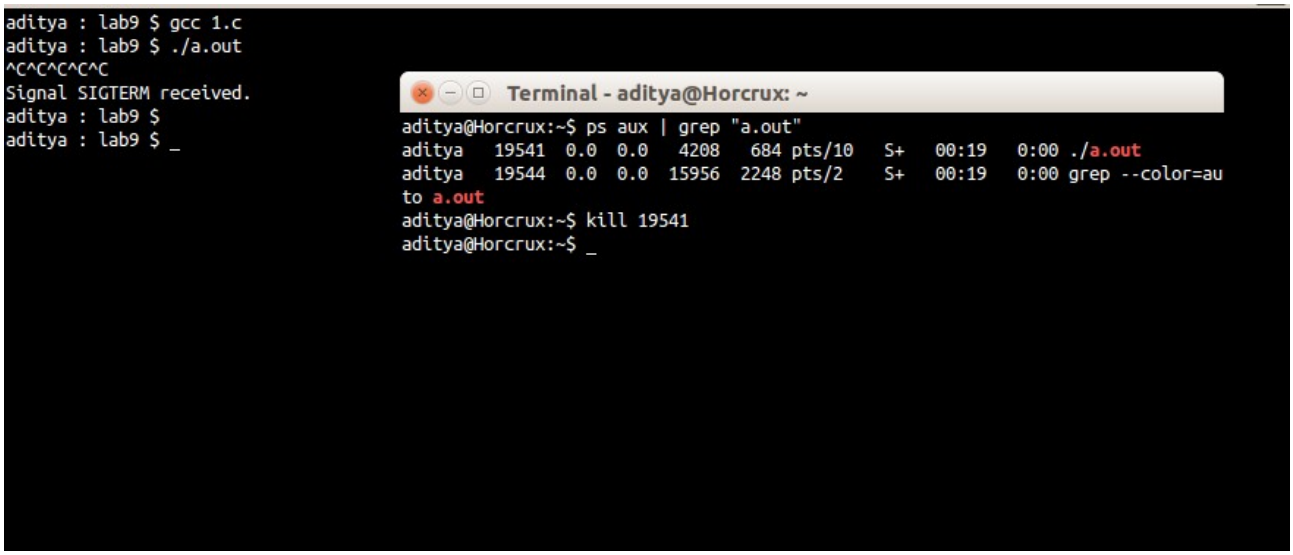
111403003 Akash Sarda
111403019 Jassim Rahman
111403023 Aditya Malu

AUP – LAB 8

Write a program to implement the following:

1. 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



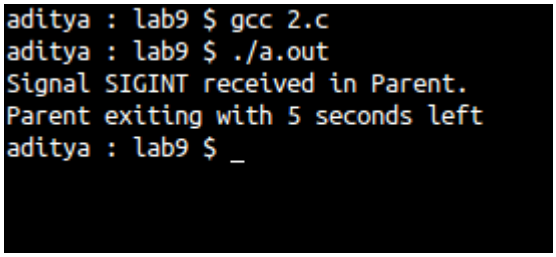
CODE

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

void err_sys(const char* x){
     perror(x);
     exit(1);
}

void signal_handler(int SIGNAL){
     if(SIGNAL == SIGTERM){
          printf("Signal SIGTERM received.\n");
     }
     else {
          printf("received signal %d\n", SIGNAL);
     }
}
```

```
int main(){
     if(signal(SIGTERM, signal_handler) == SIG_ERR){
          err_sys("Can't catch SIGTERM");
     }
     if(signal(SIGINT, SIG_IGN) == SIG_ERR){
          err_sys("Can't catch SIGINT");
     }
     if(signal(SIGSEGV, SIG_DFL) == SIG_ERR){
          err_sys("Can't catch SIGSEGV");
     }
     sleep(50);
     pause();
}
```

---

2. 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 <stdlib.h>
#include <signal.h>

void err_sys(const char* x){
     perror(x);
     exit(1);
}

void signal_handler(int SIGNAL){
     if(SIGNAL == SIGINT){
          printf("Signal SIGINT received in Parent.\n");
     }
```

```
	else {
		printf("received signal %d\n", SIGNAL);
	}
}

int main(){
	pid_t pid;
	if((pid = fork()) == -1){
		err_sys("Fork Error");
	}
	if(pid){
		if(signal(SIGINT, signal_handler) == SIG_ERR){
			err_sys("Can't catch SIGINT");
		}
		int time_remained = sleep(5);
		printf("Parent exiting with %d seconds left\n",
time_remained);
	}
	else {
		pid_t parent_pid = getppid();
		if(kill(parent_pid, SIGINT) == -1){
			err_sys("Error sending Signal");
			exit(1);
		}
	}
	exit(0);
}
```

---

3. 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.

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

```
aditya@Horcrux:~/SEM_7/AUP/lab9$ gcc 3.c
aditya@Horcrux:~/SEM_7/AUP/lab9$ ./a.out
remaining time is 3
Alarm clock
aditya@Horcrux:~/SEM_7/AUP/lab9$ _
```

CODE

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

static jmp_buf env;

int sig_alrm(int signo){
     longjmp(env, 1);
}

unsigned int mysleep(unsigned int secs){
     int time_left;
     time_left = alarm(0); /* checking if any previously set alarm
function has some unslept seconds left */
     printf("remaining time is %d\n", time_left);
     if(setjmp(env) == 0){
          alarm(secs - time_left); /* adding that time for user
set value */
          pause();
     }
}

int main(){
     alarm(3);
     mysleep(4);
}
```

---

4. “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

```
aditya : lab9 $ gcc 4.c
aditya : lab9 $ ./a.out
Send SIGQUIT signals
^\^\^\IN PARENT:
SIGNAL 0 present
SIGNAL 3 present
IN PARENT: SIGQUIT pending
IN CHILD:
SIGNAL 0 present
SIGNAL 3 present
IN CHILD: SIGQUIT not pending in CHILD
aditya : lab9 $ _
```

CODE

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

void err_sys(const char* x){
     perror(x);
     exit(1);
}

static void sig_quit(int signo){
     printf("caught SIGQUIT\n");
     if (signal(SIGQUIT, SIG_DFL) == SIG_ERR)
          err_sys("can’t reset SIGQUIT");
}

void check_sigset(sigset_t sigset){
     int i;
     for(i = 0; i < 31; i++){
          if(sigismember(&sigset, i)){
               printf("SIGNAL %d present\n", i);
          }
     }
}

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

     if (signal(SIGQUIT, sig_quit) == SIG_ERR)
          err_sys("can’t catch SIGQUIT");

     sigemptyset(&newmask);
     sigaddset(&newmask, SIGQUIT); // adding SIGQUIT to newmask
```

```
    if (sigprocmask(SIG_BLOCK, &newmask, &oldmask) < 0) // added
SIGQUIT to BLOCK
        err_sys("SIG_BLOCK error");

    printf("Send SIGQUIT signals\n");
    sleep(5);
    /* SIGQUIT here will remain pending */
    if((pid = fork()) == -1){
        err_sys("Fork Error");
    }

    if(pid){
        printf("IN PARENT:\n");
        if (sigprocmask(0, NULL, &sigset) < 0) {
            err_sys("Error getting signal mask");
        }
        else {
            check_sigset(sigset);
        }

        if (sigpending(&pendmask) < 0)
            err_sys("sigpending error");

        if (sigismember(&pendmask, SIGQUIT))
            printf("IN PARENT: SIGQUIT pending\n");

        wait();
    }
    else {
        sigset_t childsigset;
        printf("IN CHILD:\n");
        if (sigprocmask(0, NULL, &childsigset) < 0) {
            err_sys("Error getting signal mask");
        }
        else {
            check_sigset(childsigset);
        }

        if (sigpending(&pendmask) < 0){
            err_sys("sigpending error");
        }

        if (sigismember(&pendmask, SIGQUIT)){
            printf("IN CHILD: SIGQUIT pending\n");
        }
        else {
            printf("IN CHILD: SIGQUIT not pending in CHILD\n");
        }
    }
}
```