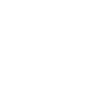
**●1.Write a C program to create „n‟ children. When the children will terminate,**

**display total cumulative time children spent in user and kernel mode\*/**



**1**

#include<sys/types.h>

#include<sys/wait.h>

#include<unistd.h>

#include<time.h>

#include<sys/times.h>

#include<stdio.h> #include<stdlib.h> int main(void)

{ int i, status; pid\_t pid; time\_t currentTime; struct tms cpuTime;

if((pid = fork())==-1) //start child process

{

perror("\nfork error"); exit(EXIT\_FAILURE);

}

else if(pid==0) //child process

{

time(&currentTime);

printf("\nChild process started at %s",ctime(&currentTime)); for(i=0;i<5;i++)

{

printf("\nCounting= %dn",i); //count for 5 seconds sleep(1);

}

time(&currentTime);

printf("\nChild process ended at %s",ctime(&currentTime));

exit(EXIT\_SUCCESS);

}

else

{ //Parent process

time(&currentTime); // gives normal time

printf("\nParent process started at %s ",ctime(&currentTime)); if(wait(&status)== -1) //wait for child process perror("\n wait error"); if(WIFEXITED(status))

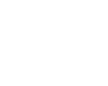
printf("\nChild process ended normally");

else

printf("\nChild process did not end normally"); if(times(&cpuTime)<0) //Get process time

perror("\nTimes error"); else

{ // \_SC\_CLK\_TCK: system configuration time: seconds clock tick printf("\nParent process user time= %fn",((double) cpuTime.tms\_utime));



**2**

printf("\nParent process system time = %fn",((double) cpuTime.tms\_stime));

printf("\nChild process user time = %fn",((double) cpuTime.tms\_cutime));

printf("\nChild process system time = %fn",((double)

cpuTime.tms\_cstime));

}

time(&currentTime);

printf("\nParent process ended at %s",ctime(&currentTime)); exit(EXIT\_SUCCESS);

}

}

**●**2.**To generate parent process to write unnamed pipe and will read from it.\*/**

#include <stdio.h>

#include<string.h>

#define READ 0

/\* The index of the “read” end of the pipe \*/ #define WRITE 1

/\* The index of the “write” end of the pipe \*/ main ()

{

char \*phrase="Dr.D.Y.Patil College"; int fd[2],bytesRead;

char message[100]; /\* Parent process’s message buffer \*/ pipe(fd); /\*Create an unnamed pipe\*/ if(fork()>0) /\* Parent Writer \*/

{

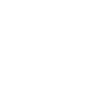
close(fd[READ]); /\* Close unused end\*/

write(fd[WRITE],phrase,strlen(phrase)+1); /\* include NULL\*/ close(fd[WRITE]);/\* Close used end\*/

}

else if (fork()==0)/\* Child Reader \*/ {

close(fd[WRITE]); /\* Close unused end\*/ bytesRead=read(fd[READ],message,100); printf("Read %d bytes: %s\n",bytesRead,message); close(fd[READ]);/\*close used end\*/



**3**

}

}

**●3.** **create a new file by named as file.txt**

#include<stdio.h>

#include<fcntl.h> #include"string.h"

int main()

{

int n=creat("file.txt","w"); char ch[16]="hello world how are"; char str[20]="od -c file.txt";

//od command in Linux is used to output the contents of a file in different //formats with the octal format being the default.

//od [OPTION]... [FILE]...

//Display contents of file in character format using -c option //change permission of file.txt with maximum access system("chmod 777 file.txt");

//write "helloworld string in file.txt

write(n,ch,16);

// to move cursor from beginning to 48th position

lseek(n,48,SEEK\_SET);

//write "helloworld string in file.txt

write(n,ch,16);

// to prompt command in command prompt system(str);

return(0);

}

**●4.LS Camand without system call**

#include<stdlib.h>

#include<stdio.h> #include<string.h>

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

{

char d[50]; if(argc==2)

{

bzero(d,sizeof(d)); strcat(d,"ls "); strcat(d,"> "); strcat(d,argv[1]); system(d);

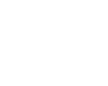
}

else

printf("\nInvalid No. of inputs");

}

**●5. To handle the two-way communication between parent and child using pipe.**



**4**

#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;

}

**●6.Exit function**

#include <stdio.h> #include <stdlib.h> void done()

{

printf("Exiting Successfully\n"); // Executed second

}

// Driver Code

int main()

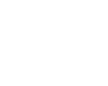
{ int value; value = atexit(done); if (value != 0) {

printf("atexit () function registration failed");

exit(1); //abnormal termination

}

printf(" Registration successful \n"); // Executed First return 0;



**5**

}

**●7.** **/\*C library function - atexit() Description**

#include <stdio.h> #include <stdlib.h> void fun1 ()

{

printf("This is fun1\n");

}

int main () {

/\* register the termination function \*/ atexit(fun1); printf("Starting main program...\n"); printf("Exiting main program...\n");

return(0);

}

**●8.** **Position**

#include <stdio.h> #include<fcntl.h>

int main()

{

int fd1,position;

fd1=open("/root/unix/f1.txt",O\_RDONLY); position=lseek(fd1,0,2);

printf("Size of file is %d bytes\n",position); return 0;

}

**●9.** **Read the current directory and display the name of the files, no of files in current directory.\*/** #include<stdlib.h>

#include<stdio.h> #include<string.h> main(int argc, char \*argv[])

{

char d[50]; int cnt=0; if(argc==2)

{

bzero(d,sizeof(d)); strcat(d,"ls -l | wc -l "); strcat(d,argv[1]); system(d);

}

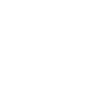
else

printf("\nInvalid No. of inputs");

//printf("No. of files in %s is %d",argv[1]);

}

**●10.Write a C program to display all the files from current directory which are created in particular**



**6**

#include<stdio.h>

#include<dirent.h>

#include<string.h>

#include<sys/stat.h>

#include<time.h> #include<stdlib.h>

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

{

char in[100],st[100],\*ch,\*ch1,c,buff[512];

DIR \*dp;

int i;

struct dirent \*ep;

struct stat sb; char mon[100]; dp=opendir("./");

if (dp != NULL)

{

while(ep =readdir(dp))

{

if(stat(ep->d\_name,&sb) == -1)

{

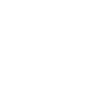
perror("stat");

exit(EXIT\_SUCCESS);

}

strcpy(mon,ctime(&sb.st\_ctime)); ch=strtok(mon," ");

/\*The first call to strtok() returns a pointer to the first token in the string pointed to by s1. Subsequent calls to strtok() must pass a NULL pointer as the first argument, in order to get the next token in the string.\*/ ch=strtok(NULL," "); if((strcmp(ch,argv[1]))==0)



**7**

{

printf("%s\t\t%s",ep->d\_name,ctime(&sb.st\_ctime));

}

}

(void)closedir(dp);

}

return 0;

}

**●11.Write a C program to implement the following unix/linux command i. ls –l > output.txt.\*/**

#include<stdlib.h>

#include<stdio.h> #include<string.h>

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

{

char d[50]; if(argc==2)

{

bzero(d,sizeof(d)); strcat(d,"ls -l "); strcat(d,"> "); strcat(d,argv[1]); system(d);

}

else

printf("\nInvalid No. of inputs");

}

**●12.**

**●13.**

**/\*Write a program that behaves like shell (Command Interpreter). It should display the command prompt as “MyShell$”.**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h> char \*buff,\*t1,\*t2,\*t3,ch;

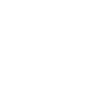
FILE \*fp;

int pid;

void count(char \*t2,char \*t3)

{

int charcount=0,wordcount=0,linecount=0;



**8**

if((fp=fopen(t3,"r"))==NULL)

printf("File not found");

else

{

while((ch=fgetc(fp))!=EOF)

{

if(ch==' ') wordcount++;

else if(ch=='\n')

{

linecount++; wordcount++;

}

else

charcount++;

}

fclose(fp); if(strcmp(t2,"c")==0)

printf("The total no. of characters :%d\n",charcount);

else if(strcmp(t2,"w")==0)

printf("The total no. of words :%d\n",wordcount); else if(strcmp(t2,"l")==0)

printf("The total no. of lines :%d\n",linecount); else

printf("Command not found");

}

}

main()

{

while(1)

{

printf("myshell$"); fflush(stdin);

t1=(char \*)malloc(80); t2=(char \*)malloc(80);

t3=(char \*)malloc(80); buff=(char \*)malloc(80); fgets(buff,80,stdin); sscanf(buff,"%s %s %s",t1,t2,t3); if(strcmp(t1,"pause")==0)

exit(0);

else if(strcmp(t1,"count")==0)

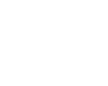
count(t2,t3);

else

{

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

printf("Child process is not created\n");



**9**

else if(pid==0)

{

//execlp("/bin",NULL) if(strcmp(t1,"exit")==0)

exit(0); system(buff);

}

else

{

wait(NULL);

exit(0);

}

}

}

}

**●14. Write a C program which receives file names as command line arguments and display those filenames in ascending order according to their sizes.\*/** #include<stdio.h>

#include<dirent.h>

#include<string.h>

#include<sys/stat.h>

#include<time.h> #include<stdlib.h>

struct filelist

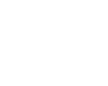
{

char fname[100];

int fsize;

};

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



**10**

{

DIR \*dp;

int i,j,k;

struct dirent \*ep;

struct stat sb; char mon[100]; struct filelist f1[100],temp; j=0; for(i=1;i<argc;i++)

{

dp=opendir("./");

if (dp!=NULL)

{

while(ep=readdir(dp))

{

if((strcmp(ep->d\_name,argv[i]))==0)

{

stat(ep->d\_name,&sb); strcpy(f1[j].fname,ep->d\_name); f1[j].fsize=sb.st\_size; j++;

break;

}

}

}

(void)closedir(dp);

} for(i=0;i<j;i++)

{

for(k=0;k<=j;k++)

{

if(f1[i].fsize < f1[k].fsize)

{

temp=f1[k]; f1[k]=f1[i];

f1[i]=temp;

}

}

} for(i=0;i<j;i++)

{

printf("%s\t%d\n",f1[i].fname,f1[i].fsize); }

return 0;

}

**15.file exist or not**

#include<stdio.h>

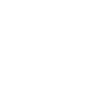
#include<stdio.h> #include<dirent.h>

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

{

DIR \*d;

struct dirent \*dir; // pointer for directory entry if(argc!=2)



**11**

{

printf("Invalid Arguments");

exit(0);

}

d=opendir("."); if(d==NULL)

{

printf("Could not open the current directory"); return(0);

}

while((dir=readdir(d))!=NULL)

{

if(strcmp(dir->d\_name ,argv[1])==0)

printf("File exist"); else

printf("File doesen't exist");

}

closedir(d);

}

**16.inode\_using\_arg**

#include<stdio.h>

#include<sys/stat.h>

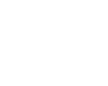
#include<unistd.h> #include<fcntl.h>

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

{ struct stat fileStat;

int ch,fd=0; FILE \*fname; if(argc!=3)

{ printf("Invalid number of arguments ");



**12**

return 1;

}

if(fname==NULL)

{

printf("Can't find the source file");

return 1;

}

//if((fd=open(filename,O\_RDONLY))==-1)

//{

//perror("open");

//}

if(fstat(fd,&fileStat)<0) return 1; printf("Information for %s\n ",filename); printf("Inode:\t%d\n",fileStat.st\_ino);

}

**17.Fork\_System \_Call**

#include<stdio.h>

#include<unistd.h>

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

int main()

{

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

{

printf("Fork Failed \n"); exit(1);

}

else if(pid==0)

{

execlp("/bin/ls","ls","-l",NULL); // Execute ls

}

else

{

wait(NULL); printf("\nChild Complete");

exit(0);

}

}

**18.Extended Shell(Toy Shell)**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

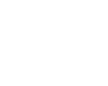
#include<unistd.h>

#include<sys/types.h>

#include<fcntl.h>

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

char cmd[80],token1[20],token2[20],token3[20],token4[20]; void count(); int main()



**13**

{

int pid; int n;

while(1)

{

printf("\nNewshell$");

gets(cmd);

n=sscanf(cmd,"%s%s%s%s",token1,token2,token3,token4);

if(strcmp(token1,"q")==0)

{

exit(0);

}

pid=fork(); if(pid==-1)

{

printf("Error\n");

}

else if(pid>0)

{

wait(NULL);

}

else

{

if(strcmp(token1,"list")==0)

{

list(token3,token2[0]);

exit(0);

}

if(strcmp(token1,"typeline")==0)

{

typeline(token3,token2);

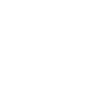
exit(0);

}

if(strcmp(token1,"search")==0)

{

search(token3,token2[0],token4); exit(0);



**14**

}

if(strcmp(token1,"count")==0)

{ count(); exit(0);

}

switch(n)

{

case 1:execlp(token1,token1,NULL); break;

case 2:execlp(token1,token1,token2,NULL);

break;

case

3:execlp(token1,token1,token2,token3,NULL);

break;

case

4:execlp(token1,token1,token2,token3,token4,NULL);

break;

}

}

}

}

void list(char \*dn, char op)

{

DIR \*dp;

struct dirent \*entry; int dc=0,fc=0; dp = opendir(dn);

if(dp==NULL)

{

printf("Dir %s not found.\n",dn);

return;

}

switch(op)

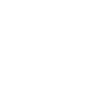
{

case 'f':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_REG) printf("%s\n",entry->d\_name);



**15**

}

break; case 'n': while(entry=readdir(dp))

{

if(entry->d\_type==DT\_DIR) dc++; if(entry->d\_type==DT\_REG) fc++;

}

printf("%d Dir(s)\t%d File(s)\n",dc,fc);

break; case 'i': while(entry=readdir(dp))

{

if(entry->d\_type==DT\_REG) printf("%s\t%d\n",entry->d\_name,entry-

>d\_fileno);

} break;

}

closedir(dp);

}

void search(char \*fn, char op, char \*pattern)

{

int fh,count=0,i=0,j=0; char buff[255],c,\*p; fh = open(fn,O\_RDONLY); if(fh==-1)

{

printf("File %s Not Found\n",fn);

return;

}

switch(op)

{ case 'f':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

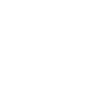
buff[j]='\0';

j=0; i++;

if(strstr(buff,pattern))

{

printf("%d: %s",i,buff);



**16**

break;

}

}

}

break; case 'c': while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j=0;

p = buff;

while(p=strstr(p,pattern))

{

count++; p++;

}

}

}

printf("Total No.of Occurrences = %d\n",count); break; case 'a': while(read(fh,&c,1))

{

buff[j++]=c; if(c=='\n')

{

buff[j]='\0';

j = 0; i++;

if(strstr(buff,pattern)) printf("%d: %s",i,buff);

}

} }//switch close(fh); }//search

void typeline(char \*fn, char \*op)

{

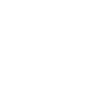
int fh,i,j,n;

char c;

fh = open(fn,O\_RDONLY); if(fh==-1)

{

printf("File %s not found.\n",fn);



**17**

return;

}

if(strcmp(op,"a")==0)

{

while(read(fh,&c,1)>0) printf("%c",c); close(fh);

return;

}

n = atoi(op);

if(n>0)

{ i=0;

while(read(fh,&c,1)>0)

{

printf("%c",c); if(c=='\n') i++;

if(i==n) break;

}

}

if(n<0)

{ i=0;

while(read(fh,&c,1)>0)

{

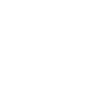
if(c=='\n') i++;

}

lseek(fh,0,SEEK\_SET);

j=1; while(read(fh,&c,1)>0) {

if(c=='\n') j++; if(j==i+n) break;



**18**

}

while(read(fh,&c,1)>0)

{

printf("%c",c);

}

}

close(fh);

}

void count()

{

int fd;

char ch;

int c=0,w=0,l=0; fd=open(token3,O\_RDONLY);

while(read(fd,&ch,1))

{

if(ch!='\n'&& ch!='\t'&&ch!=' ')

c++;

if(ch=='\n'||ch=='\t'||ch==' ')

w++;

if(ch=='\n') l++;

}

if(strcmp(token2,"c")==0) printf("\ncharacter count=%d",c); if(strcmp(token2,"w")==0)

printf("\nwordcount=%d",w); if(strcmp(token2,"l")==0)

printf("\nlinecount=%d",l);

}

**19.Pipe**

#include<stdio.h>

#include<sys/stat.h>

#include<unistd.h> #include<fcntl.h> int main()

{ struct stat fileStat; int fd=0;

FILE \*filename="infile.txt";

if((fd=open(filename,O\_RDONLY))==-1)

{

perror("open");

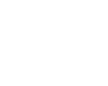
}

if(fstat(fd,&fileStat)<0) return 1; printf("Information for %s\n ",filename);

printf("Inode:\t%d\n",fileStat.st\_ino);

}

**20.Write a c program that illustrates inter process communication using shared memory.\*/** #include <stdio.h>



**19**

#include <sys/ipc.h> #include <sys/shm.h>

int main()

{

key\_t key = ftok("shmfile",65);

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

printf("MemID=%d",shmid);

char \*str = (char\*) shmat(shmid,(void\*)0,0);

printf("Data read from memory: %s\n",str); shmdt(str);

shmctl(shmid,IPC\_RMID,NULL);

return 0;

}

**21.Change the first character of the segment**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h> #define MAXSIZE 27

void die(char \*s)

{ perror(s); exit(1);

}

int main()

{ int shmid; key\_t key;

char \*shm, \*s;

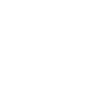
key = 5678;

if ((shmid = shmget(key, MAXSIZE, 0666)) < 0) die("shmget");

if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1) die("shmat"); for (s = shm; \*s != '\0'; s++) putchar(\*s); putchar('\n'); \*shm = '\*'; exit(0);

}

**22.** **Wait until the other process changes the first character of our memory**



**20**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h> #define MAXSIZE 27

void die(char \*s)

{ perror(s); exit(1);

}

int main()

{ char c; int shmid; key\_t key; char \*shm, \*s; key = 5678;

if ((shmid = shmget(key, MAXSIZE, IPC\_CREAT | 0666)) < 0) die("shmget");

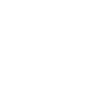
if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1) die("shmat"); s = shm;

for (c = 'a'; c <= 'z'; c++)

\*s++ = c; while (\*shm != '\*') sleep(1); exit(0);

}

**23.word\_line,Character \_count** #include <stdio.h> int main()



**21**

{

FILE \*fp; char filename[100]; char ch;

int linecount, wordcount, charcount; linecount = 0; wordcount = 0; charcount = 0;

printf("Enter a filename :"); gets(filename); fp = fopen(filename,"r");

if ( fp )

{

while ((ch=getc(fp)) != EOF) { if (ch != ' ' && ch != '\n') { ++charcount; } if (ch == ' ' || ch == '\n') { ++wordcount; } if (ch == '\n') { ++linecount; }

}

if (charcount > 0) {

++linecount;

++wordcount;

}

}

else

{

printf("Failed to open the file\n");

}

printf("Lines : %d \n", linecount); printf("Words : %d \n", wordcount); printf("Characters : %d \n", charcount); return(0);

}

**24.**

#include "apue.h" #include <fcntl.h> #define RWRWRW

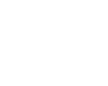
(S\_IRUSR|S\_IWUSR|S\_IRGRP|S\_IWGRP|S\_IROTH|S\_IWOTH) int main(void) { umask(0); if (creat("foo", RWRWRW) < 0) err\_sys("creat error for foo"); umask(S\_IRGRP | S\_IWGRP | S\_IROTH | S\_IWOTH); if (creat("bar",

RWRWRW) < 0) err\_sys("creat error for bar"); exit(0); }

#include "apue.h" int main(void) { struct stat statbuf; /\* turn on set-group-ID and turn off group-execute \*/ if (stat("foo", &statbuf) < 0) err\_sys("stat error for foo"); if

(chmod("foo", (statbuf.st\_mode & ˜S\_IXGRP) | S\_ISGID) < 0) err\_sys("chmod error for foo"); /\* set absolute mode to "rw-r--r--" \*/ if (chmod("bar", S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH) < 0) err\_sys("chmod error for bar"); exit(0); }

**25.Write a c program that illustrates inter process communication using shared memory.\*/**



**22**

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

int main()

{

key\_t key = ftok("shmfile",55);

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

printf("MemID=%d",shmid);

char \*str = (char\*) shmat(shmid,(void\*)0,0); printf("Write Data : "); gets(str);

printf("Data written in memory: %s\n",str);

//detach from shared memory

shmdt(str);

return 0;

}