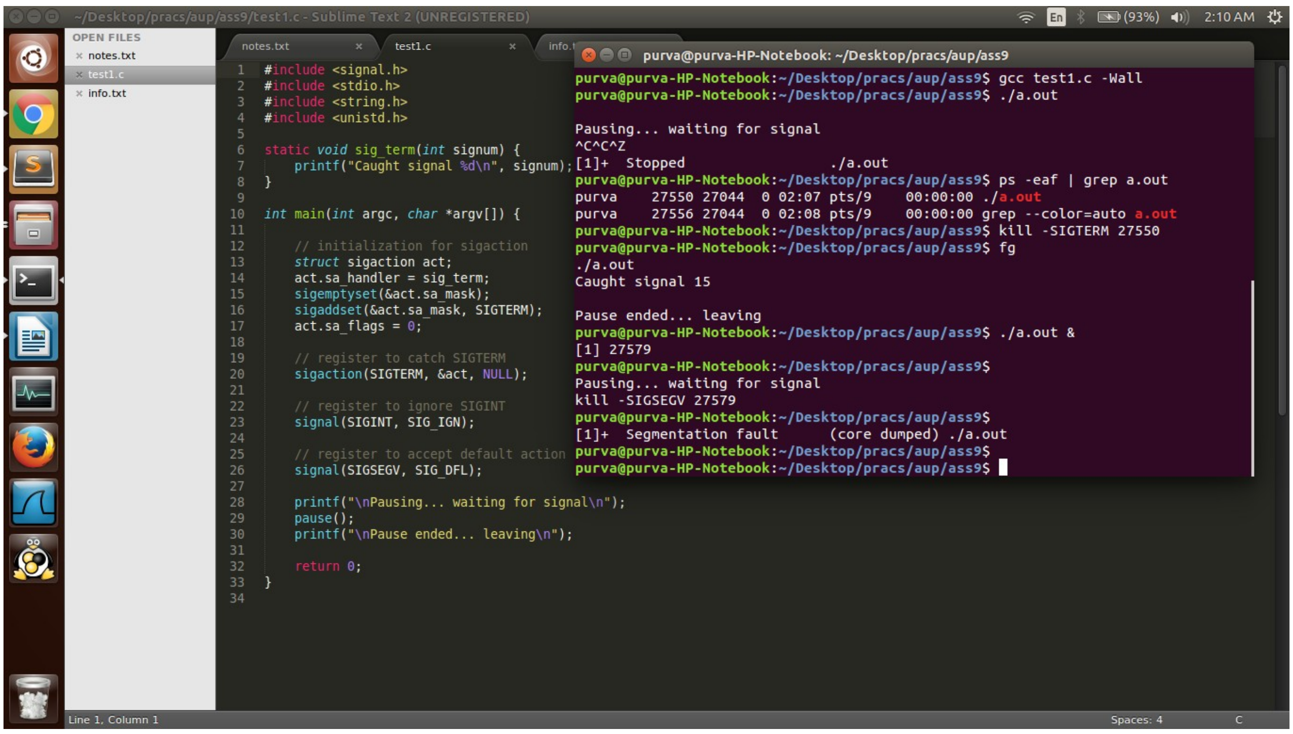
Advanced Unix Programming
Lab 9
Purva Tendulkar : 111403049

Q1. Catch the SIGTERM signal, ignore SIGINT and accept the default action for SIGSEGV. Later let the program be suspended until it is interrupted by a signal. Implement using signal and sigaction.

Code :
```
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

static void sig_term(int signum) {
   printf("Caught signal %d\n", signum);
}

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

   // initialization for sigaction
   struct sigaction act;
   act.sa_handler = sig_term;
   sigemptyset(&act.sa_mask);
   sigaddset(&act.sa_mask, SIGTERM);
   act.sa_flags = 0;

   // register to catch SIGTERM
   sigaction(SIGTERM, &act, NULL);

   // register to ignore SIGINT
   signal(SIGINT, SIG_IGN);

   // register to accept default action for SIGSEGV
   signal(SIGSEGV, SIG_DFL);

   printf("\nPausing... waiting for signal\n");
   pause();
   printf("\nPause ended... leaving\n");

   return 0;
}
```

Input & Output Screenshots :


~/Desktop/pracs/aup/ass9/test1.c - Sublime Text 2 (UNREGISTERED)
OPEN FILES
notes.txt
test1.c
info.txt
Line 1, Column 1
Spaces: 4
C
purva@purva-HP-Notebook: ~/Desktop/pracs/aup/ass9
2:10 AM

```c
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

static void sig_term(int signum) {
    printf("Caught signal %d\n", signum);
}

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

    // initialization for sigaction
    struct sigaction act;
    act.sa_handler = sig_term;
    sigemptyset(&act.sa_mask);
    sigaddset(&act.sa_mask, SIGTERM);
    act.sa_flags = 0;

    // register to catch SIGTERM
    sigaction(SIGTERM, &act, NULL);

    // register to ignore SIGINT
    signal(SIGINT, SIG_IGN);

    // register to accept default action
    signal(SIGSEGV, SIG_DFL);

    printf("\nPausing... waiting for signal\n");
    pause();
    printf("\nPause ended... leaving\n");

    return 0;
}
```

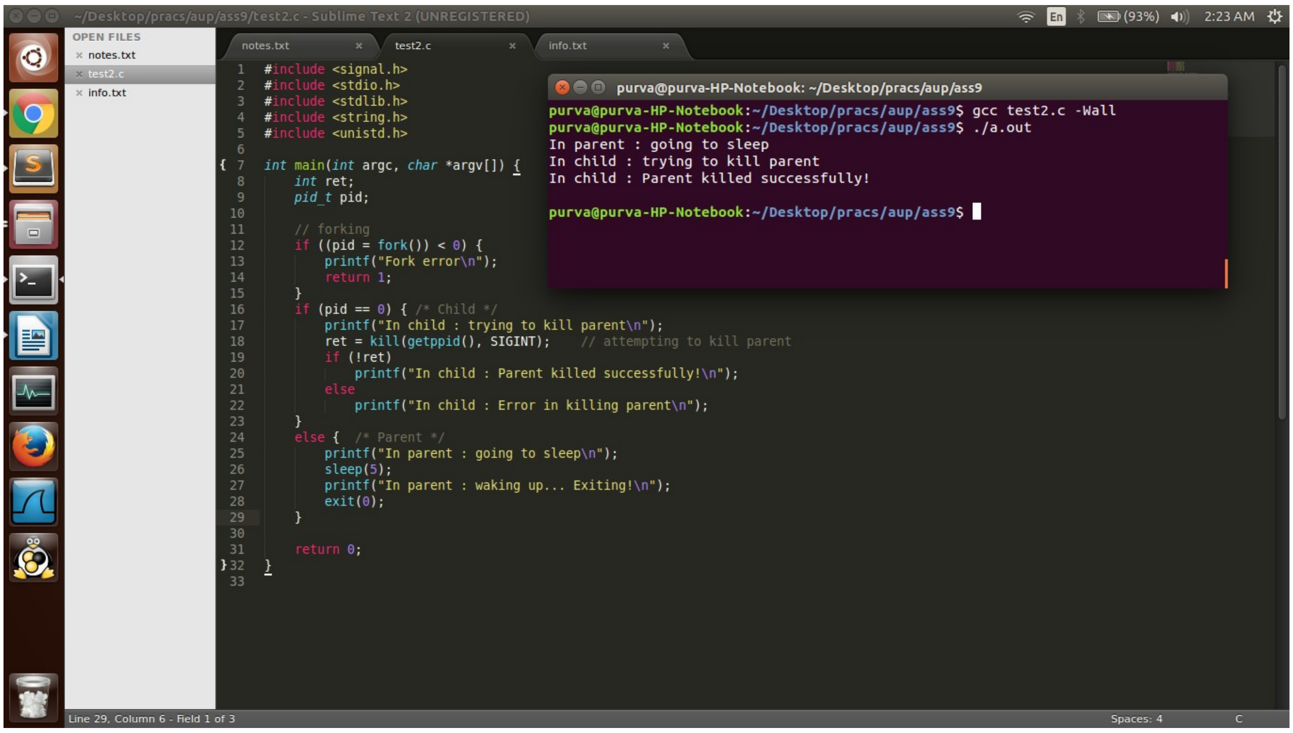
```
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ gcc test1.c -Wall
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ ./a.out

Pausing... waiting for signal
^C^C^Z
[1]+  Stopped                 ./a.out
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ ps -eaf | grep a.out
purva    27550 27044  0 02:07 pts/9    00:00:00 ./a.out
purva    27556 27044  0 02:08 pts/9    00:00:00 grep --color=auto a.out
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ kill -SIGTERM 27550
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ fg
./a.out
Caught signal 15

Pause ended... leaving
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ ./a.out &
[1] 27579
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$
Pausing... waiting for signal
kill -SIGSEGV 27579
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$
[1]+  Segmentation fault      (core dumped) ./a.out
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$
```

Q2. Create a child process. Let the parent sleeps of 5 seconds and exits. Can the child send SIGINT to its parent if exists and kill it? Verify with a sample program.

Code :

```c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    int ret;
    pid_t pid;

    // forking
    if ((pid = fork()) < 0) {
        printf("Fork error\n");
        return 1;
    }
    if (pid == 0) { /* Child */
        printf("In child : trying to kill parent\n");
        ret = kill(getppid(), SIGINT);     // attempting to kill parent
        if (!ret)
            printf("In child : Parent killed successfully!\n");
        else
            printf("In child : Error in killing parent\n");
    }
    else {  /* Parent */
        printf("In parent : going to sleep\n");
        sleep(5);
        printf("In parent : waking up... Exiting!\n");
        exit(0);
    }

    return 0;
}
```

Input & Output Screenshots :


~/Desktop/pracs/aup/ass9/test2.c - Sublime Text 2 (UNREGISTERED)
OPEN FILES
notes.txt
test2.c
info.txt
purva@purva-HP-Notebook: ~/Desktop/pracs/aup/ass9
Line 29, Column 6 - Field 1 of 3
Spaces: 4
C

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    int ret;
    pid_t pid;

    // forking
    if ((pid = fork()) < 0) {
        printf("Fork error\n");
        return 1;
    }
    if (pid == 0) { /* Child */
        printf("In child : trying to kill parent\n");
        ret = kill(getppid(), SIGINT);    // attempting to kill parent
        if (!ret)
            printf("In child : Parent killed successfully!\n");
        else
            printf("In child : Error in killing parent\n");
    }
    else {  /* Parent */
        printf("In parent : going to sleep\n");
        sleep(5);
        printf("In parent : waking up... Exiting!\n");
        exit(0);
    }

    return 0;
}
```

```
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ gcc test2.c -Wall
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ ./a.out
In parent : going to sleep
In child : trying to kill parent
In child : Parent killed successfully!

purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$
```

Explanation :
Yes. The child can kill its parent (if it exists) by sending SIGINT signal to it. Since the parent is currently sleeping, it will be awakened by the SIGINT signal sent to it and terminate.

Q3. Implement sleep using signal function which takes care of the following:

1. **If the caller has already an alarm set, that alarm is not erased by the call to alarm inside sleep implementation.**
2. **If sleep modifies the current disposition of SIGALRM, restore it**
3. **Avoid race condition between first call to alarm and pause inside sleep implementation using setjmp.**

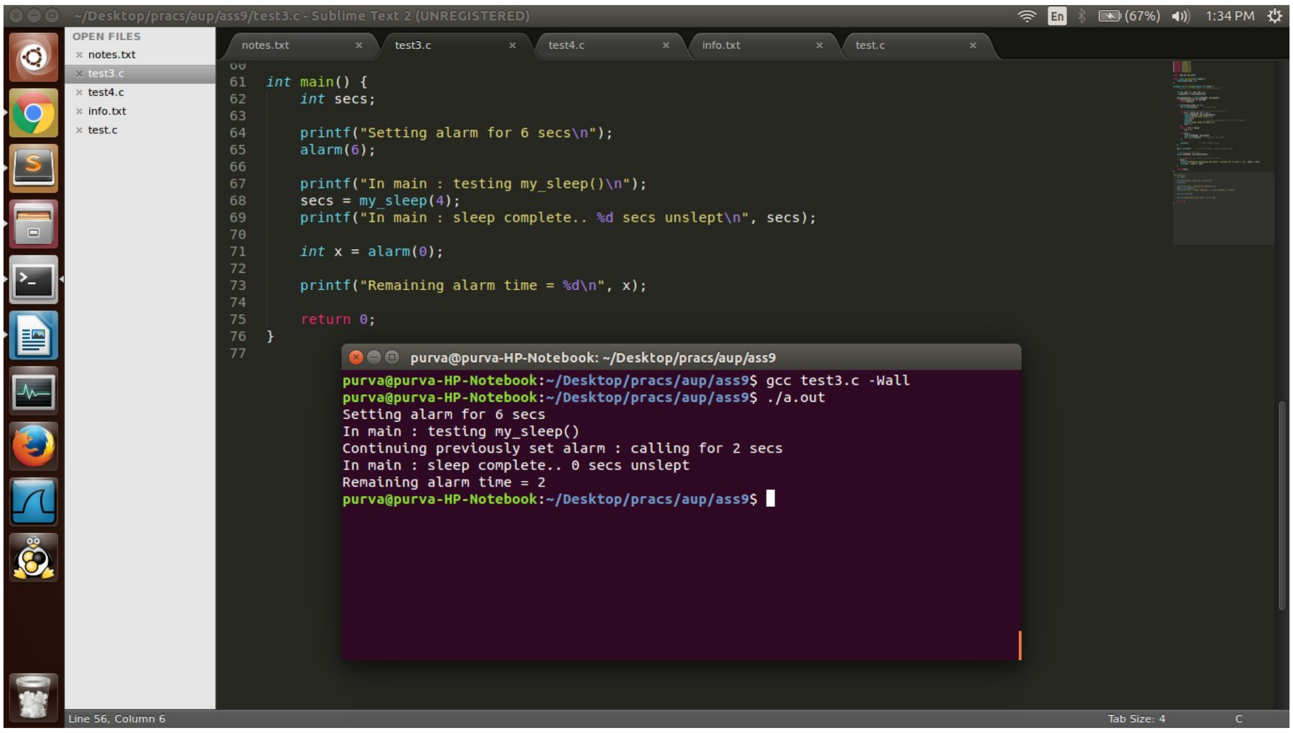
Test the implementation of sleep by invoking it in various situations.

Code :

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <setjmp.h>

static jmp_buf env_alrm;
static void sig_alarm(int signo) {
        longjmp(env_alrm, 1);
}

unsigned int my_sleep(unsigned int nsecs) {
        /* Returns 0 or the number of unslept seconds */

        int t1, set = 0, ret, old = 0;
        __sighandler_t old_disposition;

        old_disposition = signal(SIGALRM, sig_alarm);
        if (old_disposition == SIG_ERR)
                return(nsecs);

        if (setjmp(env_alrm) == 0) {
                t1 = alarm(nsecs);                // start timer

                /* If previously set alarm is to go off earlier */
                if ((t1 < nsecs) && (t1 > 0)) {
                        signal(SIGALRM, old_disposition);
                        printf("Doing older alarm\n");
                        alarm(t1);
                        pause();                  // wait for previously set alarm to complete
                        printf("Older alarm is done\n");
                        old = 1;
                }
                else if (t1 > nsecs)
                        set = 1;

                if (old) {
                        signal(SIGALRM, sig_alarm);
                        t1 = alarm(nsecs – t1);   // resetting our alarm
                }

                pause();                          // next caught signal
        }

        ret = alarm(0);          // turn off timer - return unslept time

        /* Resetting disposition */
        signal(SIGALRM, old_disposition);

        /* If previously set alarm is to go off later */
        if (set) {
                printf("Continuing previously set alarm : calling for %d secs\n", t1 - nsecs + ret);
                alarm(t1 - nsecs + ret);
        }

        return(ret);
}

int main() {
        int secs;

        printf("In main : testing my_sleep()\n");
        secs = my_sleep(4);
        printf("In main : sleep complete.. %d secs unslept\n", secs);

        return 0;
}
```

Input & Output Screenshots :

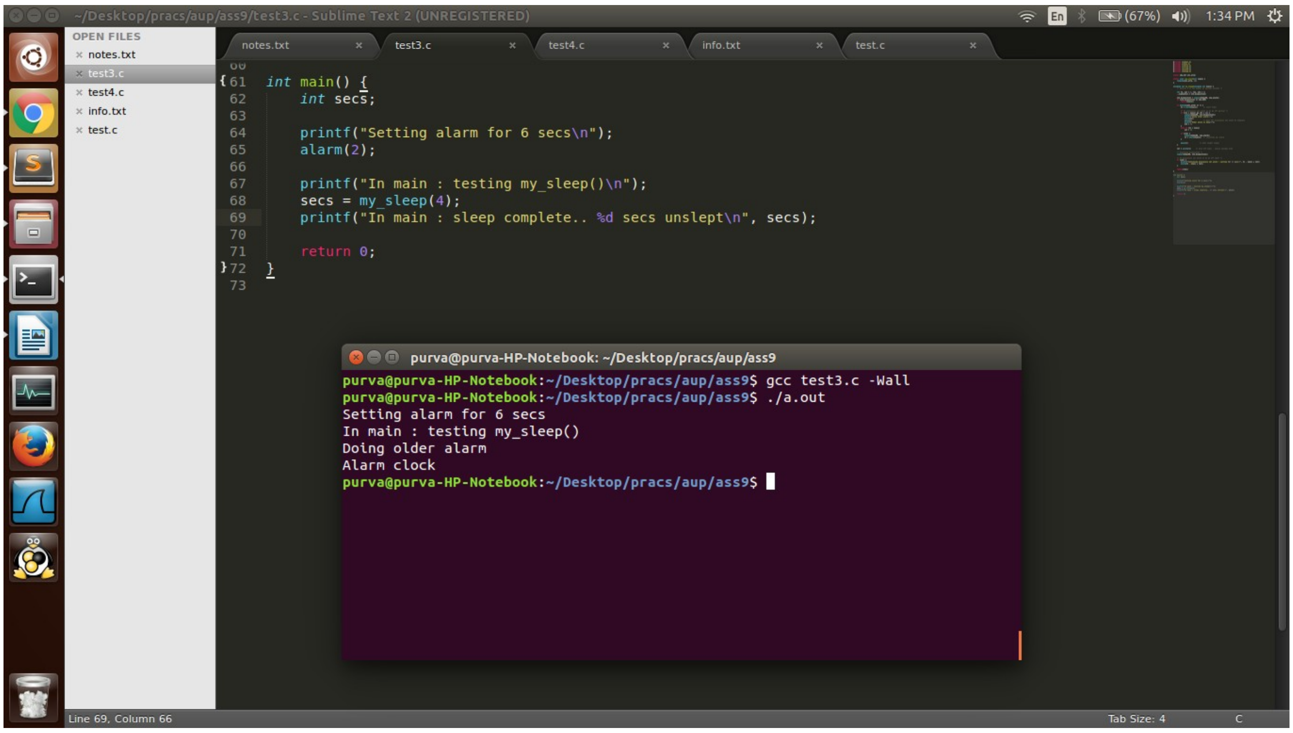
(A) Case 1 – Alarm for longer duration called before calling my_sleep



```
int main() {
    int secs;

    printf("Setting alarm for 6 secs\n");
    alarm(6);

    printf("In main : testing my_sleep()\n");
    secs = my_sleep(4);
    printf("In main : sleep complete.. %d secs unslept\n", secs);

    int x = alarm(0);

    printf("Remaining alarm time = %d\n", x);

    return 0;
}
```

```
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ gcc test3.c -Wall
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ ./a.out
Setting alarm for 6 secs
In main : testing my_sleep()
Continuing previously set alarm : calling for 2 secs
In main : sleep complete.. 0 secs unslept
Remaining alarm time = 2
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$
```

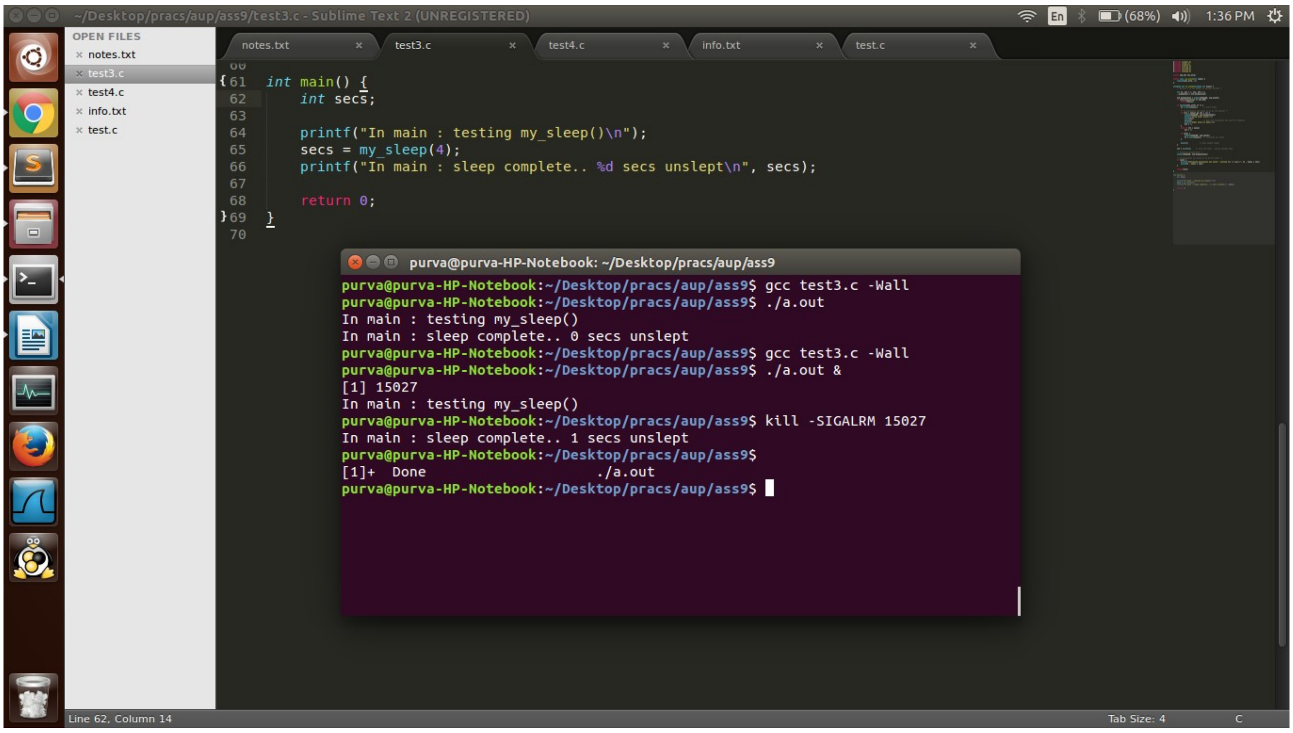
(B) Case 2 – Alarm for lesser duration called before calling my_sleep



```
int main() {
    int secs;

    printf("Setting alarm for 6 secs\n");
    alarm(2);

    printf("In main : testing my_sleep()\n");
    secs = my_sleep(4);
    printf("In main : sleep complete.. %d secs unslept\n", secs);

    return 0;
}
```

```
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ gcc test3.c -Wall
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ ./a.out
Setting alarm for 6 secs
In main : testing my_sleep()
Doing older alarm
Alarm clock
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$
```

(C) Case 3 – Only my_sleep called : with and without SIGALRM



```
int main() {
    int secs;

    printf("In main : testing my_sleep()\n");
    secs = my_sleep(4);
    printf("In main : sleep complete.. %d secs unslept\n", secs);

    return 0;
}
```

```
purva@purva-HP-Notebook: ~/Desktop/pracs/aup/ass9
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ gcc test3.c -Wall
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ ./a.out
In main : testing my_sleep()
In main : sleep complete.. 0 secs unslept
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ gcc test3.c -Wall
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ ./a.out &
[1] 15027
In main : testing my_sleep()
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ kill -SIGALRM 15027
In main : sleep complete.. 1 secs unslept
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$
[1]+  Done                    ./a.out
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$
```

Explanation :

1. **Case 1 :** my_sleep examines the previous alarm duration, and sets an alarm for the appropriate number of seconds in the future. This information about pending alarms is available in main() after calling alarm(0).
2. **Case 2 :** my_sleep examines the previous alarm duration. Since the value is lower than the duration for which sleep is called, it first sets the resets the signal disposition to its original value, calls alarm for the previous alarm duration, then again changes the disposition and calls alarm with the appropriate time duration. In this example, the original default disposition was to terminate the program.
3. **Case 3** : my_sleep works normally. In the first case, no signal is issued when the process is sleeping, In the next case, a signal (SIGALRM) is issued during sleep. Hence, the appropriate number of unslept seconds in returned.

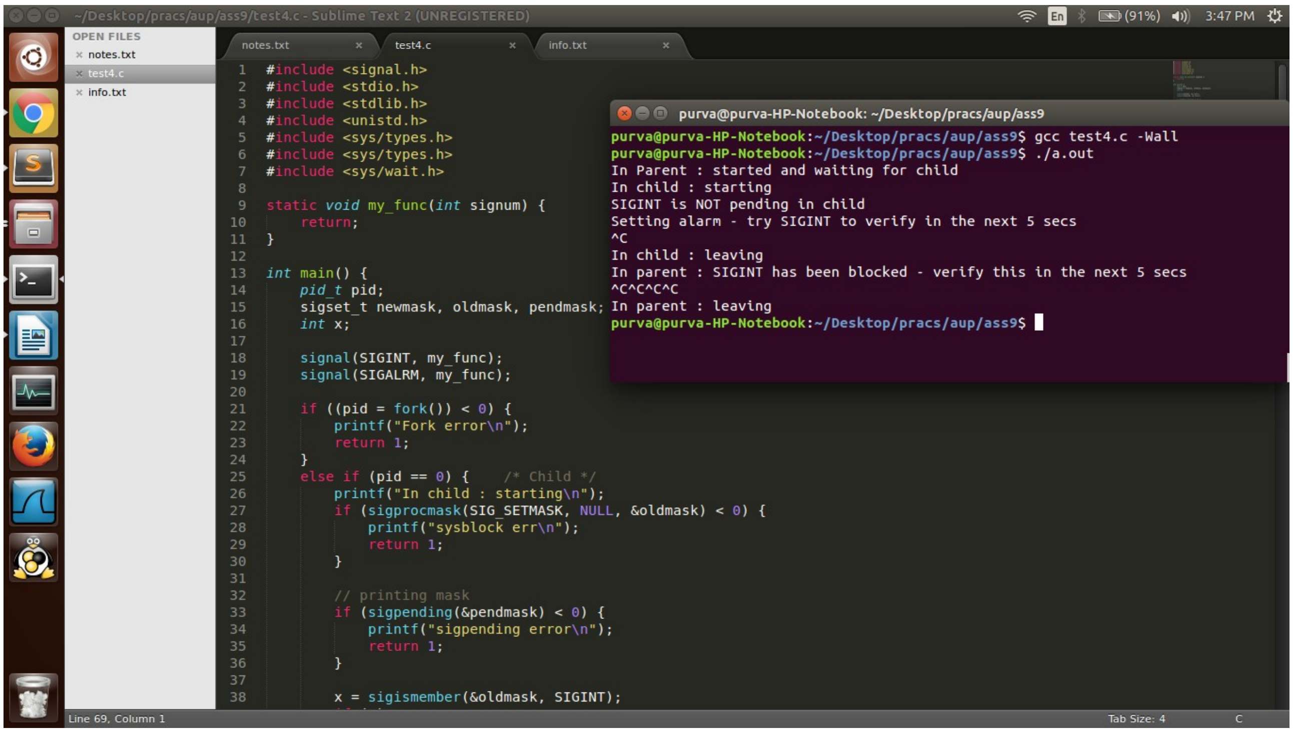
Q4. "Child inherit parent's signal mask when it is created, but pending signals for the parent process are not passed on". Write appropriate program and test with suitable inputs to verify this.

Code :

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/types.h>
#include <sys/wait.h>

static void my_func(int signum) {
        return;
}
```

```
int main() {
        pid_t pid;
        sigset_t newmask, oldmask, pendmask;
        int x;

        signal(SIGINT, my_func);
        signal(SIGALRM, my_func);

        if ((pid = fork()) < 0) {
                printf("Fork error\n");
                return 1;
        }
        else if (pid == 0) {     /* Child */
                printf("In child : starting\n");
                if (sigprocmask(SIG_SETMASK, NULL, &oldmask) < 0) {
                        printf("sysblock err\n");
                        return 1;
                }

                // printing mask
                if (sigpending(&pendmask) < 0) {
                        printf("sigpending error\n");
                        return 1;
                }

                x = sigismember(&oldmask, SIGINT);
                if (x)
                        printf("SIGINT is pending in child\n");
                else
                        printf("SIGINT is NOT pending in child\n");

                printf("Setting alarm - try SIGINT to verify in the next 5 secs\n");
                alarm(5);
                pause();
                printf("\nIn child : leaving\n");
        }
        else {  /* Parent */
                // change sigprocmask
                printf("In Parent : started and waiting for child\n");
                sigemptyset(&newmask);
                sigaddset(&newmask, SIGINT);
                if (sigprocmask(SIG_SETMASK, &newmask, NULL) < 0) {
                        printf("sysblock err_sys\n");
                        return 1;
                }

                wait(&x);

                printf("In parent : SIGINT has been blocked - verify this in the next 5 secs\n");
                alarm(5);
                pause();
                printf("\nIn parent : leaving\n");
        }

        return 0;
}
```

Input & Output Screenshots :


~/Desktop/pracs/aup/ass9/test4.c - Sublime Text 2 (UNREGISTERED)
OPEN FILES
notes.txt
test4.c
info.txt
purva@purva-HP-Notebook: ~/Desktop/pracs/aup/ass9
Line 69, Column 1
Tab Size: 4
C

```c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/types.h>
#include <sys/wait.h>

static void my_func(int signum) {
    return;
}

int main() {
    pid_t pid;
    sigset_t newmask, oldmask, pendmask;
    int x;

    signal(SIGINT, my_func);
    signal(SIGALRM, my_func);

    if ((pid = fork()) < 0) {
        printf("Fork error\n");
        return 1;
    }
    else if (pid == 0) {    /* Child */
        printf("In child : starting\n");
        if (sigprocmask(SIG_SETMASK, NULL, &oldmask) < 0) {
            printf("sysblock err\n");
            return 1;
        }

        // printing mask
        if (sigpending(&pendmask) < 0) {
            printf("sigpending error\n");
            return 1;
        }

        x = sigismember(&oldmask, SIGINT);
```

```
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ gcc test4.c -Wall
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$ ./a.out
In Parent : started and waiting for child
In child : starting
SIGINT is NOT pending in child
Setting alarm - try SIGINT to verify in the next 5 secs
^C
In child : leaving
In parent : SIGINT has been blocked - verify this in the next 5 secs
^C^C^C^C
In parent : leaving
purva@purva-HP-Notebook:~/Desktop/pracs/aup/ass9$
```