Assignment 1
CS-351
Spring 2015
Due Date: 03/07/2015 at 11:59 pm (No extensions)

Goals:

1. To gain hands-on experience with `fork()` and `exec()` system calls.
2. To master the basics of multi-process application development.
3. To appreciate the performance and fault-tolerance benefits of multi-process applications.
4. To implement a multi-process downloader application.

Overview

File downloaders are programs used for downloading files from the Internet. In this assignment you will implement two distinct types of *multi-process downloaders* (i.e. file downloaders that comprise multiple processes):

1. **a serial file downloader** which downloads files one by one.
2. **a parallel file downloader** which dowloads multiple files in parallel.

You will then compare the performance of the two types of downloaders.
Both downloaders will use the Linux `wget` program in order to perform the actual downloading. The usage of the `wget` is simple: `wget <FILE URL>`. For example, running from command line the following command:

```
wget http://releases.ubuntu.com/11.10/ubuntu-11.10-desktop-i386.iso
```

will download the Ubuntu Linux iso image to the current directory. Before proceeding with the assignment, you may want to take a moment to experiment with the `wget` command.

In your program, the parent process shall first read the file, `urls.txt`, containing the URLs of the files to be downloaded. `urls.txt` shall have the following format:

```
<URL1>
<URL2>
.
.
.
```

For example:

```
http://releases.ubuntu.com/11.10/ubuntu-11.10-desktop-i386.iso
http://releases.ubuntu.com/12.04/ubuntu-12.04.1-desktop-i386.iso
```

Next, the parent process shall fork the child processes. Each created child process shall use the `execlp()` system call to replace its executable image with that of the `wget` program. The two types downloaders are described in detail below.

The two downloaders shall be implemented as **separate programs**. The serial downloader program shall be called `serial.c` (or `.cpp` extension if you use C++). The parallel downloader program shall be called `parallel.c` (or `.cpp` extension if you use C++).

Serial Downloader

The serial downloader shall download files one by one. After the parent process has read and parsed the `urls.txt` file, it shall proceed as follows:

1. The parent process forks off a child process.
2. The child uses `execlp("/usr/bin/wget", "wget", <URL STRING1>, NULL)` system call in order to replace its program with `wget` program that will download the first file in `urls.txt` (i.e. the file at URL `<URL STRING1>`).
3. The parent executes a `wait()` system call until the child exits.
4. The parent forks off another child process which downloads the next file specified in `urls.txt`.
5. Repeat the same process until all files are downloaded.

Parallel Downloader

1. The parent forks off n children, where n is the number of URLs in `urls.txt`.
2. Each child executes `execlp("/usr/bin/wget", "wget", <URL STRING>, NULL)` system call where each `<URL STRING>` is a distinct URL in `urls.txt`.
3. The parent calls `wait()` (n times in a row) and waits for all children to terminate.
4. The parent exits.

Please note: while the parallel downloader executes, the outputs from different children may intermingle. This is acceptable.

Performance Comparison

Use the `time` program to measure the execution time for the two downloaders. For example:

```
time ./serial
real 0m10.009s
user 0m0.008s
sys 0m0.000s
```

The column titled `real` gives the execution time in seconds. Please get the execution times for both downloaders using the following `urls.txt` file:

```
http://releases.ubuntu.com/11.10/ubuntu-11.10-desktop-i386.iso
http://releases.ubuntu.com/12.04/ubuntu-12.04.1-desktop-i386.iso
```

Your execution times should be submitted along with your code (see the section titled "Submission Guidelines".

BONUS

Implement a multi-process linear search.

The parent process shall load a file of strings into an array (or a vector), split the array into n sections, and then fork off n children. Each child process shall search one of the n sections of the array, for the specified string. If the child does not find the string, it terminates with the exit code of 1. Otherwise, it terminates with the exit code of 0.

The parent, meanwhile, continuously executes `wait()`. Whenever, one of the child processes terminates, the `wait()` will unblock, and the parent process shall check the exit code of the child. The exit status of the child can be checked using the `WEXITSTATUS` macro as follows:

```
//The variable to hold the exit status int exit_status;

//Other code ....

if(wait(&exit_status) < 1) { perror("wait"); exit(-1); }
if(WEXITSTATUS(exit_status) == 0) { ...  } ...
```

If all children terminate with the code of 1, then the parent prints `"No string found"` to the terminal. Otherwise, the parent terminates all child processes and exits. A child process can be terminated using the `kill()` system call. For example, `kill(1234,SIGKILL)` will terminate the process with process id `1234` (be sure to include the header files `#include <sys/types.h> and #include <signal.h>`; otherwise your program may not compile).

Technical Details

The program shall be ran using the following command line:

```
./multi-search <FILE NAME> <KEY> <NUMBER OF PROCESSES>
```

Where `<FILE NAME>` is the name of the file containing the strings, `<NUMBER OF PROCESSES>` is the number of child processes, and `<KEY>` is the string to search for. For example, `./multi-search strings.txt abcd 10` tells the program to split the task of searching for string `abcd` in file `string.txt` amongst 10 child processes.

SUBMISSION GUIDELINES:

- ***This assignment MUST be completed using C or C++ on Linux.***
- You may work in groups of 2.
- ***Your assignment must compile and run on the TITAN server.*** Please contact the CS office to for an account.
- Please hand in your source code electronically (do not submit .o or executable code) through **TITANIUM**.
- You must make sure that the code compiles and runs correctly.
- Write a README file (text file, do not submit a .doc file) which contains
  - Your name and email address.
  - The programming language you used (i.e. C or C++).
  - How to execute your program.
  - The execution times for both downloaders.
  - Whether you implemented the extra credit.
  - Anything special about your submission that we should take note of.
- Place all your files under one directory with a unique name (such as `p1-[userid]` for assignment 1, e.g. `p1-mgofman1`).
- Tar the contents of this directory using the following command. `tar cvf [directory_name].tar [directory_name]` E.g. `tar -cvf p1-mgofman1.tar p1-mgofman1/`
- Use TITANIUM to upload the tared file you created above.

Grading guideline:

- Program compiles: 10'
- Correct serial downloader: 40'
- Correct parallel downloader: 40'
- Execution times for both downloaders: 5'
- README file: 5'
- Bonus: 15'
- Late submissions shall be penalized 10%. No assignments shall be accepted after 24 hours.

Academic Honesty:

Academic Honesty: All forms of cheating shall be treated with utmost seriousness. You may discuss the problems with other students, however, you must write your **OWN codes and solutions**. Discussing solutions to the problem is **NOT** acceptable (unless specified otherwise). Copying an assignment from another student or allowing another student to copy your work **may lead to an automatic F for this course**. Moss shall be used to detect plagiarism in programming assignments. If you have any questions about whether an act of collaboration may be treated as academic dishonesty, please consult the instructor before you collaborate. Details posted at http://www.fullerton.edu/senate/documents/PDF/300/UPS300-021.pdf.