IT383 Fall 2024

Homework#2

9/24/2024

Shane Stevens

Total: 80 HW points

Late submissions will not be accepted. **All answers except figures and equations should be typed**. Do not copy solutions obtained from the publisher, the Internet, or other students. (If you are unfamiliar with the grading policy, refer to the course syllabus). ***You need to provide detailed explanations for all questions!***

For coding questions:

* Log into one of the IT Linux servers using your ULID and password. You will need to use the following link to access IT Linux servers:
  + (<https://login.it.ilstu.edu/#/>
* Download Linux Command mini reference documents from Canvas website for IT383(filenames are shown in the following) and then save it to your home directory on IT Linux servers.
  + Filename:  davechild\_linux-command-line.pdf

Programs containing compilation errors will receive failing grades. Those containing run-time errors will incur a substantial penalty. You should make a serious effort to complete all programs on time. You will lose 10% per day (up to 3 days) if the submission is made after the deadline. Programming assignments are individual. You should complete them with your own effort. If you need help, you should come to my office hours or contact me by email. You may discuss concepts with others in the class, but not specific program code**.** The total points of this programming project are 40 programming points.

**~~NOTE: Program 0 should be done WITHTOUT any collaboration with any other student! I am required to report any cheating in programming assignment to Dean of students.~~**

**NOTE: DO NOT upload any of your programming assignments to any public Internet repository including Github.**

**Snapshots of working code must be attached with the question.**

Lab Questions

1] Write a C program called “my\_copy\_textbinary\_count.c”. This program makes a duplicate copy of a given text file or binary file. Note that the source filename and destination filenames are given as command-line arguments as shown in the following example. This program also prints out the total number of bytes from the source text file to the destination file. Example usage of the program is shown in the following. Note that $ is a shell prompt, which might be different from the prompt of the shell you are using.

$ **my\_copy\_textbinary\_count** /usr/share/dict/american-english ./dictionary.txt

/usr/share/dict/american-english was copied to ./dictionary.txt successfully.

Total number of bytes: 971578

Requirements:

0) Note that the source filename should be the first command-line argument.

1) You need to use ALL of the following C APIs (all of them!) in your program to read and write data

from/into files.

open(), close(), read(), write()

DO NOT use fopen(), fgetc(), fread(), fgets(), getline(), fputs(), fscanf(), fprintf(), in this program.

2) Your program should check the error codes returned by calls to open/read/write functions.

3) The size of the buffer (i.e., array) for file input and output operations should be 1024 bytes exactly.

In your program, you will need to use a loop where:

a) a block of data is read from the source file to the buffer(i.e., array).

b) the block of data in the buffer is written into the destination file.

c) the maximum number of bytes to be read or written specified as a parameter for read() and

write() should be 1024.

If you use a buffer of size larger than 1024 or if you do not use a loop for reading or writing data

from/to files, you will lose at least 30% of your points allocated to his program automatically.

Please note that the size of a source file can be very large (as large as hundreds Mega bytes).

1) [10 points] You need to try the following sample test cases to make sure that your program runs correctly. Note that “diff” is a Linux command to check if two files are exactly the same in terms of size and contents.

[a sample run using a sample text file as the source file]

$ **my\_copy\_textbinary\_count** SRC file DST file

/SRC was copied to ./DST successfully.

Total number of bytes: 5325648

% diff file1 file2

* Diff command should indicate that two given files are the same for each execution. Otherwise, your program didn’t work correctly! Hint: Most likely, the size of two files will be different since you write additional character(s) in the target file(s).

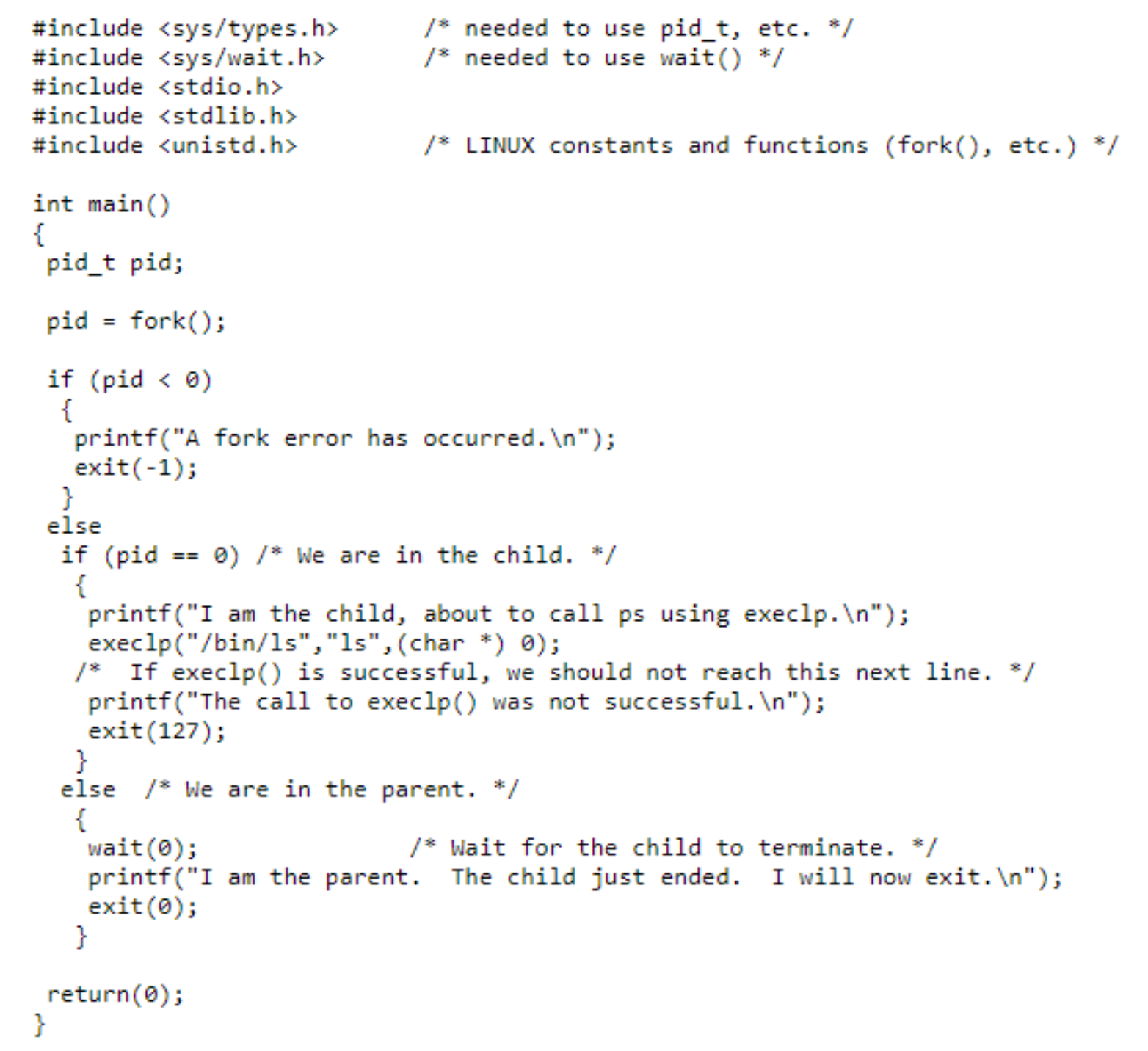
**Hint: Make sure that the total number of bytes that you read from the input file is same as the total number of bytes that you write to the output file. You should be careful about the invisible characters such as ‘\n’ and ‘\0’.**

**Hint: DO NOT ignore the return values from open(), read(), and write(). Check MAN page for the sample code using open(), read(), and write() APIs**

A computer screen with white text

Description automatically generated

2) **Type** in the C program shown in the figure above and compile/run/test the program on IT Linux system.



1. Change the variable name “pid” to “cid” in the typed C program. This change will make some clarification.

A computer screen shot of a program code

Description automatically generated

1. Change the variable name “pid” to “cid” in the given program to make it clear that the return value of fork() is not necessarily the pid of child process.

I don’t see how this is any different from the previous question

1. **Add** additional code to check the return value/error code of execlp() and wait() function calls. Without the additional code, you automatically lose 2 points. You can find how to check the return values of execlp() and wait() by using manual page for the functions by running the following commands on Linux terminal:
   1. **man execlp**
   2. **man wait**

**Explain in detail how APIs (fork(), execlp(),wait()) work in a separate plain text file (NO PDF or MS-Word).**

A screen shot of a computer code

Description automatically generated

Deliverables: source code, snapshot of the output, written document **(in plain text; at least 100 words in total; NO MS-word or PDF format!)**

3) (5 points) fork and execvp

Change the above program to run with the execvp command, refer to man page for syntax. Also, provide an essay explaining how it works.

Deliverables: source code, ~~Makefile,~~ written document (**in plain text; at least 50 words**)

**Questions: (Regarding basic Linux commands)**

1. (1.5 point) What is the purpose of ‘history’ command?

The history command shows commands that were previously executed in the terminal

1. (1.5 point) What is the purpose of ‘kill -9 *pid*’ (or '**kill** -SIGKILL pid’) command?

Kill -9 pid kills the process with the given process id immediately

1. (2 points)

6-a) What is the main purpose of ‘ps’ and ‘top’ commands?

Ps gives the process id, time and the command that the process is for, while top gives a lot more info about the processes that are running, and also shows more processes. These can be used to tell what the computer is currently doing and which processes are the most resource intensive, among other things.

6-b) What is the purpose of “pkill -u username” command?

Pkill -u username terminates all processes that were created by the given username. Unlike kill -9, it allows the processes to terminate neatly, instead of instantly killing them.

1. (1point) What is the Linux command to print the contents of a text file on console screen?

cat filename

1. (3 points) What are the three Linux commands that you need to use copy, delete, and rename a file? Also, show an example usage of each of the Linux commands.

The command for copy is cp. In this example I copy a file named test, and call the new file test2

A screenshot of a computer

Description automatically generated

The command for renaming a file is mv. In this example I rename test2 to notTest2

A screenshot of a computer program

Description automatically generated

The command for deleting a file is rm. In this example, I delete notTest2

A screenshot of a computer

Description automatically generated

1. (1 point) Show how you can use pipe (|) between two executable programs. (e.g., The output of “ls –lR /” is fed into “more” program)

The pipe is used to feed the output of one command into another command. For example, feeding ls -lR into the more program shows the files in each folder, such as in the picture

A screenshot of a computer

Description automatically generated

Assignment Questions

[1] (10 points) Including the initial parent process, how many processes are created by the program shown in the following C code:

***#include <stdio.h>***

***#include <unistd.h>***

***int main()***

***{***

***int i;***

***for (i=0; i < 10; i=i+2)***

***fork();***

***return 0;***

***}***

This program creates 1 \* 2 \* 2 \* 2 \* 2 \* 2, or 32 processes. This is because the code starts out with 1 process, and the code goes through the loop 5 times (i = 0, i = 2, i = 4, i = 6, and i = 8). Since each individual process goes through the loop, each process is creating a another new process, meaning that the number of processes double for each trip through the loop.

[2] (5 points) Describe **in detail** the actions taken by a kernel to context-switch between processes.

When a context switch occurs, the CPU saves the information, or “context” of the current process being ran in its PCB (Process Control Block). The CPU then loads the PCB of the process that it is switching to. The CPU cannot do anything else during a context switch.

[3] (5 points) What system calls have to be executed by a command interpreter or shell in order to start a new process? Describe the role of each system call in detail.

The first system call is fork. Fork creates the new process. The parent shares its data with the child process, and both processes continue following the code. The next system call is exec(). For this system call, the child’s memory space is replaced with the memory space of the new program specified in the exec() function. Once this is completed, or the exit() system call is occurs, the process terminates. The last system call is wait(). For this system call, the parent waits for the child to end before continuing on.

[4] (5 points) Describe three general methods for passing parameters to the operating system.

The first method is to pass the parameters via registers. The parameters are put in a register, which the operating system then accesses. If there are more parameters than registers, then the parameters are stored in blocks. This is method 2, where the parameters are stored in blocks or tables, and then passed through registers. The last approach is to use the stack. In this scenario, parameters are pushed onto the stack, and popped off by the operating system.

[5] (5 points) When a process creates a new process using the fork() operation, which of the following states is shared between the parent process and the child process? You need to **support your answer** by some explanations.

(a) stack (b) Heap (c) shared memory segments (d) text (e) Data

The only state shared between parent and child processes is the shared memory segments. This is because the shared memory segments are data that is specified to be shared between the parent and child. Each process has its own copy of the stack and heap. I guess data would be shared since both processes are running the same program, and are impacting the same variables. Text would also be shared since the processes would be working with the same text

[6] (5 points) Which of the following components of program state are shared across threads in a multithreaded process? Support your answer in detail.

a. Register values – Threads do not share registers since a register can only work with one thread at a time, making it impossible for them to share registers.

b. Heap memory – Heap memory is shared by all threads since one of the benefits of threads is that they share most of their memory. This is because threads share the same heap memory.

c. Global variables – global variables are shared by all threads because the whole point of global variables is that they are shared by all of a program.

d. Stack memory- Each thread has its own stack so that it can perform operations on its own. If all threads shared the same stack, it would be very difficult for all of the threads to work at the same time.

[7] (5 points) What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

User threads are controlled by the user while kernel threads are controlled by the kernel. Another major difference is that in order for user-level threads to run, they must be mapped to kernel-level threads. User threads also cannot truly run in parallel, since the kernel does not actually see the multiple threads, but instead just sees one process. This makes user threads better for when the user wants more control over the threads, while kernel level threads are better for faster execution, due to them being able to actually run in parallel.

[8] (5 points) Describe the actions taken by a kernel to context-switch between kernel-level threads.

Switching kernel-level threads is similar to switching processes, except the only data that needs to be stored is the registers, program counter, and stack, since the rest of the data is shared between all of the threads.

[9] (5 points) Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system? Explain in detail.

A multithreaded solution using multiple user-level threads can achieve better performance than a multiprocessor system on a single-processor system. The key to why is context switching. Both systems can only really run one thread/process at a time, but since threads take less time for context switching, the multithreaded system will have better performance.

[10] (5 points) Discuss the number of memory copies needed in (a) IPC using shared memory and (b) IPC using message passing. Also, discuss the other types of overheads in IPC.

In a shared memory system, only one copy of memory is needed, since all processes share the memory. On the other hand, message passing requires a copy for each process that is using the data. Message passing also has the added overhead of the system calls that are often used for message passing. On the other hand, shared memory can have conflicts if multiple processes try to access memory at the same time, and may have added overhead because one process may be waiting for another process to finish accessing memory.