Question 1

inter.c

#include<sys/stat.h>

#include<sys/types.h>

#include<fcntl.h>

#include<unistd.h>

#include<stdio.h>

int main()

{

pid\_t pid;

if((pid=fork())<0)

printf("error");

else if(pid==0)

if(execl("textinterpreter","test","myarg1","myarg2","myarg4",(char \*)0)<0)

printf("error1");

if(waitpid(pid,NULL,0)<0)

printf("wait eror");

system("ls> list");

return 0;

}

text interpreter

#! /home/guest1/echoarg my2

echoarg.c

#include<sys/stat.h>

#include<sys/types.h>

#include<fcntl.h>

#include<unistd.h>

#include<stdio.h>

main(int argc,char \*argv[])

{

int i;

for(i=0;i<argc;i++)

{

printf("argv[%d]=%s",i,argv[i]);

}

}

output

cc echoarg.c -o echoarg

chmod 777 textinterpreter

gcc inter.c

./a.out

Question 2

. Addition of two numbers using coprocessor

#include<sys/stat.h>

#include<sys/types.h>

#include<fcntl.h>

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define maxline 2000

int main()

{

int n,int1,int2;

char line[maxline];

while((n=read(STDIN\_FILENO, line,maxline))>0)

{

line[n]=0;

if(sscanf(line,"%d%d",&int1,&int2)==2)

{

sprintf(line,"%d\n",int1+int2);

n=strlen(line);

if(write(STDOUT\_FILENO,line,n)!=n)

printf("write error");

}

else

{

if(write(STDOUT\_FILENO,"invalid arg\n",13)!=13)

printf("write error");

}}

exit(0);

}

Question 3: concatenation of two strings using coprocessor

#include<sys/stat.h>

#include<sys/types.h>

#include<fcntl.h>

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define maxline 2000

int main()

{

int n;

char int1[maxline],int2[maxline];

char line[maxline];

char \*str;

while((n=read(STDIN\_FILENO, line,maxline))>0)

{

line[n]=0;

if(sscanf(line,"%s%s",int1,int2)==2)

{

str=strcat(int1,int2);

sprintf(line,"%s\n",str);

n=strlen(line);

if(write(STDOUT\_FILENO,line,n)!=n)

printf("write error");

}

else

{

if(write(STDOUT\_FILENO,"invalid arg\n",13)!=13)

printf("write error");

}}

exit(0);

}

O/P

gcc concat.c

$ ./a.out

hi hello

hihello

ms rit

msrit

^C

Question 4: Pipe – Parent child interaction

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<sys/types.h>

#include<string.h>

#include<ctype.h>

int main()

{

int n,fd[2];

char buf[30];

pid\_t pid;

if(pipe(fd)==-1)

{

printf("error\n");

exit(1);

}

pid=fork();

if(pid<0)

printf("fork error\n");

if(pid>0) /\*parent \*/

{

close(fd[0]);

write(fd[1],"hola!\n",7);

}

else

{

close(fd[1]); /\*child\*/

n=read(fd[0],buf,7);

write(STDOUT\_FILENO,buf,n);

if(islower(buf[0]))

{

buf[0]=toupper(buf[0]);

printf("%c", buf[0]);

}

printf("%s", buf);

write(STDOUT\_FILENO,buf,n);

}

exit(0);

}

Question 5: Illustrate client and server communication

client.c

#include<stdio.h>

#include<stdlib.h>

#include<sys/stat.h>

#include<sys/types.h>

#include<unistd.h>

int main(int argc,char \*argv[])

{

FILE \*fp;

if(argc!=2)

{

printf("usage:fifo client string \n");

exit(1);

}

if((fp=fopen("MY\_FIFO","w"))==NULL)

{

printf("fopen error");

exit(1);

}

fputs(argv[1],fp);

fclose(fp);

return 0;

}

server.c

#include<stdio.h>

#include<stdlib.h>

#include<sys/stat.h>

#include<sys/types.h>

#include<unistd.h>

int main()

{

FILE \*fp;

char readbuf[80];

umask(0);

mknod("MY\_FIFO",S\_IFIFO|0666,0);

while(1)

{

fp=fopen("MY\_FIFO","r");

fgets(readbuf,80,fp);

printf("received string %s\n",readbuf);

fclose(fp);

}

return 0;

}

O/P

1. open two terminals run client.c and server.c seperately

2. terminal 1

cc server.c -o server

./server

3. terminal 2

cc client.c -o client

$ ./client lkfjgljdflg

Question 666: Daemon

#include<stdio.h>

#include<stdlib.h>

#include<sys/stat.h>

#include<sys/types.h>

#include<unistd.h>

int deamon\_init()

{

pid\_t pid;

if((pid=fork())<0)

return -1;

else if(pid!=0)

exit(0);

setsid();

chdir("\\");

umask(0);

return (0);

}

void main()

{

int x=deamon\_init();

system("ps -axj");

}

Signals

1. **Experiment signals on fork and exec.**

**Setup a timer - setup handler for the timer - fork (or exec) – check who gets the timer signal**

1. **Write a program (use signal system call)**
2. **which calls a signal handler on SIGINT signal and then reset the default action of the SIGINT signal**
3. **which ignores SIGINT signal and then reset the default action of SIGINT signal**

**Rewrite the program using sigaction system call**

**Question 7: 1 (signal on fork) Solution**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <signal.h>

void timer\_handler(int signo)

{

printf("pid : %d timer recd\n", getpid());

}

int main()

{

int pid;

printf("parent pid : %d\n", getpid());

signal(SIGALRM, timer\_handler);

alarm(10);

pid = fork();

if(pid < 0)

{

perror("fork"); exit(1);

}

else if(pid)

{

sleep(20);

//sleep(5);

}

else

{

sleep(20);

}

}

// fork

// timer : only to parent

//

**Question 8: 1 (signal on exec) Solution**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <signal.h>

void timer\_handler(int signo)

{

printf("pid : %d timer recd\n", getpid());

}

int main()

{

int pid;

printf("parent pid : %d\n", getpid());

signal(SIGALRM, timer\_handler);

alarm(5);

if(execl("/bin/sleep", "sleep", "20", (void\*)0) < 0)

{

perror("execl"); exit(1);

}

}

// fork

// timer : only to parent

// exec

// : delivered to execed process

**Question 9: 2a Solution**

#include <signal.h>

#include <stdio.h>

#include <unistd.h>

void gotit(int sig)

{

printf("Got the signal!!!\n");

(void)signal(SIGINT,SIG\_DFL);

}

int main()

{

(void)signal(SIGINT,gotit);

while(1)

{

printf("Hello World\n");

sleep(1);

}

}

**Question 10: 2b Solution**

#include <signal.h>

#include <stdio.h>

#include <unistd.h>

int main()

{

(void)signal(SIGINT,SIG\_IGN);

int count=0;

while(1)

{

printf("Hello World\n");

count++;

if(count>10)

(void)signal(SIGINT,SIG\_DFL);

sleep(1);

}

}

**Question 11: 2c Solution**

#include <signal.h>

#include <stdio.h>

#include <unistd.h>

struct sigaction new\_sig;

void gotit(int sig)

{

printf("Got the signal!!!\n");

new\_sig.sa\_handler = SIG\_DFL;

sigaction(SIGINT,&new\_sig,0);

}

int main()

{

new\_sig.sa\_flags=0;

sigemptyset(&new\_sig.sa\_mask);

sigaddset(&new\_sig.sa\_mask, SIGINT);

new\_sig.sa\_handler = gotit;

sigaction(SIGINT,&new\_sig,0);

while(1)

{

printf("Hello World\n");

sleep(1);

}

}