**CS 367 - Recitation #10** **Signals/Unix-IO**

Adilet Kuroda

**Name**_____ **G#**_____

Thor Page

**Group Member Name:** _____

**Group Member Name:** _____

**Today's Goals:** We want to get comfortable with basic signal handling and Unix-I/O.

**Work in groups of 2-3 students.** Every group will turn in what they've got to Blackboard.

**Grading is based on participation.** Get as much done as you can. You will also be given feedback in the form of a 'score' (1-3) and possibly some comments. This doesn't affect your grade – it is solely for feedback. A score of 3 means everything looks great. A score of two indicates some minor problems. And a score of one indicates that there were some major issues. If you get a 1, don't panic - go see your prof or a GTA to get more extensive feedback.

---

# Signal Handling (Chapter 8.5)

1. Explain below in plain words.
   (i) What does it mean to "install" a signal handler?

When pass the address of handler function to a signal function

   (ii) What is "catching" or "handling" the signal?

When handler is invoked, it is called catching the signal and when the handler is executed, it is called handling the signal.

2. Below is a simple alarm code using signal. Write the code corresponding to the boxes.

```
/* SIGALRM handler */
        (c)
{
    printf("Rrrrrrr!\n");
    exit(0);
}

int main()
{
    /* No turning off the alarm: Ignore SIGINT (ctrl-c)*/
        (a)
    alarm(3*60); //set alarm
    /* Install the SIGALRM handler */
        (b)
    return 0;
}
```

**Reference:**
*int sigaction(int sig, const struct sigaction *restrict act, struct sigaction *restrict oact); // Registers handler*
Note: **struct sigaction** needs to be initialized to 0s (hint: memset(void *s, int c, size_t n) )
Note: struct sigaction has one important member: **sa_handler** to set to the handler function.

*void sighandler (int); // Signature for the handler*

**(a)** `/*Ignore SIGINT (ctrl-c)*/` (hint: use `SIG_IGN` as the function to ignore a signal)

```
struct sigaction action;
memset(&action, 0, sizeof(struct sigaction));
action.sa_handler = SIG_IGN;
sigaction(SIGINT, &action, NULL);
```

**(b)** `/* Install the SIGALRM handler */`

```
struct sigaction action_1;
memset(&action_1, 0, sizeof(struct sigaction));
action_1.sa_handler = alarm_handler;
sigaction(SIGALRM, &action_1, NULL);
```

**(c)** `/* SIGALRM handler signature */`

```
void alarm_handler(int sig)
```

# Unix I/O (Redirects) Chapter 10.9

3. Explain below in plain words.
   (i) What does it mean to "redirect" a stream from stdout to a file?

Instead of printing to terminal, data gets written to a file.

   (ii) What does the **dup2** function do?

System call that creates a copy of the file descriptor by using smallest available file descriptor.

4. Below is a simple redirect out sample program.

```
int main() {
    int fd = -1;
    pid_t child = 0;
   // Open the "out.txt" file for writing (O_WRONLY, O_CREAT, O_TRUNC flags)
   // Use this value for the permissions mode: 0644
    [          (a)          ]

    // fork to create a new process that will be redirected out.
    pid_t child = fork();

    // If it is the child...
    if(child == 0) {
      // Use dup2 to set fd to replace STDOUT_FILENO

      [          (b)          ]
      // Execute the process with path "/usr/bin/ls" and the command "ls"
      [          (c)          ]
    }
```

*int open(const char *pathname, int flags,int mode); // Open a device and returns an FD*
*Note: Flags…*

- *O_RDONLY read only*
- *O_WRONLY write only*
- *O_RDWR read and write*
- *O_CREAT If it doesn't exist, create a new file*
- *O_TRUNC If it does exist, erase it out before writing to it*
- *O_APPEND If it does exist, add the new stuff to the end*

*int execl(const char *path, const char *arg, ...); // path includes name, arg0 is the name, NULL to end the list.*

- *The … here means you can keep adding args one at a time, separated by a comma, until you're done.*
- *Then you use NULL to end it.*
- *Example: execl("/usr/bin/ls", "ls", "-a", "-l", NULL);*
- *execv is another version of exec that may be easier to use when actually programming.*

*int dup2(int oldfd, int newfd); // Create a copy of the file descriptor.*

- *Remember, oldfd is the one you want to install. newfd is where you want to install it.*
- *Eg. dup2(fd, STDIN_FILENO); // This will set STDIN of the process to fd.*
- *Valid newfds are: STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO*

**(a)**
```
// Open the "out.txt" file for writing (O_WRONLY, O_CREAT, O_TRUNC flags)
fd = open("out.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
if (fd < 0){
printf("Error");
return -1;
}
```

**(b)**
```
// Use dup2 to set fd to replace STDOUT_FILENO
dup2(fd, STDIN_FILENO);
```

**(c)**
```
// Execute the process with path "/usr/bin/ls" and the command "ls"
    execl("/usr/bin/ls", "ls", NULL);
```

Time remaining? Try these last two programs out on Zeus! You'll need the following #includes:
**<stdio.h>, <stdlib.h>, <sys/types.h>, <sys/stat.h>, <fcntl.h>, <unistd.h>, <signal.h>**