**1.Write a C program that takes one or  
more file/directory names as command line input and reports the following information  
on the file:  
o File type  
o Number of links  
o Time of last access  
o Read, write, and execute  
permission**#include <unistd.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

char \* error\_msg[] = {

"\nUsage: ./12filestat <file|dir> [<file|dir>]\n\n", "\nFile does not exist !!\n\n", "\nError doing 'stat' on file\n\n"

};

void print\_error(int msg\_num, int exit\_code, int exit\_flag);

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

{

int i;

mode\_t file\_perm;

struct stat file\_details;

char success\_msg[] = "\nCommand executed successfully\n\n";

if ( argc < 2 )

print\_error(0,2,1);

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

{

printf("\n%s\n%s\n%s\n","----------------",argv[i],"----------------");

if ( access(argv[i],F\_OK) == -1 )

{

print\_error(1,3,0);

continue;

}

if ( lstat(argv[i],&file\_details) < 0 )

{

print\_error(2,4,0);

continue; }

if ( S\_ISREG(file\_details.st\_mode) )

printf("File type : Regular\n");

else if ( S\_ISDIR(file\_details.st\_mode) )

printf("File type : Directory\n");

else if ( S\_ISLNK(file\_details.st\_mode) )

printf("File type : Symbolic link\n");

else

printf("File type : Other");

printf("Number of links : %d\n", (int)file\_details.st\_nlink);

printf("Time of last access : %s",ctime(&file\_details.st\_atime));

printf("File Permissions:\n");

file\_perm = file\_details.st\_mode & ~S\_IFMT;

printf("\tUser : ");

if ( file\_perm & S\_IRUSR ) printf("Readable, ");

else printf("Not readable, ");

if ( file\_perm & S\_IWUSR ) printf("Writable, ");

else printf("Not writable, ");

if ( file\_perm & S\_IXUSR ) printf("Executable\n");

else printf("Not executable\n");

printf("\tGroup : ");

if ( file\_perm & S\_IRGRP ) printf("Readable, ");

else printf("Not readable, ");

if ( file\_perm & S\_IWGRP ) printf("Writable, ");

else printf("Not writable, ");

if ( file\_perm & S\_IXGRP ) printf("Executable\n");

else printf("Not executable\n");

printf("\tOthers : ");

if ( file\_perm & S\_IROTH ) printf("Readable, ");

else printf("Not readable, ");

if ( file\_perm & S\_IWOTH ) printf("Writable, ");

else printf("Not writable, ");

if ( file\_perm & S\_IXOTH ) printf("Executable\n");

else printf("Not executable\n");

}

printf("%s", success\_msg);

return 1;

}

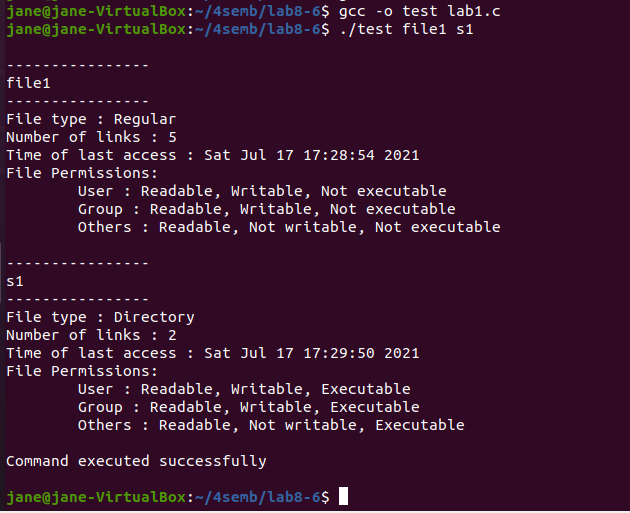
void print\_error(int error\_index, int exit\_code, int exit\_flag)

{

fprintf(stderr, "%s\n",error\_msg[error\_index]);

if (exit\_flag) exit(exit\_code);

}

**  
  
  
2.                   
Write a program in C that illustrates how to execute two commands concurrently with a command pipe.**#include<stdio.h>

#include<fcntl.h>

int main()

{

int pfd[2],p;

pipe(pfd);

p=fork();

if(p==0)

{

close(pfd[0]);

close(1);

dup(pfd[1]);

execlp("ls","ls","-l",(char\*)0);

}

else

{

close(pfd[1]);

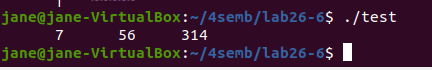
close(0);

dup(pfd[0]);

execlp("wc","wc",(char\*)0);

}

}

**  
  
  
3.      Write a C Program that makes a copy of a file using standard I/O and system calls**

#include <stdio.h>

#include <unistd.h>

#include <fcntl.h>

#include<stdlib.h>

void typefile (char \*filename)

{

int fd, nread;

char buf[1024];

fd = open (filename, O\_RDONLY);

if (fd == -1) {

perror (filename);

return;

}

while ((nread = read (fd, buf, sizeof (buf))) > 0)

write (1, buf, nread);

sleep(1);

close (fd);

}

int

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

{

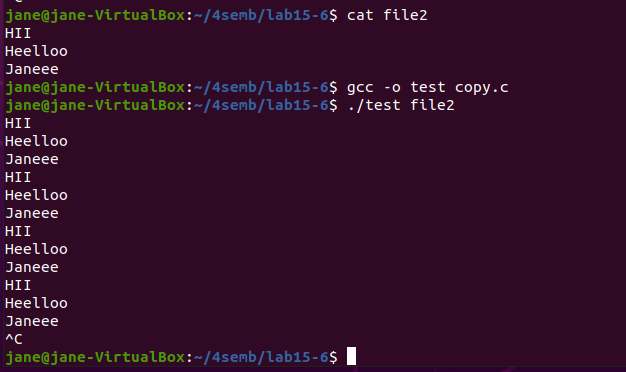
int argno;

for (argno = 1; argno < argc; argno )

typefile (argv[argno]);

exit (0);

}

**  
  
  
4. Implement in C the following Unix commands using system calls  
  
  
    A)Cat  B)mv c)cp d)ln**

**A)Cat**

#include<sys/types.h>

#include<sys/stat.h>

#include<unistd.h>

#include<stdio.h>

#include<fcntl.h>

int main( int argc,char \*argv[3] ){

int fd,i;

char buf[2];

fd=open(argv[1],O\_RDONLY,0777);

if(fd==-argc)

{

printf("file open error");

}

else

{

while((i=read(fd,buf,1))>0)

{

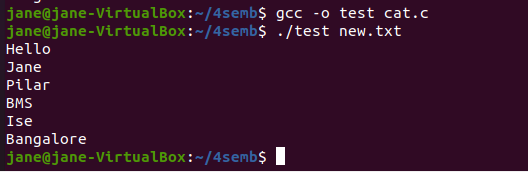
printf("%c",buf[0]);

}

close(fd);

}

}

****

**b)mv**

#include<sys/types.h>

#include<sys/stat.h>

#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

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

{

int i,fd1,fd2;

char \*file1,\*file2,buf[2];

file1=argv[1];

file2=argv[2];

printf("file1=%s file2=%s",file1,file2);

fd1=open(file1,O\_RDONLY,0777);

fd2=creat(file2,0777);

while(i=read(fd1,buf,1)>0)

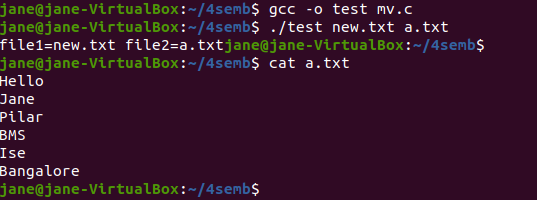
write(fd2,buf,1);

remove(file1);

close(fd1);

close(fd2);

}

****

**c)cp**

****

**In)**

#include<stdio.h>

#include<fcntl.h>

#include<unistd.h>

#include<sys/stat.h>

#include<stdlib.h>

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

{

int i;

struct stat s;

if (argc<3)

{

perror("ERROR:Too Few Arguments");

exit(1);

}

if(argc>4)

{

perror("ERROR:Too Many Arguments");

exit(1);

}

if(argc==3)

i=0;

else

i=1;

if(i && !(strcmp(argv[1],"-s")==0)) {

perror("ERROR:Invalid Syntax");

exit(1);

}

if(access(argv[i+1],F\_OK))

{

perror("ERROR:File name not Found");

exit(1);

}

if(!access(argv[i+2],F\_OK))

{

perror("ERROR:File Name already exist");

exit(1);

}

if(stat(argv[i+1],&s)<0)

{

perror("ERROR:Unable to reterive stat information"); exit(1);

}

if(!S\_ISREG(s.st\_mode))

{

perror("ERROR:Not a Regular File");

exit(1); }

if(argc==3)

if(link(argv[i+1],argv[i+2])<0)

{

perror("ERROR:Unable to create the Link");

exit(1);

}

if(argc==4)

if(symlink(argv[i+1],argv[i+2])<0)

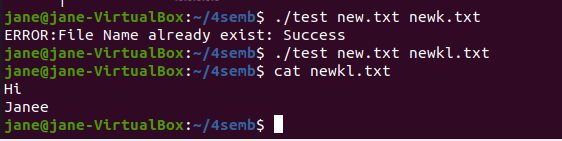
{

perror("ERROR:Unable to create the Link");

exit(1);

}

}

  
 **5. Write a C program to list for every file in a directory, its inode number and file name.**

#include<stdlib.h>

#include<stdio.h>

#include<string.h>

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

{

char d[50];

if(argc==2)

{

bzero(d,sizeof(d));

strcat(d,"ls ");

strcat(d,"-i ");

strcat(d,argv[1]);

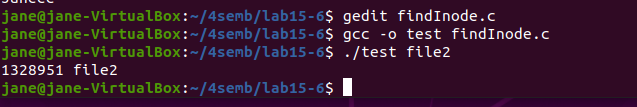
system(d);

}

else

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

}

**  
  
6. Write  
a C Program that demonstrates redirection of standard output to a file  
.EX:ls>f1.**

#include<stdlib.h>

#include<stdio.h>

#include<string.h>

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

{

char d[50];

if(argc==2)

{

bzero(d,sizeof(d));

strcat(d,"ls ");

strcat(d,"> ");

strcat(d,argv[1]);

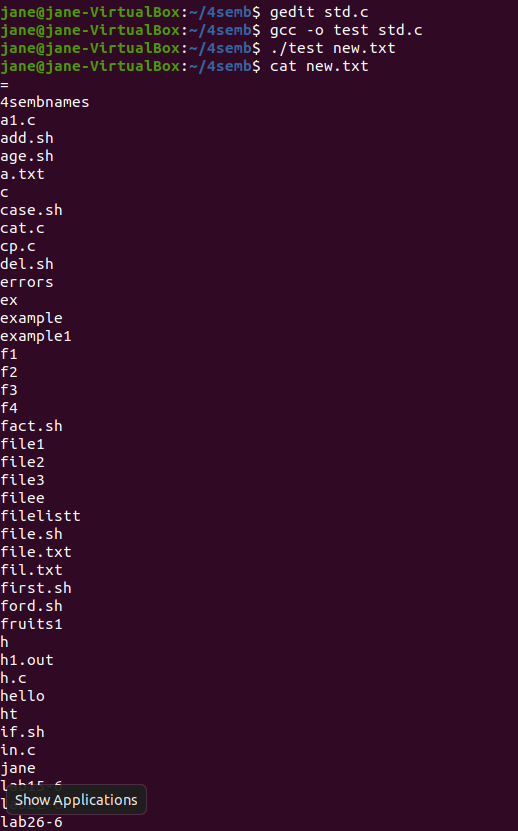
system(d);

}

else

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

}

  
 **7. Write  
a C program to create a child process and allow the parent to display “parent”  
and the child to display “child” on the screen** #include <stdio.h>

#include <sys/wait.h> /\* contains prototype for wait \*/

int main(void)

{

int pid;

int status;

printf("Hello World!\n");

pid = fork( );

if(pid == -1) /\* check for error in fork \*/

{

perror("bad fork");

exit(1);

}

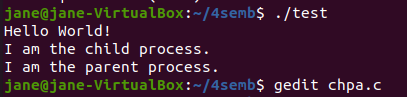
if (pid == 0)

printf("I am the child process.\n");

else { wait(&status); /\* parent waits for child to finish \*/

printf("I am the parent process.\n");

}

}  
**  
8.Write  
a C program to create a Zombie process.**#include <stdlib.h>

#include<stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main ()

{

int pid\_t,child\_pid;

child\_pid = fork ();

if (child\_pid > 0) {

sleep (10);

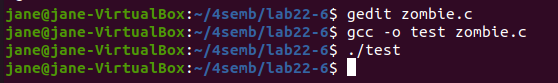
}

else {

exit (0);

}

return 0;

}  
 **  
  
  
9.Write  
a C program that illustrates how an orphan is created.**#include <stdio.h>

int main()

{

int pid ;

printf("I'am the original process with PID %d and PPID %d.\n",getpid(),getppid());

pid=fork();

if(pid!=0 )

{

printf("I'am the parent with PID %d and PPID %d.\n",getpid(),getppid());

printf("My child's PID is %d\n",pid) ;

}

else

{

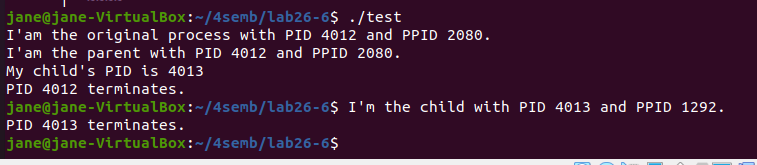
sleep(4);

printf("I'm the child with PID %d and PPID %d.\n",getpid(), getppid()) ;

}

printf ("PID %d terminates.\n", getpid()) ;

}

**  
   
  
  
10. Write  
a program that illustrates how to execute two commands concurrently with a  
command pipe.**

#include<stdio.h>

#include<fcntl.h>

main()

{

int pfd[2],p;

pipe(pfd);

p=fork();

if(p==0)

{

close(pfd[0]);

close(1);

dup(pfd[1]);

execlp("ls","ls","-l",(char\*)0);

}

else

{

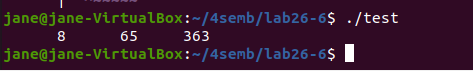
close(pfd[1]);

close(0);

dup(pfd[0]);

execlp("wc","wc",(char\*)0);

}

} **  
   
  
  
11. Write  
a C programs that illustrate communication between two unrelated processes  
using named pipe.**#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<string.h>

#include<fcntl.h>

void server(int,int);

void client(int,int);

int main()

{

int p1[2],p2[2],pid;

pipe(p1);

pipe(p2);

pid=fork();

if(pid==0)

{

close(p1[1]);

close(p2[0]);

server(p1[0],p2[1]);

return 0;}

close(p1[0]);

close(p2[1]);

client(p1[1],p2[0]);

wait();

return 0;

}

void client(int wfd,int rfd)

{

int i,j,n;

char fname[2000];

char buff[2000];

printf("ENTER THE FILE NAME :");

scanf("%s",fname);

printf("CLIENT SENDING THE REQUEST.....PLEASE WAIT\n");

sleep(10);

write(wfd,fname,2000);

n=read(rfd,buff,2000);

buff[n]='\0';

printf("THE RESULTS OF CLIENTS ARE......\n");

write(1,buff,n);

}

void server(int rfd,int wfd)

{

int i,j,n;

char fname[2000];

char buff[2000];

n=read(rfd,fname,2000);

fname[n]='\0';

int fd=open(fname,O\_RDONLY);

sleep(10);

if(fd<0)

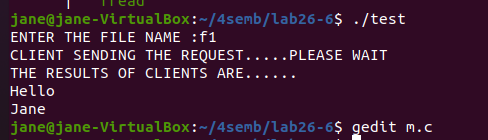
write(wfd,"can't open",9);

else

n=read(fd,buff,2000);

write(wfd,buff,n);

}

**  
   
  
  
12. Write  
a C program that illustrates suspending and resuming processes using signals**#include<stdio.h>

#include<signal.h>

#include<stdlib.h>

#include<unistd.h>

void alarms(int signo)

{

printf("alarmfunction.signo=%d\n,",signo);

}

int main()

{

if(signal(SIGALRM,alarms)==SIG\_ERR)

printf("Signalerror");

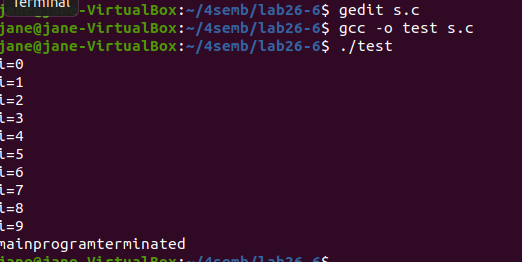
alarm(5);

for(int i=0;i<10;i++)

printf("i=%d\n",i);

printf("mainprogramterminated\n");

return 0;

} **  
  
  
13. Write a separate program using signal system call to catch the  
following signals.  
  
  
o   SIGSEGV**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <signal.h>

void sighandler(int signum)

{

printf("Process %d got signal %d\n", getpid(), signum);

signal(signum, SIG\_DFL);

kill(getpid(), signum);

}

int main()

{

signal(SIGSEGV, sighandler);

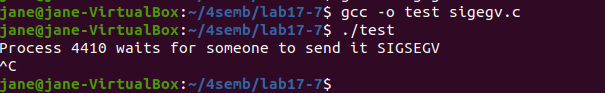
printf("Process %d waits for someone to send it SIGSEGV\n",

getpid());

sleep(1000);

return 0;

}

**  
  
o   SIGINT**#include<stdio.h>

#include<signal.h>

#include<unistd.h>

void sig\_handler(int signo)

{

if (signo == SIGINT)

printf("received SIGINT\n");

}

int main(void)

{

if (signal(SIGINT, sig\_handler) == SIG\_ERR) printf("\ncan't catch SIGINT\n");

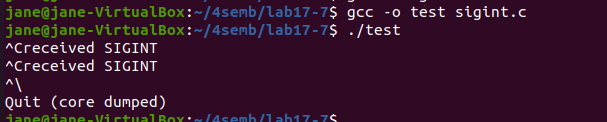
// A long long wait so that we can easily issue a signal to this process

while(1)

sleep(1);

return 0;

}

**  
o   SIGFPE**

#include <stdio.h>

#include <signal.h>

#include <setjmp.h>

#include <string.h>

jmp\_buf fpe;

void handler(int signum)

{

// Do stuff here then return to execution below

longjmp(fpe, 1);

}

int main()

{

volatile int i, j;

for(i = 0; i < 10; i++)

{

// Call signal handler for SIGFPE

struct sigaction act;

struct sigaction oldact;

memset(&act, 0, sizeof(act));

act.sa\_handler = handler;

act.sa\_flags = SA\_NODEFER | SA\_NOMASK;

sigaction(SIGFPE, &act, &oldact);

if (0 == setjmp(fpe))

{

j = i / 0;

sigaction(SIGFPE, &oldact, &act);

} else {

printf("handling SIGFPE....\n");

sigaction(SIGFPE, &oldact, &act);

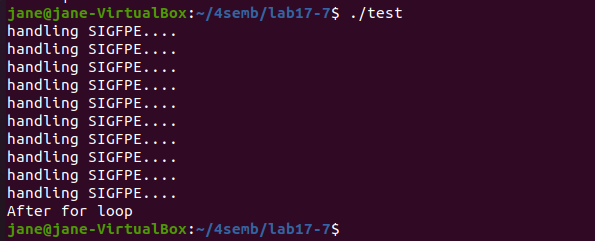
/\* handle SIGFPE \*/

}

}

printf("After for loop\n"); return 0;

}

**  
  
  
o   SIGALRM (use alarm system  
call)**#include<stdio.h>

#include<unistd.h>

#include<signal.h>

void sig\_handler(int signum){

printf("Inside handler function\n");

}

int main(){

signal(SIGALRM,sig\_handler); // Register signal handler

alarm(4); // Scheduled alarm after 4 seconds

alarm(1); // Scheduled alarm after 1 seconds

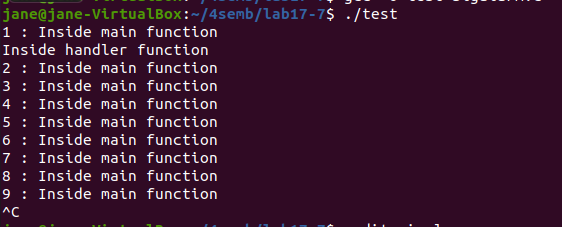
for(int i=1;;i++){

printf("%d : Inside main function\n",i);

sleep(1); // Delay for 1 second

}

return 0;

} **  
o   SIGALRM (use setitimer  
system call)**  
#include <sys/time.h> /\* for setitimer \*/

#include <unistd.h> /\* for pause \*/

#include <signal.h> /\* for signal \*/

#define INTERVAL 500 /\* number of milliseconds to go off \*/

/\* function prototype \*/

void DoStuff(void);

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

struct itimerval it\_val; /\* for setting itimer \*/

/\* Upon SIGALRM, call DoStuff().

\* Set interval timer. We want frequency in ms,

\* but the setitimer call needs seconds and useconds. \*/

if (signal(SIGALRM, (void (\*)(int)) DoStuff) == SIG\_ERR) {

perror("Unable to catch SIGALRM");

exit(1);

}

it\_val.it\_value.tv\_sec = INTERVAL/1000;

it\_val.it\_value.tv\_usec = (INTERVAL\*1000) % 1000000;

it\_val.it\_interval = it\_val.it\_value;

if (setitimer(ITIMER\_REAL, &it\_val, NULL) == -1) {

perror("error calling setitimer()");

exit(1);

}

while (1)

pause();

}

/\*

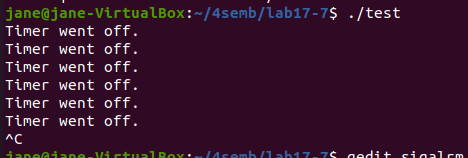
\* DoStuff

\*/

void DoStuff(void) {

printf("Timer went off.\n");

}

**  
o   SIGVTALRM (use setitimer  
system call)**#include <signal.h>

#include <stdio.h>

#include <string.h>

#include <sys/time.h>

void timer\_handler (int signum)

{

static int count = 0;

printf ("timer expired %d times\n", ++count);

}

int main ()

{

struct sigaction sa;

struct itimerval timer;

/\* Install timer\_handler as the signal handler for SIGVTALRM. \*/

memset (&sa, 0, sizeof (sa));

sa.sa\_handler = &timer\_handler;

sigaction (SIGVTALRM, &sa, NULL);

/\* Configure the timer to expire after 250 msec... \*/

timer.it\_value.tv\_sec = 0;

timer.it\_value.tv\_usec = 250000;

/\* ... and every 250 msec after that. \*/

timer.it\_interval.tv\_sec = 0;

timer.it\_interval.tv\_usec = 250000;

/\* Start a virtual timer. It counts down whenever this process is

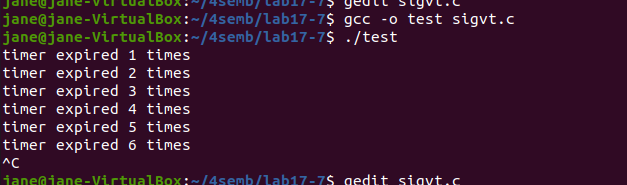
executing. \*/

setitimer (ITIMER\_VIRTUAL, &timer, NULL);

/\* Do busy work. \*/

while (1);

}

**  
  
o   SIGPROF (use setitimer  
system call).**

#include <signal.h>

#include <stdio.h>

#include <string.h>

#include <sys/time.h>

void timer\_handler (int signum)

{

static int count = 0;

printf ("sigprof timer expired %d times\n", ++count);

}

int main ()

{

struct sigaction sa;

struct itimerval timer;

/\* Install timer\_handler as the signal handler for SIGVTALRM. \*/

memset (&sa, 0, sizeof (sa));

sa.sa\_handler = &timer\_handler;

sigaction (SIGPROF, &sa, NULL);

/\* Configure the timer to expire after 250 msec... \*/

timer.it\_value.tv\_sec = 0;

timer.it\_value.tv\_usec = 250000;

/\* ... and every 250 msec after that. \*/

timer.it\_interval.tv\_sec = 0;

timer.it\_interval.tv\_usec = 250000;

/\* Start a virtual timer. It counts down whenever this process is

executing. \*/

setitimer (ITIMER\_PROF, &timer, NULL);

/\* Do busy work. \*/

while (1);

}

