Project: Timing a Linux Process

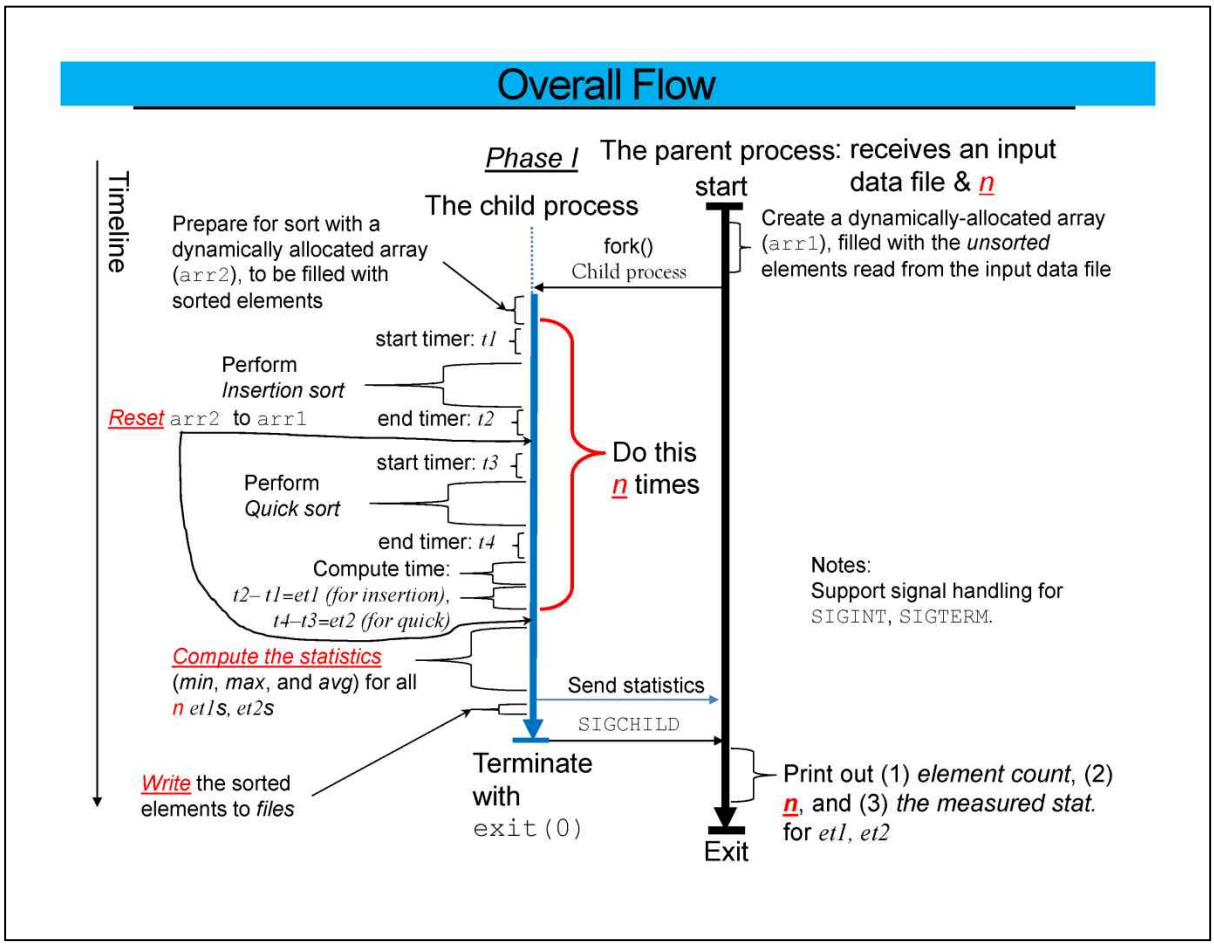
ELEC462004 System Programming – Term Project

(Instructor: Prof. Suh, Young-Kyoon)

Phase I: Due: 01:59pm November 22, 2017

In this project, you are to implement the timing program on measuring the elapsed time (ET) (Phase I) and process time (PT) (Phase II) of a portion of running an algorithm. Your project must be implemented in **C** and be running on **Linux** (preferably Ubuntu).

Task description



- Create a C program, named **`procTimer`** (the parent in the above), which prints out the *minimum*, *maximum*, and *average* execution time of a sorting task "in memory" (done in the child in the above). It takes two arguments. One is the name of a datafile for the sorting, and the other is how many times the sorting should be performed. Below are the expected command-line arguments.

  `[Usage]./procTimer` input_data_file number_of_repetitions

  (`argv[1]`: data file name, `argv[2]`: how many times the sorting is performed.)

- **procTimer** creates an unsorted data array (named `dataArr`). **procTimer** fills in the unsorted array the elements while reading from the input data file. **procTimer** also creates another array (named `sortedArr`) to contain the elements to be sorted. **procTimer** initializes `sortedArr` by copying all the elements in `dataArr`.

- **procTimer** creates a **CHILD** process (via fork).

- For each repetition, the child process measures and records the **elapsed time** of sorting the (unsorted) elements in `sortedArr`. For elapsed time (ET) measurement, use **gettimeofday** `()` to get the timestamp before and after the sorting.

- The child process must use two sorting algorithms: *insertion sort* and *quick sort*. The child should first perform insertion sort and then quick sort.

- After each sort or repetition, you **MUST RESET** `sortedArr` so that you can repeat the next measurement; that is, copy `dataArr` into `sortedArr`.

- This measurement should continue until as many times as the specified times by `argv[2]`. For instance, if `argv[2] == 1000`, then you must have recorded 1,000 ETs for insertion sort and 1,000 ETs for quick sort. After all the repetitions are completed, the child process computes the *minimum*, *maximum*, and *average* elapsed times (in integer) for *insertion sort* and *quick sort*. The times are in **MILLISECONDS** (`msec`).

- And the child produces the sorted output files for each type of sort: one 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.

- The child must send up to the parent the summary of the measured statistics.

- Once the child successfully produces the two output files and finishes sending the summary, then it dies. In the meantime, the parent MUST be waiting until the child dies so.

- The parent process then MUST print 1) *how many elements in the input data file exist* 2) *how many it repeats the measurement*, and finally 3) *the received statistics of the measured ETs for each type of sort*. The output looks like the following:

```
$./procTimer sorting_input.txt 10000

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.
- You 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_phase1` directory.

<WARNING!!>

- Suppose that your leader's studentID is '**2014123123**' and **PROJP1_HOME**=/usr/share/fall17/elec462004/project_phase1.
- Create your directory named the leader's student ID under HW5_HOME: specifically, the name of the submission directory will be **$PROJP1_HOME**/2014123123 in the above example.
- To complete your submission, put all of your files into the directory that you will have created above: that is, **$PROJP1_HOME**/2014123123 in the above example.
- 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. You can ask questions to TA by Email (kyo1363@naver.com), by visiting IT-3-308A.

Late Day Policy

All exercises are due at 13:59pm 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.