Unix Term works

Termwork-1

Write a C/C++ POSIX compliant program to check the following limits: (i) No. of clock ticks (ii) Max. no. of child processes (iii) Max. path length (iv) Max. no. of characters in a file name (v) Max. no. of open files/ process

Source Code:

```
#define _POSIX_C_SOURCE 199309L
#include<iostream>
#include<unistd.h>
#include<limits.h>
using namespace std;
int main(){
        int choice,res=0;
        while(1){
        cout<<"1. Compile Time Values\n2. Run Time Values\n3.Exit\nEnter your choice:";
                cin>>choice;
                switch(choice){
                        case 1: cout<<"Compile Time Values.\n";
                                #ifdef _POSIX_CLK_TCK
                cout<<"No. of Clock Ticks per second is: "<<_POSIX_CLK_TCK<<endl;
                                #else
                                        cout<<"_POSIX_CLK_TCK not defined\n";
                                #endif
                                #ifdef _POSIX_CHILD_MAX
cout<<"Max number of child processes at any time is: "<<_POSIX_CHILD_MAX<<endl;
                                #else
                                        cout<<"_POSIX_CHILD_MAX not defined\n";
                                #endif
                                #ifdef _POSIX_PATH_MAX
                                cout<<"Max path name is: "<<_POSIX_PATH_MAX<<endl;
                                #else
```

```
                    cout<<"_POSIX_PATH_MAX not defined\n";
               #endif
               #ifdef _POSIX_NAME_MAX
     cout<<"Max no. of characters in file name: "<<_POSIX_NAME_MAX<<endl;
               #else
                    cout<<"_POSIX_NAME_MAX not defined\n";
               #endif
               #ifdef _POSIX_OPEN_MAX
cout<<"Max no. of files simultaneously opened is: "<<_POSIX_OPEN_MAX<<endl;
               #else
                    cout<<"_POSIX_OPEN_MAX not defined\n";
               #endif
               break;
          case 2:   cout<<"Run Time Values.\n";
               if((res=sysconf(_SC_CLK_TCK))==-1)
                    perror("sysconf");
               else
                    cout<<"No. of Clock Ticks per second is: "<<res<<endl;
               if((res=sysconf(_SC_CHILD_MAX))==-1)
                    perror("sysconf");
               else
          cout<<"Max number of child processes at any time is: "<<res<<endl;
               if((res=pathconf("/",_PC_PATH_MAX))==-1)
                    perror("pathconf");
               else
                    cout<<"Max path name is: "<<res<<endl;
               if((res=pathconf("/",_PC_NAME_MAX))==-1)
                    perror("pathconf");
               else
                    cout<<"Max no. of characters in file name: "<<res<<endl;
                    if((res=sysconf(_SC_OPEN_MAX))==-1)
                        perror("sysconf");
                    else
                    cout<<"Max no. of files simultaneously opened is:"<<res<<endl;
                    break;
                case 3: exit(0);
                default : cout<<"Invalid Choice.\n";
            }
        }
        return 0;
}
```

Termwork-2

Write a C/C++ POSIX compliant program that prints the POSIX defined configuration options supported on any given system using feature test macros.

Source Code:

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include<unistd.h>
#include<iostream>
using namespace std;
int main(){
        #ifdef _POSIX_JOB_CONTROL
                cout<<"System Supports Job Control feature"<<endl;
        #else
                cout<<"System does not support job control\n";
        #endif

        #ifdef _POSIX_SAVED_IDS
                cout<<"System Supports saved set-UID and saved set-GID"<<endl;
        #else
```

```
            cout<<"System does not support saved set-UID\n";
        #endif

        #ifdef _POSIX_CHOWN_RESTRICTED
            cout<<"System Supports change ownership feature"<<endl;
        #else
            cout<<"System does not support change ownership feature\n";
        #endif

        #ifdef _POSIX_NO_TRUNC
            cout<<"System Supports path truncation option."<<endl;
        #else
            cout<<"System does not support path truncation\n";
        #endif

        #ifdef _POSIX_VDISABLE
            cout<<"System Supports disable character for files."<<endl;
        #else
            cout<<"System does not support disable character\n";
        #endif
        return 0;
}
```

Termwork-3

Consider the last 100 bytes as a region. Write a C/C++ program to check whether the region is locked or not. If the region is locked, print pid of the process which has locked. If the region is not locked, lock the region with an exclusive lock, read the last 50 bytes and unlock the region.

Source Code:

```
#include<stdio.h>
#include<sys/types.h>
```

```
#include<unistd.h>
#include<fcntl.h>
int main(int argc,char *argv[]){
        char temp[1000];
        setbuf(stdout,temp);
        struct flock fvar;
        int fdesc,rc;
        char buf;
        off_t offset;
        pid_t pid=fork();
        fdesc=open(argv[1],O_RDWR);
        offset=lseek(fdesc,-100,SEEK_END);
        fvar.l_type=F_WRLCK;
        fvar.l_whence=SEEK_CUR;
        fvar.l_start=0;
        fvar.l_len=100;
        if(fcntl(fdesc,F_SETLK,&fvar)==-1){
                printf("\n--------------------\nFile has been locked by:\n");
                while(fcntl(fdesc,F_GETLK,&fvar)!=-1 && fvar.l_type!=F_UNLCK){
                        printf("\nFile: %s is locked by process with pid: %u",argv[1],fvar.l_pid);
                        printf(" from %ld the byte in the file for: %ld",fvar.l_start,fvar.l_len);
                printf(" number of bytes, for %s\n",(fvar.l_type==F_WRLCK?"write":"read"));
                        if(!fvar.l_len) break;
                        fvar.l_start+=fvar.l_len;
                        fvar.l_len=0;
                }
        }
        else{
                printf("\n--------------------\n");
        printf("\n\nFile: %s was not locked and acquiring of Exclusive lock was",argv[1]);
```

```
                printf(" successful by Process ID: %u",getpid());
                offset=lseek(fdesc,-50,SEEK_END);
                printf("\n\nLast 50 bytes of file: %s = \n",argv[1]);
                while((rc=read(fdesc,&buf,1))>0)        printf("%c",buf);
                offset=lseek(fdesc,-100,SEEK_END);
                fvar.l_type=F_UNLCK;
                fvar.l_whence=SEEK_CUR;
                fvar.l_start=0;
                fvar.l_len=100;
                fcntl(fdesc,F_SETLK,&fvar);
                printf("\nFile unlocked successfully.\n");
        }
        return 0;
}
```

Termwork-4

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

Source Code:

Server Side:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<string.h>
#define FIFO1 "fifo1"
#define FIFO2 "fifo2"
#define PERMS 0666
char fname[256];
int main(){
        int readfd,writefd,fd;
        ssize_t n;
        char buff[512];
        if(mkfifo(FIFO1,PERMS)<0)
                printf("Cant Create FIFO files\n");
        if(mkfifo(FIFO2,PERMS)<0)
                printf("Cant Create FIFO files\n");
        printf("Waiting for connection Request.\n");
        readfd=open(FIFO1,O_RDONLY,0);
        writefd=open(FIFO2,O_WRONLY,0);
        printf("Connection Established\n");
        read(readfd,fname,255);
        printf("Client has requested file %s\n",fname);
        if((fd=open(fname,O_RDWR))<0){
                strcpy(buff,"File does not exist.\n");
                write(writefd,buff,strlen(buff));
        } else {
                while((n=read(fd,buff,512))>0)
                        write(writefd,buff,n);
        }
        close(readfd);unlink(FIFO1);
        close(writefd);unlink(FIFO2);
}
```

Client Side:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<string.h>
#define FIFO1 "fifo1"
#define FIFO2 "fifo2"
```

```
#define PERMS 0666
char fname[256];
int main(){
	ssize_t n;
	char buff[512];
	int readfd,writefd;
	printf("Trying to connect to server.\n");
	writefd=open(FIFO1,O_WRONLY,0);
	readfd=open(FIFO2,O_RDONLY,0);
	printf("Connected...\n");
	printf("Enter the filename to request from server:");
	scanf("%s",fname);
	write(writefd,fname,strlen(fname));
	printf("Waiting for server to reply..\n");
	while((n=read(readfd,buff,512))>0)
		write(1,buff,n);
	close(readfd);
	close(writefd);
	return 0;
}
```

Termwork-5

a) Write a C/C++ program that outputs the contents of its Environment list

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main(int argc,char *argv[]){
	int i;
	char **ptr;
	extern char **environ;
	for(ptr=environ;*ptr;ptr++)
		printf("%s\n",*ptr);
	exit(0);
}
```

b) Write a C / C++ program to emulate the unix ln command

Source Code:

```
#include<stdio.h>
#include<unistd.h>
int main(int argc,char *argv[]){
	if(argc!=3){
		printf("Usage: %s <src_file><dest_file>\n",argv[0]);
		return 0;
	}
	if(link(argv[1],argv[2])==-1){
		printf("Link Error\n");
		return 1;
	}
	else
		printf("Hard link created successfully.\n"); //link success
	return 0;
}
```

Termwork-6

Write a C/C++ program to illustrate the race condition.

Source Code:

```
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
static void charatatime(char *);
```

```
int main(){
        int pid,i;
        for(i=0;i<3;i++){
        for(i=0;i<3;i++){
        if((pid=fork())<0)
                printf("fork error.\n");
        else if(pid==0)
                charatatime("output from child\n");
        else
                charatatime("output from parent\n");
}
}
        _exit(0);
}
static void charatatime(char *str){
        char *ptr;
        int c;
        setbuf(stdout,NULL);
        for(ptr=str;(c=*ptr++)!=0;)
                putc(c,stdout);
}
```

Termwork-7

Write a C/C++ program that creates a zombie and then calls system to execute the ps command to Verify that the process is zombie

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main(){
        pid_t pid;
```

```
	if((pid=fork())<0)
		perror("fork error");
	else if (pid==0)
		_exit(0);  //child
	sleep(4); // parent
	system("ps -o pid,ppid,state,tty,command");
	_exit(0);
}
```

Termwork-8

Write a C/C++ program to avoid zombie process by forking twice

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main(){
	int pid=fork();
	if (pid==0){
		//first child
		pid=fork();//creates second child or grandchild
		if(pid==0){
			//second child
			sleep(1);
			printf("Second child: My Parent PID is %d\n",getppid());
		}
	}
	else{
		//Parent process
		wait(NULL);//will block parent process until any of its children has finished
		sleep(2);
```

```
                system("ps -o pid,ppid,state,tty,command");
        }
        return 0;
}
```

Termwork-9

Write a C/C++ program to implement 'system' function.

Source Code:

```
#include<sys/wait.h>
#include<errno.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
int system1(const char *cmdstring){
        pid_t pid;
        int status;
        if(cmdstring==NULL)
                return (1);
        if((pid=fork())<0)
                status=-1;
        else if(pid==0){
                execl("/bin/sh","sh","-c",cmdstring,(char *)0);
                _exit(127);
        }
        else
                while(waitpid(pid,&status,0)<0){
                        if(errno != EINTR)
                                status=-1;
                        break;
                }
        return (status);

}
int main(){
        int status;
        if((status=system1("date"))<0)
                printf("system() error");
        if((status=system1("who"))<0)
                printf("system() error");
        exit(0);
}
```

Termwork-10

Write a C/C++ program to set up a real-time clock interval timer using the alarm API.

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<signal.h>
#define INTERVAL 5
void callme(int sig_no){
        alarm(INTERVAL);
        printf("Hello!!\n");
}
int main(){
        struct sigaction action;
        action.sa_handler=(void(*)(int))callme;
        sigaction(SIGALRM,&action,0);
        alarm(2);
        sleep(5);
        return 0;
}
```