CMSC 216 Quiz 7 Worksheet

The next quiz for the course will be on Mon, May 4. The following list provides additional information about the quiz:

- Do not post any solutions to this worksheet in Piazza. That represents an academic integrity violation.
- The quiz will be a written quiz (no computer).
- Closed book, closed notes quiz.
- Answers must be neat and legible.
- Quiz instructions can be found at http://www.cs.umd.edu/~nelson/classes/utilities/examRules.html
- Make sure you know your section number and your TA's name.

We will include the following cheat sheet in the quiz.

Process Cheat Sheet

- `pid_t fork(void);`
- `pid_t wait(int *status);`
- `pid_t waitpid(pid_t pid, int *status, int options);`
- `int execvp(const char *file, char *const argv[]);`
- `int execlp(const char *file, const char *arg, ...);`
- `int open(const char *filename, int flags);`
- `int open(const char *filename, int flags, mode_t mode);`
- `int close(int fd);`
- `ssize_t read(int fd, void *buffer, size_t n);`
- `ssize_t write(int fd, const void *buffer, size_t n);`
- `int pipe(int pipefd[2]);`
- `int dup2(int old_fd, int new_fd);`

End of Process Cheat Sheet

Exercises

For problems requiring you to draw a process diagram use a diagram similar to one shown at:

http://www.cs.umd.edu/class/spring2015/cmsc216/content/resources/ProcessDiagramExample.pdf

1. Briefly explain the following terms:
    a. Kernel/system time
    b. User time
    c. Wall time
2. What is a file descriptor? What is the type associated with a file descriptor?
3. What is the difference between status and WEXITSTATUS(status)?
4. A process has opened two files and a pipe. Draw a diagram that illustrates the contents of the file descriptor table of the process and the contents of the kernel open file table. Specify the reference count for all open file descriptions.
5. Why do we need to create a pipe before we fork a process (instead of forking a process and then creating a pipe)?
6. When we dup2(fd, STDOUT_FILENO) we lose the reference to standard output. How can you restore file descriptor 1? Hint: Open a file.

7. Modify the following program so one process performs the sum(1, x/2) task and a second process the sum(x/2 + 1, x) task. You may not modify the sum function and you may not modify main.

```
int sum(int start, int end) {
   int answer = 0, i;

   for (i = start; i <= end; i++) {
      answer += i;
   }

   return answer;
}

int process(int x) {
   return sum(1, x / 2) + sum(x / 2 + 1, x);
}

int main() {
   int value;

   scanf("%d", &value);
   printf("%d\n", process(value));

   return 0;
}
```

8. Write a program that computes the average of integer values in a file. For this problem:

   - You will use only two processes: the parent and a child.
   - The parent process will:
     - Read the name of the file with integer values to average. Use the message "Enter filename:"
     - Create a child
     - Receive the average value computed by the child using a pipe
     - Display the message "Average value for file ***FILENAME*** is ***AVERAGE_VALUE"*** where ***FILENAME*** and ***AVERAGE_VALUE*** represent the filename and average value, respectively
   - The child process will:
     - Compute the average by calling the **compute_average()** function. **NOTICE THAT YOU MAY NOT MODIFY THIS FUNCTION.**
     - Return to the parent the average by using a pipe
   - **The parent will never open any file.**
   - The child will open the file.
   - You can assume the pipe can handle any amount of data we sent.
   - You may not add any functions beside main().

Below you will find an example of running the program you are expected to write. The executable's name is average, the file data.txt has the values to average, and the unix prompt is represented by a %.

```
% cat data.txt
2
4
12
% average
Enter filename: data.txt
Average value for file data.txt is 6
%
```

```
#define OPEN_FLAGS O_RDONLY
#define MAX_LENGTH 80

/* YOU MAY NOT MODIFY THIS FUNCTION */
int compute_average() {
   int count = 0, value, sum = 0;
   while(scanf("%d", &value) != EOF) {
      sum += value;
      count++;
   }
   return sum / count;
}

int main() {
   /* FUNCTION YOU NEED TO IMPLEMENT */
   return 0;
}
```