Lecture 3#

- Processes and Job Control
- Privileged ,User and Group Accounts
- Logs and audits

Processes and Job Control

- A Process is simply an instance of a running a program
- A process is said to be born when program starts executions and remains alive as long as the program is active. So after execution is complete a process is said to be die.
- A process also has name, usually the name of the program being executed. For e.g. when you execute the grep command a process name grep is created
- However a process can be considered synonymous with a program when 2 users run the same program, there one program on disks but 2 process in memory

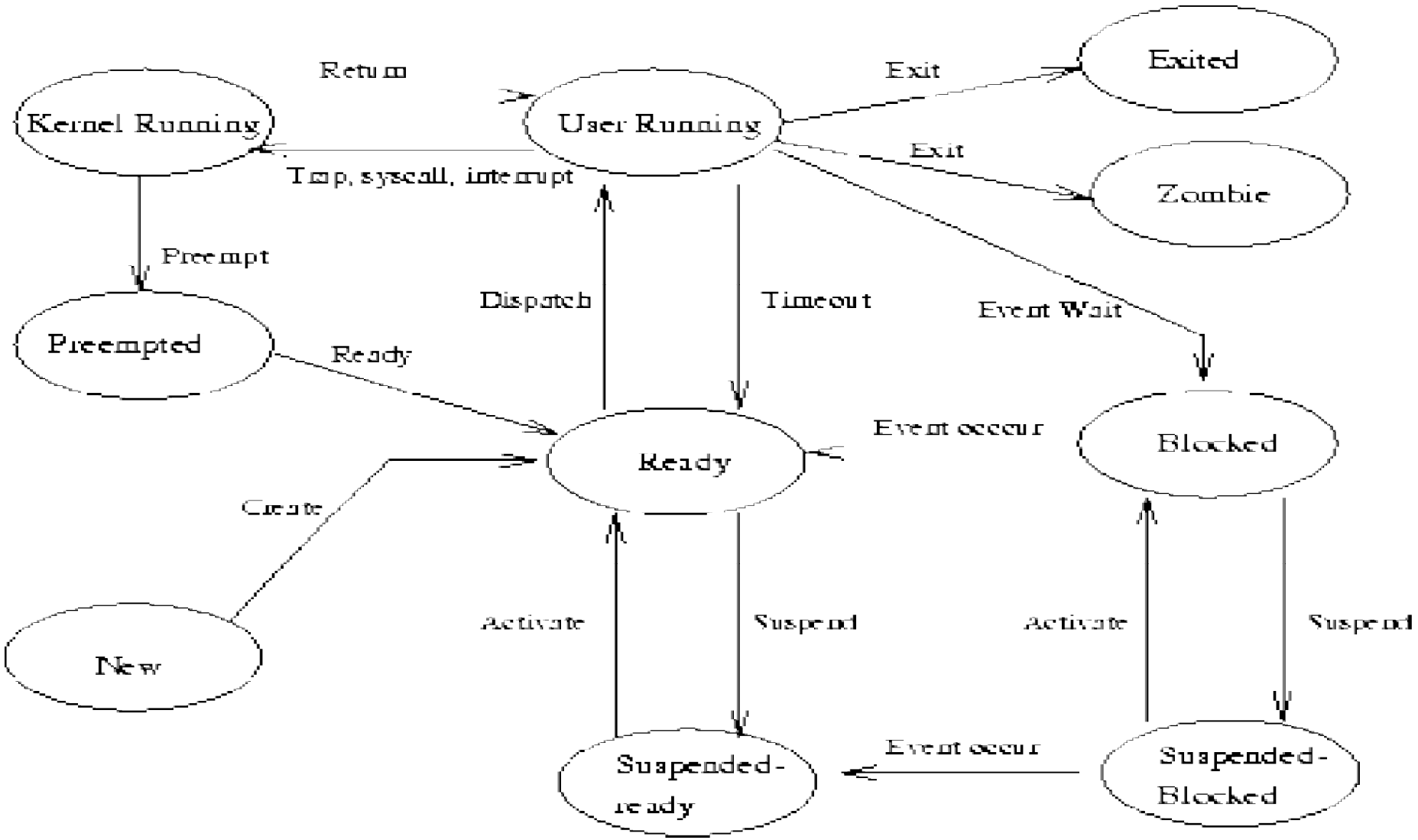
1The UNIX process model

- New State:- the process being created
- Ready State: - the process is waiting to be assigned to a processor.
- Blocked State:- the process is in main memory and waiting for an events
- Blocked Suspended: - the process is in secondary memory and waiting for an event.
- Suspend / Ready:- the process is in secondary memory but is available for execution as soon as it is loaded into memory

- Exited State: - it is a process for which parent have decided not to waits for them after a process die for their cleaning purpose. So this process does not have any entry in process table.
- Zombie:- it is a process for which there parent have to waits for the completions (possibly to clean after them) and system maintain its entry in process tables
- Preempted State:- it is a special case pf blocked state a process retaining from system calls (hence after having run in kernel modes) to immediately blocked and put the ready process queue instead of returning ,leaving CPU to another process.

- All processes in UNIX are created using the fork() system call . UNIX implements through the fork() and exec() system calls an elegant two-step mechanism for process creation and execution. fork() is used to create the image of a process using the one of an existing one, and exec is used to execute a program by overwriting that image with the program's one.

- A call to fork() of the form:
- #include <sys/types.h>pid_t childpid;...childpid = fork(); /* child's pid in the parent, 0 in the child */... creates (if it succeeds) a new process, which a child of the caller's, and is an exact copy of of the (parent) caller itself. By exact copy we mean that it's image is a physical bitwise copy of the parent's


Return
Exit
Exited
Kernel Running
User Running
Exit
Trap, syscall, interrupt
Zombie
Preempt
Dispatch
Timeout
Event Wait
Preempted
Ready
Event occeur
Blocked
Ready
Create
New
Activate
Suspend
Activate
Suspend
Suspended-
ready
Event occeur
Suspended-
Blocked

UNIX Command for process:-

- Every Unix process has a process ID (PID) which can be used to refer to it, suspend it or kill it entirely
- Processes can be stopped and started, or killed once and for all. The kill command does this and more. The kill command takes a number called a *signal* as an argument and another number
- called the *process identifier* or *PID* for short. Kill send signals to processes. Some of these are fatal and some are for information only. The two commands
- kill -15 127
- kill 127
- are identical. They both send signal 15 to PID 127. This is the normal *termination* signal and it is often enough to stop any process from running.

Process management command

- ps :To display the currently working processes
- top :Display all running process
- kill pid :Kill the process with given pid
- killall proc: Kill all the process named proc
- pkill pattern :Will kill all processes matching the pattern
- bg :List stopped or background jobs,resume a stopped job in the background
- fg: Brings the most recent job to foreground

- fg n: Brings job n to the foreground
- ps –x: gives information about currently-running processes that you own. These may be from other UNIX sessions than your current UNIX session. The name of each process is in the far right column, and the process id for each process is in the first column. (BEWARE: the options for ps vary on different flavors of UNIX and Linux

Process control:

		sample usage
`file`	*type of file*	`file *`
`jobs`	*in current shell*	`jobs`
`kill`	*stop process*	`kill 23098` `kill -9 23098`
`nice`	*process priority*	`nice ./a.out`
`ps`	*list processes*	`ps -ef | sort | more`
`which`	*command filename*	`which netscape`

Child processes and zombies

- When we start a process, the new process becomes a *child* of the original. If one of the children starts a new process then it will be a child of the child (a grandchild). Processes therefore form *hierarchies*. Several children can have a common *parent*.
- All Unix user-processes are children of the initial process init, with process ID 1. If we kill a parent, then (unless the child has detached itself from the parent) all of its children die too. If a child dies, the parent is not affected. Sometimes when a child is killed, it does not die but becomes *defunct* or a *zombie* process.

- This means that the child has a parent which is *waiting* for it to finish. If the parent has not yet been informed that the child has died, because it has been suspended itself for instance, then the dead child is not completely removed from the kernel's process table. When the parent wakes up and receives the message that the child has terminated (and its exit status), the process entry for the dead child can be remove
- Most UNIX processes go through a zombie state, but most terminate so quickly that they cannot be seen.
- It is not possible to kill a zombie process, since it is already dead. The only way to remove a zombie is to either reactivate the process which is waiting for it, or to kill that process

The Windows process model

- Like UNIX, processes under Windows/NT can live in the foreground or in the background, though unlike UNIX, Windows does not fork processes by replicating existing ones. A background process can be started with
- start /B
- In order to kill the process it is necessary to purchase the Resource kit which contains a kill command. A background process detaches itself from a login session and can continue to run even when the user is logged out.

Privileged accounts

- Operating systems that restrict user privileges need an account which can be used to configure and maintain the system. Such an account must have access to the whole system, without regard for restrictions. It is therefore called a privileged account.
- In UNIX the privileged account is called root, also referred to colloquially as the super-user. In Windows, the Administrator account is similar to UNIX’s root, except that the administrator does not have automatic access to everything as does root. Instead he/she must be first granted access to an object. However the Administrator always has the right to grant them self access to a resource

- No one should use a privileged root or Administrator account as a user account. To do so is to place the system in jeopardy. Privilege should be exercised only when absolutely necessary.

Logs and audits

- Operating system kernels share resources and offer services. They can be asked to keep lists of transactions which have taken place so that one can later go back and see exactly what happened at a given time. This is called logging or auditing
- Full system auditing involves logging every single operation that the computer performs. This consumes vast amounts of disk space and CPU time and is generally inadvisable unless one has a specific reason to audit the system.

- Auditing has become an issue again in connection with security. Organizations have become afraid of break-ins from system crackers and want to be able to trace the activities of the system in order to be able to look back and find out the identity of a cracker.
- One use for auditing is so-called non-repudiation, or non-denial.