CS 551 Systems Programming, Summer 2024

Homework Assignment 2

Out: 7/17/2024 Wed.

Due: 7/24/2024 Wed. 23:59:59

PART I: Q&A (50 points)

I.1. (5 points) A program performs the following steps:

(1) Opens file `x.dat` with flags `O_WRONLY|O_CREAT` resulting in file descriptor `fd1`.

(2) Opens file `x.dat` with flags `O_WRONLY|O_CREAT` resulting in file descriptor `fd2`.

(3) Writes 6 'A' characters to `fd1`.

(4) Writes 3 'B' characters to `fd2`.

(5) Does a `dup2(fd1, fd2)`.

(6) Writes 3 'B' characters to `fd2`.

(7) Closes descriptors fd1 and `fd2`.

Assuming that `x.dat` does not initially exist and that there are no errors, show the contents of `x.dat` after **each** of the above steps, and **explain why**.

I.2. (5 points) The following simple program copies standard input to standard output:

```
#include <stdio.h>
int main()
{
    int c;
    while ((c = getchar()) != EOF) putchar(c);
    return 0;
}
```

When compiled into executable t and run as "`./t`", the program faithfully echoes each line back right after it is typed in. However, when run as "`./t | ./t`", input lines are not echoed immediately but in batches only after some number of lines have been typed in. Explain the difference in behavior between the two executions.

I.3. (10 points) List bugs and inadequacies in the following function:

```
/** Return number of sym-links in specified directory dir. */
int count_sym_links(const char *dir)
{
    DIR *dir = opendir(dir);
    int count = 0;

    for (dirent dP = readdir(dir); dP; dP = readdir(dir))
    {
        const char *name = dP->d_name;
```

```
            struct stat *statP;

            char dir_name[strlen(dir) + 1 + strlen(name)];
            sprintf("%s/%s", dir, name);
            stat(dir_name, statP);
            if (S_ISLNK(statP->st_mode))
                count++;
        }

        return count;
    }
```

Assume that all required header files have been included. It is not necessary to report the lack of checking for errors on system or library calls.

I.4. (12 points) User `u1` belongs to primary group `g1` and supplementary group `g2`, and user `u2` belongs only to primary group `g2`. Neither `u1` nor `u2` are super-users. Given the following `ls -l` listing:

```
-rwxr-xr-x    1 u2     g2      14096 Sep  8 22:38 exec1
-rwsr--r-x    1 u1     g1      44096 Sep 25 01:36 exec2
-rw-r--rw-    1 u2     g2       4012 Sep 23 01:12 data1
-rw-r---w-    1 u1     g1       8222 Sep  8 17:13 data2
```

Say user `u1` and `u2` try to execute the executable files `exec1` and `exec2`, in which they try to read or write the data file `data1` and `data2`. Fill in the following matrix with a '`Y`', '`N`' or '`-`' depending on whether the execution specified by the row can (`Y`) or cannot (`N`) make the access specified by the column (`R` denotes read, `W` denotes write), or such access does not make sense (`-`) (for exmaple, if you think the user cannot execute the executable, then fill a '`-`' in the corresponding row).

	data1 R	**data1 W**	**data2 R**	**data2 W**
`u1` runs `exec1`				
`u2` runs `exec1`				
`u1` runs `exec2`				
`u2` runs `exec2`				

Please justify you answers.

I.5. (8 points) Among the commands provided by a shell (bash for example), some of them are built-in commands, which means they are part of the shell program and are called directly by the shell. The other commands are external programs that can be loaded and executed by the shell program (using `fork`(), `exec`(), etc.). For each of the following commands, discuss if it **must** be a built-in command , or can be an external program, and why.

(1) **`cd`** (changes the current working directory)

(2) **`pwd`** (prints the current working directory)

(3) **`exec`** (replaces the shell by another program)

(4) **`exit`** (exits the shell)

I.6. (10 points) Discuss the validity of the following statements

(1) Writing non-empty data to a file descriptor will always change the file pointer associated with the file descriptor.
(2) If a user with a particular user id creates a file and the permissions/ownership on that file are never changed, then that user can also remove the file.
(3) The user and group ownership of a new file is always set to the effective uid and gid of the process which created it.
(4) Moving a file from one directory to another is done by copying the file and removing the original.

PART II: Programming tasks (50 points)

II.1. (20 points) In this task, you are going to write a C program using universal file I/O functions to reproduce a file of records.

The scenario is we have a data file containing multiple rows of contents (rows are separated by a newline character (\n)). Each row contains multiple records that are separated by spaces or tabs. The file does not contain any holes, which means there is no NULL byte in the file. You need to write a program, which switches the two records in each row according to the user's intention, and appends the new content to the original data file. Say the program name is "switch_column", then this program is used in the following way,

```
./switch_column  path_to_file  col_1_idx  col_2_idx
```

where `path_to_file` specifies the path to the data file, `col_1_idx` and `col_2_idx` are the indices of the two records (in the same row) that are going to be switched. Indices of records within each row start from 1. If the number of records in a row is smaller than either `col_1_idx` or `col_2_idx`, then the row keeps the same (i.e., no switching happens).

For example, we have a data file name "data_file" in the same directory as the program, and the data file has the following content:

```
ab cd ef gh ij
AB CD EF GH IJ KL
12 34 56 78
```

If the program is run as

```
./switch_column  data_file  2  3
```

then the content of the data file will change to:

```
ab cd ef gh ij
AB CD EF GH IJ KL
12 34 56 78
===
ab ef cd gh ij
AB EF CD GH IJ KL
12 56 34 78
```

If the program is run as

```
./switch_column  data_file  6  2
```

then the content of the data file will change to:

```
ab cd ef gh ij
AB CD EF GH IJ KL
12 34 56 78
===
ab cd ef gh ij
AB KL EF GH IJ CD
12 34 56 78
```

More specs and requirements of the task

- Each row in the data file will contain no more than 64 characters.
- Your program should NOT `malloc` space that is larger than 64 bytes (i.e., should not be larger than line size).
- Records in the new content are separated by a single space (i.e., you don't need to preserve the white spaces as in the original content).
- Your should not use an intermediate file to complete the task.

II.2. (30 points) Suppose that we have three processes related as grandparent, parent, and child, and that the grandparent doesn't immediately perform a `wait()` after the parent exits, so that the parent becomes a zombie. When do you expect the grandchild to be adopted by `init`: after the parent terminates or after the grandparent does a `wait()`? Write a program to verify your answer.

In this program,

- The parent process sleeps 3 seconds after it forks the child process and regains the CPU, and it terminates after waking up.
- The grandparent process sleeps for 6 seconds after it forks the parent process and regains the CPU, and it performs a `wait()` on the parent process after waking up.
- The child process performs busy polling, and output a message once it detects it has been adopted by the `init` process.
- All processes should be terminated after the program exits.
- Report the following events to the standard output
  - process creation, also report the process ID in this case
  - process termination
  - `wait()` returns
- Each output message should include the time interval (in μs, i.e., microseconds) between the time when the program starts running and the time when the message is output. (Approximate time intervals would be sufficient. For example, you can perform a `gettimeofday()` after the program starts running, and this will be the time when the program starts. Before you want to output a message, perform another `gettimeofday()`, and calculate the difference as the time interval.)

- The output messages should be nicely organized such that it is easy to understand what the program is supposed to demonstrate.

The name of the compiled program should be "oz_demo".

Submission instructions

1. For PART I:
    - Type your answers using whatever text editor you like, remember to include the index number of each question.
    - Export the file to PDF format.
    - Name the PDF file based on your BU email ID. For example, if your BU email is "abc@binghamton.edu", then the PDF file should be named as "hw2_abc.pdf".
    - Submit the PDF file to Brightspace website before the deadline.
2. For PART II:
    - Write a Makefile that generates the executable programs for task II.1 and II.2.
        - For II.1, the program name should be "switch_column".
        - For II.2, the program name should be "oz_demo"
    - Compress the Makefile and all the C source files into a ZIP file. Name the ZIP file based on your BU email ID. For example, if your BU email is "abc@binghamton.edu", then the zip file should be "hw2_code_abc.zip".
    - Do not put executables in the above folder. Source files only.
    - Submit the ZIP file to Brightspace before the deadline.

Grading guidelines:

(1) Compress the PART II ZIP file on a Linux machine. If your zip file cannot be uncompressed, 5 points off.

(2) If the submitted PDF file/ZIP file/source code files included in the ZIP file are not named as specified above (so that it causes problems for TA's automated grading scripts), 10 points off.

(3) If the submitted code does not compile:

```
TA will try to fix the problem (for no more than 3 minutes);
if (problem solved)
  1%-10% points off (based on how complex the fix is, TA's discretion);
else
  TA may contact the student by email or schedule a demo to fix the problem;
  if (problem solved)
    11%-20% points off (based on how complex the fix is, TA's discretion);
  else
    All points off;
```

So in the case that TA contacts you to fix a problem, please respond to TA's email promptly or show up at the demo appointment on time; otherwise the line 9 above will be effective.

(4) If the code is not working as required in this spec, the TA should take points based on the assigned full points of the task and the actual problem.

(5) Lastly but not the least, stick to the collaboration policy stated in the syllabus: you may discuss with your fellow students, but code should absolutely be kept private.