Université de Strasbourg
UFAZ

System Programming
2020/2021

Final exam - december 2021 - 1h30

Lecture notes and printed material (including solutions to PW exercices) are allowed. This is **individual work** (no talking, no files sharing, etc. allowed during the exam).

The answers to the questions should be written in C language, using exclusively system primitives (except for displaying information, e.g. `printf`).

The goal of this exam is to write a program which renames all the files contained in a given directory :

```
    rename dir old new
```

It renames all files matching `*.old` present in the directory `dir` into files `*.new`. Let us suppose the directory `rep` contains the following files :

```
toto.abc  a.out  titi.ab  tata.abc
```

Running the command

```
    rename rep abc jpeg
```

will produce the following result on the file system :

```
toto.jpeg  a.out  titi.ab  tata.jpeg
```

Your program shall be structured into 3 processes (a parent process and 2 children processes).

A process (the first child) opens the directory given as a parameter and transfers, via a pipe, the names of the files using the old suffix to a second process (the second child). When all the files are processed, the process terminates with a return code equal to the number of files transmitted into the pipe. We assume that there is never more than 255 files to deal with.

Another process (the second child) will read the names of the files from the pipe and rename each received file using the new suffix. Each time a file is renamed, a signal `SIGUSR1` must be transmitted to the main process (the parent). When all names transmitted through the pipe are treated, the process sends a signal `SIGTERM` to the parent process and terminates with a return code equal to the number of renamed files.

The parent process, after having created the 2 children processes, shall get ready and wait to receive signals.

Upon receiving the signal `SIGTERM`, the parent process will wait for its 2 children to complete the tasks and it will compare the return code of each child to the number of occurrences of the signal `SIGUSR1` it has received. If these values are equal, the main program will terminate with the return code 0. If these values are different, the return code should be different from 0.

If the parent process is too slow, it may receive signals faster than it can handle them. This means the reception of 2 successive identical signals may be counted as one single signal. To avoid that, the parent process must acknowledge that it has received the signal by sending back the signal `SIGUSR2` to the child process.

The child process must not send a new signal before it has received the acknowledgement signal. In practise, the child process must wait for the acknowledgement signal before sending a new signal.

You should only use system primitives, except to deal with strings (e.g. you can use the library function printf). Errors should be handled properly, in the following way :

```
    if (result == -1) error (...)       // no need to write an explicit message
```

Exercice 1 – 2 points

Write a function `void prepare (void)` which prepares a process to be ready to receive signals `SIGUSR1`, `SIGUSR2` and `SIGTERM`. To make things easier, you may assume the setting is the same for all 3 processes.

Exercice 2 – 2 points

Write a function `int wait_signals (pid_t pid)` which waits for signals `SIGUSR1` and `SIGTERM`. You should make sure not to wait forever for the expected signals. When receiving the signal `SIGUSR1`, the current process must send the signal `SIGUSR2` to the process `pid`. When receiving the signal `SIGTERM`, the function must terminate and return the number of times the current process has received the signal `SIGUSR1`.

Exercice 3 – 4 points

Write a function `int traverse (const char *filename, const char *osufx, int desc)` which traverses the directory denoted by `filename` and writes into the descriptor `desc` the full name of each file whose suffix corresponds to `osufx`. To make things easier, we assume the directory `filename` only contains regular files. We assume there exists a function

```
int endswith (const char * name, const char * sufx)
```

which returns 1 if `name` ends with the character « . » followed by the string pointed by `sufx`, 0 otherwise. Once all files are treated, the function returns the number of files whose name has been written into the pipe.

Explain shortly how a filename of arbitrary length can be written (or read) on the descriptor desc.

Exercice 4 – 3 points

Write a function `void rename (const char *filename, const char *osufx, const char *nsufx)` which renames a file `filename` by replacing its current suffix (which corresponds to `osufx`) by the one which corresponds to `nsufx`). In the function `rename`, we know that `filename` ends with a character « . » followed by the string pointed by `osufx` because it was already tested in the function `endswith`.

Exercice 5 – 4 points

Write a function `int modify (const char *osufx, const char *nsufx, int desc, pid_t pid)` which reads the names of the files from the descriptor `desc`. Each file received must be renamed using the function `rename` (written in the previous question). For each renamed file, your program must send a signal `SIGUSR1` to the process `pid`. When all files are dealt with, the signal `SIGTERM` must be sent to the process `pid` and the function must return the number of renamed files.

Exercice 6 – 2 points

Write a function `int wait_children (int nb_files)` which waits for the 2 children to terminate. If the return code of both children is the same and equal to `nb_files` , then this function must return 0. In all other cases, it must return 1.

Exercice 7 – 3 points

Write a function `int main (int argc, char * argv []).`

Your program takes as argument the name of a directory, a suffix for files you want to rename and the new suffix to use. If the number of treated files does not correspond to the number of signals received by the parent process, then your program must terminate with a non-0 return code.