**Module 7**

1)

#include<stdio.h>

#include<errno.h>

#include<sys/file.h>

void main()

{

int p;

p=open("filename.txt",O\_RDONLY);

if(p==-1)

{

printf("errno = %d\n",errno);

perror("main");

}

p=open("/",O\_WRONLY);

if(p==-1)

{

printf("errno = %d\n",errno);

perror("main");

}

p=open("filename.txt",O\_RDONLY|O\_CREAT);

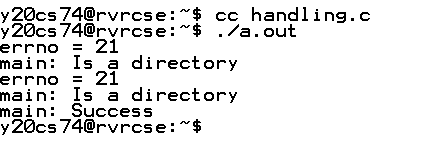
printf("errno = %d\n",errno);

perror("main");

errno=0;

perror("main");

}



#include<stdio.h>

#include<fcntl.h>

#include<stdlib.h>

int main()

{

int fd1 = open("foo1.txt", O\_RDONLY|O\_CREAT);

if (fd1==-1)

{

perror("c1");

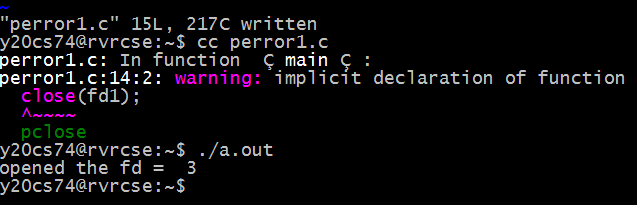
exit(1);

}

printf("opened the fd = % d\n", fd1);

close(fd1);

}



#include<stdio.h>

#include<fcntl.h>

#include<stdlib.h>

int main()

{

int fd1 = open("foo1.txt”);

if (fd1==-1)

{

perror("c1");

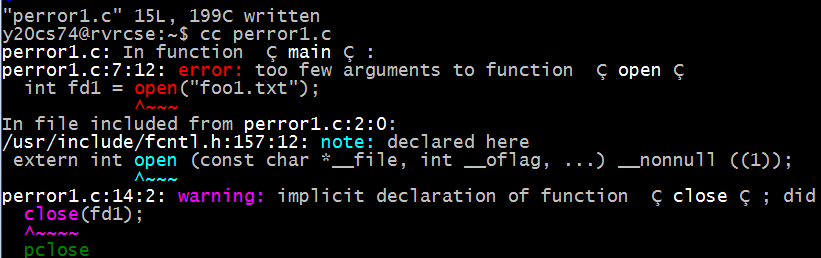
exit(1);

}

printf("opened the fd = % d\n", fd1);

close(fd1);

}



2)

#include<stdio.h>

#include<errno.h>

#include<sys/file.h>

#include<stdlib.h>

void main()

{

int p=open("text.txt",O\_WRONLY);

if(p==-1)

{

printf("File doesnot exit");

}

else

{

char s[]="Unix System Programming";

write(p,s,sizeof(s));

}

close(p);

int fd=open("text.txt",O\_RDONLY);

if(fd==-1)

{

printf("File doesnot exists");

}

else

{

char \*s=(char \*)malloc(20\*sizeof(char));

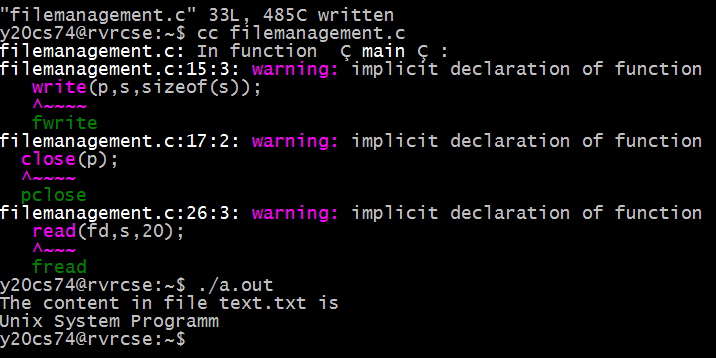
read(fd,s,20);

printf("The content in file text.txt is\n%s\n",s);

}

close(fd);

}



3)

#include<stdio.h>

#include<errno.h>

#include<sys/file.h>

#include<stdlib.h>

void main()

{

int fd1,fd2,fd3;

fd1=open("txt.txt",O\_WRONLY|O\_CREAT);

if(fd1==-1)

{

printf("File doesn't exists\n");

}

else

{

printf("fd1 = %d\n",fd1);

fd2=dup(fd1);

printf("fd2 = %d\n",fd2);

write(fd2,"linux programming",20);

close(fd2);

fd3=dup(fd1);

printf("fd3 = %d\n",fd3);

write(fd3,"SOC",3);

close(fd3);

dup2(fd1,1);

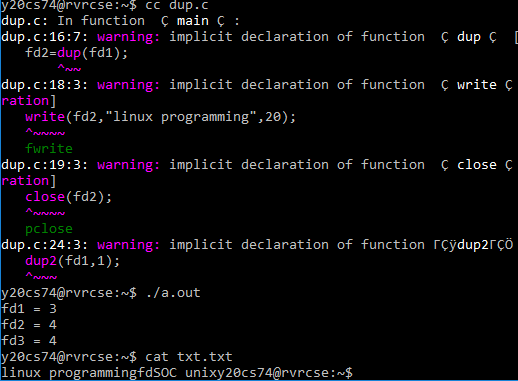
write(1," unix \n",5);

}

close(fd1);

}





**Module-8**

**1)**

#include<stdio.h>

#include<unistd.h>

void main()

{

int n,sum;

printf("Enter a number\n");

scanf("%d",&n);

if(fork()==0)

{

sum=0;

for(int i=1;i<=n;i++)

{

if(i%2==0)

sum=sum+i;

}

printf("Even sum %d\n",sum);

}

else

{

sum=0;

for(int i=1;i<=n;i++)

{

if(i%2!=0)

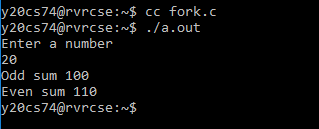
sum=sum+i;

}

printf("Odd sum %d\n",sum);

}

}



2)

#include<unistd.h>

void main()

{

int n=fork();

if(n==0)

{

printf("This is child process with id = %d\n",getpid());

printf("The parent id of the child process p\_id = %d\n",getppid());

sleep(5);

printf("After sleep :\nThe child process id = %d\n",getpid());

printf("The rearranged parent p\_id = %d\n",getppid());

}

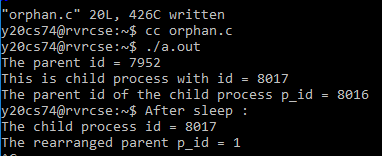
else

{

printf("The parent id = %d\n",getppid());

}

}



3)

#include<stdio.h>

#include<unistd.h>

void main()

{

pid\_t p;

printf("The Process id is %d\n",getpid());

p=fork();

if(p<0)

{

printf("Fork Failure\n");

}

else if(p==0)

{

printf("Child Process\n");

printf("The process id is %d\n",getpid());

sleep(20);

}

else

{

wait();

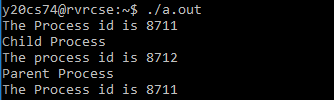
printf("Parent Process\n");

printf("The Process id is %d\n",getpid());

sleep(30);

}

}



3)

#include<stdio.h>

#include<unistd.h>

void main()

{

pid\_t p;

printf("The Process id is %d\n",getpid());

p=fork();

if(p<0)

{

printf("Fork Failure\n");

}

else if(p==0)

{

printf("Child Process\n");

printf("The process id is %d\n",getpid());

sleep(20);

}

else

{

wait();

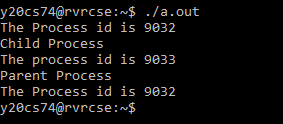
printf("Parent Process\n");

printf("The Process id is %d\n",getpid());

sleep(30);

}

}



4)

#include<stdio.h>

#include<unistd.h>

#include<signal.h>

void sig\_handler(int signum){

printf("Inside handler function\n");

}

int main(){

signal(SIGALRM,sig\_handler); // Register signal handler

alarm(4); // Scheduled alarm after 4 seconds

alarm(1); // Scheduled alarm after 1 seconds

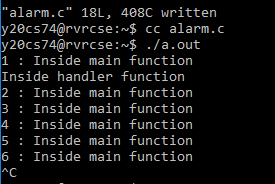
for(int i=1;;i++){

printf("%d : Inside main function\n",i);

sleep(1); // Delay for 1 second

}

}



Module 9

1)

#include<stdio.h>

#include<signal.h>

int main(){

int pid1,pid2,i,j;

pid1=fork();

if(pid1==0){

for (i=1;i<10;i++){

printf("p1:%d",i);

sleep(1);

}

}

pid2=fork();

if(pid2==0){

for (j=10;j>0;j--){

printf("p2:%d",j);

sleep(1);

}

}

printf("parent sleeping:\n");

sleep(3);

printf("killing first child\n");

kill(pid1,SIGSTOP);

printf("resuming the first child\n");

sleep(3);

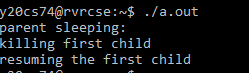
kill(pid1,SIGCONT);

sleep(3);

kill(pid1,SIGINT);

kill(pid2,SIGINT);

}



2)

#include<stdio.h>

#include<unistd.h>

int main() {

int pipefds[2];

int returnstatus;

char writemessages[2][20]={"Hi", "Hello"};

char readmessage[20];

returnstatus = pipe(pipefds);

if (returnstatus == -1) {

printf("Unable to create pipe\n");

return 1;

}

printf("Writing to pipe - Message 1 is %s\n", writemessages[0]);

write(pipefds[1], writemessages[0], sizeof(writemessages[0]));

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Reading from pipe message 1 is %s\n", readmessage);

printf("Writing to pipe - Message 2 is %s\n", writemessages[0]);

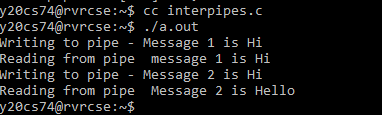
write(pipefds[1], writemessages[1], sizeof(writemessages[0]));

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Reading from pipe Message 2 is %s\n", readmessage);

return 0;

}



3)

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/shm.h>

#include<string.h>

int main()

{

int i;

void \*shared\_memory;

char buff[100];

int shmid;

shmid=shmget((key\_t)2345, 1024, 0666|IPC\_CREAT);

printf("Key of shared memory is %d\n",shmid);

shared\_memory=shmat(shmid,NULL,0);

printf("Process attached at %p\n",shared\_memory);

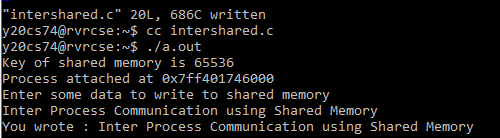
printf("Enter some data to write to shared memory\n");

read(0,buff,100); //get some input from user

strcpy(shared\_memory,buff); //data written to shared memory

printf("You wrote : %s\n",(char \*)shared\_memory);

}



Module 10