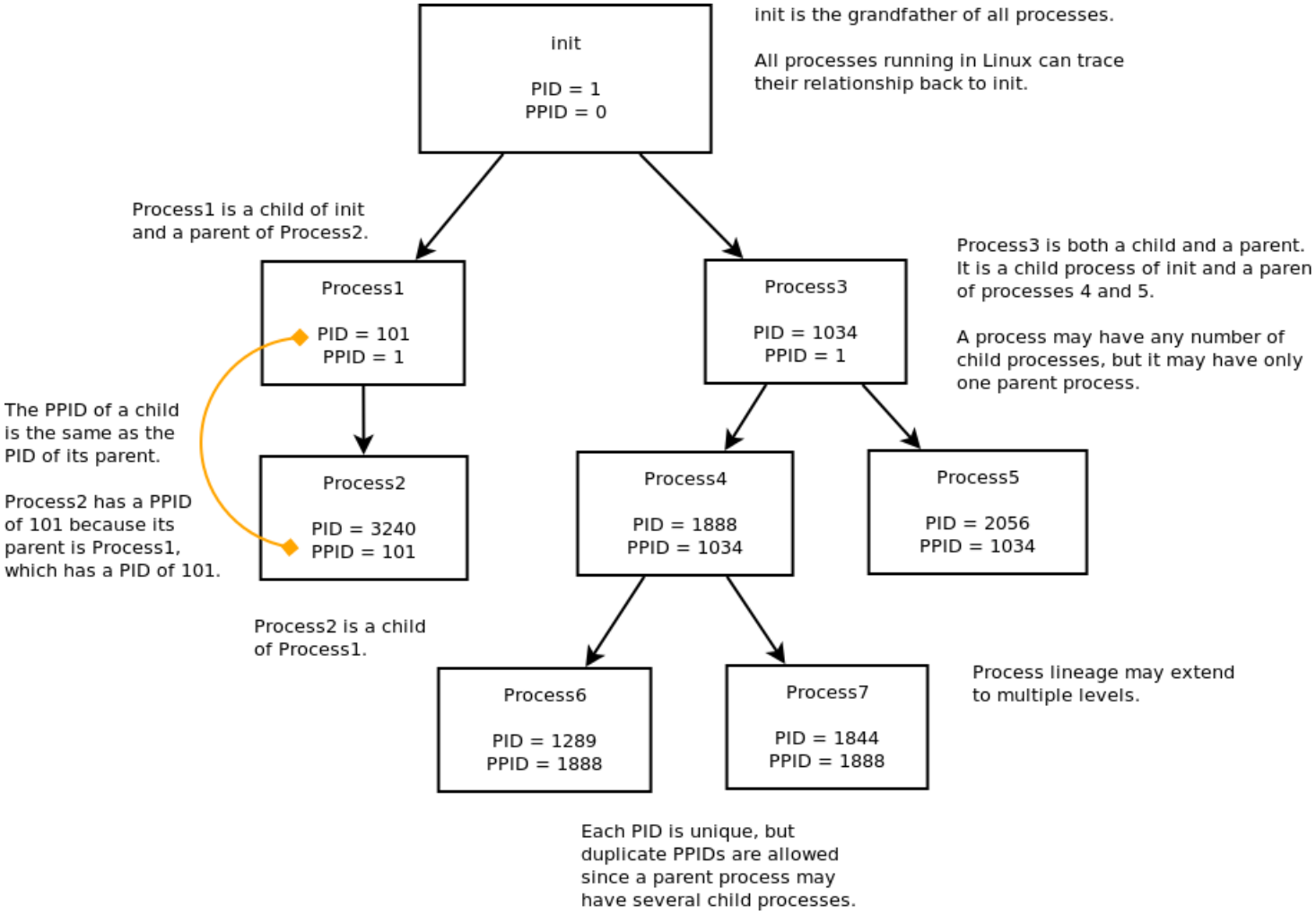
Every process has a parent process identity ppid

The first process has a pid 1 and is the ancestor of all other processes

Therefore, each process can trace itself back to the first process. See next slide:

https://delightlylinux.files.wordpress.com/2012/06/pid.png

Parents & Children



Linux Processes

- The first process (often called **init**) is the ancestor of all subsequent process; init starts running at the end of the boot sequence. It has a **pid** 1

- Each process has a parent and a **ppid** (parent process identity). The parent spawns the child process. The following is a simple example of a process spawning another which in turn spawns another, etc.

PID	**PPID**	**CMD**
26906	25804	csh
26919	26906	ksh
26926	26919	sh
26927	26926	ps

Process Commands

There are a number of commands available for process management.

ps (process status) gives information on the current status of processes

Some variations are:

ps – A

ps –l

ps –x

Enter **man ps** to get more information on this command.

Process Commands (cont.)

kill

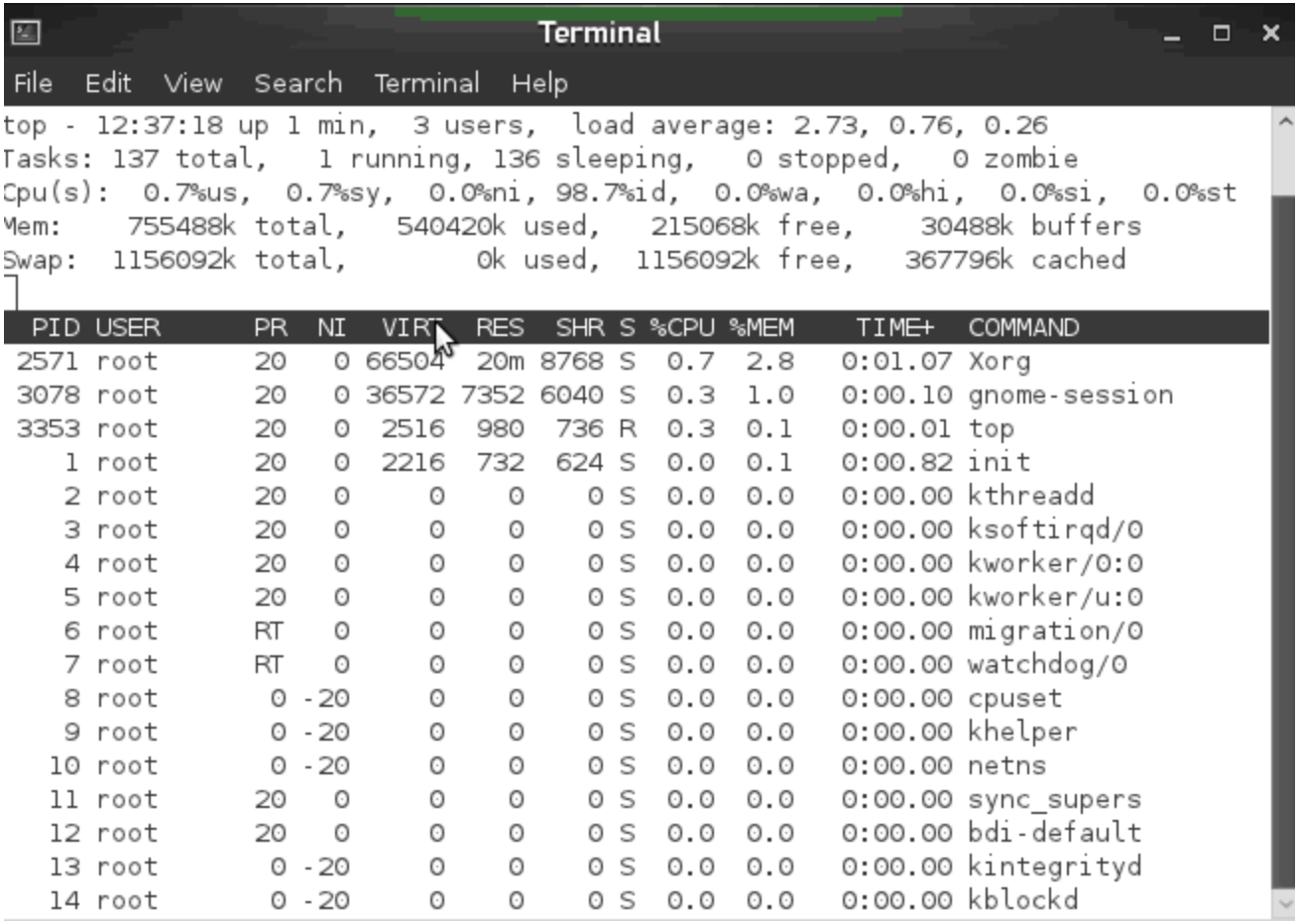
- The command **kill -9 process_pid** kills a process and all its descendants.
- Try killing some processes from the previous example.

Process Commands (cont.)

top

- Display **top cpu** processes This provides an ongoing look at processor activity in real time.
- This displays a useful overview of current activity:

The top Command



The top Command (cont.)

- **uptime - t**his displays the time the system has been up, and the three load averages for the system. The load averages are the average number of process ready to run during the last 1, 5 and 15 minutes.

- **processes - t**he total number of processes running at the time of the last update. This is also broken down into the number of tasks which are running, sleeping, stopped, or undead.

- **CPU states** this shows the percentage of CPU time in user mode, system mode, niced tasks, iowait and idle. (Niced tasks are only those whose nice value is positive.) Time spent in niced tasks will also be counted in system and user time, so the total will be more than 100%.

The top Command (cont.)

- **Mem -** contains statistics on memory usage, including total available memory, free memory, used memory, shared memory, and memory used for buffers.
- **Swap** - contains statistics on swap space, including total swap space, available swap space, and used swap space.
- **VIRT -** Virtual Image (kb) The represents the total amount of **virtual** memory used by the task. It includes all code, data and shared libraries plus pages that have been swapped out.
- **SWAP -** Swapped size (kb)
- This is the swapped out portion of a task's total **virtual** memory image.

The top Command (cont.)

- **RES -** Resident size (kb) The non-swapped **physical** memory a task has used.
- **SHR --** Shared Mem size (kb) This is the amount of **shared** memory used by a task. It simply reflects memory that could be potentially shared with other processes.
- **PR -** Priority This displays the priority of the task.
- **NI -** Nice value This displays the nice value of the task. A negative nice value means higher priority, whereas a positive nice value means lower priority

pstree Command

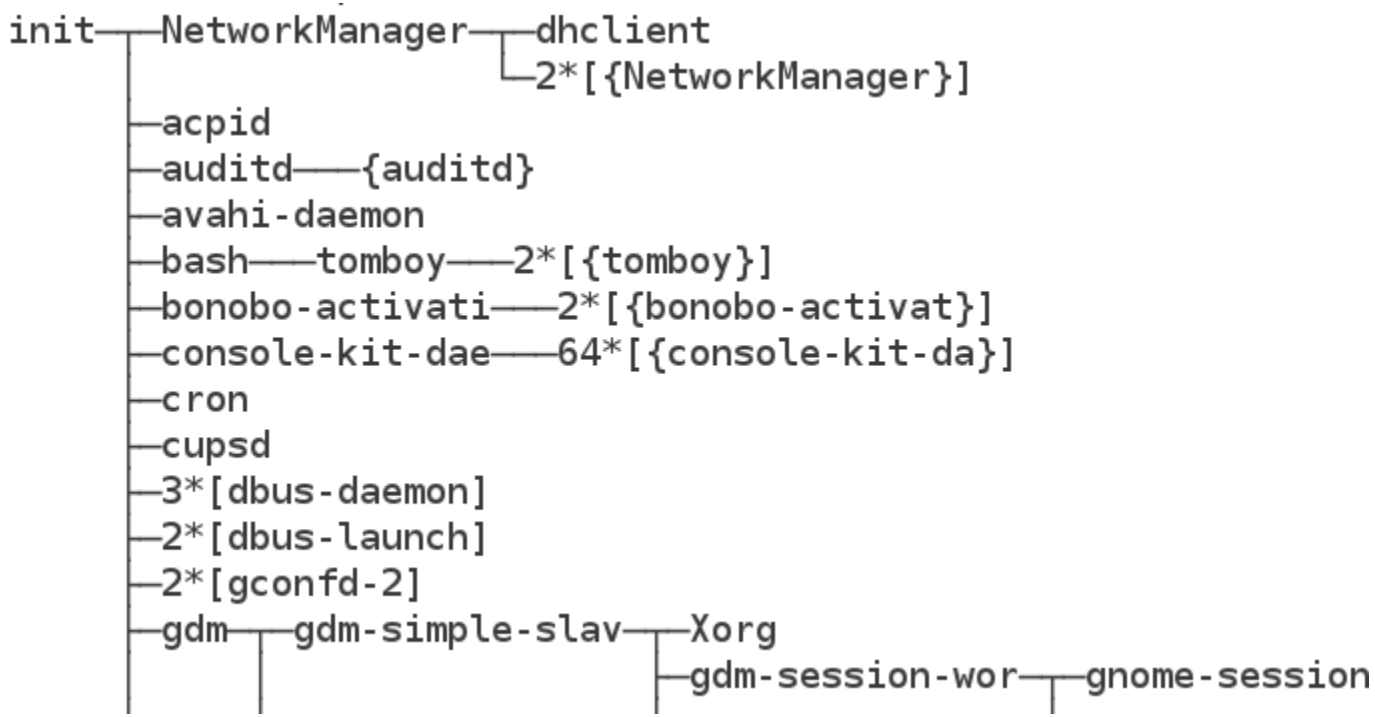
The pstree command displays a list of processes in an ancestral tree format.

It indicates parents and children.

It traces each process back to the first process.

pstree Command (cont.)

Example: pstree output



Linux Daemons

A daemon is a special type of Linux process:

A daemon runs in the background – often with the user unaware of its existence until its required.

Daemons perform a service

A daemon usually starts automatically at boot time.

Linux Daemons (cont.)

A Linux administrator can install, start, stop and remove daemons.

The at daemon can be installed like any other software e.g.

apt-get install at

The administrator can also decide which services to start (or not) automatically.

Examples of Linux Services

- cups – Printing Service, a Linux machine can act as a print server.
- apache – Web Server
- samba – Services for Windows clients
- cron – Scheduling service

See https://wiki.archlinux.org/index.php/Daemons_List **for more.**

The cron daemon

In many distributions the cron daemon starts automatically.

cron is used to schedule a job e.g.

run a backup of the system every day at 23.00

The cron daemon (cont.)

Daemon commands (in SUSE Linux)

To check if cron is running: rccron status

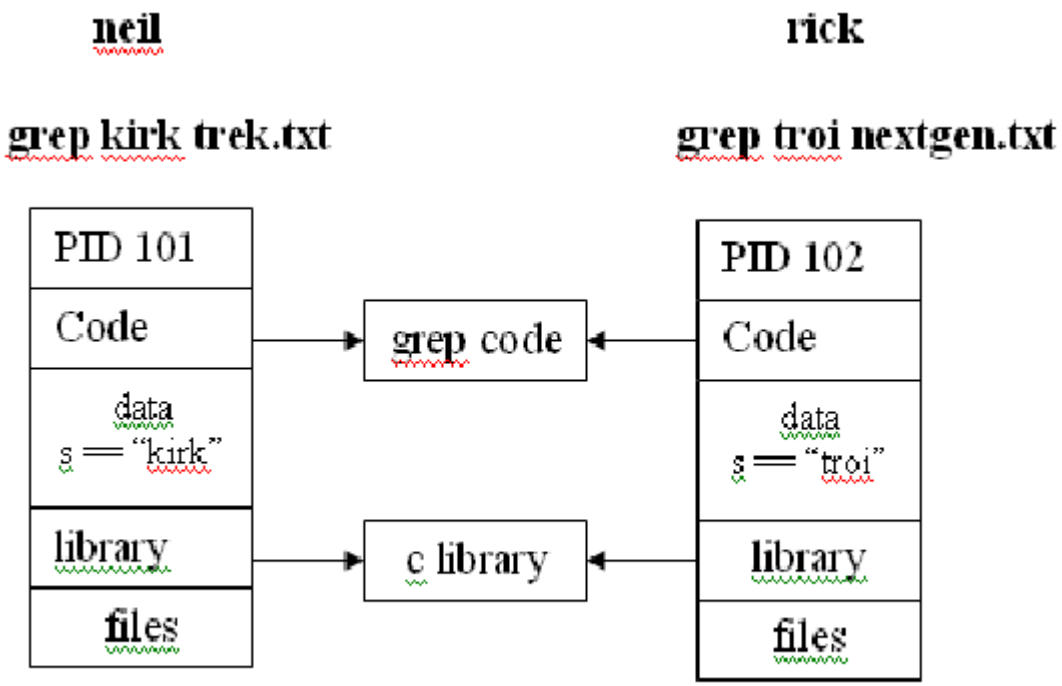
To start cron: rccron start

To stop cron: rccron start

Process Sharing

- Two Users
- Looking for different things in different files
- Sharing grep

Process Sharing



Process Sharing

If we run the **ps** command, described as follows, the output will look something like this:

ps –af

UID	PID	PPID	STIME	TTY	TIME	CMD
rick	101	96	18:24	tty2	00:00:00	grep pid_t/usr/include/sys/*.h
neil	102	9	18:24	tty4	00:00:00	grep XOPEN /usr/include/features.h

Process Sharing

Each process is allocated a PID. The next unused number in sequence is chosen. In this example, the two processes started by **neil** and **rick** have been allocated the identifiers 101 and 102.

Each process calls the cron command.

Process Sharing

- The program code that will be executed by the **grep** command is stored in a disk file. The code is loaded into memory as read-only. This area can't be written to but it can safely be shared.

- The system libraries can also be shared. There need be only one copy of **printf** in memory, even if many running programs call it.

Process Sharing

Not everything can be shared. The variables that it uses are distinct for each process. In this example, we see that the search string passed to the **grep** command appears as a variable, **s**, in the data space of each process.

These are separate and usually can't be read by other processes. The files that are being used in the two **grep** commands are also different.