

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 implemented using signal :

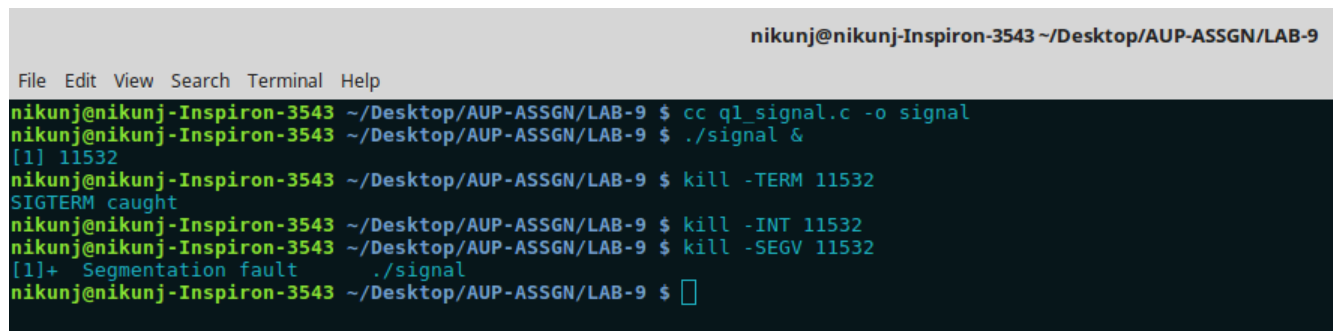
```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

static void sig_term(int signo);

int main(int argc, char *argv[]) {
    if(signal(SIGTERM, sig_term) == SIG_ERR) {
        printf("Couldn't catch SIGTERM\n");
    }
    if(signal(SIGINT, SIG_IGN) == SIG_ERR) {
        printf("Couldn't ignore SIGINT\n");
    }
    if(signal(SIGSEGV, SIG_DFL) == SIG_ERR) {
        printf("Couldn't perform default action for SIGSEGV\n");
    }
    for( ; ; ) {
        // Suspend the process until a signal comes
        pause();
    }
    return 0;
}

static void sig_term(int signo) {
    // reestablishing signal handler
    signal(SIGTERM, sig_term);
    if(signo == SIGTERM) {
        printf("SIGTERM caught\n");
    }
    return;
}
```

Execution:



```
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9
File Edit View Search Terminal Help
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9 $ cc q1_signal.c -o signal
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9 $ ./signal &
[1] 11532
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9 $ kill -TERM 11532
SIGTERM caught
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9 $ kill -INT 11532
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9 $ kill -SEGV 11532
[1]+  Segmentation fault      ./signal
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9 $
```

Here ./signal is signal executable of implementation using signal function. 11532 is pid of the process. SIGTERM is caught. SIGINT is ignored. SIGSEGV does it's default action give a segmenation fault.

### Code implemented using sigaction

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
void sig_term(int signo);

int main(int argc, char *argv[]) {
    struct sigaction a1, a2, a3;
    a1.sa_handler = &sig_term;
    a2.sa_handler = SIG_IGN;
    a3.sa_handler = SIG_DFL;
    // block every signal during the handler
    sigfillset(&a1.sa_mask);
    sigfillset(&a2.sa_mask);
    sigfillset(&a3.sa_mask);
    if(sigaction(SIGTERM, &a1, NULL) < 0) {
        printf("Cannot handle SIGTERM\n");
    }
    if(sigaction(SIGINT, &a2, NULL) < 0) {
        printf("Cannot handle SIGINT");
    }
    if(sigaction(SIGSEGV, &a3, NULL) < 0) {
        printf("Cannot handle SIGSEGV\n");
    }
    for(;;) {
        pause();
    }
    return 0;
}

void sig_term(int signal) {
    if(signal == SIGTERM) {
        printf("SIGTERM caught\n");
    }
    return;
}
```

Execution :



```
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9
File Edit View Search Terminal Help
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9 $ ./sigaction &
[1] 12004
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9 $ kill -TERM 12004
SIGTERM caught
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9 $ kill -INT 12004
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9 $ kill -SEGV 12004
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9 $ kill -SEGV 12004
bash: kill: (12004) - No such process
[1]+  Segmentation fault ./sigaction
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-9 $
```

sigaction() function helps in overcoming the limitations caused by signal() function in earlier systems. 12004 is the pid of the background process. Kill -TERM 12004 is caught by the handler. SIGINT is ignored and SIGSEGV leads to a segmentation fault.

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:

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

void sig_handler(int signo) {
    if((signal(SIGINT, sig_handler)) == SIG_ERR) {
        perror("Unable to set signal handler again");
    }
    printf("Caught SIGNINT\n");
    return;
}

int main() {
    pid_t child;

    if((child = fork()) < 0) {
        perror("Unable to create child process");
        return -1;
    }

    if(child == 0) {
        sleep(2);
        if((kill(getppid(), SIGINT)) == -1) {
            if(errno == ESRCH) {
                perror("Parent already died :-(");
                return -1;
            }
            else {
                perror("Unable to send signal to parent");
                return -1;
            }
        }
        return 0;
    }
    else {
        if((signal(SIGINT, sig_handler)) == SIG_ERR) {
            perror("Unable to set signal handler");
        }
        sleep(5);
        return 0;
    }

    return 0;
}
```

Execution:

```
sagar5mati@sagar5mati-HP-15-Notebook-PC: ~/btechcomp/aup/git/LAB-9
sagar5mati@sagar5mati-HP-15-Notebook-PC:~/btechcomp/aup/git/LAB-9$ ./2
Caught SIGINT
sagar5mati@sagar5mati-HP-15-Notebook-PC:~/btechcomp/aup/git/LAB-9$
```

Explanation:

The child can indeed send SIGINT to the parent. This is verified as the parent catches the signal which in default case would have led to its termination. We wait inside the child so that the parent can install the signal handler. If we did this beforehand then even the child would have the same signal handler.

Q3)

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

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

Code:

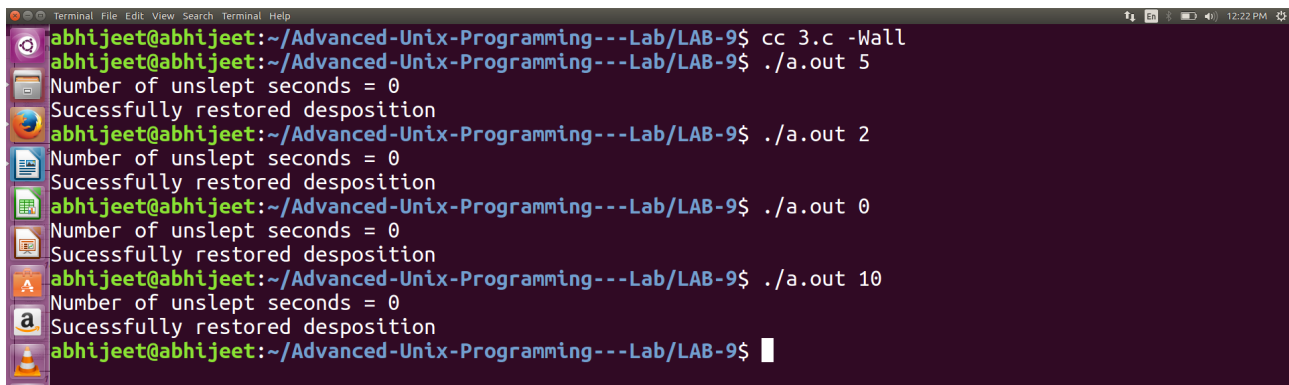
```
#include <setjmp.h>
#include <signal.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <ctype.h>
#include <limits.h>
jmp_buf env;
int charToInt(char *str) {
    int result = 0, count = 0;
    while(str[count]) {
        if(isdigit(str[count])) {
            result = result * 10 + (str[count] - '0');
        }
        else {
            return INT_MIN;
        }
        count++;
    }
    return result;
}
static void sig_alm(int signo) {
    longjmp(env, 1);
}
```

```

}
unsigned int sleep_aup(unsigned int seconds) {
    unsigned int diff;
    void (*handler)(int);
    handler = signal(SIGALRM, sig_alm);
    if(handler == SIG_ERR) {
        return seconds;
    }
    if(setjmp(env) == 0) {
        diff = alarm(3600);
        if(diff != 0) {
            alarm(diff);
        }
        else {
            alarm(seconds);
        }
        pause();
    }
    signal(SIGALRM, handler);
    return alarm(0);
}
void pass(int signo) {
    printf("You are in the user defined signal handler.\n");
    return;
}
int main(int argc, char *argv[]) {
    if(argc < 2) {
        printf("Not enough arguments\n");
        return 1;
    }
    unsigned int a, unslept;
    a = charToInt(argv[1]);
    if(a == INT_MIN) {
        printf("Enter a valid possitive number\n");
        return 1;
    }
    alarm(3);
    signal(SIGALRM, pass);
    unslept = sleep_aup(a);
    printf("Number of unslept seconds = %d\n", unslept);
    if(signal(SIGALRM, SIG_DFL) == pass) {
        printf("Sucessfully restored old desposition\n");
    }
    return 0;
}

```

Execution:

A terminal window with a dark purple background and a sidebar of application icons on the left. The terminal shows a user named 'abhiijeet' at a prompt in the directory '~/Advanced-Unix-Programming---Lab/LAB-9'. The user runs 'cc 3.c -Wall', then executes './a.out 5', './a.out 2', './a.out 0', and './a.out 10'. Each execution prints 'Number of unslept seconds = 0' and 'Sucessfully restored desposition' (note the spelling error in the original image).

```
abhiijeet@abhiijeet:~/Advanced-Unix-Programming---Lab/LAB-9$ cc 3.c -Wall
abhiijeet@abhiijeet:~/Advanced-Unix-Programming---Lab/LAB-9$ ./a.out 5
Number of unslept seconds = 0
Sucessfully restored desposition
abhiijeet@abhiijeet:~/Advanced-Unix-Programming---Lab/LAB-9$ ./a.out 2
Number of unslept seconds = 0
Sucessfully restored desposition
abhiijeet@abhiijeet:~/Advanced-Unix-Programming---Lab/LAB-9$ ./a.out 0
Number of unslept seconds = 0
Sucessfully restored desposition
abhiijeet@abhiijeet:~/Advanced-Unix-Programming---Lab/LAB-9$ ./a.out 10
Number of unslept seconds = 0
Sucessfully restored desposition
abhiijeet@abhiijeet:~/Advanced-Unix-Programming---Lab/LAB-9$
```

The above program takes care of all the points mentioned in the question. If an alarm has already been set which is yet to go off, then that alarm is not overridden. Also if there is a desposition already set for the signal, it is restored. The possibility of a race condition is taken care of with the help of setjmp and longjmp. The above program takes the number of seconds to sleep as a comand line argument.

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:

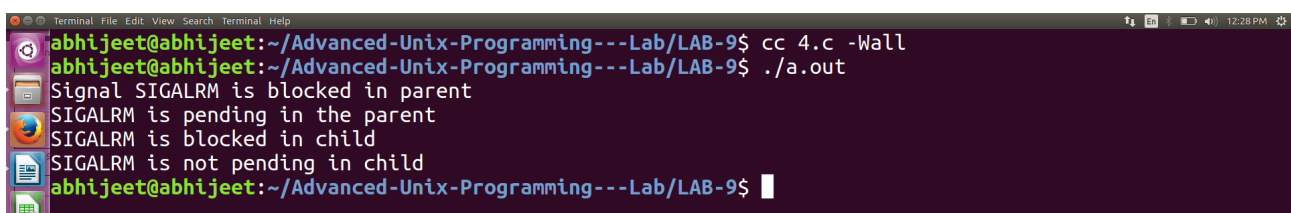
```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
int main(int argc, char *argv[]) {
    int pid;
    sigset_t newset, oldset, set;
    sigemptyset(&newset);
    if(sigaddset(&newset, SIGALRM) == -1) {
        perror("Sigaddset Error ");
        return 1;
    }
    if(sigprocmask(SIG_BLOCK, &newset, &oldset) == -1) {
        perror("Sigprocmask error ");
        return 1;
    }
    printf("Signal SIGALRM is blocked in parent\n");
    raise(SIGALRM);
    pid = fork();
    if(pid == 0) {
        if(sigprocmask(SIG_SETMASK, NULL, &oldset) == -1) {
            perror("Sigprocmask error in child ");
            return 1;
        }
        if(sigismember(&oldset, SIGALRM)) {
            printf("SIGALRM is blocked in child\n");
        }
    }
}
```

```

    }
    if(sigpending(&set) == -1) {
        perror("sigpending error in child ");
        return 1;
    }
    if(!sigismember(&set, SIGALRM)) {
        printf("SIGALRM is not pending in child\n");
    }
    else {
        printf("SIGALRM is pening in child\n");
    }
}
else if(pid > 0){
    if(sigpending(&set) == -1) {
        perror("Error in sigpending ");
        return 1;
    }
    if(!sigismember(&set, SIGALRM)) {
        printf("SIGALRM signal is not pending in the parent\n");
    }
    else {
        printf("SIGALRM is pending in the parent\n");
    }
}
else {
    perror("Fork failed ");
    return 1;
}
return 0;
}

```

Execution:



```

abhijeet@abhijeet:~/Advanced-Unix-Programming---Lab/LAB-9$ cc 4.c -Wall
abhijeet@abhijeet:~/Advanced-Unix-Programming---Lab/LAB-9$ ./a.out
Signal SIGALRM is blocked in parent
SIGALRM is pending in the parent
SIGALRM is blocked in child
SIGALRM is not pending in child
abhijeet@abhijeet:~/Advanced-Unix-Programming---Lab/LAB-9$

```

For the above example, I have blocked SIGALRM in the parent with the help of `sigprocmask` function. After this, the SIGALRM signal is raised. A fork is called to create a child process, which gets the oldmask with `sigprocmask`. The child process checks if the SIGALRM is blocked in the mask, and then checks if it is pending. During this time the parent also checks for a pending SIGALRM signal. As we can see in the output, the signal mask is passed on by the parent to it's child where as the pending signals have not.