1. Program to check POSIX compliance

#define \_POSIX\_SOURCE

#define \_POSIX\_C\_SOURCE 199309L

#include<iostream.h>

#include <unistd.h>

int main() {

#ifdef \_POSIX\_VERSION

cout << "System conforms to POSIX: " << \_POSIX\_VERSION << endl;

#else

cout << "\_POSIX\_VERSION is undefined\n";

#endif

return 0;

}

1. Write a C/C++ program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.

**Writer Process**

#define \_POSIX\_SOURCE

#define \_POSIX\_C\_SOURCE 200809L

#include<sys/types.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<stdio.h>

int main()

{

int fd;

char buf[5]="HELLO";

if(mkfifo("fifo1",0777)==0)

printf("FIFO file created");

fd=open("fifo1",O\_WRONLY,0);

write(fd,buf,5);

printf("run the reader process");

close(fd);

}

**Reader Process**

#define \_POSIX\_SOURCE

#define \_POSIX\_C\_SOURCE 200809L

#include<sys/types.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<stdio.h>

int main()

{

int fd,n;

char buf[5];

fd=open("fifo1",O\_RDONLY,0);

while((n=read(fd,buf,sizeof(buf)))>0)

write(1,buf,n);

close(fd);

return 0;

}

1. IPC program using pipes

#include<stdio.h>

#include<unistd.h>

int main() {

int pipefds1[2], pipefds2[2];

int returnstatus1, returnstatus2;

int pid;

char pipe1writemessage[20] = "Hi";

char pipe2writemessage[20] = "Hello";

char readmessage[20];

returnstatus1 = pipe(pipefds1);

if (returnstatus1 == -1) {

printf("Unable to create pipe 1 \n");

return 1;

}

returnstatus2 = pipe(pipefds2);

if (returnstatus2 == -1) {

printf("Unable to create pipe 2 \n");

return 1;

}

pid = fork();

if (pid != 0) // Parent process {

close(pipefds1[0]); // Close the unwanted pipe1 read side

close(pipefds2[1]); // Close the unwanted pipe2 write side

printf("In Parent: Writing to pipe 1 – Message is %s\n", pipe1writemessage);

write(pipefds1[1], pipe1writemessage, sizeof(pipe1writemessage));

read(pipefds2[0], readmessage, sizeof(readmessage));

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

} else { //child process

close(pipefds1[1]); // Close the unwanted pipe1 write side

close(pipefds2[0]); // Close the unwanted pipe2 read side

read(pipefds1[0], readmessage, sizeof(readmessage));

printf("In Child: Reading from pipe 1 – Message is %s\n", readmessage);

printf("In Child: Writing to pipe 2 – Message is %s\n", pipe2writemessage);

write(pipefds2[1], pipe2writemessage, sizeof(pipe2writemessage));

}

return 0;

}