**Question 1(a)- Same program-Different code (Parent-Child)**



#include<iostream>

#include<unistd.h>

using namespace std;

//same program, different code for parent & child, running as concurrent processes

int main()

{

 int pid=fork();

 if(pid<0)

 {

 cout<<"\nUnsuccessful\n";

 return -1;

 }

 else if(pid==0)

 {

 cout<<"I am child process \n";

 }

 else

 {

 sleep(10);

 cout<<"I am parent process \n";

 }

 return 0;

}

**Question 1(b**)- **Same program-Different code,but parent waits for child (Parent-Child)**

//PARENT WAITS FOR CHILD BEFORE FINISHING ITS TASK

//same program, different code for parent & child but parent waits for child to finish

#include<iostream>

#include<sys/types.h>

#include<sys/wait.h>

#include<unistd.h>

#include<stdio.h>

using namespace std;

int main()

{

 int status;

 int pid= fork();

 int x;

 if(pid<0)

 {

 cout<<"\nChild process can not be created \n";

 return -1;

 }

 else if(pid==0)

 {

 cout<<"\nI am child : "<<pid<<"\n";

 cout<<"\nI am child. Child process id: "<<getpid()<<"\n";

 execlp("/bin/ls", "ls", NULL);

 }

 else

 {

 x=wait(&status);

 cout<<"\nChild complete \n";

 cout<<"\nI am parent. Parent process id: "<<getpid()<<"\n";

 cout<<"\nInfo returned by wait(&status): "<<x<<" which is = child processid\n";

 }

 return 0;

}



**Question 1(c)- Same program-Same code(Parent-Child)**

//SAME PROGRAM, SAME CODE, concurrent processes

#include<iostream>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

using namespace std;

int main()

{

 int code=fork();

 int status;

 if(code<0)

 {

 cout<<"\nUnsuccessful \n";

 }

 else // run same code for child and parent process

 {

cout <<"pid = " << getpid() << " return code: "<< code<<"\n";

 }

 return 0;

}



**Question 1(d)- Same program – Same code, but parent waits for child (Parent-Child)**

//SAME PROGRAM, SAME CODE, parent process waits for child to finish :-

#include<iostream>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

using namespace std;

int main()

{

 int code=fork();

 int status;

 if(code<0)

 {

 cout<<"\nUnsuccessful \n";

 }

 else // run same code for child and parent process

 {

 wait(NULL);

 cout <<"pid = " << getpid() << " return code: "<< code<<"\n";

 }

 return 0;

}



**Question 1(e)- Diffrent program-Diffreant code(Parent-Child)**

//Different programs

// parent.cpp: the parent program

/\*

To compile & execute the parent-child programs, steps are:

step 1) g++ child.cpp -o child

step 2) g++ parent.cpp

step 3) ./a.out

\*/

#include <iostream>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <unistd.h>

using namespace std;

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

{

 int i = 0;

 long sum;

 int pid;

 int status, ret;

 cout<<"Parent: Hello, World!\n";

 pid = fork ();

 if (pid == 0) {

 // I am the child: loading a different program in the child process

 execvp ("./child", argv);

 }

 // I am the parent

 cout<<"Parent: Waiting for Child to complete.\n";

 if ((ret = waitpid (pid, &status, 0)) == -1)

 cout<<"parent:error\n";

 if (ret == pid)

 cout<<"Parent: Child process waited for.\n";

}

// child.cpp: the child program

#include <iostream>

#include <stdlib.h>

#include <string.h>

#define A 500

#define B 600

#define C 700

using namespace std;

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

{

 int i, j;

 long sum;

 // Some arbitrary work done by the child

 cout<<"Child: Hello, World!\n";

 for (j = 0; j < 30; j++ ) {

 for (i =0; i < 900000; i++) {

 sum = A \* i + B \* i \* i + C;

 sum %= 543;

 }

 }

 cout<<"Child: Work completed!\n";

 cout<<"Child: Bye now.\n";

 exit (0);

}



**Question 2 – Pipes**

/\*

WAP to demonstrate Inter-Process Communication (IPC) between parent

and child using pipe system call

\*/

#include<iostream>

#include<unistd.h>

#include<string.h>

#include<sys/types.h>

#define BUFFERSIZE 25

#define READEND 0

#define WRITEEND 1

using namespace std;

int main(void)

{

 char writemsg[BUFFERSIZE] = "Greetings";

 char readmsg[BUFFERSIZE];

 int fd[2];

 pid\_t pid;

 if (pipe(fd) == -1) /\* create the pipe \*/

{

 cout<<"Pipe failed!";

 return 1;

 }

 pid = fork();/\* fork a child process \*/

 if (pid < 0)

 {

 cout<<"Fork Failed!"; /\* error occurred \*/

 return 1;

 }

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

 {

 close(fd[READEND]); /\* close the unused end of the pipe \*/

 write(fd[WRITEEND], writemsg, strlen(writemsg)+1); /\* write to the pipe \*/

 close(fd[WRITEEND]); /\* close the write end of the pipe \*/

 }

 else /\* child process \*/

{

 close(fd[WRITEEND]); /\* close the unused end of the pipe \*/

 read(fd[READEND], readmsg, BUFFERSIZE); /\* read from the pipe \*/

 cout<<"read: "<<readmsg;

 close(fd[READEND]); /\* close the write end of the pipe \*/

 }

 return 0;

}



**Question 3** – **Kernel**

/\*

WAP to display the following:

a. Kernel version

b. CPU type and model

c. Information on configured memory, amount of free and used memory

\*/

#include<iostream>

#include<stdlib.h>

#include<stdio.h>

using namespace std;

int main()

{

 cout<<"\nKernel version is:\n";

 system("cat /proc/sys/kernel/osrelease");

 cout<<"\nCPU space: \n";

 system("cat /proc/cpuinfo |awk 'NR==3,NR==4{print}' \n");

 cout<<"\n Configured memory is :\n";

 system("cat /proc/meminfo |awk 'NR==1{print $2}'\n");

 cout<<"\n Amount of free memory is :\n";

 system("cat /proc/meminfo |awk 'NR==2{print $2}'\n");

 cout<<"\n Amount of used memory is :\n";

 system("cat /proc/meminfo |awk '{if (NR==1) a=$2; if (NR==2) b=$2 } END {print a-b}'\n");

 return 0;

}



**Question 4** –**Consumer-Producer**

//WAP to demonstrate producer-consumer problem using shared memory.

//C++ program for Consumer process illustrating POSIX shared-memory API.

#include<iostream>

#include<stdlib.h> // used for shm\_open

#include<string.h>

#include<fcntl.h>

#include<sys/shm.h>

#include<sys/stat.h>

#include<sys/mman.h>

#include<sys/unistd.h>

#include<sys/types.h>

#include<stdio.h>

using namespace std;

//To run, use:

// g++ consumer.cpp -lrt

int main()

{

 const int SIZE =4096;

 const char \*name ="/my\_eg";

int shm\_fd; /\* shared memory file descriptor \*/

 void\* ptr; /\* pointer to shared memory object \*/

 shm\_fd = shm\_open(name, O\_RDONLY, 0666); /\* open the shared memory object \*/

 if(shm\_fd<0)

 { cout<<"\nERROR\n"; }

 else

 {

 /\* memory map the shared memory object \*/

 ptr = mmap(0, SIZE, PROT\_READ, MAP\_SHARED, shm\_fd, 0);

 /\* read from the shared memory object \*/

 cout<<(char\*)ptr;

 /\* remove the shared memory object \*/

 shm\_unlink(name);

 }

 return 0;

}



**Question 5** – **pthread**

//Write program to understand working of Pthread library.

/\* To compile & run the program, do following:

step 1) g++ -pthread Ques5\_pthread.cpp

step 2) ./a.out 5

\*/

#include<pthread.h>

#include<stdio.h>

#include<stdlib.h>

#include<iostream>

using namespace std;

int sum;

void\* runner(void\* param);

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

{

 pthread\_t tid;

 pthread\_attr\_t attr;

 if(argc!=2)

 {

 cout<<"\nUsage :a.out<integer value>\n";

 return -1;

 }

 if(atoi(argv[1])<0)

 {

 cout<<"\n%d must be >=0\n"<<atoi((const char\*)(argv[1]));

 return -1;

 }

 //get the default attributes

 pthread\_attr\_init(&attr);

 //create the thread:

 pthread\_create(&tid,&attr,runner,argv[1]);

 //parent waits for the child thread to finish

 pthread\_join(tid,NULL);

 //output the value of shared data "sum"

 cout<<"\nSUM is: "<<sum<<endl;

 return 0;

}

//child thread will begin execution here:

void\* runner(void\* param)

{

 int i,upper=atoi((const char\*)param);

 sum=0;

 for(i=1;i<=upper;i++)

 sum+=i;

 pthread\_exit(0);

}



**Question 6** – **File Details**

**/\***

**WAP to print file details including owner access permissions, file**

**access time, where file name is given as a command line argument.**

**\*/**

**#include<iostream>**

**#include<stdlib.h>**

**#include<stdio.h>**

**#include<unistd.h>**

**#include <sys/stat.h>**

**#include <sys/types.h>**

**using namespace std;**

**/\***

**Steps to compile and run the program are:**

**step 1) g++ Ques6.cpp**

**step 2) ./a.out <name of any file>**

**For eg:- ./a.out producer.cpp**

**\*/**

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

**{**

 **if(argc !=2)**

 **{**

 **cout<<"\nEnter file name!\n";**

 **return 1;**

 **}**

 **struct stat fileStat;**

 **if(stat(argv[1],&fileStat) < 0)**

 **return 1;**

 **cout<<"\nFile details for "<< argv[1]<<" are :\n";**

 **cout<<"File Size: "<<fileStat.st\_size<<" bytes\n";**

 **printf(" time of last access: %ld : %s", fileStat.st\_atime, ctime(&fileStat.st\_atime));**

 **printf(" time of last modification: %ld : %s", fileStat.st\_mtime, ctime(&fileStat.st\_mtime));**

 **printf(" time of last change: %ld : %s", fileStat.st\_ctime, ctime(&fileStat.st\_ctime));**

 **//system("ls -l "argv[1]);**

 **cout<<"File Permissions: \t";**

 **printf( (S\_ISDIR(fileStat.st\_mode)) ? "d" : "-");**

 **printf( (fileStat.st\_mode & S\_IRUSR) ? "r" : "-");**

 **printf( (fileStat.st\_mode & S\_IWUSR) ? "w" : "-");**

 **printf( (fileStat.st\_mode & S\_IXUSR) ? "x" : "-");**

 **printf( (fileStat.st\_mode & S\_IRGRP) ? "r" : "-");**

 **printf( (fileStat.st\_mode & S\_IWGRP) ? "w" : "-");**

 **printf( (fileStat.st\_mode & S\_IXGRP) ? "x" : "-");**

 **printf( (fileStat.st\_mode & S\_IROTH) ? "r" : "-");**

 **printf( (fileStat.st\_mode & S\_IWOTH) ? "w" : "-");**

 **printf( (fileStat.st\_mode & S\_IXOTH) ? "x" : "-");**

 **cout<<endl;**

 **return 0;**

**}**



**Question 7** – **Storing our content in file**

/\*

WAP to copy a source file into the target file and display the

target file using system calls.

Steps to compile and execute the program:

step 1) g++ Ques\_7.cpp -o cpcmd

step 2) ./cpcmd src.txt dest.txt

Error opening file src.txt errno = 2

step 3) touch src.txt

step 4) echo "This program is a simulation of the Linux copy command "cp" using I/O system calls. We copy the content from one file to another " > src.txt

step 5) ./cpcmd src.txt dest.txt

step 6) cat dest.txt

This program is a simulation of the Linux copy command "cp" using I/O system calls. We copy the content from one file to another \*/

#include <iostream>

#include <stdlib.h>

#include <fcntl.h>

#include <errno.h>

#include<unistd.h>

#include<sys/types.h>

#define BUFF\_SIZE 1024

using namespace std;

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

{

 int srcFD,destFD,nbread,nbwrite;

 char \*buff[BUFF\_SIZE];

 /\*Check if both src & dest files are received or --help is received to get usage\*/

 if(argc != 3 || argv[1] == "--help")

 {

 cout<<"\nUsage: cpcmd source\_file destination\_file\n";

 exit(EXIT\_FAILURE);

 }

 /\*Open source file\*/

 srcFD = open(argv[1],O\_RDONLY);

 if(srcFD == -1)

 {

 cout<<"\nError opening file "<<argv[1]<<" errno = \n"<<errno;

 exit(EXIT\_FAILURE);

 }

 /\*Open destination file with respective flags & modes

 O\_CREAT & O\_TRUNC is to truncate existing file or create a new file

 S\_IXXXX are file permissions for the user,groups & others\*/

 destFD = open(argv[2],O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IWGRP | S\_IROTH | S\_IWOTH);

 if(destFD == -1)

 {

 cout<<"\nError opening file "<<argv[2]<<" errno = \n"<<errno;

 exit(EXIT\_FAILURE);

 }

 /\*Start data transfer from src file to dest file till it reaches EOF\*/

 while((nbread = read(srcFD,buff,BUFF\_SIZE)) > 0)

 {

 if(write(destFD,buff,nbread) != nbread)

 cout<<"\nError in writing data to \n"<<argv[2];

 }

 if(nbread == -1)

 cout<<"\nError in reading data from \n"<<argv[1];

 if(close(srcFD) == -1)

 cout<<"\nError in closing file \n"<<argv[1];

 if(close(destFD) == -1)

 cout<<"\nError in closing file \n"<<argv[2];

 exit(EXIT\_SUCCESS);

}

