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

* * * * *

2410BCS515C72702

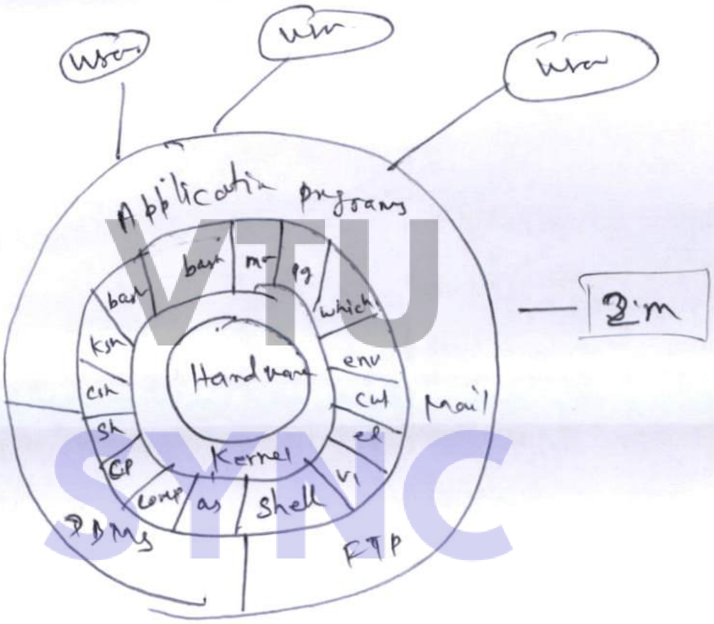
Visvesvaraya Technological University

Belagavi, Karnataka - 590 018.

Scheme & Solutions

Signature of Scrutinizer

Subject Title : UNIX SYSTEM PROGRAMMING **Subject Code :** BCS515C

Question Number	Solution	Marks Allocated
1.(a)	Explain the Kernel & shell relationship in UNIX operating system with a neat diagram.	10M
Ans	 user user user Application programs bash mr pg which bash ksh env Hardware csh cut mail sh el Kernel cp vi comp as shell DBMS FTP — 2m Explanation kernel, shell, Files & process and system calls — 4×2 = 8m	
b	(i) who → display list of internet of users $who root - console Oct 28 10:10 (:0) (ii) ls → listing files $ls $ls -l chap x usp c++ Java iii) passwd → changing user password $passwd enter password to change - ***.. (iv) echo → print/ display $echo "Hi" → Hi (v) date - display current date & time $date → MON OCT 28 10:30:30 IST 2024	2m×5 = 10 10m

Subject Title : UNIX SYSTEM PROGRAMMING **Subject Code :** BCS515C

Question Number	Solution	Marks Allocated
2 (a)	- file related commands are:- i) ls iv) mv ii) cat v) rm → 2m×5=10 iii) cp i) ls → display files & directories in specified direc $ls, $ls -l, $ls -a. ii) cat → display content of a file. $cat filename.txt iii) cp → copy file/directory to other location $cp source.txt destination.txt $cp -r /home/rr/a /hom/rr/b (iv) mv → move/rename file/directorie $mv oldname.txt newname.txt $mv abc.txt xyz.txt (v) rm → delete file/directories $rm filename.txt $rm abc.txt.	10m
(b)	Explain any four silent features of UNIX i) Portable ii) multiuser system iii) multitasking iv) Networking v) Building-Block Approach, vi) UNIX Toolkit vii) pattern matching viii) language script	1m×4 =4m
(C)	file types or Categories - explanation of below * ordinary files < text file / binary file 2m×3=6m * Directory file * Device file < CD ROM / drive, etc	6m
	Module 2	
3(a)	Chmod Relative permission → change the permission specified in the command line and leave other permission unchanged. Syntax → $chmod category operation permission filename(s)	

Question Number	Solution	Marks Allocated

	user category (user, group, others)
	Opertation (assign or remove permission)
	type (read, write, execute)
	ex $chmod u+x test # assign (+) x (execute) to u (user)
	$ chmod ugo+x test # assign (+) x to ugo(user, group & other)
	$ chmod u+x test, test2, test3
	Absolute permission:- Give permission explicitly String of three octal digits.
	Read permission – 4 (octal 100)
	Write " – 2 (octal 010)
	Execute " – 1 (octal 001)
	ex $chmod 666 test
	$ chmod 777 test
	$ chmod 761 test
		5M×2
		=
		10m

(b)	ls → list/display all files & directories
	syntax $ ls [option] [argument]
	options:-
	-a., -F, -u, -i, -d, -R, -t, -l } write example of each (any five)
	ex $ls -l chap
	$ ls -x
	$ ls -Fx
	- ls -a
	- ls -axF
		2m×5
		=10M

Subject Title : UNIX System Programming Subject Code : BCS515C

Question Number	Solution	Marks Allocated
4(a)	grep → display lines containg the pattern syntax – $grep options pattern filename(s) eg $grep "sales" emp.lst # disply all lines containg sales in Emp.lst any five examples by using below option i) ~~grep~~ –i → iggnore case $\boxed{2m \times 5}$ ii) –v – don't disply lines. iii) –n → display line numbes along with lines iv) –l → display list of filename only v) –e → exp specific expression with option vi) –x → file ~~matches~~ ~~with~~ patterns with . entire line vii) –f → pattern from file viii) –E → treats pattern as an extended ERE ix) –F → matches multiple fixed string. eg $grep –i "Unix Programming" emp.lst	10m
(b)	Explanation of three standard files i) standard input → The file(stream) representing input connected to the keyboard (0) ii) standard output → The file (stream) representating output connected to display (1) iii) standard Error → The file (stream) representing error message (2)	2m×3 = 6m
(c)	Shell Interpretive Cycle: – → Shell sits between users & OS acting as command interpreter. → read input & translate the comment into action → shell is analogue to command in DOS. – Every UNIX platform will either have Bourne shell – $, #, %.	4m

Subject Title : UNIX System Programming   Subject Code : BCS515C

Question Number	Solution	Marks Allocated

Module-3

5(a) POSIX → a set of standardized specification to ensure compatibility between operating system

- process management — fork(), exec(), wait()
- File I/O operation → open(), read(), write

4M

SUS (Single Unix specification)

→ It maintained by open group & serves as a certification program for operating systems.

shell utilities — ls, grep, find & basic commands

(b)

```
 # include< studio.h>
 # include< fcntl.h>
 # include < unistd.h>
 # define BUFFER_SIZE 1024
 int main()
 {
   int ad;
   char buf [BUFFER_SIZE];
   ssize_t br;
  ad = open(" abc.txt", O_RDONLY);
  if (ad < 0){
    perror (" failed to open file")
    exit (EXIT_FAILURE);
  }
 printf(" file opened successfully, Reading contents");
While ( br = read (ad, buf, sizeof(buf)-1)) >0) {
      buf [br] = '\0';
      printf("%s", buf);
   }
 if ( br<0) {
    perror(" failed to read from file")
    close(ad);
    exit(EXIT_FAILURE);
 }
close(ad);
printf(" \n file read & closed successfully");
return 0;
}
```

10M

Subject Title : Unix System Programming

Subject Code : BCS515C

Question Number	Solution	Marks Allocated

(c)

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) {
       printf(" Directory %s' Created successfully\n", dir);
   }
   else {
       perror(" mkdir failed");
       }
   return 0;
 }
```

6 M

rmdir() → delete an empty directory.

Syntax int rmdir(const char *pathname);

ex

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

6(a)

~~Character~~	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/null	/dev/sda, /dev/sr0
suitable for	Interactive device	Storage device
Buffering	No buffering	Buffered I/O

6M

Subject Code : BCS515C

Subject Title : UNIX System Programming

Question Number	Solution	Marks Allocated
(b)		10M

```
#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);
   }
  printf(" 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;
}
```

Subject Title : UNIX System programmig

Subject Code : BCS515C

Question Number	Solution	Marks Allocated
(C)	memory layout of c program . includes:- i) Text (code) segment ii) Data segment iii) Heap segment iv) Stack segment v) Command line argument & Environment variable	4m
	Module - 4	

7(a)	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, process 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 & ~~return~~ same return value as fork. The vfork function organises with 2.9BSD.
	#include<stdio.h>
	int main()
	{ int a=10; 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)
	}
		10m

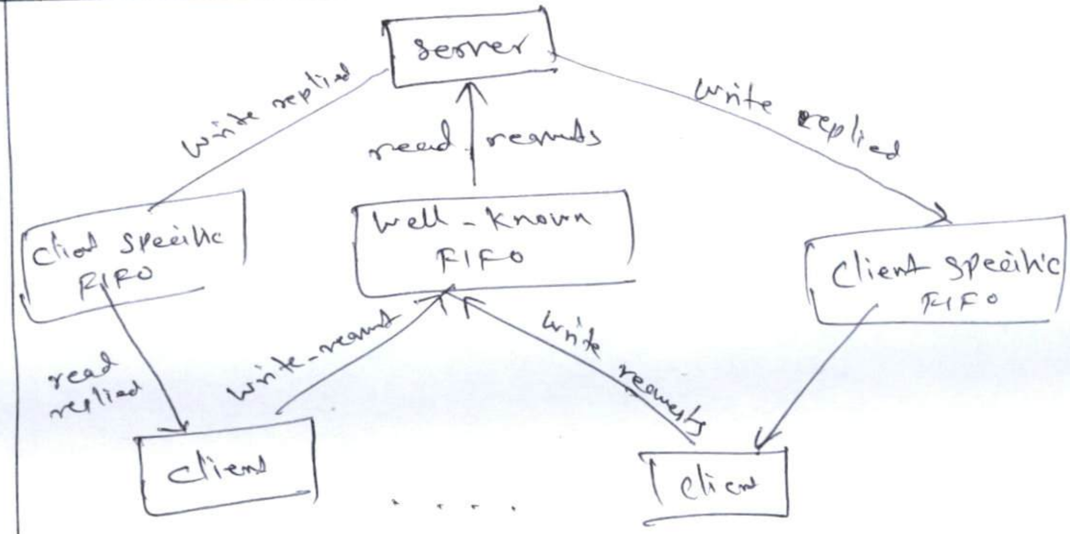
Subject Title : Unix System Programming Subject Code : BCS515C

Question Number	Solution	Marks Allocated
8 (a)	Pipes are used for communicating between UNIX process two limitation:- i) pipes are half duplex ii) pipes can be used to communicate only between two process that have a common ancestor (4M)	10m

```
int main()
{ int n;
  int fd[2];
  pid_t pid;
  char line [MAXLINE];
  if (pipe (fd) < 0)
    printf ("Error in creating pipe\n");
  if ((pid = fork()) < 0)
  printf(" Error in creating process\n");
  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 (1, line, n);
  }
  exit (0);
}
```

(b) FIFOs another means of inter-process communication in UNIX. They are also called named pipes. pipe can be used only between related process when a common ancestor has created the pipe. with FIFOs, however, unrelated processes can exchange data.

FIFO is to send data between client & a server.



10m

explanation of above client server communication

Module 5

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

ctrl + c
ctrl + Z
ctrl + \

-2m

program

```
    #include< stdio.h>
    #include< signal.h>
    # include< setjmp.h>

sigjmp_buf jmp_buffer;
void handle-sigint(int sig)
{printf("caught sigint (signal (%d) jumping back to saved state",
        sig);
  siglongjmp(jump-buffer, 1);
 }
void hand-sigabrt(int sis)
 { printf("caught sigabrt (signal %d). program aborted"; sig);
   exit( EXIT_ FAILURE);
 }
```

Subject Title : Unix System programming

Subject Code : BCS515C

Question Number	Solution	Marks Allocated

```
int main()
{
 signal(~~signal~~ SIGINT , handle_sigint);
 signal ( SIGABTT , handle_sigabrt );
 if(sigsetjmp( jump_buffer, 1)==0) {
   printf(" Jump point saved press ctrl+c to trigger");
 else
   printf(" Back to SIGINT);
 }
 while(1)
 {
   char input;
  printf(" enter 'q' to abort");
  scanf("%c", &input);
  if (input == 'q');
   printf(" Aborting the program using abort() \n");
   abort();
  else
     printf(" continuing execution", press ctrl+c to
     send SIGINT");
   }
 return 0;
}
```

(8m)

10M

(b) 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 & it terminates with the system shut down. — 2(m)

program

```
    #include<sys/types.h>
    #include<sys/stat.h>
    #include<studio.h>
    #include<unistd.h>
    #include<fcntl.h>
 int main()
 { pid_t pid;
```

Subject Title : Unix System Programming Subject Code : BCS515C

Question Number	Solution	Marks Allocated

	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);
	/* do its daemon thing */
	return 0;
	}
		8M

10 (a)	sigprocmask() - block specific signals, unblock signals and check.
	int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
	siglongjmp() → restor the context saves by sigsetjmp()
	void siglongjmp(sigjmp_buf env, int val);
	#include stdio.h>
	#include<unistd.h>
	sigjmp_buf jmp_buffer;
	void sigint_handle(int sig)
	{ printf("caught signal);
	siglongjmp(jump_buffer, 1);	

Subject Title : Unix System Programming **Subject Code :** BCS515C

Question Number	Solution	Marks Allocated

	int main() {
	sigset_t new mask, old mask;
	signal (sigint , sigint_handler);
	sigemptyset (&new_mask);
	sigaddset(&new_mask, sigint);
	if(sigprocmask(sig_block, &new_mask, &old mask)) < 0)
	{ perror ("sigprocmask");
	exit (EXIT_FAILURE);
	}
	if (sigsetjmp(jump_buffer, 1) != 0)
	printf(" jump point saved");
	else
	printf(" back from sigint);
	if (sigprocmask(sig_setmask, &old_mask, NULL)<0)
	{ perror("sigprocmask");
	exit (EXIT_FAILURE);
	while (1);
	pause( );
	}
	return 0;
	}
		80m

(b) 8	Coding rules:- i) fork process to create child ii) changing working directory to / (root) iii) close standard files descriptors. iv) set File permission using umask(); v) create new session with setsid()	10m
	Error logging: 8 m x 2 i) use syslog() ii) open the log using openlog() iii) use syslog for message reporting iv) close the log with closelog() any example program.	