Gebze Technical University
Department of Computer Engineering - CSE 344 System Programming
Spring 2018-19, HW #4 (Due March 31th @23:55)
NO LATE SUBMISSIONS

This HW is based on the homework two. Again, you will create a new process for each directory to get the size of that directory. Do not use exec, just use fork. Each created processes will be responsible for:

- Finding the size of the directory given by parent process,
- Creating new processes to find the sizes of subdirectories. The created child processes must be able to run in parallel,
- Writing the PID of the process, size in kilobytes and path of the directory to a single global FIFO "<#yourstudentid>sizes"(example: 111044002sizes). As multiple processes will write the same FIFO at the same time, your operations must be **atomic**. Do not use any synchronization tool if it is not needed(read FIFO documentations). Remove the FIFO after execution.

If -z option is given, the processes will also be responsible for:

- Receiving the total size of each subdirectory from pipe connected to child processes,
- Sending the total size of current directory to parent process via pipe,

After your child processes write all the size of all directories, main process will read the FIFO, do any operations if needed and output the results to standard output. The order of output is important and it should be post-order. Finally, it will output the number of created child processes and exit.

Do not add the sizes of files or the directories pointed by a symbolic link. Just say that there is a special file. Do not show size of any file explicitly, just directories.

The program `buNeDuFPF` with the argument `rootpath` will be called like below. The `rootpath` can be any path on the system(except the paths requiring root priviledges), relative or absolute.

```
./buNeDuFPF [-z] rootpath
```

When none or meaningless command line arguments are given to the program(s) the output should warn the user and print a usage, informing the user how the program should actually be called.

There shouldn't be any zombie or orphan processes when the execution is finished. Also there shouldn't be any memory leak. You can test these with valgrind.

Your homework will be tested by our test directories and files.

Ask your questions in the Moodle forum "HW4 Questions" by opening a new topic.

Example is given in the next page. Your homework should output by the same style given in the example.

Homework format:

Inside zip file→ Your .c and .h files and a makefile named Makefile. You makefile must **just** compile your program when "make" command is given from terminal. There shouldn't be and directory inside zip file and your makefile or program should not generate a directory.

If you submit your homework 1 or 2, you grade will be -100. This homework must use pipes and FIFO. Do not use any code from internet or friends, there is no need for that.

DO NOT FORGET TO READ ALL OF THE HOMEWORK CAREFULLY AND EXAMINE THE EXAMPLE IN THE NEXT PAGE.

An example:

A (directory)
|- B (directory)
|- kagurachan.exe (15 MB)
|- C (directory)
|- beware(a special file like smybolic link or pipe) (assume 0 KB)
|- squirtle.pu (300 KB)
|- gintoki.png (3 MB)
|- blabla.txt (2 MB)
|- lalala.exe (8 MB)
|- D (directory)
|- beware2(a special file like smybolic link or pipe) (assume 0 KB)
|- okletsgo.sh (1 MB)

Output of “buNeDuPDF -z A” gives total sizes:

PID	SIZE	PATH
1415	-1	Special_file_beware
1415	300	A/B/C
1414	18732	A/B
1413	0	A/D
1412	-1	Special_file_beware2
1412	29996	A

4 child processes created. Main process is 1411.

Output of “ buNeDuPDF A” don’t add subdirectory sizes:

PID	SIZE	PATH
1415	-1	Special_file_beware
1415	300	A/B/C
1414	18432	A/B
1413	0	A/D
1412	-1	Special_file_beware2
1412	11264	A

4 child processes created. Main process is 1411.