CBCS SCHEME

USN

BCS515C

Fifth Semester B.E./B.Tech. Degree Examination, Dec.2024/Jan.2025
UNIX System Programming

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	Explain the Kernel and Shell relationship in UNIX operating system with a neat diagram.	10	L1	CO1
	b.	Explain the following UNIX commands with syntax and examples: i) who ii) ls iii) passwd iv) echo v) date	10	L2	CO1
		OR			
Q.2	a.	Explain any five file related commands with syntax and example of each.	10	L2	CO1
	b.	Explain the salient features of UNIX operating system.	04	L1	CO1
	c.	Explain the file types or categories.	06	L2	CO1
		Module – 2			
Q.3	a.	Explain the use of chmod command to change file permission using both absolute and relative methods.	10	L2	CO2
	b.	Explain ls commands with all the options and examples.	10	L2	CO2
		OR			
Q.4	a.	Explain grep commands with all its options.	10	L2	CO2
	b.	Explain three standard files in UNIX.	06	L2	CO2
	c.	Explain the steps of shell interpretive cycle.	04	L2	CO2
		Module – 3			
Q.5	a.	Explain POSIX and SUS (Single UNIX Specification) standards.	04	L2	CO3
	b.	Develop a C program to demonstrate the use of open( ) and read( ) system call in UNIX.	10	L3	CO3
	c.	Explain the use of mkdir( ) and rmdir( ) function in managing directories.	06	L2	CO3
		OR			
Q.6	a.	Differentiate between character special files and block special files.	06	L2	CO3
	b.	Develop a c program to demonstrate the chdir( ) and fchdir( ) functions in UNIX.	10	L3	CO3
	c.	Explain the memory layout of a C program in UNIX.	04	L2	CO3
		Module – 4			
Q.7	a.	Develop both the fork and vfork function in a example program.	10	L3	CO4
	b.	Explain briefly with an example two system v IPC mechanism: i) Message Queues ii) Semaphores	10	L2	CO4
		OR			
Q.8	a.	Explain pipes and its limitations upon developing a program to send data from parent to child over a pipe.	10	L2	CO4
	b.	Explain the client server communication using FIFO with a neat diagram.	10	L2	CO4
		Module – 5			
Q.9	a.	Illustrate signal in UNIX and develop program to setup signal handlers for sigsetsmp( ) and abort( ).	10	L3	CO5
	b.	Explain Daemon process by developing program to transform a normal user into a Daemon process.	10	L3	CO5
		OR			
Q.10	a.	Explain implement SIGPROCMASK and SIGCONGJMP functions with examples.	10	L2	CO5
	b.	Explain coding rules and error logging for Daemon process with neat diagram.	10	L2	CO5

* * * * *

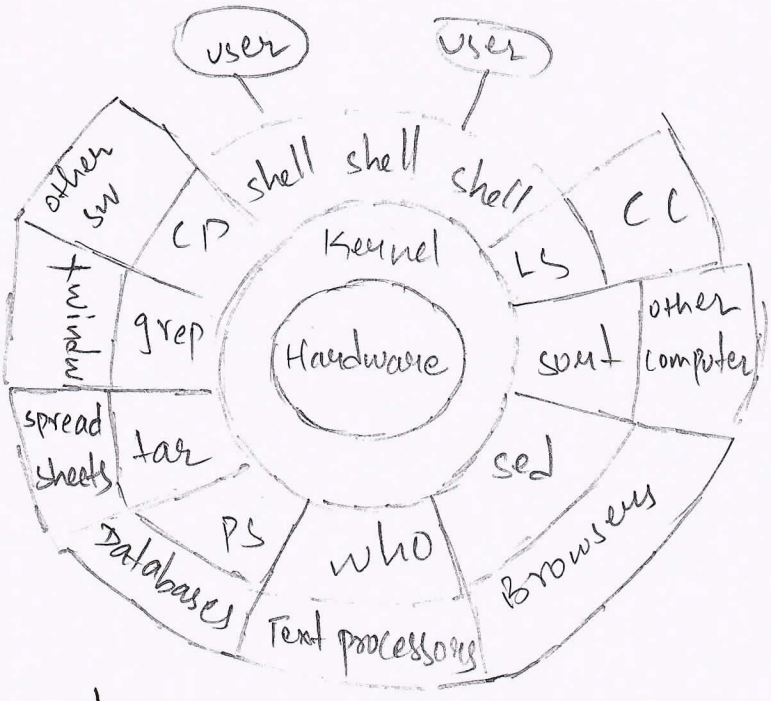
KLS VDIT HALIYAL

DEPT. OF. CSE.

QUESTION PAPER SOLUTIONS OF UNIX S/M PROG.

BCS515C.

1a.



→ The main concept in the unix architure is the division of labour between two agencies the KERNEL and SHELL.

→ The kernel interacts with hardware and shell interacts with user.

→ The kernel is the core of the operating system. It is loaded into memory when the system is booted and communicates directly with the hardware.

→ The Program access the kernel throug a set of function called System calls.

→ The shell is the command interpreter, it translates command into action. It is the interface between user and kernel

→ System contains only one kernel but there may be several shells.

→ For eg. consider a echo command which has lot of spaces between the arguments

eg: $echo sun solaris

sun solaris.

1b.

i) who

→ who are the users?

→ Unix maintains an account of all users, who are logged on to the systems.

syntax: $who.

→ eg: $who

abc	pts/1	Jan 1	20:25 (:0)
xyz	pts/10	Jan 1	14:49 (heavens.com)

ii) ls

→ listing files

→ It is possible to list the names of the files in the directory using ls command.

eg: $ls

syntax: $ls

README
chap01
chap02
chap03

iii) passwd

→ Changing password

→ we can use passwd command to change the password

Eg $passwd

Passwd : Changing password for xyz
Enter login password : *****
New password : ******
Re-enter new password : ******
Passwd (SYSTEM) : Passwd successfully changed for xyz

iv) echo

→ Displaying a message
→ echo command can be used to
- To display a message
- To evaluate shell variables.

→ Some escape sequences are

/a	– Bell	/n	– new line
/b	– Backspace	/t	– Tab
/c	– No newline	/v	– vertical Tab
/f	– Form feed	//	– Backslash.

Syntax : $echo
eg : $echo Sun Solaris

Sun Solaris.

v) Date

→ to display date and time
→ The unix system maintains an internal clock meant to run perpetually
Syntax : $date
eg : $date

Tue Aug 28 1:58:03 IST 2018

Options with date:

+%a - day
+%h - Month name
+%d - date
+%T - time
+%S - second
+%Z - time zone
+%Y - Year
+%H - Hour
+%M - Minute.

eg: $date +%h
Aug

$date +"%h %d"
Aug 28

2a.

i) cat:

→ Displaying and creating files:

→ cat command is used to display the contents of a small file on a termina

```
$cat cprogram.c
#include <stdio.h>
void main()
{
Print ("hello");
}
```

→ As like other files cat accepts more than one filename as arguments

$cat ch1 ch2

It contains the contents of chapter1
It contains the contents of chapter2

→ If this contents of the second files are shown immediatly after the first file without any header information.

ii) cp

→ copy file/directory to other location.

→ The cp command copies a file or group of file from source to destination

→ The syntax takes two filenames to specified in the command line

→ Syntax: $cp source.txt destination.txt

eg: $cp -u /home/u1/a /home/u1/b.

iii) mv:

→ The mv command move/rename file (or) directories

→ It doesn't create a copy of the file. It merely renames it.

Syntax: $ mv oldname.txt newname.txt

eg: $ mv abc.txt xyz.txt.

→ A group of files can be moved to a directory

ex: moves 3 files ch1, ch2, ch3 to the directory module

$ mv ch1 ch2 ch3 module.

iv) rm :→

→ delete file (or) directory.

→ It deletes all files in the current directory and all its subdirectories.

Syntax: $rm filename.txt

eg: $rm abc.txt.

v) ls

→ displaying files and directories in specified manner (or) find.

Syntax: $ ls.

options used with ls.

-a : Displays all files including hidden files, current directory & parent directory

-l : Displays all files with seven attributes such as permissions, links, ownername, group.

-i : display file with inode number.

-t : sorts file based on last modified time

eg: $ls -l

Q6

i. Portable

ii. Multiuser system: Unix is a multiuser system i.e multiple user can use the system at a time.

iii. Multitasking system

iv. Building Block approach: Unix contains several commands each perfo one simple job.

v. Unix tool kit: Unix contains set of tools like compilers, interpreters network network applications etc..

vi. Pattern matching: Unix contains pattern matching features.

vii. Programming facility: Unix shell is also a programming language

Q7. Basic file types

i. Ordinary File.: A ordinary file (or) regular file is the most common file type. An ordinary file itself can be divided into two types.

a) Text file: A text file contains only printable characters & you can often view the contents and make sense out of them.

b) Binary file: A binary file contains printable & unprintable characters that cover the entire ASCII range. Most Unix command are binary files.

ii. Directory File: A directory contains no data, but keeps some details of the files and subdirectories that it contains. A directory file contain an entry for every file & subdirectory that it houses. Each entry has two components i. The file name ii. The inode number.

iii. Device File: You'll also be printing files, installing software from CD-ROMs (or) backing up files to tape. All of these activities are performed by reading (or) writing the file representing the device.

→ Device file name are generally found inside a single directory structure, /dev. A device file is indeed special its not really a stream of character In fact, it doesn't contain anything at all.

3a.

* chmod :

→ A file (or) a directory is created with a default set of permissions which can be determined by umask. Using chmod command we can change the file permissions & allow the owner to execute his file.

i) Relative permissions : Change the permissions specified in command line and leaves the other permissions unchanged.

Syntax : $chmod category operation permission filename(s)

→ category : u - user , g - group , o - others, a - all (ugo)

→ operations : + → assign , − → remove , = → absolute

→ permissions : r - read , w - write , x − execute.

ex : Initially

-rw-r--r--1 kumar metal 1906 sep 23:38 xstart

$chmod u+x xstart

-rwxr--r--1 kumar metal 1906 sep 23:38 xstart.

ii) Absolute permissions : specifying the final permissions. we can set all nine permissi -n explicitly. A string of 3 octal digits is used as an expression.

Read permission - 4 (octal 100)
Write permission - 2 (octal 010)
Execute permission - 1 (octal 001)

ex: $chmod 666 xstart
$ chmod 777 xstart
$ chmod 761 xstart.

octal	Permissions	significance.
0	---	No permissions
1	--x	Execute only
2	-w-	write only
3	-wx	write & Execute only
4	r--	read only
5	r-x	Read & Execute
6	rw-	Read & write
7	rwx	Read, write & Execute

3b ls command : The ls command is to obtain a list of all file name in the current directory.

Syntax : $ls [option] [arguments]

option	Description.
-a	shows all filenames beginning with a dot including . & ..
-F	Marks executables with *, directories with / & symbolic link with

-u	sorts filenames by last access time
-i	Displays inode number
-R	Recursive list
-t	sorts filenames by last modification time
-l	long listing in ASCII collating sequence seven attributes of a fil

ex: $ls -l

-rw-r--r--1 kumar metal 19514 may 10 13:45 Chap01.

→ $ls -x → $ls -a

→ $ls -Fx → $ls -axF

4a.

grep - searching for a pattern.

→ displaying lines containing the pattern.

Syntax: $grep options pattern filename(s)

ex: $grep "sales" emp.list

display all lines containing sales in emp.list.

* grep options

Option	significance
-i	Ignores case for matching
-v	Doesn't display lines matching expressions
-n	Display line numbers along with lines
-c	Displays count of no. of occurrences
-l	Displays list of filenames only
-e exp	Matches multiple patterns
-f filename	Takes pattern from file, one per line
-E	Treats pattern as an ERE
-F	Matches multiple fixed strings.

eg: $grep -i "unix programming" emp.list.

4b Three standard files.

→ The shell associates 3 files with the terminal - two for display & one for keyboard.

→ When a user logs in, the shell makes available 3 files representing 3 stream

i) Standard input: The file (stream) representing input, connected to the keyboard.

ii) Standard output: The file (stream) representing output, connected to the display.

iii) Standard error: The file (stream) representing error messages that emanate from the command (or) shell, connected to the display

→ The first 3 slots are generally allocated to the 3 standard stream as

0 → standard input
1 → standard output
2 → Standard error.

4c. Shell Interpretive cycle

→ Shell sits between users and OS acting as command interpreter

→ Read input and translate the command into action.

→ Shell is analogous to command in DOS.

→ Every unix platform will either have Bourne shell

→ $, #, %

→ The shell provides the user interface to the rich set to GNU utilities.

5a.

POSIX - a set of standardlized specificate to ensure compatibility bet^n OS.

→ process management - fork(), exec(), wait()

→ File I/O operation - open(), read(), write().

SUS - Single Unix specification.

→ It maintained by open group & servers as a certification program for OS

→ Shell utilities - ls, grep, find & basic command.

5b
```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#define BUFFER_SIZE 1024
int main()
{
    int ad;
    char buf[BUFFER_SIZE];
    ssize_t bw;
    ad = open("abc.txt", RDONLY);
    if (ad < 0)
    {
        P.error("failed to open file");
        exit(EXIT_FAILURE);
    }
    Printf("file opened successfully, Reading content");
    while (bw = read(ad, buf, sizeof(buf)-1)) > 0)
    {
        buf[bw] = "\0";
        Printf("%.s", buf);
    }
    if (bw < 0) {
        Perror("failed to read from file")
        close(ad);
        exit(EXIT_FAILURE);
    }
    close(ad);
    Printf("\n file read & closed successfully");
    return 0;
}
```

5c) mkdir: used to create a new directory.

Syntax: int mkdir(const char * pathname, mode_t mode);

```
int main()
{
    const char * dir = "new-directory";
    if(mkdir(dir, 0755) == 0) {
        mk printf("Directory '%.s' created successfully\n", dir);
    }
    else {
```

```
    perror("mkdir failed");
  }
  return 0;
}
```

rmdir() - delete an empty directory

Syntax: int rmdir(const char *pathname);

ex:
```
int main()
{
  const char *dir = "new directory";
  if (rmdir(dir) == 0)
  {
    printf("Directory '%s' removed successfully\n", dir);
  }
  else {
    perror("rmdir failed");
  }
  return 0;
}
```

6a.

	Character special file	Block special files
Data transfer	one byte at a time	in blocks
usage	Terminals, printers, serial ports	Hard disk, SSD, USB drive
Performance	Good for real time input/output	optimized for bulk data transfer
examples.	/dev/tty, /dev/mal	/dev/sda, /dev/sr0
suitable for	Interactive device	storage device
Buffering	No buffering	Buffered I/O.

6b.
```
#include<stdio.h>
#include<unistd.h>
#define buf_size 1024
int main()
{
  char cwd[buf_size];
  int fd;
  if(getcwd(cwd, sizeof(cwd))==NULL){
    perror("getcwd failed");
    exit(EXIT_FAILURE);
```

```
Printf (" current directory: %s\n", cwd );
if( chdir ( " /tmp " ) != 0) {
    Perror ( " chdir failed " );
    exit ( EXIT_FAILURE );
}
Print ( " changed directory to /tmp using chdir\n " );
fd = open (cwd , O_RDONLY );
if (fd < 0 )
{
  Perror( " failed to open previous directory " );
  exit ( EXIT_FAILURE
if ( fchdir ( fd ) != 0) {
    Perror ( " fchdir failed " );
    close ( fd );
    exit ( EXIT_FAILURE );
}
Printf " changed back to %s using fchdir\n " , cwd );
close ( fd );
if( getcwd ( cwd , sizeof ( cwd ) == NULL){
    Perror ( " getcwd failed " );
    exit ( EXIT_FAILURE);
}
Printf ( " current directory after fchdir (): %s \n", cwd );
return 0;
}
```

6. Memory layout of C Program.

i. Text (code) segment. : It consisting of the machine instructions that the CPU executes
→ Text segment is often readonly, to prevent a gg program from accidently modifying its instructions.

ii. Initialized data segment : simply the data segment, containing variables that are specifically initialized in the program.

iii. Uninitialized data segment, often called "bss" segment. Data in this segment is intialized by kernel to arithmetic 0 (or) null pointers before the program starts.

iv. Stack : where automatic variables are stored, along the information that is saved each time a function is called.

Heap : where dynamic memory allocation usually takes place. the heap has been located between uninitialized data and the stack.

7a.

fork() : existing process can create a new one by calling the fork function.

Prototype : #include <unistd.h>

pid_t fork (void);

return 0 in child, processes ID of child in parent

-1 on error.

ex:
```
int main ()
{
  int a=10; pid;
  if ((pid = fork())>0)
  { Printf(" error ");
    return -1;
  }
  else
  { a = a+1;
    Printf("child process a=%d\n", a);
  }
  Printf("parent process a=%d\n", a);
}
```

Vfork() : it has the same calling sequence and same return value as fork.

The vfork function organises with 2.9050.

```
#include <stdio.h>
int main()
{ int a=0; pid;
  if((pid = vfork())<0)
  { Printf("error");
    return -1;
  }
  else
  { a = a+1;
    Printf("child process =%d\n", a);
  }
  Printf("parent process a=%d\n", a);
}
```

3b) Msg Queue & Semaphore :-

* Semaphores

→ Semaphores are system-implemented data structures that are shared between processes.

→ Semget

function

→ To create a semaphore, (or) gain access to one that exits

Include : <sys/types.h> <sys/ipc.h> <sys/sem.h>

comand : int semget(key_t key, int nsems, int semflg):

Returns : Success : the semaphore identifier (semid);

Failure : -1, sets errno : yes.

Arguments

→ key_t key : used to identify a semaphore set

→ int nsems : the no. of semaphores in the set

→ int semflg : specify access permissions and/or special creation condition.

* Message Queue

→ A message queue is a linked list of messages stored within the kernal and identified by a message queue identifier.

→ msgget() : A new queue is created (or) an existing queue is opened

→ msgsnd() : new messages are added to the end of a queue

→ msgrcv() : messages are fetched from a queue

→ msgctl() : It performs various operations on a queue

→ The message queues are powerful IPC tools that allow processes to send and receive messages in the FIFO

→ They are particularly useful in the distributed system where processes need to communicate across different machines.

8a.

→ Pipes are the oldest form of UNIX System IPC and are provided by all UNIX systems

→ Pipes are used for communicating between UNIX process.

limitation :

i) Historically, they have been half duplex. Some systems now provide full-duplex pipes, but for maximum portability, we should never assumess that this is the case

ii) Pipes can be used only between processes that have a common ancestor, Normally a pipe is created by a process, that process calls fork, and the pipe is used between the parent and the child.

→ send data from parent to child over pipe.

```
#include "apue.h"
int main(void)
{
  int n;
  int fd[2];
  pid_t pid;
  char line[MAXLINE];
  if( pipe(fd) < 0)
      err_sys("pipe error");
  if((pid = fork()) < 0) {
      err_sys("fork error");
    }
  else if (pid > 0) {
      close(fd[0]);
      write[fd[1], "hello world\n", 12);
    }
  else {
      close(fd[1]);
      n = read(fd[0], line, MAXLINE);
      write(STDOUT_FILENO, line, n);
    }
  exit(0);
}
```

8b.

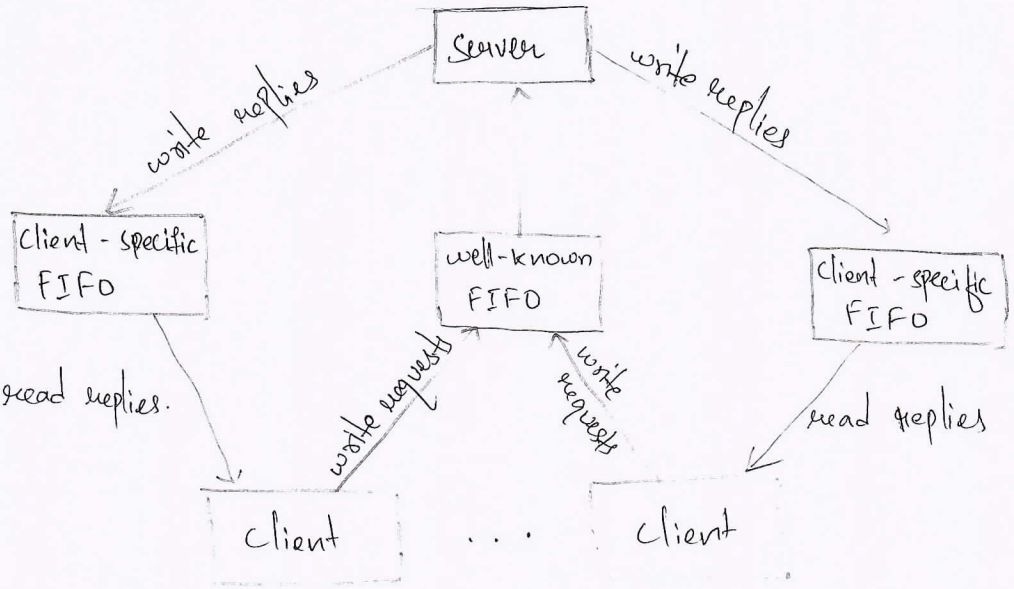
→ FIFO's is used to send data between a client and server.

→ FIFO's another means of inter-process communication in UNIX. They are called named pipes. Pipe can be used only between related process when a common ancestor

has created the pipe. with FIFO's.

→ If we have a server that is contacted by numerous clients, each client can write its request to a well-known FIFO that the server creates.

→ The problem in using FIFO's for this type of client-server communication is how to send replies back from the server to each client. One solution is for each client to send its process ID with the request.



9a

Signals are software interrupts. Signals provide a way of handling asynchronous events

→ ctrl+c

→ ctrl+z

→ ctrl-\

```
#include <stdio.h>
#include <signal.h>
#include <setjmp.h>
sigjmp_buf jmp_buffer;
void handle_sigint(int sig)
{
printf("caught signal (signal (%d) jumping back to saved ", sig);
  siglongjmp(jump_buffer, 1);
}
void hand_sigabort(int sig)
{
printf("caught sigabort(signal %d), Program aborted"; sig);
} exit(EXIT_FAILURE);
```

```
int main()
{
 signal(SIGINT, handle_sigint);
 signal(SIGARRT, handle_sigabrt);
 if(sigsetjmp(jump_buffer,1)==0){
   Printf("jump point saved  ctrl+c to trigger");
  else
   Printf("Back to SIGINT);
  }
while(1)
{ char input;
 Printf("enter 'a' to about");
 scanf("%c", &input);
 if(input=='a')
   Printf("Aborting the program using about()\n");
   about();
 else
   printf(""continuing executing", prev ctrl+c to send SIGINT");
 }
 return 0;
}
```

9b Daemon process is a background process that is not under the direct control of the user. This process is usually started when the system is bootstrapped & is terminated with the system shut down.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <unistd.h>
#include <cntl.h>
int main()
{ pid_t pid;
  int i;
  Pid = fork();
  if( pid==1)
     return -1;
 else if( Pid!=0)
```

```
exit (EXIT_SUCCESS);
if (setsid () == -1)
  return -1;
if (chdir ("/") == -1)
  return -1;
for (i=0; i< NR_OPEN; i++)
  close (i);
open ("/dev/null", O_RDWR);
dup (0);
dup (0);
return 0;
}
```

10a. SigProcmask() - block specific signals, unblock signals and check signals.

Prototype:

```
int sigprocmask (int how, const sigset_t *set, sigset_t *oldset);
```

siglongjmp() - restores the content saved by sigsetjmp()

Prototype

```
void siglongjmp (sigjmp_buf env, int val);
```

```
#include <stdio.h>
#include <unistd.h>
sigjmp_buf jmp_buffer;
void sigint_handler (int sig)
{ printf ("caught signal");
  siglongjmp (jmp_buffer, 1);
int main ()
{ sigset_t (new mass, old mask);
signal (signal, signal_handler);
sigemptyset (& new mask);
Sigaddset (& newmask, SigInt);
if (sigprocmask (sig_block, & newmass, & old mask)) < 0)
{ Perror ("sigprocmask")
  exit (EXIT_FAILURE);
}
```

```
if( sigsetjmp(jump_buffer, 1) == 0)
   Printf(" Jump    saved ");
else
    printf(" back from sigint");
if( sigprocmask( sigset mass, &old_mask, NULL) < 0 )
{
 Perror(" sigprocmask ");
 exit(EXIT_FAILURE);
 while(1)
 Pause();
}
 return 0;
}
```

(b). coding rules.

1. First thing to do is call umask to set the file mode creation mask to 0. The file mode creation mask that's inherited could set to deny certain permission.
2. Call fork and have the parent exit.
3. Call setsid to create a new session.
4. Change the current directory to the root directory. The current working directory inherited from the parent could be on a mounted file system.
5. Unneed file descriptors should be closed. This prevents the daemon from holding open any descripter.

Error logging

→ use syslog(): daemon read all theses forms & messages
→ open the log using openlog()
→ use syslog for message reporting
→ close the log with closelog();

#include <syslog.h>

open void openlog (const char *ident, int option, int facility);

void syslog (int priority, const char * format, ...);

void closelog (void);

int setlogmask (int maskpri);

Prof. F. N. NADAF

HOD
Computer Science & Engineering
KLS Vishwanathrao Deshpande
Institute of Technology, Haliyal.