BCA: II SEM:IV
UNIX SHELL PRGRAMMING

Dr.T.LOGESWARI
DEPT OF COMPUTER SCIENCE
NEW HORIZON COLLEGE

Unix Programming

Unit –I :- Introduction

- UNIX is a computer Operating System which is capable of handling activities from multiple users at the same time. Unix was originated around in 1969 at AT&T Bell Labs by Ken Thompson and Dennis Ritchie.

What is Unix ?

- The UNIX operating system is a set of programs that act as a link between the computer and the user.
- The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the operating system or kernel.
- Users communicate with the kernel through a program known as the shell.
- The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

Feature of Unix

Feature of Unix

- Unix is a computer operating system. The salient feature of Unix Operating system are as follows
  - Multi User
  - Multi Tasking
  - Portability
  - Communication
  - Security
  - Machine Independent

- Multi User
  - It is a multi user operating system
  - It support multiple user at time(it share the resource like printer, hard disk etc)
  - The main computer is connected to a number of terminals, which in turn consist of a keyboard and monitor
  - There are several types of terminals can be connected to the server. They are
    - Dumb Terminals (consist of keyboard & monitor do not have hard disk or memory
    - Terminals Emulator(consist of own memory, MP and disk drive it transmit its job to server for processing)
    - Dial in terminals (Connect to server through telephone lines

- Multi tasking
  - This feature enables Unix Os to support concurrent execution of multiple processes
  - Each command takes small fraction of times to process.
- Portability
  - The Unix operating system is highly portable
  - It can easily be implemented on different hardware platform with little or no modification
- Communication
  - It is highly effective in Unix
  - Communication can be established between user working on different terminals connected to the same or different machine

- Security
  - Unix is a multi user system, so there might be possibility of intruder
  - So unix provide high security
  - First at the system start up level it provide login names and password for user
  - Second file protection is provided by granting different permission to each file. Read = only to read ; write = permission only to write in a file
  - Unix support file encryption(unreadable format)

  When required you decrypt

- Machine Independent
  - It hides the machine architecture from the user. So it is easier to write a program that run on different hardware implementation
  - Unix Os treats everything including memory and I/O devices as files
  - The Unix file system allows easy and efficient maintenance of files

- Paging
  - If the available memory is full. LINUX then look for 4kb pages of memory which can be freed. Pages, with content already on hard disk are discarded. If one of the pages is to be accessed, it has to be reloaded. This procedure is called paging

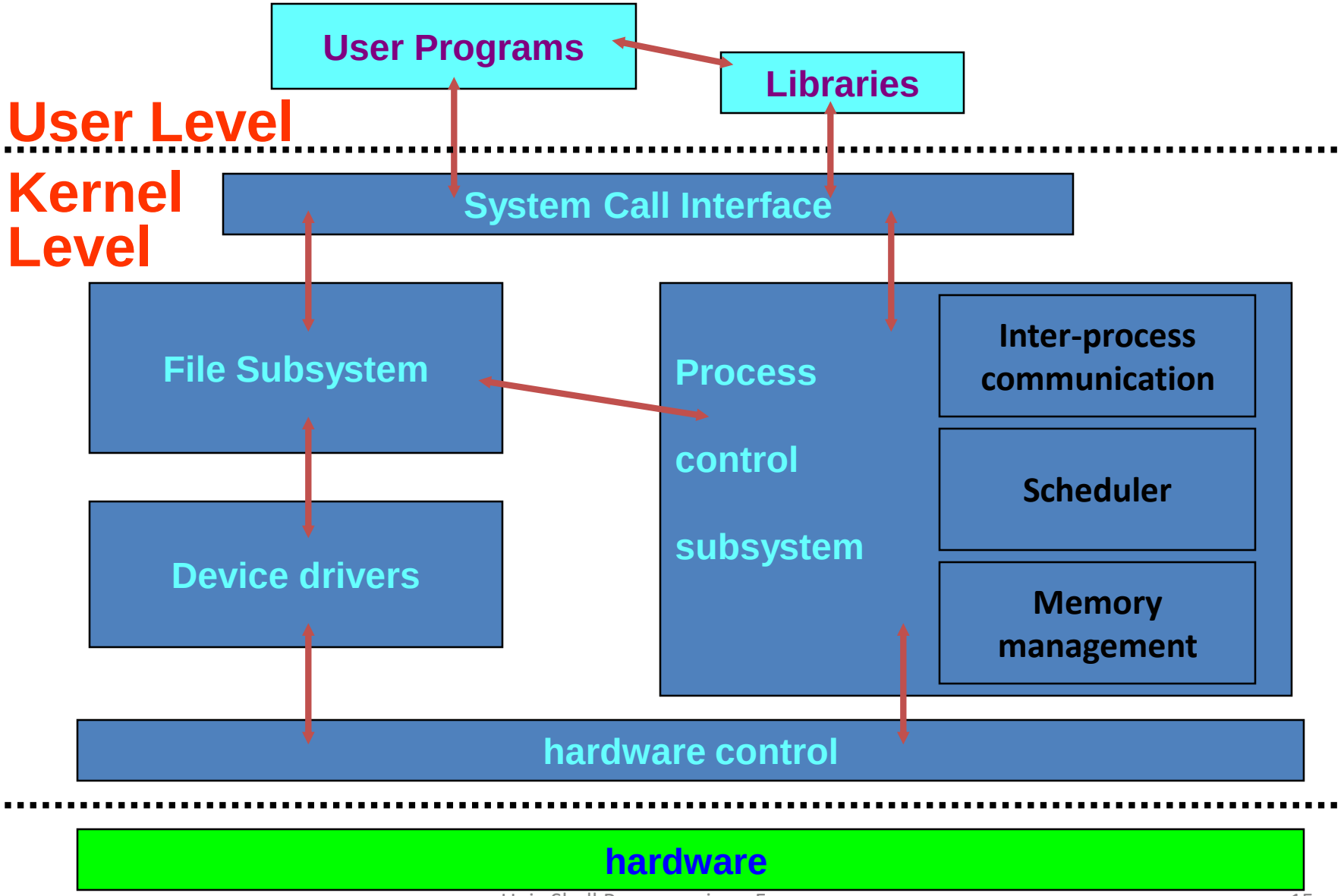
Other features

- Distributed data processing capabilities
- Open Source code
- Shell Scripts
- Powerful Pipes and Filters
- E-Mail Facility
- Support to most programming languages

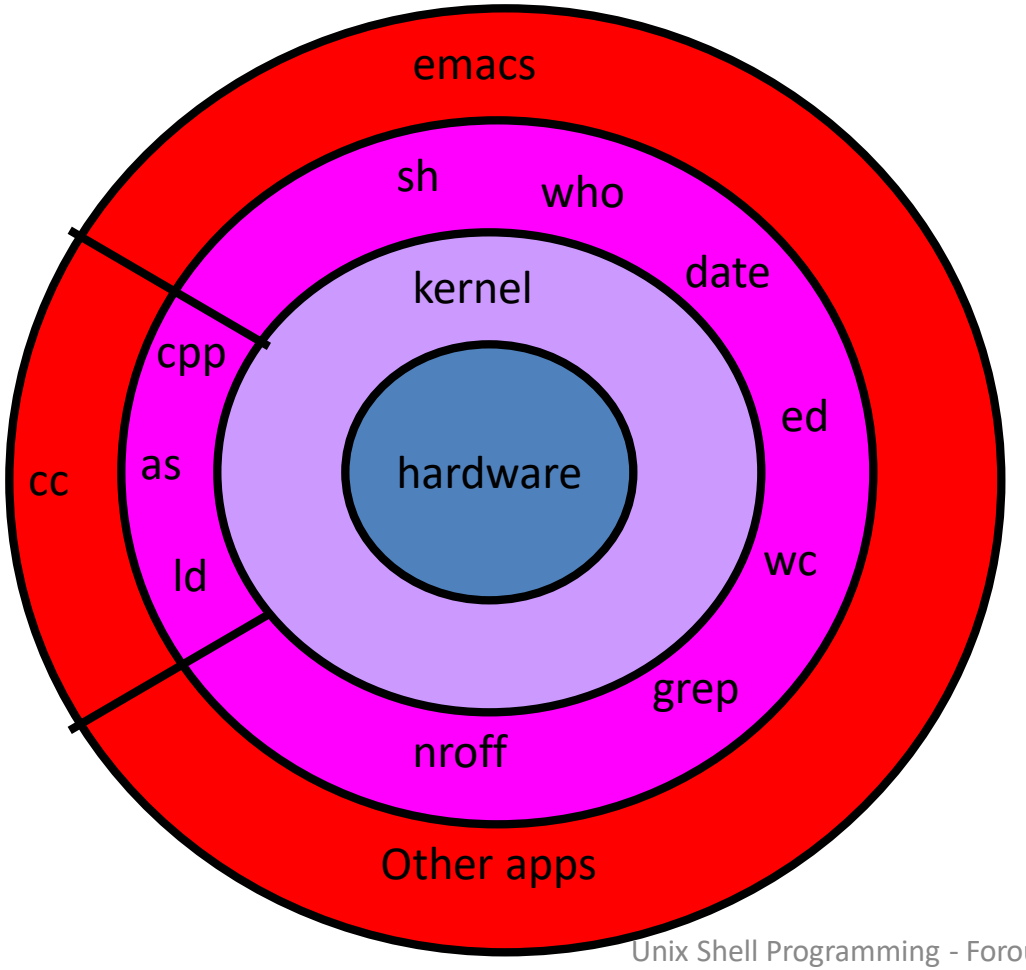
Unix System Architecture / Unix Component

- A Unix operating system consist of three layer
  - Hardware
  - Kernel mode
  - User mode
- the major component of unix system are
    - Kernel
    - Shell & GUI
    - File system

Block Diagram of System Kernel



Architecture of Unix System



- OS interacts directly with the hardware
- Such OS is called system kernel

- The main concept that unites all versions of UNIX is the following four basics −
- **Kernel:** The kernel is the heart of the operating system.
- It interacts with hardware and most of the tasks like memory management, task scheduling and file management.
- It is a collection of program and routine written in C
- When the system is booted the kernel gets loaded into memory and communicate directly with hardware

Function of Kernel

- Three major tasks of kernel:
  - Process Management
  - Device Management
  - File Management
- Three additional Services for Kernel:
  - Virtual Memory
  - Networking
  - Network File Systems
- Experimental Kernel Features:
  - Multiprocessor support
  - Lightweight process (thread) support

- **Shell:** The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands.
- C Shell, Bourne Shell and Korn Shell are most famous shells which are available with most of the Unix variants.
- The shell invoke a command line prompt which is usaually $ or % where the user can type a unix command
- The bourne shell(sh) is the widely used and popular shell

- **Commands and Utilities:** There are various command and utilities which you would use in your day to day activities. **cp, mv, cat** and **grep** etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various optional options.
- Command utilities or system utilities also include server program called daemons(which provide remote network and administration services)
- **Files and Directories:** All data in UNIX is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system.

- When you work with UNIX, one way or another you spend most of your time working with files. This would teach you how to create and remove files, copy and rename them, create links to them etc.

- In UNIX there are three basic types of files −
- **Ordinary Files** − An ordinary file is a file on the system that contains data, text, or program instructions.

- **Directories** − Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders.

- **Special Files** − Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names.

Linux System

- It is a free open source Unix Operating System
- Developed in 1991 by Linux Torvalds
- Open source means source code is freely available so anyone can add feature and correct bugs
- Features
  - It support many programming languages like ada, C, C++. Java
  - It includes Intel C++ Compiler, sun Studio and IBM XL C C++ Compiler

Other Features

- Configurability
- Convenience
- Stability
- Freedom

Advantages & Disadvantages

- It is free
- It is portable to any hardware platform
- It is secure and versatile
- It is scalable

➢ More technical ability needed

➢ Not all hardware compatible

Unix command format

Unix Command format

- A unix command line consist of the name of the unix command followed by arguments
  - The syntax

    $command-options target

  - Command is the name of built in shell command
  - Option is a special argument which should always gives as minus to differentiate from file name
  - Target is a filename or expression

- All unix command must written in lower case
- Whenever the option are specified there must be space or tab between the command name and the option
- A unix command can contain zero, one or more arguments
- A unix command can terminate by delete or Ctrl-u
- When unix command contain more than 80 character and overflow to next time, it is indicated by special prompt >, which appear at the beginning of the next line. This special prompt is known as secondary prompt

Types of Unix Command

- External
  - It exist independently as a separate file
  - The command such as cat or ls is issued
  - The shell searches for these command files using a system variable called PATH variable and execute it
  - Most of the unix command are external in nature

- Internal
  - It does not exist independently
  - They are part of another program or routine
  - Example echo command
  - Internal command are also known as built in commands

Unix Advantages

- It is fully multi tasking with protected memory
- Very efficient virtual memory
- Access control and security (valid account and password)
- Optimized for program development
- Unix is free software
- Unix is very stable (never crash)
- It can smoothly manage extremely huge amount of data

Unix Drawbacks

- Commands have cryptic names and give very little response to tell the user what they are doing
- Much use of special keyboard characters
- Well Understand the main feature of design feature
- Documentation is short on examples and tutorials to help how to use.

Unix File System

Dr.T.Logeswari

- Unix File system is generally divided into four part
  - The Boot Block
  - The Super Block
  - The inode Table
  - The data block

Disk Arrangement

- The boot block
  - It is located at the beginning of the file system
  - It can be accessed by code incorporated by computer ROM bios
  - The boot block is part of disk label
  - It contain a program called bootstrap to boot the OS

- The super block
  - It contain statistical information to keep track of the entries in the file system
  - Whenever disk manipulation is required the super block is accessed
  - Always the copy kept in RAM

The super block contain information

- Size of the file system
  - The storage size of the device or current partition
- List of storage blocks
  - The storage space is divided into series of standard size block
  - When data moved to or from the file system ie block
- Number of free block on the file system
- A list of free block with their location

- Size of the inode list
  - The inode list is initialized to track the maximum number of files which cannot be more than maximum number of storage blocks
- Number of free inode on the file system
- A list of free inodes
- Index to next free inode on the list
- Lock fields for free blocks and free inode lists
- Flag to indicate modification of super blocks

- The inode table
  - Information about each file in the file system is kept in a special kernel structure called inode
  - The inode contain pointer to disk block containing data in the file
  - It contain other information like file size, file modification time, permission bit owner, group etc
  - It does not contain name of file
  - The name of file is listed in directory
  - The directory contain the file name and their associated inode

- A inode size of 64 bytes for a file contain the information
  - File owner type
    - This is number id used as password to find user of the system
  - Group id
    - This is group to access the owner file
  - File type
    - it indicate whether inode represent a file, a directory or block device
    - If the type value is zero then inode is free

- File Access Permission
  - User access
    - Access the file(creator)
  - Group Access
    - Access the member of specified group
  - Other Access
    - Rest of the world
- Three types of Access
  - Read
  - Write
  - Execute

- Each file contain 9 access permission for read write and execute
    - The format rwx-rwx-rwx
  - Date & time of last file access
  - Date & time of last modified
- Inode modification time
  - Whenever the file is accessed or modified or when inode is modified, the content of the file changed – the inode modification time also change

- Number of links
  - It gives the number of directory entries referencing the same inode
- Size of the file
  - This gives the size of the file in bytes
- Table of disk address
  - The data is stored on the storage device
- The block storing the data file data in which to retrieve
- Totally there is 13 address or pointer

- The first 10 point directly
- If additional storage is required 11th pointer is used(256 space)
- If additional storage is required 12th pointer is used(256 *256 space)
- If additional storage is required 13th pointer is used(256*256*256 space)

- The data block
  - It contain the actual data in the file or directories
  - It follow inode table and occupy most storage space
  - The file allotted for one file cannot allotted for other file unless the two files are linked

- Advantages
  - Data in small files can be accessed directly from inode
  - Larger files can be accessed efficiently
  - Disk can be filled completely
- Disadvantage
  - Inode information kept separately from data often requires a long seek when file accessed
  - Inode of files in a common directory are not kept together so it case low performance
  - Original file system uses 512 bytes block , an inefficient transfer size

Types of files

- The Unix OS is built the concept of file system
- It is used to store all the information including
  - OS kernel itself
  - Executable file command to support OS
  - OS configuration information
  - Temporary work file
  - User data
  - Control access to system hardware and system function

- Files can be classified into four categories
  - Ordinary or Regular file
  - Directory file
  - Device file or special file
  - Hidden file

- Ordinary or Regular file
  - It contain text, data or program information
  - It cannot contain other file or directories detail
  - Unix file name not broken into name part and an extension part(extension used to classify file)

- It contain any character except / .
- The character such as *,?,# and & have special meaning in shell so you t not suppose to use in file
- You can use _ underscore symbol
- The ordinary file has two type
- Text file
  - It contain only printable character
  - It contain line of character each terminate by newline(enter) character
  - Eg
  - Text file include C and Java program in Shell and Perl

- Binary Files
  - It contain both printable and non printable character (0 to 255 codes)
  - Eg
    - Unix command, object command of C program
  - Directory file
    - Directories are container or folder that hold files and other directories
    - Directories point to other directories which is known as sub directories
    - It also contain link information
    - Normally a directories contain two piece of information
      - File name
      - A unique identification number(called inode number)

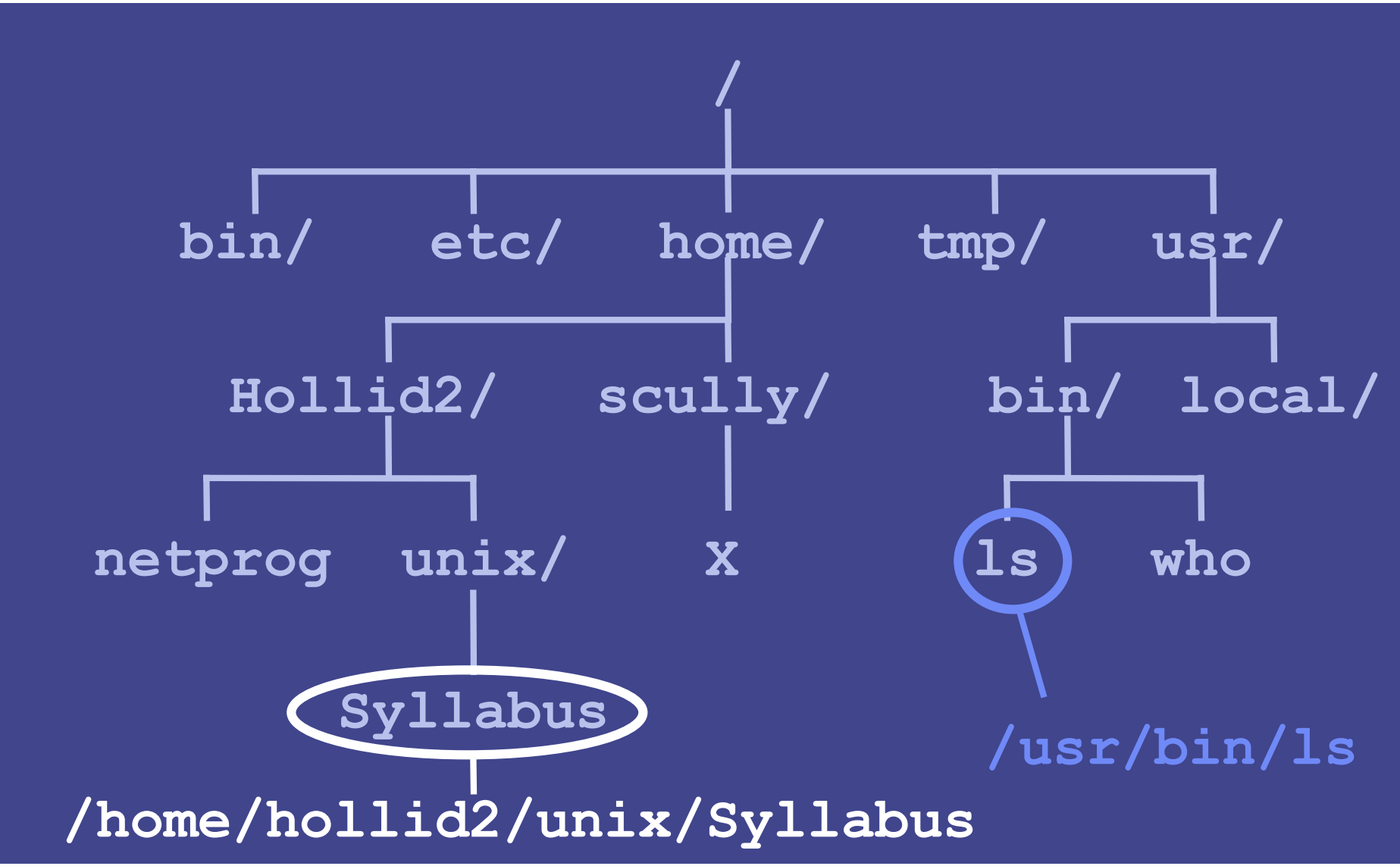
- Device Files
  - To provide application, with easy access to harware device, unix allow the same way as ordinary file
  - The video screen of your PC, RAM, disk Drive and other device are accessed through device file
  - Two types of device files
    - Block Oriented device – it transfer data into block
    - Character oriented device – it transfer data on the byte by byte basics(eg modem, printer and network)

- Hidden files
  - A file name begin with . (dot) character such file is known as hidden file
  - It is used to store specific information ie startup information
  - It is not displayed using ls command

PATHNAME

- To reach for a particular directory or file in the file system a specific route is required.
- The route taken to reach a file is a file system is known as path to a file
- To specify the location the path name is used
- To specify the path the source to destination must include
- It is separated by / (it act as delimiter as file and directory
- Two types of path 1) absolute 2) relative
- It can be defined by absolute path from root or relative path from current working directories (it does not contain / as the first character)
- If we use relative path the “.”for current directory and “..” for parent directory

Pathname Examples



Relative Pathnames

Prefixed w/the current directory, $PWD

So, relative to the current working directory

```
$ cd /home/hollid2
$ pwd
/home/hollid2
$ ls unix/Syllabus
unix/Syllabus
$ ls X
ls: X: No such file or directory
$ ls /home/scully/X
/home/scully/X
```

Home directory

- When you login to unix the current directory is your home directory
- It is created by the system when the user account is opened
- If your login name banu then automatically placed the directory that could have pathname /home/banu
- The tilde character is the unix shorthand for your home directory
- Eg
  - $echo $HOME

  /home/banu

  Or

Process Management

The Process

- As unix is a multitasking and multiuser operating system
- It can run number of program simultaneously
- All program that are loaded into memory for execution are known as processes
- A process can be defined as a program in execution
- Each task in unix is represented by a process

Process state

- The life time of the process can be divided into number of states
- If process start the execution from one state to another
- Five state model in OS
  - New – the process is created
  - Running – being executed
  - Waiting – it is waiting for some event
  - Ready – it is ready to process
  - Terminated – after execution over come to terminite

Unix process State

- It consist of five model with additional swapping and zombie state
  - Create
  - Ready (in memory)
  - Ready (swapped)
  - Asleep (in memory)
  - Asleep (swapped)
  - Running(kernel)
  - Running(user)
  - Zombie(without process terminate the parent process exit)
  - Preempted(returining from kernel to user mode)

Parent and Child Processes

- A parent generate another process in UNIX
- The process that generate or create another process is called parent of newly generated process called child
- Ex

$cat fruits

Sh is parent cat is child

Sh is parent cat and grep are children

- Therefore the shell process sh is parent and cat process child process
- If the process is running it is alive after the process over said to dead
- Ex
  - $cat fruit|grep orange fruits

Unix Process Creation

- In unix a parent process create child processes resulting in a tree of processes
  - The original process is called parent
  - The new process is called child
  - The child is the copy of parent
  - The parent can either wait for the child or continue the job parallel
  - in unix a process create a child process using system call called fork()
  - Fork ()return 0 in child
  - Fork() return the process ID of the new child in parent process
  - If fork()system call is not successful it return -1

- Resource sharing
  - A process need certain resource such as CPU time, memory , file, I/O device to complete the task
  - When a process create child process, the child directly obtain the resource from the OS or share a subset of its parent resource
  - If so many children means the parent process share his resources
- Exec() family of system call is used after fork() to start another completely
- Ps command is used to display a listing of currently active process

- Four principal event cause process to created
  - System initialization
  - Execution of process creation system call by running a process
  - A user request to create a new process
  - Initiation of a batch job
  - When an operation system is booted several process are created
  - ie foreground process and background process

Data Structure of a process

- Each process in the system is represented by a data structure called process control block(PCB) or process descriptor in linux
- It contain some basic information about job includes
  - The process ID (unique no to identify process)
  - Code (program code for execution)
  - Data (data being used for execution)
  - Register value (the value used in CPU register)
  - PC value(address stored in program counter)

- Eg of unix process
  - Each process has unique PID
  - It has parent child relationship
  - The tree structure of all the process in the system of the root with special process called init(it start running when OS is first booted)
  - It is the one for creating and initializing all process
  - init create login password for children
  - Each login create user shell process

Context of a process

- The context of a process is its state or mode
- The process run in two mode
  - Kernel mode
  - User mode
- Kernel mode
  - It is referred to as supervisor mode
  - System processes such as swapping, memory allocation, house keeping run in kernel mode
- User Mode
  - The user process run
  - User process such as application program, utility program run in user mode

Kernel versus user

Kernel mode	User mode
The processor execute all instruction in hardware	It execute only subset
If mode is set to kernel the processor can execute either kernel or user	But it can refer only user mode
The part of critical OS operation execute in kernel mode	The application and other software execute in user mode
The kernel code run fast	It is slow

Type of process

- Three broad categories
  - Interactive process
  - Non interactive process
  - Daemons
- Foreground process or interactive process
  - All the user processes that are created by user using shell and attached to the terminals are known as interactive or foreground process

- Non interactive process or background process
  - Some process run without using terminals such process is called Non interactive process or background process
  - The background processes take their input from a files process them without holding up the terminal and write the output to another file
  - Eg
    - Sorting
    - Searching in a large file system for a file

    The command to run background process by ending command line with &

    Eg

    $sort -0 std.sort.dat&

    5423 (PID NO)

- Some limitation
  - It wont show either successful or unsuccessful
  - To find out execution over by giving PID no
  - The output of the file is redirected to another file
  - If too many process is running background the efficiency is less
  - There is danger when u logout some process is still running

Daemon Process

- It is the processes that constantly running without using an associated terminals or login shell
- It wait to get some instruction either in system or user and start performing the task
- It alive until process should shutdown
- The important process are swapper, init, cross, vhand etc

Feature of daemons

- They start running as soon as it initialized
- The lifetime of daemon is as long as system is running
- The daemon cannot kill prematurely
- Init process is one of the first program loaded after bootstrapping
- The scheduler process is used to manage and schedule other process
- Vhand (virtual memory handler) loaded into the system to swap active processes between memory and disk

- Run away process
  - When the user set a process to run in the background by adding an & at the end of the command and logout without closing the program or killing the program such process is runaway process

Process Management

Dr.T.Logeswari

The Process

- As Unix is a multitasking and multiuser operating system
- It can run number of program simultaneously
- When a program is loaded into memory for execution it is known as processes
- A process can be defined as a program in execution
- Each task in unix is represented by a process

- Every process has two important attributes
  - The process ID(PID)
    - Each process is uniquely identified by a unique integer called PID that is allocated by kernel when the process is born
    - It is needed to control process
  - The parent PID(PPID)
    - The PID of the parent is also available as a process attributes

- The Shell (SH) Process
  - When you login into unix system a process is immediately set up the kernel
  - This represent a unix command which may be sh(bourne shell),Ksh(korne shell), csh(C shell)or bash(bourne again shell
  - Any command given at the command prompt is actually the standard input to the shell process.
  - This process remain active till the user log out when it is killed by the kernel
  - The shell maintain a set of variable that are available to the user like PATH and SHELL
  - The shell pathname stored in SHELL, but PID stored in a special variable $$

To know your PID of your shell

```
$echo $$
```

Output

```
291
$
```

The 291 is the process ID of the currently running command line shell

Parent and Child Processes

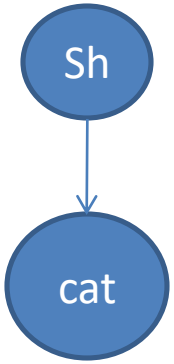
- When you start the computer it first start the kernel.
- The kernel in turn is responsible for starting the first process, which is normally init process. This process is responsible for all other processes
- When starting a process, init start the process as a child of its own. For instance from init the mingetty process is started. Which is responsible for opening a login shell
- From mingetty, the bourne shell process is started to allow user to work in unix command line
- From that it clearly shows unix process management ; there is

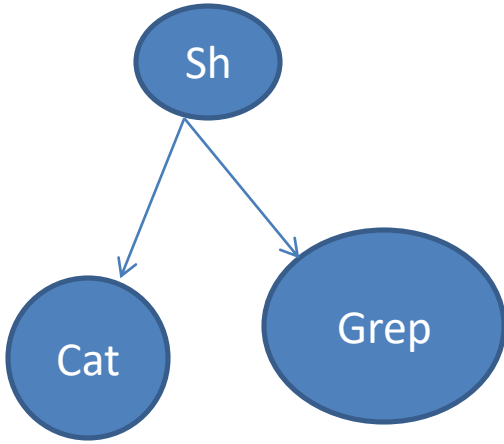
- Ex

  $cat fruits

Sh is parent cat is child

Sh is parent cat and grep are children





- Therefore the shell process sh is parent and cat process child process
- If the process is running till it is alive .it die after the completion of the process

UNIX Process Management

- Most of OS executes within user processes
- Uses two categories of processes:
  - System "processes"
    - run in kernel mode for housekeeping functions (memory allocation, process swapping...)
  - User processes
    - run in user mode for user programs
    - run in kernel mode for system calls, traps, and interrupts inside the user's process image

System call

it represent the border between the user program and the kernel

Two set of system call

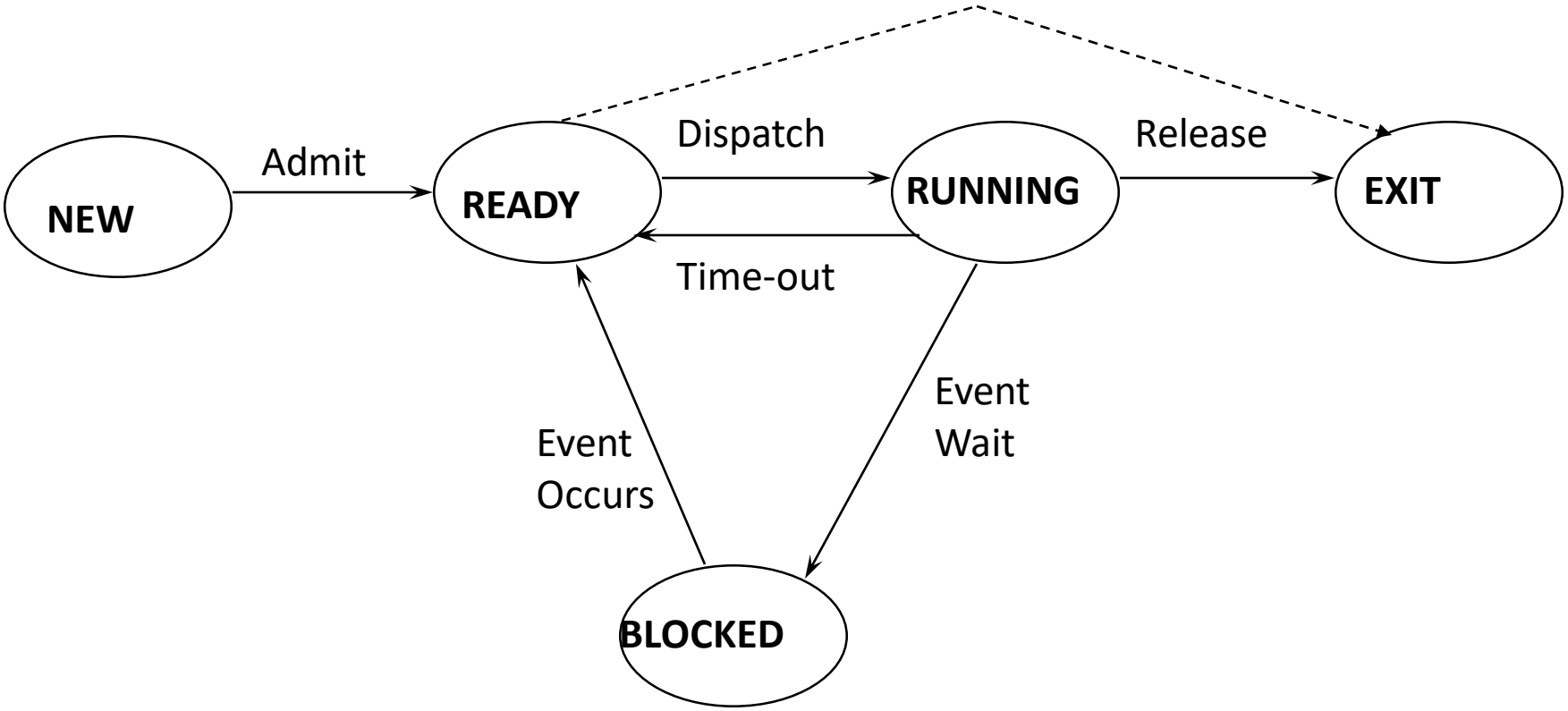
one set interact with the file subsystem

another interact with the process subsystem

A five-state process model

- Five states: New, Ready, Running, Blocked, Exit
- **New** : A process has been created but has not yet been admitted to the pool of executable processes.
- **Ready** : Processes that are prepared to run if given an opportunity. That is, they are not waiting on anything except the CPU availability.
- **Running**: The process that is currently being executed. (Assume single processor for simplicity.)
- **Blocked** : A process that cannot execute until a specified event such as an IO completion occurs.
- **Exit**: A process that has been released by OS either after normal termination or after abnormal termination (error).

State Transition Diagram

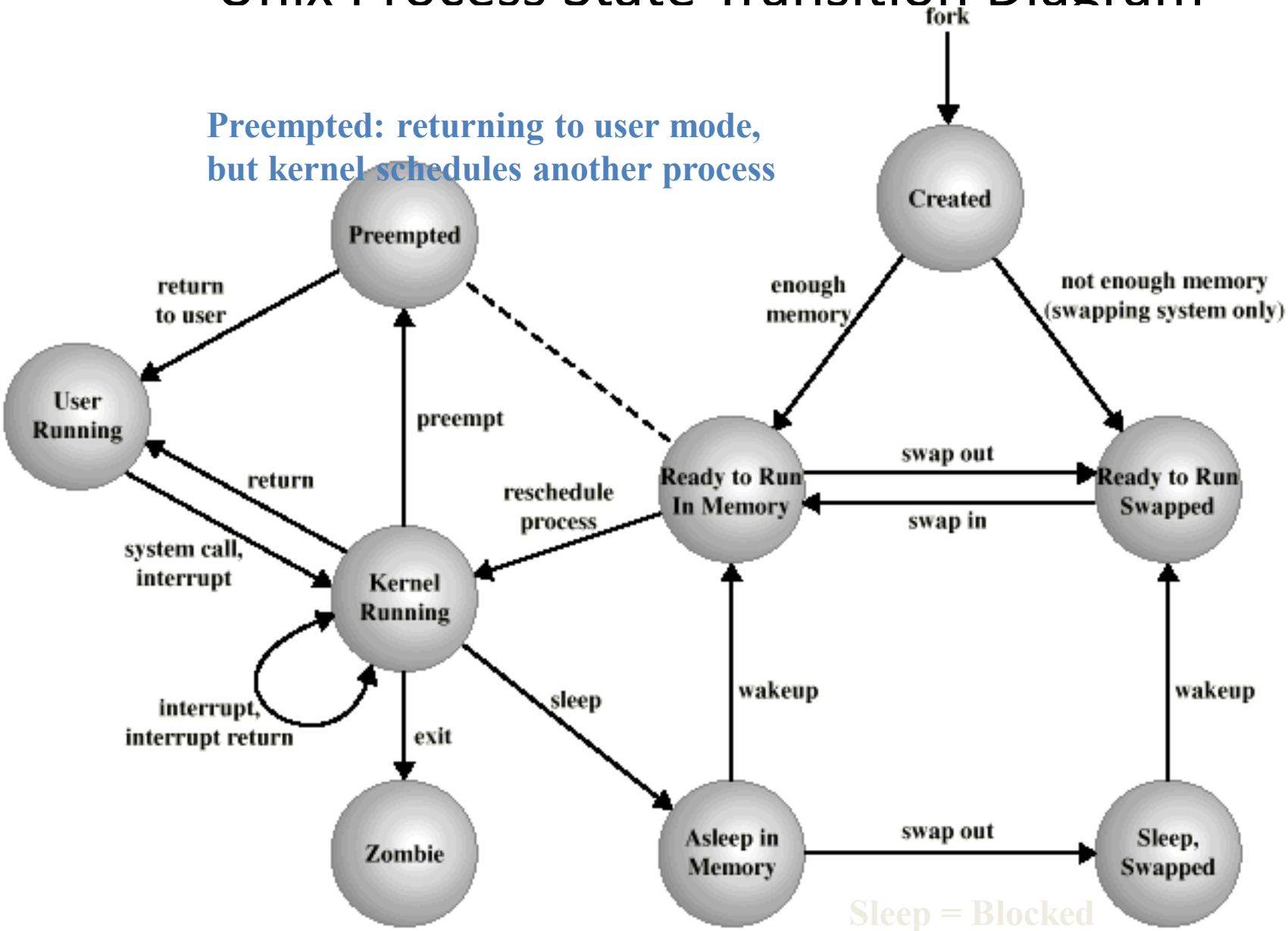


States of a UNIX Process

- User running: Process executes in user mode
- Kernel running: Process executes in kernel mode
- Ready to run in memory: process is waiting to be scheduled
- A sleep in memory: waiting for an event
- Ready to run swapped: ready to run but requires swapping in
- Preempted: Process is returning from kernel to user-mode but the system has scheduled another process instead
- Created: Process is newly created and not ready to run
- Zombie: Process no longer exists, but it leaves a record for its parent process to collect.

– See Process State Diagram!!

Unix Process State Transition Diagram



Unix Process Creation

- A parent generate another process in UNIX
- The process that generate or create another process is called parent of newly generated process called child

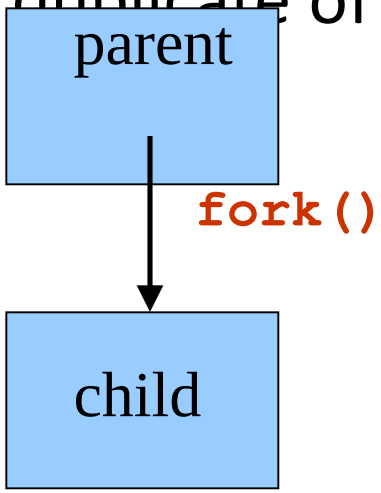
Unix Process Creation

- In unix a parent process create child processes resulting in a tree of processes
  - The original process is called parent
  - The new process is called child
  - The child is the copy of parent
  - The parent can either wait for the child or continue the job parallel

- Resource sharing
  - A process need certain resource such as CPU time, memory , file, I/O device to complete the task
  - When a process create child process, the child directly obtain the resource from the OS or share a subset of its parent resource

How To Create New Processes?

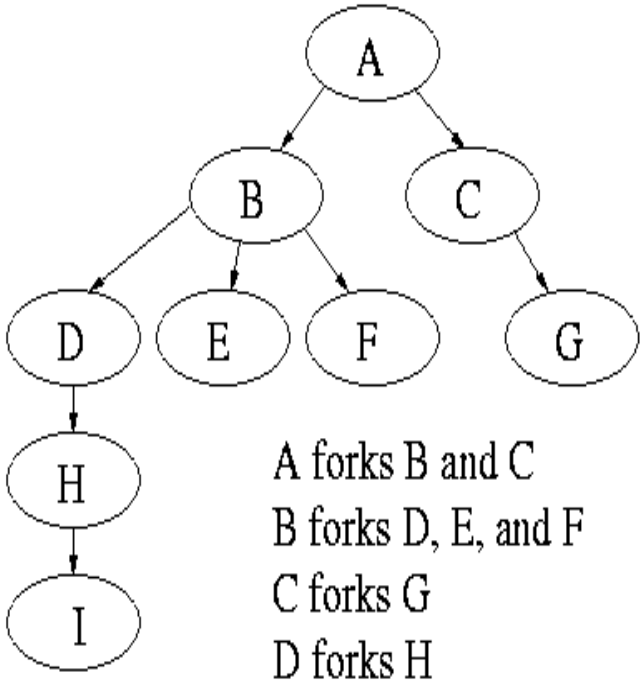
- Underlying mechanism
  - A process runs fork to create a child process
  - Parent and children execute concurrently
  - Child process is a duplicate of the parent process



Process Creation

- After a **fork**, both parent and child keep running, and each can fork off other processes.
- A process tree results. The root of the tree is a special process created by the OS during startup.

- A process can *choose* to wait for children to terminate. For example, if C issued a () system call, it would block until G finished.



- Three distinct phases in the creation of process in unix
  - Forking
  - Overlaying and execution
  - Waiting
- These three phases executed using system call
  - Fork (to create a process)
  - Wait(to wait for child process to complete)
  - Exec(used after fork to start another completely different program)
  - Exit(terminates the child process)

Fork () System Call

Syntax: int pid = fork()

- In unix a process create a child process using system call called fork()
- Fork ()return 0 in child
- Fork() return the process ID of the new child in parent process
- If fork()system call is not successful it return -1

Exec() System call

Syntax: exec*(“program”[,argvp,envp]);

- it allows the replacement of current process with a named program

- the process created with exec() has the same PID as the child that was just forked

- Wait () system call

Syntax: int = wait(&status)

  - The parent then execute the wait() system call to keep waiting for the child process to complete
  - When a parent exit without waiting and the child want to exit then the child called zombie
  - It remains in the system until it is waited

- Exit() system call

Syntax: exit(status);

Once the child process complete execution a call is made to exit() system call that terminate or end the child process and sent signal to the parent

- Four principal event cause process to created
  - System initialization
  - Execution of process creation system call by running a process
  - A user request to create a new process
  - Initiation of a batch job
    - When an operation system is booted several process are created
    - ie foreground process and background process

Summary

- Fork
  - Creates a duplicate of the calling process
  - The result is two processes: parent and child
  - Both continue executing from the same point
- Exit
  - Orderly program termination
  - Unblocks waiting parent
- Wait
  - Used by parent
  - Waits for child to finish execution

Data Structure of a process

- Each process in the system is represented by a data structure called process control block(PCB) or process descriptor in linux
- Kernel maintain 2 data structure to describe the process
  - PCB (it contain field always accessible by kernel)
  - U – area(it can accessible only to the running process)
- It contain some basic information about PCB
  - The process ID (unique no to identify process)
  - Code (program code for execution)
  - Data (data being used for execution)
  - Register value (the value used in CPU register)
  - PC value(address stored in program counter)

- User Area
  - Each process has only one user table
  - It contain some information that must be accessible while process is in execution
    - A pointer to the process table slot
    - Parameter of the current system call, return value error codes
    - File descriptor for all open files
    - Current directory and current root
    - Process and file size limit
  - User table is an extension of the process table

- Eg of unix process
  - Each process has unique PID
  - It has parent child relationship
  - The tree structure of all the process in the system of the root with special process called init(it start running when OS is first booted)
  - It is the one for creating and initializing all process
  - init create login password for children
  - Each login create user shell process

Context of a Process

- A context switch (also referred to as process switch or a task switch) is the switching of the CPU from the one process to another.
- Context switching is an essential feature of multitasking operating system
- The different context of a process are
  - User Context (consist of process text, data, user stack and shared memory that occupy the virtual address space of the process

- Kernel Context (maintained and accessible only to kernel. It has information that the kernel needs to keep track of the process and to stop and restart the process while other processes are allowed to execute
- When context of switching take place the kernel perform the following activities ie CPU has to switch from the process A to Process B

Kernel versus user

Kernel mode	User mode
The processor execute all instruction in hardware	It execute only subset
If mode is set to kernel the processor can execute either kernel or user	But it can refer only user mode
The part of critical OS operation execute in kernel mode	The application and other software execute in user mode
The kernel code run fast	It is slow
Example: I/O Instruction	Application program

Type of process

- Three broad categories
  - Interactive process
  - Non interactive process
  - Automatic or batch process
  - Daemons
- Foreground process or interactive process
  - All the user processes that are created by user using shell and attached to the terminals are known as interactive or foreground process

- Non interactive process or background process
  - Some process run without using terminals such process is called Non interactive process or background process
  - The background processes take their input from a files process them without holding up the terminal and write the output to another file
  - Eg
    - Sorting
    - Searching in a large file system for a file
  - The command to run background process by ending command line with &

    Eg

    $sort -0 std.sort.dat&

    5423 (PID NO)

- Some limitation
  - It wont show either successful or unsuccessful
  - To find out execution over by giving PID no
  - The output of the file is redirected to another file
  - If too many process is running background the efficiency is less
  - There is danger when u logout some process is still running

- Automatic or batch process
  - It is not connected to a terminal
  - Rather these task can be queued into spooler area (FIFO)basics
- Daemon process
  - It is the Server processes that constantly running without using an associated terminals or login shell
  - It wait to get some instruction either in system or user and start performing the task
  - It alive until process should shutdown
  - The important process are scheduler process, init, vhand (virtual memory handler) etc

Feature of daemons

- They start running as soon as it initialized
- The lifetime of daemon is as long as system is running
- The daemon cannot kill prematurely
- Init process is one of the first program loaded after bootstrapping
- The scheduler process is used to manage and schedule other process
- Vhand (virtual memory handler) loaded into the system to swap active processes between memory and disk

- Run away process
  - When the user set a process to run in the background by adding an & at the end of the command and logout without closing the program or killing the program such process is runaway process

Process Related Commands

- Ps Command
- Syntax: ps [-option][user]
  - Ps command is used to display the attribute of the process
  - It display PID, TTY, TIME and CMD(name of the process
    - $ps
    - $ps-f
    - $ps-a

- Nohup: Log out safely
  - When working with unix OS there will be a time to run the command even after the user log out or unplanned login session terminate
  - If you issue the command at command prompt the shell create a child prompt for the command you have issued
- Example

  $nohup sort –o empsort.lst emp.lst &

  Ouput

  567

  Sending output to empsort.lst

- Top command
  - It provides an ongoing look at processor activity in real time
  - It shows how much processing power and memory are being used as well as other information about the running processes
  - It is useful for system administrator

    $top

    Output

    PID, Username, PRI,Size,Res,State,Time,CPU, MEM

- Nice command
  - The nice command is used to run a command at priority lower than the command normal priority
- Syntax: nice[-increment]command
  - If you do not specify an increment value the command default to an increment of 10
  - You must have root user authority to run a command at a higher priority
  - The priority of the process is called nice value

$nice wc-|manual

output

  - It run with lower priority. The increment value is not specified. so it reduce the wc command by 10 unit

- Time command

Syntax: time[option]command[arguments..]

- The time command run the specified program command with the given argument
- When command finishes, time write a message to standard output giving timing statistics about program run
- It accept the entire command line as argument and time it
  - Real time(the elapsed time between innovation and termination)
  - User time(to execute itself)
  - System time(kernel to execute the user process)
    - $time ./script1.sh

Process Termination

- When the terminal hangs
- When the user log off
- When a program execution has gone into an endless loop
- The process has exceeded its time limit
- Memory is unavailable
- An I/O failure
- Interrupt by the operating system

The process is terminated the following action takes place:

- All the file opened by the terminated process are closed
- Exit status of the terminal process is saved
- If the process had any child processes, the kernel makes init process as the parent of all live child processes
- Process state is changed to zombie
- Release memory resource utilized by the process

UNIX Signals

- Signals are a UNIX mechanism for controlling processes
- A signal is a message to a process that requires *immediate* attention
- Signals are generated by exceptions, e.g.:
  - Attempts to use illegal instructions
  - The user pressing an interrupt key
  - Window resize events
  - A child process calling exit or terminating abnormally

Signal Numbers

- Each signal has a default action associated with it
- Most signals can be caught from within a program. A programmer can then:
  - Ignore signal
  - Perform the default action
  - Execute a program specified function
- The default action can be
  - **Term** Terminate the process.
  - **Ign** Ignore the signal.
  - **Core** Terminate the process and dump core.
  - **Stop** Stop the process.

Signal Numbers

Signal Name	Number	Default Action	Meaning
SIGHUP	1	Term	Hangup (sent to a process when a modem or network connection is lost, terminal is closed, etc)
SIGINT	2	Term	Interrupt (generated by Ctrl-C)
SIGTRAP	5	Core	Trace trap
SIGKILL	9	Term	Kill
SIGBUS	10	Core	Bus error (invalid memory reference)
SIGSEGV	11	Core	Segmentation violation
SIGTERM	15	Term	Software termination signal (default kill signal)

For a complete reference see the section 7 of the manual on signal

```
$ man 7 signal
```

kill and *killall*

- In order to communicate with an executing process from the shell, you must send a signal to that process.
- The *kill* command sends a signal to the process.
- You can direct the signal to a particular process by using its *pid*

- The *kill* command has the following format:

  kill [options] pid

  - -l lists all the signals you can send
  - -p (on some systems prints process information)
  - -*signal* is a signal number

- To terminate a process you can send the *HUB* signal, which is signal number 9.

- Example: *kill -9 24607* will force the process with pid 24607 to terminate.

- On LINUX you have the option to use *killall* to kill all processes that are running and are owned by you.

USE THE *KILL-COMMAND* TO TERMINATE ANY UNWANTED BACKGROUND PROCESSES !!!

Scheduling Processes - at

- You can schedule something to happen once using at
- at TIME will execute at given TIME the commands given in Standard input.
- It's often more comfortable to use

  at TIME < filename

  at TIME –f filename

Scheduling Processes - at (2)

```
$ at now + 1 min
$ at> who | logged
$ at> ls myDir | listing.txt
$ at> <EOT>
job 1171280502.a at Mon Feb 12 11:41:42 2007
$ at 3am < commands
job 8 at 2007-03-21 03:00
```

Scheduling Processes - batch

- The **batch** command can be used to queue up jobs:

```
$ batch
at> ls myDir > listing.txt
at> <EOT>
$
```

- These jobs will be run as soon as the system has the resources to do so

Scheduling Processes - cron

- Processes can be scheduled to run at a periodic intervals:
  - Use the **cron** daemon
  - With this, users can schedule processes to run periodically, or at specified times
  - Create a text file called **crontab.cron** which contains lines with a date/time and command line

Scheduling Processes - cron (2)

- Cron jobs are allowed or denied by system administrators using the **cron.allow** and **cron.deny** files in either **/var/spool/cron** or **/etc/crond.d**
- You have to register your crontab using the command **crontab crontab.cron** in order for the cron daemon to activate your crontab

Scheduling Processes - cron (3)

- Each line in crontab.cron has five fields:
  - Minute - (0-59)
  - Hour - (0-23)
  - Day of the month - (1-31)
  - Month of the year - (1-12)
  - Day of the week - (0-6) (Sunday is 0)
  - Command line - the command to be executed

Using cron

- Edit your crontab.cron file to contain what you want it to do:

```
0,30 * * * 1-5 date >> datelog
```

  - This cron job will record the date it was run every 30 minutes from Monday to Friday, in the file datelog
- Register your crontab:

```
$ crontab crontab.cron
```