Working with processes

Laboratory 2
Duration: 2 weeks

This assignment will help us learn how to create new processes, how to have them do certain work, and how to kill them when they are no longer required. To learn these, we will take up the simple application of searching for a pattern (string) in a file.

Part I

- Clone the repository from here.
    - The repository contains a file named `file.txt`. Please do not include it in your own repository that you will maintain on github / gitea .
- Make a copy of searcher.cpp as part1_searcher.cpp .
- To only compile: `make build-part1` (creates ./part1.out)
- To compile and run: `make run-part1`
- Complete the function main() in part1_searcher.cpp to search for the given pattern in the given portion of the given file. For example, the command `./part1.out file.txt NGTNIJGK 0 67108863` is expected to search for the pattern *NGTNIJGK* in the range of characters starting from position 0 to position 67108863 in the file *file.txt* .
    - If the pattern is found, print
        [<pid>] found at <position>
      and return 1
    - Else, print
        [<pid>] didn't find
      and return 0

Part II

- Let us now have multiple processes searching for the pattern in different chunks of the file. A chunk is a contiguous part of the file. Chunks are disjoint.

  We start with one instance of part2_partitioner.out , and ask it to search in the whole file. For example, `./part2_partitioner.out somefile.txt NGTNIJGK 0 127 32` is expected to search for the pattern NGTNIJGK in the file somefile.txt whose size is 128 Bytes between position 0 and position 127. We also pass a command line argument indicating the maximum chunk size (32 in this example). If the region to be searched is greater than the maximum chunk size, then part2_partitioner.out spawns two child processes, both instances of part2_partitioner.out . In our example, left_child is told to search in the range 0 – 63 and right_child is told to search in the range 64 – 127. Maximum chunk size stays the same in both the instances. The parent process then waits for its children to complete – it does not do any work while it waits. Continuing with the example, the right_child finds that its range is greater than the maximum chunk size,

so it again spawns two children – left_child searches in the range 64 – 95 and right_child searches in the range 96 – 127. Now, in these processes, the range is not greater than the maximum chunk size. So they perform the search by spawning a single child process each: part2_searcher.out . The code for part2_searcher is the same as you had done in Part I.

- Make a copy of searcher.cpp as part2_searcher.cpp . Use the same code that you wrote in Part I.
- Make a copy of partitioner.cpp as part2_partitioner.cpp . Complete the function main(). Add print statements to indicate the occurrence of various events. Template print statements are provided as comments in partitioner.cpp . A sample expected output is given in Appendix A.
- Update the makefile to cleanly build and run your solution. Work with the same input file and pattern as in Part I. Use a maximum chunk size of 8388608 .

Part III

- You may have noticed that in this chunking approach to searching, even after the pattern is found in one chunk, the search continues in other chunks. This is wasteful. So, once a searcher process finds the pattern, we want all other processes to be killed. Achieve this by cleverly sending kill signals to processes.
- A sample expected output is given in Appendix B.
- Make a copy of searcher.cpp as part3_searcher.cpp and modify this.
- Make a copy of partitioner.cpp as part3_partitioner.cpp and modify this.
- Update the makefile to cleanly build and run your solution.

Documentation lookup

- File opening and reading in C++
- ifstream class : for all things related to file reading
- Fork: run `man fork` or check this online manual page
- Exec and its variants: run `man exec` or check this online manual page
- Wait and its variants: run `man wait` or check this online manual page
- Sending signals to processes: run `man kill` or check this online manual page
- signal handlers

Submission

- Source code for all parts with suitable makefile
- Report clearly indicating the order in which processes were created and killed in Part II and Part III. Draw the process tree.

Appendix A: Sample output for Part II

```
[174271] start position = 0 ; end position = 67108863
[174271] forked left child 174272
[174271] forked right child 174273
[174272] start position = 0 ; end position = 33554431
[174273] start position = 33554432 ; end position = 67108863
[174272] forked left child 174274
[174273] forked left child 174275
[174272] forked right child 174276
[174273] forked right child 174277
[174276] start position = 16777216 ; end position = 33554431
[174277] start position = 50331648 ; end position = 67108863
[174274] start position = 0 ; end position = 16777215
[174275] start position = 33554432 ; end position = 50331647
[174276] forked left child 174278
[174277] forked left child 174279
[174275] forked left child 174280
[174274] forked left child 174281
[174276] forked right child 174282
[174277] forked right child 174283
[174275] forked right child 174284
[174274] forked right child 174285
[174278] start position = 16777216 ; end position = 25165823
[174279] start position = 50331648 ; end position = 58720255
[174278] forked searcher child 174286
[174282] start position = 25165824 ; end position = 33554431
[174279] forked searcher child 174287
[174284] start position = 41943040 ; end position = 50331647
[174282] forked searcher child 174288
[174281] start position = 0 ; end position = 8388607
[174280] start position = 33554432 ; end position = 41943039
[174283] start position = 58720256 ; end position = 67108863
[174284] forked searcher child 174289
[174281] forked searcher child 174290
[174280] forked searcher child 174291
[174285] start position = 8388608 ; end position = 16777215
[174283] forked searcher child 174292
[174285] forked searcher child 174293
[174292] found at 64520807
[174283] searcher child returned
[174277] right child returned
[174288] didn't find
```

[174282] searcher child returned
[174276] right child returned
[174286] didn't find
[174287] didn't find
[174278] searcher child returned
[174279] searcher child returned
[174291] didn't find
[174277] left child returned
[174276] left child returned
[174280] searcher child returned
[174275] left child returned
[174273] right child returned
[174272] right child returned
[174293] didn't find
[174285] searcher child returned
[174274] right child returned
[174289] didn't find
[174284] searcher child returned
[174275] right child returned
[174273] left child returned
[174271] right child returned
[174290] didn't find
[174281] searcher child returned
[174274] left child returned
[174272] left child returned
[174271] left child returned

Appendix B: Sample output for Part III

```
[226345] start position = 0 ; end position = 67108863
[226345] forked left child 226346
[226345] forked right child 226347
[226346] start position = 0 ; end position = 33554431
[226346] forked left child 226348
[226346] forked right child 226349
[226347] start position = 33554432 ; end position = 67108863
[226347] forked left child 226350
[226347] forked right child 226351
[226348] start position = 0 ; end position = 16777215
[226348] forked left child 226352
[226348] forked right child 226353
[226349] start position = 16777216 ; end position = 33554431
[226350] start position = 33554432 ; end position = 50331647
[226349] forked left child 226354
[226350] forked left child 226355
[226350] forked right child 226356
[226349] forked right child 226357
[226351] start position = 50331648 ; end position = 67108863
[226351] forked left child 226358
[226351] forked right child 226359
[226353] start position = 8388608 ; end position = 16777215
[226353] forked searcher child 226360
[226352] start position = 0 ; end position = 8388607
[226354] start position = 16777216 ; end position = 25165823
[226352] forked searcher child 226361
[226356] start position = 41943040 ; end position = 50331647
[226354] forked searcher child 226362
[226356] forked searcher child 226363
[226357] start position = 25165824 ; end position = 33554431
[226357] forked searcher child 226364
[226355] start position = 33554432 ; end position = 41943039
[226355] forked searcher child 226365
[226358] start position = 50331648 ; end position = 58720255
[226359] start position = 58720256 ; end position = 67108863
[226358] forked searcher child 226366
[226359] forked searcher child 226367
[226367] found at 64520807
[226359] searcher child returned
[226351] right child returned
[226358] received SIGTERM
```

[226366] received SIGTERM
[226347] right child returned
[226350] received SIGTERM
[226355] received SIGTERM
[226356] received SIGTERM
[226365] received SIGTERM
[226363] received SIGTERM
[226345] right child returned
[226346] received SIGTERM
[226348] received SIGTERM
[226349] received SIGTERM
[226357] received SIGTERM
[226353] received SIGTERM
[226354] received SIGTERM
[226364] received SIGTERM
[226360] received SIGTERM
[226362] received SIGTERM
[226352] received SIGTERM
[226361] received SIGTERM