CMSC 216 Quiz 6 Worksheet

The next quiz for the course will be on Mon, May 2. The following list provides additional information about the quiz:

- The quiz will be a written quiz (no computer).
- The quiz will be in lab session.
- 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.
- You must take your quiz in your assigned lab/discussion section and not show up to a random discussion section. We will not grade quizzes taken in the incorrect section.
- **Regarding Piazza -** Feel free to post questions in Piazza regarding the worksheet and possible solutions to problems, but for coding questions please do not post code. You can post suggestions on how to solve coding problems, but your classmates will benefit more if they themselves actually solve the problems. Pretend you are a TA while addressing or providing help in Piazza ☺

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

1. When will fork() fail?

2. Provide an explanation and a diagram for each of the following:

    a. File descriptor table
    b. Kernel open file table
    c. Open file description
    d. Inode
    e. Reference count in an open file description

3. A process has opened two files. Draw a diagram that illustrates the contents of the file descriptor table of the process, and the contents of the kernel open file table. What is the reference count for each opened file?

4. Draw a diagram that:

    a. Illustrates the contents of the file descriptor table for a process that has created a pipe.
    b. Modify the diagram in a. after a fork() has taken place.

5. What is the purpose of dup2?

6. Why do we usually call dup2 before we call one of the exec* system calls?

7. Draw a diagram that:

    a. Illustrates the contents of the file descriptor table for a process that has opened a file.
    b. Modify the diagram in a. after a dup2 call has been used with the opened file and STDIN_FILENO.

8. Is Domino's Pizza better than Papa Johns?

9. Write a program where a parent process creates a child. The parent will send to the child (using a pipe) a number. The child will print the square root of the number sent by the parent.

10. Suppose a process creates a child. Which of the following two options do you think are best for the parent process? Could you think of a scenario where one could be preferable to the other?

```
/* Option 1 */
wait(NULL)
read(fd, buffer, 1024);

/* Option 2 */
read(fd, buffer, 1024);
wait(NULL);
```

11. 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;
}
```

Previous Quiz Question and Solution

Implement a program that reads a file name and writes to a file the result of calling the function print_powers we have provided. For this problem:

- The parent process will create a single child
- Parent's processing
  - It will read the name of a file using scanf
  - It will send that name to a child process using a pipe
  - You can assume the maximum filename is 80
- Child's processing
  - It will read the name of the file from the pipe
  - Using the provided file name, the child will open a file for writing using the options O_CREAT, O_WRONLY and 0666
  - The child will call the print_powers function. After the function has been called, the results must appear in the file that was opened
  - **You may not modify the print_powers function in any way. If you do, you will receive 0 pts for this problem**
- You don't need to use good variable names, but we expect good indentation.

```
void print_powers() {
   int i = 0, limit = 4;

   for(i = 0; i <= limit; i++) {
      printf("%d\n", i * i);
   }
}

int main() {
```

Answer:

```
int main() {
    pid_t child_pid;
    int pipe_fd[2];
    char filename[MAX_LEN + 1];

    pipe(pipe_fd);
    child_pid = fork();

    if (child_pid) { /* parent code */
       int status;

       close(pipe_fd[0]); /* closing pipe's read end */
       scanf("%s", filename);  /* reading filename */
       /* sending filename */
       write(pipe_fd[1], filename, strlen(filename) + 1);
       close(pipe_fd[1]); /* closing pipe's write end */
       wait(&status); /* reaping */
    } else { /* child code */
       int fd;

       close(pipe_fd[1]); /* closing pipe's write end */
       read(pipe_fd[0], filename, MAX_LEN + 1); /* reading file name */
       close(pipe_fd[0]); /* closing pipe's read end */
       fd = open(filename, O_CREAT | O_WRONLY, 0666);
       dup2(fd, STDOUT_FILENO); /* redirecting */
       close(fd); /* releasing resource */

       print_powers();
    }

    return 0;
}
```