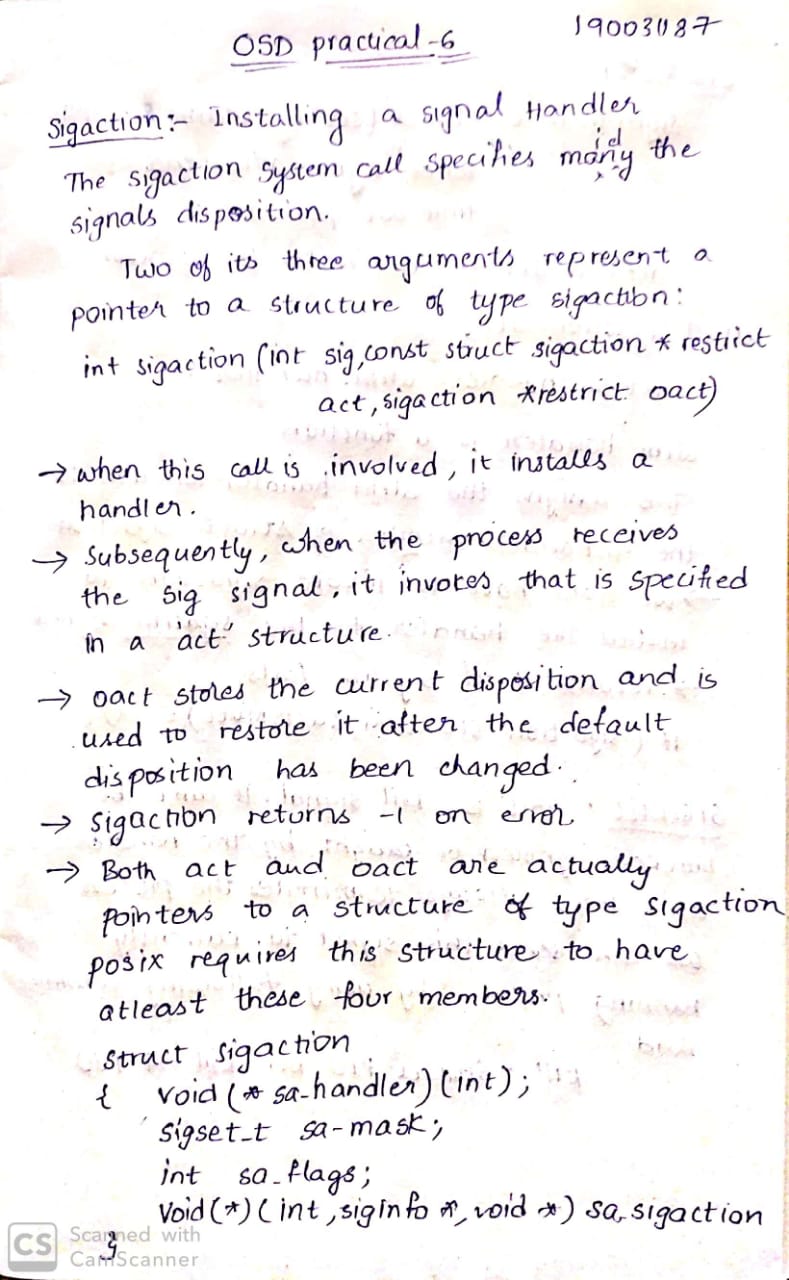
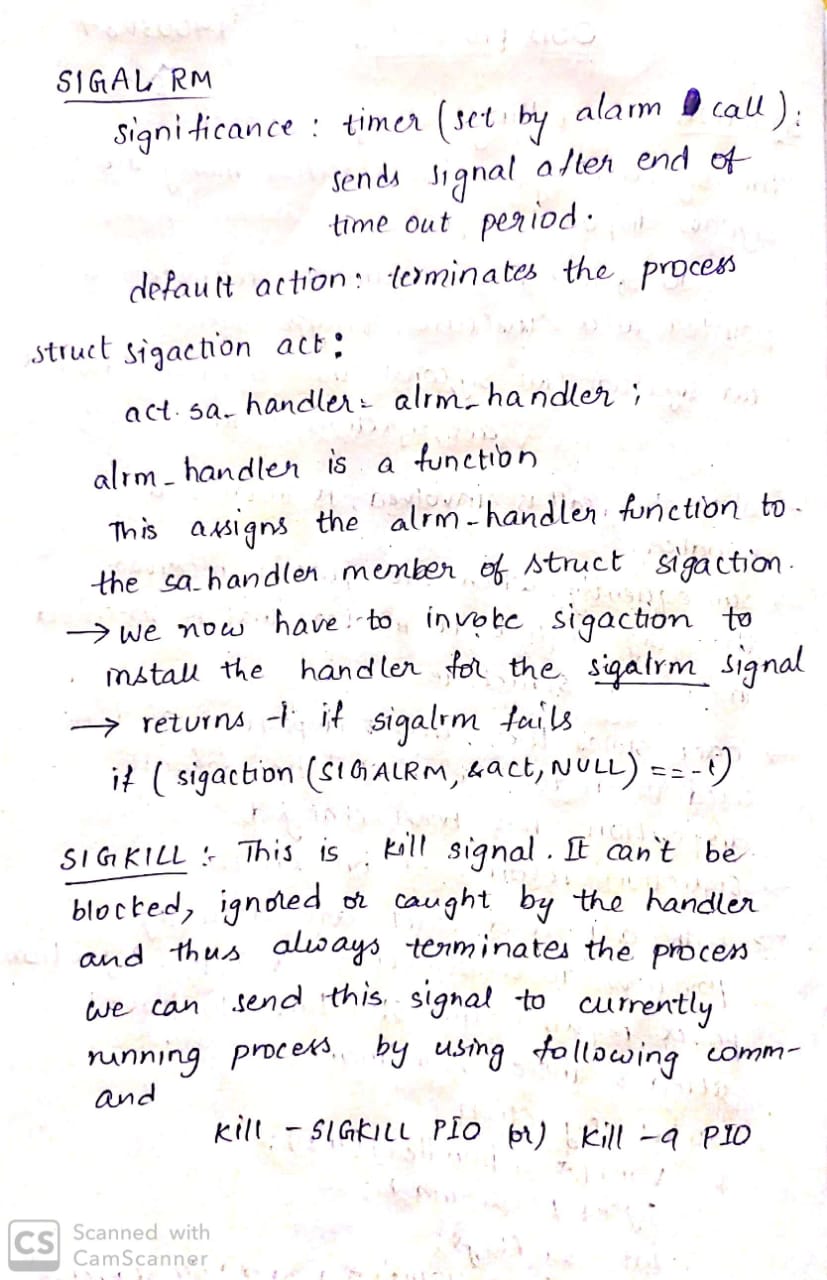
**Operating System and Design (19CS2106S)**

**Lab- 6**

**Pre-Lab**

POSIX signals. sigaction: This call specifies the signal handler. Two of the arguments to this call specify a structure that is also named sigaction. alarm: The alarm call is used in the next example to set a timer that generates the SIGALRM signal after the timeout period. The library function sleep uses alarm.pause: This is somewhat like the shell’s read statement. It holds up program execution until a signal is received. kill: You can send a signal to a process using this system call. A library function, raise, uses kill to send any signal to the current process.





**In-Lab**

1. signal.c -- Waits for 5 seconds for user input and then
2. Generates SIGALRM that has a handler specifiedkillproce ss.c -- Uses fork and exec to run a user-defined programand kills it if it doesn’t complete in 5 seconds.

1) signal.c :

Code:

[#include](tg://search_hashtag/?hashtag=include) <stdio.h>

[#include](tg://search_hashtag/?hashtag=include) <sys/stat.h> /\* For struct stat \*/

[#include](tg://search_hashtag/?hashtag=include) <stdarg.h>

[#include](tg://search_hashtag/?hashtag=include) <stdlib.h>

[#include](tg://search_hashtag/?hashtag=include) <time.h>

[#include](tg://search_hashtag/?hashtag=include) <unistd.h>

[#include](tg://search_hashtag/?hashtag=include) <signal.h> [#define](tg://search_hashtag/?hashtag=define) BUFSIZE 100 void alrm\_handler(int signo); /\* Prototype declaration \*/ char buf[BUFSIZE] = "foo\0"; /\* Global variable \*/

void quit(char \*message, int exit\_status)

{

printf(" %s",message);

exit(exit\_status);

}

int main (void) { int n; struct sigaction act; [act.sa](http://act.sa/)\_handler = alrm\_handler; /\* Specify handler \*/ if (sigaction(SIGALRM, &act, NULL) == -1) /\* Install handler \*/ quit("sigalrm", 1); fprintf(stderr, "Enter filename: "); alarm(5); /\* Set alarm clock; will deliver \*/

n = read(STDIN\_FILENO, buf, BUFSIZE); /\* SIGALRM in 5 seconds \*/ if (n > 1) /\* Will come here if user inputs \*/

fprintf(stderr, "Filename: %s\n", buf); /\* string within 5 seconds \*/ exit(0);

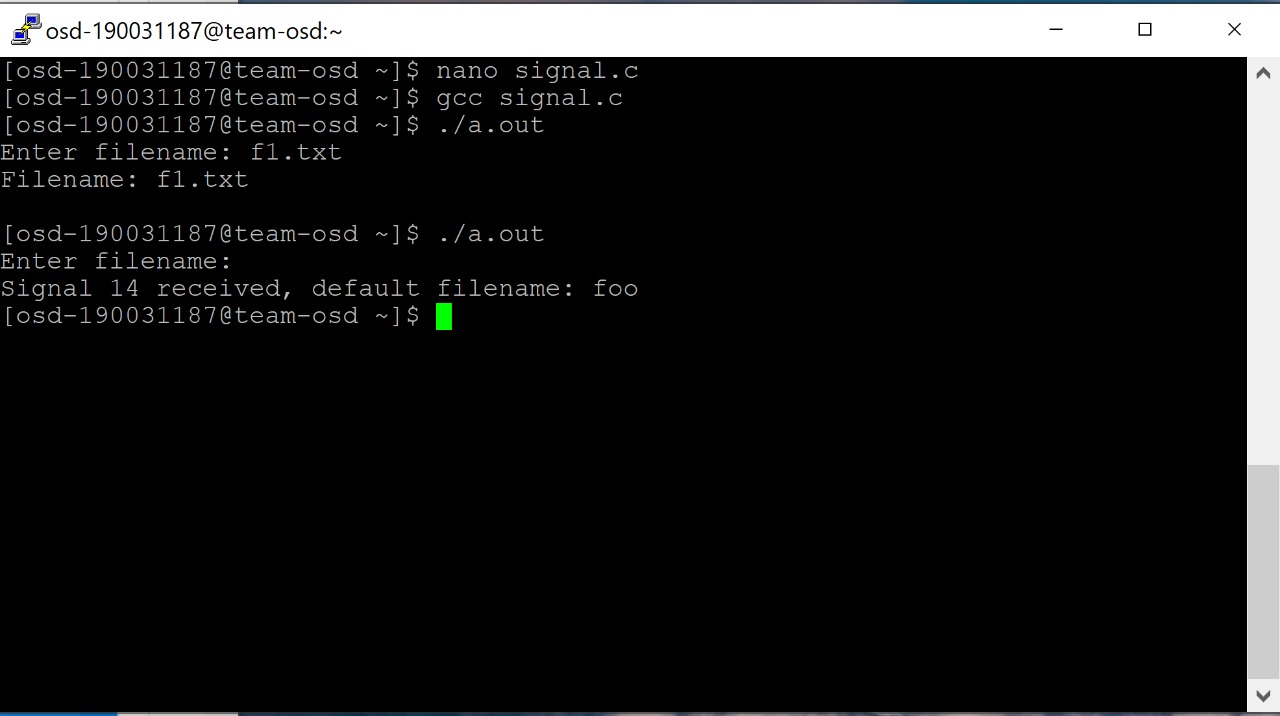
}

void alrm\_handler(int signo)

{

fprintf(stderr, "\nSignal %d received, default filename: %s\n", signo, buf); exit(1); }

Output:-



2) killprocess.c:

Code:-

[#include](tg://search_hashtag/?hashtag=include) <stdio.h>

[#include](tg://search_hashtag/?hashtag=include) <sys/types.h>

[#include](tg://search_hashtag/?hashtag=include) <sys/wait.h>

[#include](tg://search_hashtag/?hashtag=include) <signal.h> [#include](tg://search_hashtag/?hashtag=include) <stdlib.h>

pid\_t pid;

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

int i, status;

void death\_handler(int signo); /\* A common signal handler this time \*/

struct sigaction act; [act.sa](http://act.sa/)\_handler = death\_handler; sigaction(SIGCHLD, &act, NULL); /\* Disposition for these two signals \*/ sigaction(SIGALRM, &act, NULL); /\* set to enter a single handler \*/

switch (pid = fork()) { case -1: fprintf(stderr, "Fork error\n"); case 0: execvp(argv[1], &argv[1]); /\* Execute command \*/ perror("exec"); break; default: alarm(5); /\* Will send SIGALRM after 5 seconds \*/ pause(); /\* Will return when SIGCHLD signal is received \*/

fprintf(stderr, "Parent dies\n");

} exit(0);

}

void death\_handler(int signo) { /\* This common handler picks up the \*/

int status; /\* exit status for normal termination \*/ /\* but sends the SIGTERM signal if \*/ switch (signo) { /\* command doesn’t complete in 5 seconds \*/ case SIGCHLD: waitpid(-1, &status, 0); /\* Same as wait(&status); \*/

fprintf(stderr, "Child dies; exit status: %d\n",

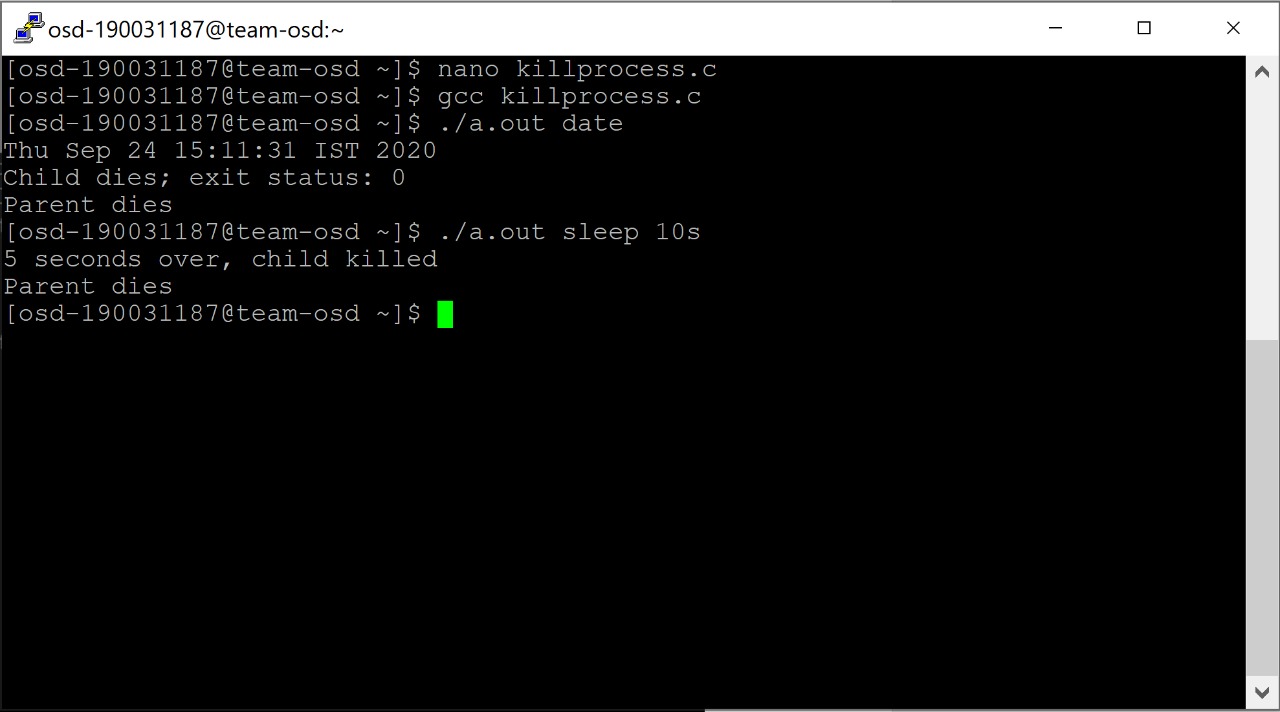
WEXITSTATUS(status)); break; case SIGALRM: if (kill(pid, SIGTERM) == 0)

fprintf(stderr, "5 seconds over, child killed\n");

}

}

Output:-



**Post-Lab**

1. mynice.c: A child process inherits its priority value from its parent, and change it by using nice ()
2. program to demonstrate time and times System Call.

**mynice.c:**

**Code:**

[#include](tg://search_hashtag/?hashtag=include) <stdio.h> main ()

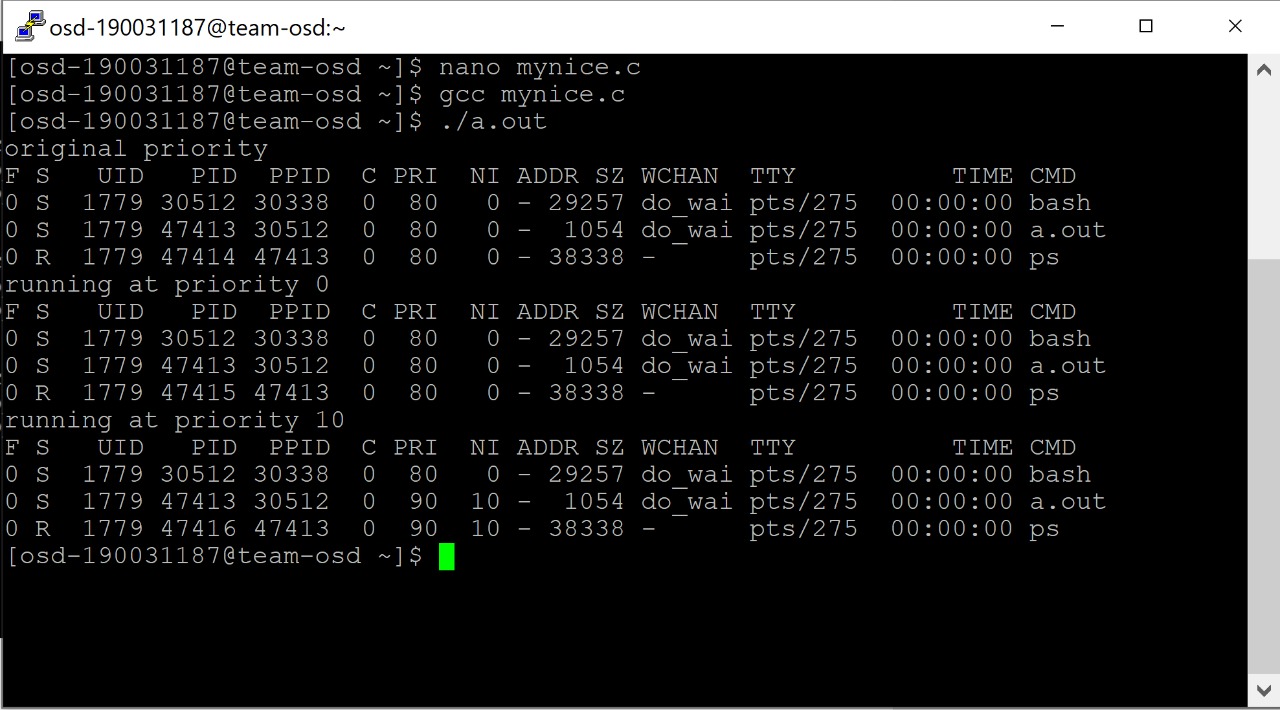
{

printf ("original priority\n"); system ("ps -l"); /\* Execute a ps \*/ nice (0); /\* Add 0 to my priority \*/ printf ("running at priority 0\n"); system ("ps -l"); /\* Execute another ps \*/ nice (10); /\* Add 10 to my priority \*/ printf ("running at priority 10\n");

system ("ps -l"); /\* Execute the last ps \*/

}

Output:



2) **time.c:**

**Code:**

[#include](tg://search_hashtag/?hashtag=include) <stdio.h> /\* printf \*/

[#include](tg://search_hashtag/?hashtag=include) <time.h> /\* time\_t, struct tm, difftime, time, mktime \*/

int main ()

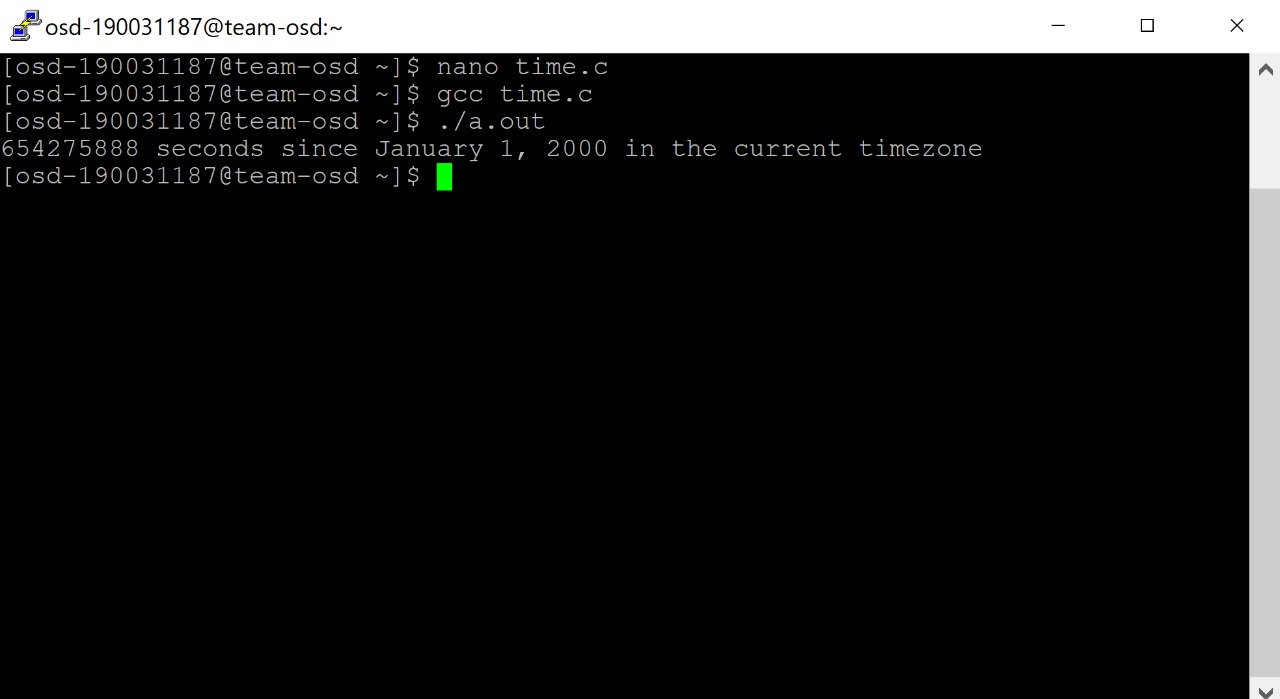
{

time\_t timer; struct tm y2k = {0}; double seconds; [y2k.tm](http://y2k.tm/)\_hour = 0; [y2k.tm\_](http://y2k.tm/)min = 0; [y2k.tm\_](http://y2k.tm/)sec = 0; [y2k.tm](http://y2k.tm/)\_year = 100; [y2k.tm](http://y2k.tm/)\_mon = 0; [y2k.tm](http://y2k.tm/)\_mday = 1; time(&timer); /\* get current time; same as: timer = time(NULL) \*/ seconds = difftime(timer,mktime(&y2k));

printf ("%.f seconds since January 1, 2000 in the current timezone", seconds); return 0;

}

Output:



**times.c:**

**Code:**

[#include](tg://search_hashtag/?hashtag=include) <stdio.h>

[#include](tg://search_hashtag/?hashtag=include) <unistd.h>

[#include](tg://search_hashtag/?hashtag=include) <sys/times.h> [#include](tg://search_hashtag/?hashtag=include) <time.h>

int main() { struct tms times\_start, times\_end; clock\_t times\_start\_retval, times\_end\_retval; clock\_t clock\_start, clock\_end;

int i;

/\* clock called first and last, so estimates using "clock" should be slightly longer than estimates using "times" \*/ if((clock\_start = clock()) == -1) { perror("starting clock"); return -1;

}

if((times\_start\_retval = times(&times\_start)) == -1) { perror("starting times");

return -1;

}

for(i = 100000000; i; i--); // do work

if((times\_end\_retval = times(&times\_end)) == -1) { perror("ending timer"); return -1;

}

printf("Times using the clock system call\n"); printf("clock start: %li\nclock end: %li\n", clock\_start, clock\_end); printf("elapsed: %li\n\n", clock\_end - clock\_start);

printf("Times using the times system call\n"); printf("System start: %li\nUser start: %li, start retval: %li\n", times\_start.tms\_stime, times\_start.tms\_utime, times\_start\_retval); printf("System end: %li\nUser end: %li, end retval: %li\n", times\_end.tms\_stime, times\_end.tms\_utime, times\_end\_retval); printf("elapsed: %li\n\n", times\_end\_retval - times\_start\_retval);

return 0;

}

Output:

