PII: Timing a Linux Process via Socket

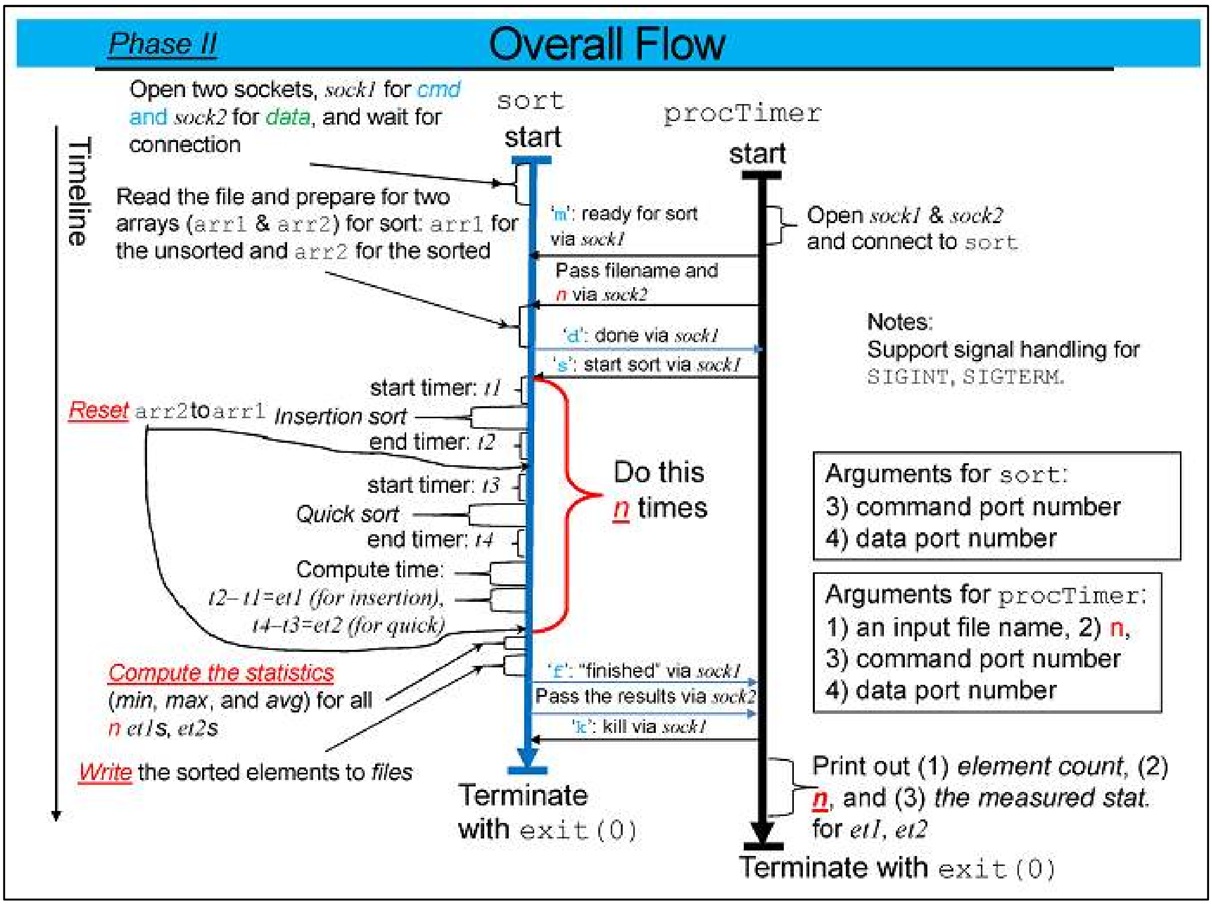
ELEC462004 System Programming – Term Project

(Instructor: Prof. Suh, Young-Kyoon)

Project Phase II: Due:11:59pm (midnight) December 6, 2017

In Phase II, you are to implement the timing program on measuring the elapsed time (ET) (Phase I) of a portion of running an algorithm **via SOCKET**. (Note that the previous phase II that was described before is replaced by this.) Your project must be implemented in **C** and be running on **Linux** (preferably Ubuntu).

Task description



Phase II
Overall Flow
Timeline
Open two sockets, sock1 for cmd and sock2 for data, and wait for connection
sort
start
procTimer
start
Read the file and prepare for two arrays (arr1 & arr2) for sort: arr1 for the unsorted and arr2 for the sorted
'm': ready for sort via sock1
Open sock1 & sock2 and connect to sort
Pass filename and n via sock2
Notes:
Support signal handling for SIGINT, SIGTERM.
'd': done via sock1
's': start sort via sock1
start timer: t1
Reset arr2 to arr1
Insertion sort
end timer: t2
start timer: t3
Quick sort
end timer: t4
Do this n times
Arguments for sort:
3) command port number
4) data port number
Compute time:
t2−t1=et1 (for insertion),
t4−t3=et2 (for quick)
Arguments for procTimer:
1) an input file name, 2) n,
3) command port number
4) data port number
Compute the statistics (min, max, and avg) for all n et1s, et2s
'f': "finished" via sock1
Pass the results via sock2
'k': kill via sock1
Write the sorted elements to files
Terminate with exit(0)
Print out (1) element count, (2) n, and (3) the measured stat. for et1, et2
Terminate with exit(0)

- Write a C program, named `sort`, to perform insertion and quick sorting tasks on a given datafile. It takes two arguments: one (*port1*) for command and another (*port2*) for data. It then opens two sockets: *sock1* at *port1* and *sock2* at *port2*. It then waits for the connection from the below program.

  `[Usage]./sort` port_number_for_command port_number_for_data

  Ex. `./sort 2400 2401`

- Write another C program, named procTimer, which prints out the *minimum*, *maximum*, and *average* execution time of a sorting task "in memory" (done in another program in the above). It takes four arguments: one for the name of a datafile for the sorting, another for how many times the sorting should be performed (***n***), and the other two for command and data ports, port1 and port2. Note that *port1* and *port2* should be the same as those two from sort. Below are the expected command-line arguments.

  [Usage]./procTimer input_data_file number_of_repetitions port_number_for_command port_number_for_data

  Ex. ./procTimer sample.txt 1000 2400 2401

  procTimer opens two sockets: *sock1* at *port1* and *sock2* at *port2*. It then connects to sort through both sockets. procTimer sends one character of '**m**' to sort via ***sock1*** to make it ready for its forthcoming sorting task. It subsequently sends (i) the name of the given input data file (e.g. sample.txt) and (ii) the number of repetitions (e.g. *n*) to sort via ***sock2***.

- sort reads the input data file (named sample.txt) and then fills all the elements in an unsorted data array (named arr1). It also creates another array (named arr2) to contain the elements to be sorted. sort initializes arr2 by copying all the elements in arr1 by the *reset* operation. It then sends one character of '**d**' to procTimer via ***sock1*** to say that it's ready for sort.

- procTimer sends one character of '**s**' to **sort** via ***sock1*** to tell it to "start" the sorting.

- Finish the sorting at **sort** as done in Phase I. Make sure that you should reset arr2 to arr1 after each repetition for each sort.

- Once **sort** finishes the measurement, it computes the statistics (the minimum, maximum, and average) for each sort. **sort** then writes the sorted elements into the files, named, insertion_sort_res.txt for insertion sort, and another, named quick_sort_res.txt for quick sort. Both files must have the same sorted data files: use 'diff' to make certain that both have no difference.

- Finally, sort sends one character of '**f**' to procTimer via ***sock1*** to say that the sorting is "finished." It then sends the measured data to **procTimer** via ***sock2***.

- procTimer shows on screen the receive data through standard out. Finally, procTimer sends one character of '**k**' to sort via ***sock1*** to kill it and then it terminates with an exit value of 0. Then sort terminates with an exit value of 0 on receipt of the character ('**k**') sent from procTimer.

- The output looks like the following:

```
// one terminal

~$./sort 2400 2401 (or '&' can be appended too.)

(Now it waits…)

Received 'm'

Received 's'

Received 'k'

(It dies now.)

~$
```

```
// another   terminal

~$./procTimer sorting_input.txt 10000 2400 2401

Received 'd'

Received 'f'

### Received the measurement statistics ###

The elements in the input data file: 100000

The number of repetitions: 10000

Insertion sort

- The minimum elapsed time: 10000 millseconds

- The maximum elapsed time: 11000 millseconds

- The average elapsed time: 11000 millseconds

Quick sort

- The minimum elapsed time: 9000 millseconds

- The maximum elapsed time: 10000 millseconds

- The average elapsed time: 9500 millseconds

~$
```

Notes

- Your code MUST use dynamic allocation. We'll check memory leak after running your code.
- Again, `sort` and `procTimer` must support signal handling on the following signals: `SIGINT` and `SIGTERM`.
    - Those two signals must be ignored.
- We provide you with an input file having 100,000 integers. See the `project_phase2` directory.
- [**IMPORTANT**!!] You must include into your submission directory (to be described shortly) your **Makefile** to produce `sort` and `procTimer`.
    - If you don't have your make file, then you get **ZERO credit** for this phase 2.

<WARNING!!>

- Suppose that your leader's studentID is '**2014123123**' and the home directory for phase 2: `PROJP2_HOME=/usr/share/fall17/elec462004/project_phase2.`

- Create your directory named the leader's student ID under `PROJP2_HOME`: specifically, the name of the submission directory will be `$PROJP2_HOME/2014123123` in the above example.

- To complete your submission, put all relevant project code into the directory that you just created above, which will be like `$PROJP2_HOME/2014123123` in the above example.

- Make sure you have your **Makefile** to generate the executables: `sort` and `procTimer`.

- If you use the wrong name of the directory or place your files in the wrong directory,

NO grade will be given.

- Since this is in-memory sorting, we'll run your code against a HUGE VOLUME of elements (at least 50K elements) stored in a data file.

- - Before you finish all the dynamic data structures, you should free them. We will check if you have any **memory leak** at the end of the program via `valgrind`. We'll give you a penalty of **5%** if the leak is detected.

QnA

Feel free to ask whatever questions you may have to me by email. Or you could visit my office by appointment.

Late Day Policy

All exercises are due at midnight on the scheduled due date. A grading penalty will be applied to late assignments. Any assignment turned in late will be penalized 50% per late day.

Plagiarism

No plagiarism will be tolerated. If the assignment is to be worked on your team own, please respect it. If the instructor determines that there are substantial similarities exceeding the likelihood of such an event, he will call the two (or more) students to explain them and possibly to take an immediate test (or assignment, at the discretion of the instructor) to determine the student's abilities related to the offending work.