Assignment 1: Process and Summarize

Due date: *Wednesday, December 21, 2024 at 22:00*

> *This is an individual assignment. You may discuss it with others, but your code and documentation must be written on your own.*

In a source file called `multiproc.c` write a C program that performs a given processing operation on a set of files, and presents the results using a summarization program. The processing and summarization operations should proceed in parallel, which means that you have to create separate processes for both processing and summarization. The general usage of the `multiproc` program is as follows:

```
Usage: ./multiproc [options] [file]...

Arguments:
  [file]...       The list of regular files on which the program should
                  apply the processing and summarize operations. The
                  list can also contain some directories. In case of a
                  directory, the program will apply the processing and
                  summarizing operation on all regular files within the
                  directory and its sub-directories.

Options:
  -p <prog>, --proc <prog>              The program for the processing
                                        operation (default: cat)
  -s <prog>, --summary <prog>           The program for the summarize
                                        operation (default: cat)
  -w <workers>, --workers <workers>     The number of forked processes
                                        (default: 5, max: 20)
  -h, --help                            Display this help and exit
```

The program takes, as command-line arguments, zero or more *options* and zero or more *file names.* In case of zero file names, the program should not output anything. A file name can be either a regular file or a directory. For each file name, the program should apply the processing. In case of a directory, the program should apply the processing to all regular files within the given directory and its sub-directories. You can use the `stat` POSIX function do determine weather a given file name identifies a directory or a regular file. Also, you can use the `opendir` and `readdir` functions to iterate over the files within a directory.

The processing consists of running the processing program with a file name as an argument. A processing program is an executable installed on the host. In practice, you should fork a new "worker" process, and execute the processing program with the file name as first argument within that process. Then, the program should pass the output of the worker to the summarization program, which should also run in a separate child process. The summarization program is also an executable. The main program should be able to search both the processing and summarization programs using the `PATH` environment variable. The processing program has a line-based output, meaning that it will output some text separated by new lines. The output lines will be at most 1024 character long.

The `-w` option specifies the maximum number of worker processes that the can exist at any given time. In any case, a user is not allowed to specify more than 20 workers, and your program must check that `-w` option does not exceeds this limit. When all the workers are allocated, the program must ensure the progress of each worker. That means that your program should pass the lines of output from the workers to the summarization process as

soon as they become available. When there are more input files than available workers and a worker terminates—meaning that the processing program has terminated for the given file—the main program must fork another worker program to process another file.

When given an invalid option, the program should output its help page to `stderr`, and terminate with status code `EXIT_FAILURE`. Similarly, any error during the program execution should output an error message to `stderr` and terminate with status code `EXIT_FAILURE` without leaving any zombie process. In case of success, the program should terminate with `EXIT_SUCCESS` status code leaving no zombies.

To test your implementation, we made available a small set of automated tests. Obviously, you are allowed to extend with tests you consider useful for your implementation.

A possible program usage, could be computing the maximum compression ratio for all files within a directory. The processing would be a program that uses a compression tool (`gzip` for example) to compress a given file, and obtain the compression ratio. The summarization program could be a program that given a list of numbers returns the maximum value. For instance, the program execution on the command line might look as follows:

```
./multiproc -p ./tests/procs/compress_ratio.sh \
  -s ./tests/procs/max.sh -w 20 \
  ./tests/compress/ ./tests/lorem_ipsum.txt
14
```

The above command, applies the `./tests/procs/compress_ratio.sh` processing program to all files within the `./tests/compress/` directory, and to the `./tests/lorem_ipsum.txt` file using at most 20 processing workers. `./tests/procs/compress_ratio.sh` is a script that compresses a file using `gzip`, and outputs the compression ration. Then, it uses the `./tests/procs/max.sh` summarization program to present the results to the user. Also `./tests/procs/max.sh` is a script that computes the maximum value in a list of numbers. Both the test files and the processing/summarization programs are available in the tests package. Finally, the `14` is the maximum compression rate for the given set of files, i.e. the output of the summarization program.

Submission Instructions

Submit one source file, as required, through the iCorsi system. Add comments to your code to explain sections of the code that might not be clear. You must also add comments at the beginning of the source file to properly acknowledge any and all external sources of information you may have used, including code, suggestions, and comments from other students. If your implementation has limitations and errors you are aware of (and were unable to fix), then list those as well in the initial comments.

You may use an integrated development environment (IDE) of your choice. However, *do not submit any IDE-specific file,* such as project description files. Also, *make absolutely sure that the file you submit can be compiled and tested with a simple invocation of the standard C compiler.*