**CSCI 4210 — Operating Systems**
**CSCI 6140 — Computer Operating Systems**
**Homework 2 (document version 1.0)**
**Files and Processes in C**

## Overview

- This homework is due by 11:59:59 PM on Thursday, September 24, 2015. Homeworks are to be submitted electronically.
- This homework will count as 4% of your final course grade.
- This homework is to be completed **individually**. Do not share your code with anyone else.
- You **must** use C for this homework assignment.
- Your program **must** successfully compile and run on Ubuntu v14.04.3 LTS or v15.04.
- Your program **must** successfully compile via `gcc` with no warning messages when using the `-Wall` compiler option.

## Homework Specifications

In this second homework, you will use C to implement a fork-based file-echo program. The goal is to work with files, as well as the `fork()` and `wait()` system calls.

Overall, your program will open a text file for reading, call `fork()` to create child processes to read equally sized "chunks" of the file, display each chunk to the user via the child processes, and call `wait()` to acknowledge when each child process terminates.

Unlike Homework 1, to open a file for this assignment, use `open()`. Further, use `read()` to read from the file and `printf()` to write file contents (and other output) to `stdout` (file descriptor `1`).

The file to read is specified on the command-line as the first argument. The second argument specifies the number of bytes each child process is responsible for (hint: use `atoi()` to convert to `int`). As an example, you could execute your program to read `filename.txt` with each child process reading and processing `10` bytes (i.e., a chunk) as follows:

```
bash$ ./a.out filename.txt 10
```

Note that you open the file only once in the initial parent process. Remember that this will create an entry in the file descriptor table, which will then be copied to (inherited by) each child process. Do not open the file again in any child process.

Your initial parent process must loop and create enough child processes (via `fork()`) to read the entire file. For example, if the input file contains `168` bytes and the chunk size is `10`, then the parent process must create `17` child processes, the last of which will process only `8` bytes.

To obtain the size of the given input file, use the `lstat()` system call, which will populate (if it successfully finds the file) the `struct stat` buffer containing the given file's size.

### Required Output

When you execute your program, first display the given file's size and how many child processes will be necessary to handle the file (see below for formatting details). Do not include any other debugging statements, though you should be in the habit of using `#ifdef DEBUG_MODE` to organize your debugging code. See the `main-with-debug.c` example on the course website.

Each chunk that a child process prints should be displayed with the child process's PID, as follows:

```
CHILD <pid> CHUNK: <chunk>
```

Note that chunks may contain `'\n'` (newline) characters.

Given the following example `hw2-sample-input.txt` file:

```
Once when a Lion was asleep a little Mouse began running up and down upon him;
this soon wakened the Lion, who placed his huge paw upon him, and opened his
big jaws to swallow him. "Pardon, O King," cried the little Mouse: "forgive me
this time, I shall never forget it: who knows but what I may be able to do you
a turn some of these days?" The Lion was so tickled at the idea of the Mouse
being able to help him, that he lifted up his paw and let him go. Some time
after the Lion was caught in a trap, and the hunters, who desired to carry him
alive to the King, tied him to a tree while they went in search of a wagon
to carry him on. Just then the little Mouse happened to pass by, and seeing
the sad plight in which the Lion was, sent up to him and soon gnawed away the
ropes that bound the King of the Beasts. "Was I not right?" said the little Mouse.
         "LITTLE FRIENDS MAY PROVE GREAT FRIENDS."
```

Running the code as follows should display the output shown below (though with different PIDs):

```
bash$ ./a.out hw2-sample-input.txt 42
PARENT: File 'hw2-sample-input.txt' contains 909 bytes
PARENT: ... and will be processed via 22 child processes
CHILD 42942 CHUNK: Once when a Lion was asleep a little Mouse
CHILD 42943 CHUNK:  began running up and down upon him;
this
CHILD 42944 CHUNK: soon wakened the Lion, who placed his huge
CHILD 42945 CHUNK:  paw upon him, and opened his
big jaws to
CHILD 42946 CHUNK: swallow him. "Pardon, O King," cried the l
etc.
```

See the course website for these sample input and output files. What happens when you comment out the parent's call to `wait()`? Submit your assignment **with** the call to `wait()` in place.

## Handling Errors

Your program must ensure that the correct number of command-line arguments are included. If not, display an error message and usage information as follows:

```
ERROR: Invalid arguments
USAGE: ./a.out <input-file> <chunk-size>
```

If system calls fail, use `perror()` to display the appropriate error message, then exit the program by returning `EXIT_FAILURE`.

Upon return from `wait()`, if the child process terminated due to a signal, display the following error message (but do not exit):

```
PARENT: child <child-pid> terminated abnormally
```

If the child process terminated normally, but its exit status was not `0`, display the following error message (but do not exit):

```
PARENT: child <child-pid> terminated with nonzero exit status <exit-status>
```

## Submission Instructions

To submit your homework, please create a single compressed and zipped file (using `tar` and `gzip`). Please include only source and documentation files (i.e., do not include executables or binary files!).

To package up your submission, use `tar` and `gzip` to create a compressed `tar` file using your RCS userid, as in `goldsd.tar.gz`, that contains your source files (e.g., `main.c` and `file2.c`); include a `readme.txt` file only if necessary.

Here's an example showing how to create this file:

```
bash$ tar cvf goldsd.tar main.c file2.c readme.txt
main.c
file2.c
readme.txt
bash$ gzip -9 goldsd.tar
```

Submit the resulting `goldsd.tar.gz` file via the corresponding homework submission link available in LMS (`http://lms9.rpi.edu`). The link is in the Assignments section.

As noted above, please submit your assignment **with** the call to `wait()` in place.