National University
of Computer & Emerging Sciences
Peshawar Campus

Problem Set:	Assignment 04	**Semester**:	Spring 2013
Points:	4		
Date Set:	March 11, 2013	**Due Date**:	March 20, 2013
Course:	CS206 Operating Systems	**Instructor**:	Nauman

Note: Code in the following is intentionally left low-quality. Please write the code yourself.

1. The following code deals with signal basic handling.

   (a) Write the following program and observe and *explain* the output. Hint: This is an example of *null pointer dereferencing*. Explain what that means.

```
/* an example that dereferences a null pointer */

int
main(int argc, char **argv) {
        int a;

        a = *(int *)0;  /* read address 0 */
}
```

   (b) Modify the program to the following:

```
/* signal example #1: trap an illegal address reference */

#include <stdio.h>      /* for printf */
#include <stdlib.h>     /* for exit */
#include <signal.h>     /* defines signals and the signal() function */

int
main(int argc, char **argv) {
        void catch(int);
        int a;

        signal(SIGSEGV, catch); /* catch SIGSEGV - segmentation violation */
        a = *(int *)0;  /* read address 0 */
}

void
catch(int snum) {
        printf("got signal %d\n", snum);
        exit(0);
}
```

   (c) Run both versions and include screenshot of the output.

2. The following code demonstrates sending signal from one process to another. Write the code, modify it somehow to include your NUCES ID somewhere in the output, compile, execute and submit the screenshots demonstrating a test run.

```c
/* signal example 2: catch an external signal */
#include <stdio.h>
#include <signal.h>
#include <sys/wait.h>

int
main(int argc, char **argv) {
        void catch(int);
        int waitstat;

        signal(SIGUSR1, catch);
        printf("Run this in another window: kill -s USR1 %d\n", getpid());
        for (;;) {
                printf("Going to sleep\n");
                sleep(3600);      /* for a good long time */
        }
}

void
catch(int snum) {
        printf("got signal %d\n", snum);
        /* on SunOS systems, we have to  reset the signal catcher */
        signal(SIGUSR1, catch); /* don't need this on Linux or OS X (but doesn't hurt) */
}
```

3. This last piece of assignment demonstrates waiting for a signal from a child.

   (a) Compile and execute the following code:

```c
#include <stdlib.h>      /* needed to define exit() */
#include <unistd.h>      /* needed for fork() and getpid() */
#include <signal.h>      /* needed for signal */
#include <stdio.h>       /* needed for printf() */

int
main(int argc, char **argv) {
        void catch(int);          /* signal handler */
        void child(void);         /* the child calls this */
        void parent(int pid);     /* the parent calls this */
        int pid;          /* process ID */

        signal(SIGCHLD, catch); /* detect child termination */

        switch (pid = fork()) {
        case 0:           /* a fork returns 0 to the child */
                child();
                break;

        default:          /* a fork returns a pid to the parent */
                parent(pid);
                break;

        case -1:          /* something went wrong */
                perror("fork");
                exit(1);
        }
        exit(0);
}
```

```
void
child(void) {
        printf("        child: I'm the child\n");
        sleep(3);       /* do nothing for 3 seconds */
        printf("        child: I'm exiting\n");
        exit(123);      /* exit with a return code of 123 */
}

void
parent(int pid) {
        printf("parent: I'm the parent\n");
        sleep(10);      /* do nothing for five seconds */
        printf("parent: exiting\n");
}

void
catch(int snum) {
        int pid;
        int status;

        pid = wait(&status);
        printf("parent: child process exited with value %d\n", WEXITSTATUS(status));
}
```

(b) Now modify the program to create 5 children using a loop and wait for all of them to finish (hint: do not use the *wait* syscall). Return different exit codes from each child and output these codes from your main process. Compile and execute the program and submit screenshots of test runs.