	B.M.S COLLEGE OF ENGINEERING, BANGALORE-19 (Autonomous Institute, Affiliated to VTU) **Computer Science & Engineering**

INTERNALS-3		
Course Code : 20CS5PCUSP	**Course Title : Unix Shell and System Programming**	
Semester : 5 A/B/C/D	**Maximum Marks: 40**	**Date: 18-1-2022**
Faculty Handling the Course:	Dr. Kayarvizhy N, Dr. Nandini V , Dr. Manjunath D R	
Instructions: ***Internal choice is provided in Part C.***		

PART-A

Total 5 Marks (No Choice)

No.	Question	Marks
1	Explain Lock promotion and Lock Splitting If the process has already a read lock on the region and request for read lock on the same region, then instead of giving two locks for the same region giving one write lock can cover both the locks. This is called as **lock promotion** If a process sets a read lock on a file, for example from address 0 to 256, after a while the process unblocks the file from 128 to 480, it will own two write locks on the file: one from 0 to 127 and the other from 481 to 512. This process is called "**Lock Splitting**".	5

PART-B

Total 15 Marks (No Choice)

No.	Question	Marks

2a	The following is the parent process
	int increment=0;
	ans=fork();
	if (ans ==0)
	{
	printf ("Increment value in child program is =%d", increment);
		5

	}
	else
	{
	increment =increment +1;
	}
	Analyse the above code and write the value of increment for child and parent process after execution of above code.
	Child value =0
	Parent value =1
	Explantiion	

2b	Analyse below code using API used and fill up the blanks and identify the functionality of the program
	`#include <fcntl.h>`
	`int main(int argc, char *argv[])`
	`{`
	`if (argc != 2)`
	`printf ("usage: a.out <pathname>");`
	`if (access(argv[1], _____) < ___)`
	`printf ("access error for %s", argv[1]);`
	`else`
	`printf("read is allowed \n");`
	`if (open(argv[1], _____) < ____)`
	`printf ("file open error for %s", argv[1]);`
	`else`
	`printf("file open is allowed \n");`
	`exit(0);`
	`}`
	• **Checking the access privilege of the file**
	• **if (access (argv[1], R_OK) < 0)**
	• **if (open (argv[1], O_RDONLY <0)**
		5

2c	Consider below API and analyse its behaviour when it takes absolute path and relative path as an parmeter. `int mkfifoat(int fd, const char *path, mode_t mode);` • If the path parameter specifies an absolute pathname, then the fd parameter is ignored and the mkfifoat function behaves like the mkfifo function. • If the path parameter specifies a relative pathname and the fd	**5**

	parameter is a valid file descriptor for an open directory, the pathname is evaluated relative to this directory. • If the path parameter specifies a relative pathname and the fd parameter has the special value AT_FDCWD, the pathname is evaluated starting in the current working directory, and mkfifoat behaves like mkfifo.	

PART- C

Total 20 Marks (Choice between question 3a & 3b, choice between question 4a & 4b)

3a	Write a c/c++ program which takes n number of filenames as argument and lists it current access and modification time and change its access and modification time to current time
	
	#include <time.h>
	#include <stdio.h>
	#include <sys/types.h>
	#include <sys/stat.h>
	#include <fcntl.h>
	
	main() {
	int file_descriptor;
	char fn[]="argv[1]";
	struct utimbuf ubuf;
	struct stat info;
	
	if ((file_descriptor = creat(fn, S_IWUSR)) < 0)
	perror("creat() error");
	else {
	close(file_descriptor);
	puts("before utime()");
	stat(fn,&info);
	printf(" The file's modification time is %ld\n",info.st_mtime);
	printf(" The file's access time is %ld\n",info.st_atime);
		10

```
    ubuf.modtime = 0;       /* set modification time to Epoch */
    ubuf.actime=0;
    time(&ubuf.actime);
    if (utime(fn, &ubuf) != 0)
      perror("utime() error");
    else {
      puts("after utime()");
      stat(fn,&info);
      printf("  utime.file modification time is %ld\n", info.st_mtime);
    }
    unlink(fn);
  }
}
```

OR

3b **10**

Write a program to register exit handlers. exit handlers should perform the following

i. Frees the memory allocated by the user

ii. The variable “Lock” which was set to 1 should be reset to 0

```
#include "ourhdr.h"

static void my_exit1(void), my_exit2(void);

int *p;

int lock=1;

int main(void)

{

  p =(int *) malloc (sizeof(100));

if (atexit(my_exit2) != 0)

  err_sys("can't register my_exit2");

if (atexit(my_exit1) != 0)

  err_sys("can't register my_exit1");

  return(0);

}

static void my_exit1(void)
```

{

free(p);

}

static void my_exit2(void)

{

lock=0;

}

4a Write a c/c++ program which takes a directory as argument and list the files. If the directory does not have any files (empty directory) delete the directory **10**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
void listDir(char *dirName)
{
   DIR* dir;
   struct dirent *dirEntry;
   struct stat inode;
   char name[1000];
   dir = opendir(dirName);
   if (dir == 0) {
      perror ("Eroare deschidere fisier");
      exit(1);
   }
   if(rmdir(dir)==0)
   {
    Printf("Empty directory - removed")
   }
   else
   {
   while ((dirEntry=readdir(dir)) != 0) {
      sprintf(name,"%s/%s",dirName,dirEntry->d_name);
      }
}}
```

OR

4b Write c/c++ program to simulate the following and print appropriate messages **10**

i. Check for the existence of lock
ii. Get lock for the specified region
iii. Release the lock

```
#include <stdio.h>
 #include <stdlib.h>
 #include <unistd.h>
 #include <fcntl.h>
 #include <errno.h>
 int main(int argc,char *argv[])
```

```
{
        int fd;
        char buffer[255];
        struct flock fvar;
        if(argc==1)
        {
                printf("usage: %s filename\n",argv[0]);
                return -1;
        }
        if((fd=open(argv[1],O_RDWR))==-1)
        {
                perror("open");
                exit(1);
        }
        fvar.l_type=F_WRLCK;
        fvar.l_whence=SEEK_END;
        fvar.l_start=SEEK_END-100;
        fvar.l_len=100;
        printf("press enter to set lock\n");
        getchar();
        printf("trying to get lock..\n");
        if((fcntl(fd,F_SETLK,&fvar))==-1)
        {     fcntl(fd,F_GETLK,&fvar);
              printf("\nFile already locked by process (pid):    \t%d\n",fvar.l_pid);
              return -1;
        }
        printf("locked\n");
        if((lseek(fd,SEEK_END-50,SEEK_END))==-1)
        {
                perror("lseek");
                exit(1);
        }
        if((read(fd,buffer,100))==-1)
        {
                perror("read");
                exit(1);
        }
        printf("data read from file..\n");
        puts(buffer);
        printf("press enter to release lock\n");
        getchar();
        fvar.l_type = F_UNLCK;
        fvar.l_whence = SEEK_SET;
        fvar.l_start = 0;
        fvar.l_len = 0;
        if((fcntl(fd,F_UNLCK,&fvar))==-1)
        {
                perror("fcntl");
                exit(0);
        }
        printf("Unlocked\n");
        close(fd);
        return 0;
}
```

ALL THE BEST