**QUESTION-1**

**Ques (a) - Same program, different code for parent & child, running as concurrent processes**

#include<iostream>

#include<unistd.h>

using namespace std;

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;

}



**Ques (b) - 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 process id\n";

 }

return 0;

}



**Ques (c)-**  **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;

}



**Ques(d) -**  **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;

}



**Ques(e) -**

 **child.cpp: Program for Child Process**

#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...\n";

 exit (0);

}

 **parent.cpp: Program for Parent Process**

#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";

}



 **QUESTION-2**

**Write a program 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] = " Pipe Program ";

 char readmsg[BUFFERSIZE];

 int fd[2];

 pid\_t pid;

 /\* create the pipe \*/

 if (pipe(fd) == -1) {

 cout<<"Pipe failed!";

 return 1;

 }

 /\* fork a child process \*/

 pid = fork();

 if (pid < 0) { /\* error occurred \*/

 cout<<"Fork Failed!";

 return 1;

 }

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

 // close the unused end of the pipe

 close(fd[READEND]);

 // write to the pipe

 write(fd[WRITEEND], writemsg, strlen(writemsg)+1);

 // close the write end of the pipe

 close(fd[WRITEEND]);

 }

 else { /\* child process \*/

 // close the unused end of the pipe

 close(fd[WRITEEND]);

 // read from the pipe

 read(fd[READEND], readmsg, BUFFERSIZE);

 cout<<"read: "<<readmsg;

 // close the write end of the pipe

 close(fd[READEND]);

 }

 return 0;

}



 **QUESTION-3**

**Write a program 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**

**Write a program to demonstrate producer-consumer problem using shared memory**

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

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <fcntl.h>

#include <sys/shm.h>

#include <sys/stat.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <errno.h>

#include <iostream>

using namespace std;

int main()

{

 // the size (in bytes) of shared memory object

 const int SIZE = 4096;

 // name of the shared memory object

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

 // strings written to shared memory

 const char\* message\_0 = "Hello,";

 const char\* message\_1 = " Have a Nice day!";

 // shared memory file descriptor

 int shm\_fd;

 // pointer to shared memory obect

 void\* ptr;

 // create the shared memory object

 shm\_fd = shm\_open(name, O\_CREAT | O\_RDWR, 0666);

 if (shm\_fd < 0)

 {

 cout<<"In shm\_open() of producer.";

 exit(1);

 }

 // configure the size of the shared memory object

 ftruncate(shm\_fd, SIZE);

 // memory map the shared memory object

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

if (ptr == NULL)

 { cout<<"Error In mmap() requesting .";

 exit(1);

 }

 char \*cptr=(char\*)ptr;

 // write to the shared memory object

 cptr=strcat(cptr,message\_0);

 cptr=strcat(cptr,message\_1);

return 0;

}

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;

int main()

{

 const int SIZE =4096;

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

 // shared memory file descriptor

 int shm\_fd;

 // pointer to shared memory object

 void\* ptr;

 // open the shared memory object

 shm\_fd = shm\_open(name, O\_RDONLY, 0666);

 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

 //printf("%s", (char\*)ptr);

 cout<<(char\*)ptr;

// remove the shared memory object

 shm\_unlink(name);

 }

 return 0;

}



 **QUESTION-5**

**Write a program to understand working of Pthread library.**

#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**

**Write a program 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;

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**

**Write a program to copy a source file into the target file and display the target file using system calls.**

#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);

}



**----------------------------------------------------------------------------------**

 **THE END**

**----------------------------------------------------------------------------------**