İ.T.Ü.
Computer Engineering Department

13.03.2013

BLG 312E – Computer Operating Systems

Homework 2

Submission Deadline: **03.04.2013**, **Wednesday**, until **17.00**.

- Homeworks submitted after the deadline and/or not submitted via Ninova system will NOT be accepted.
- You are required to submit all 3 homeworks. Please note that you have to achieve at least 20 out of 100 points on a homework for its submission to be accepted. Homeworks with lower grades will NOT be considered as submitted. Submitting parts of the codes provided in class will NOT be sufficient to achieve a grade of 20.
- You are expected to work individually on all exams and homeworks. All forms of collaboration are discouraged and will be treated as cheating.

Where to submit: You should submit your homework program code via Ninova system: ninova.itu.edu.tr

What to submit: You should submit your source file via Ninova system. (No additional report file is required; however, it is expected from you to include code comments in your source file)

Program: Write and test a C program that implements the described behaviors below:

- **Program input parameters:** `m, n, inputFileName, outputFileName.` These should be provided as command line arguments to the program.
- Note: `m` and `n` parameters should always satisfy the inequality: `m < n`
- `m` number of processes should be created by a process $\mathcal{P}$ using the fork system call. These processes should have associated index values: `1` … `m`. (Notice that the first process created should have an index of 1.) (The $\mathcal{P}$ process then becomes the reporter process.)
- Define an integer array (`array`) containing `n` elements in the shared memory and fill up its elements with the data read from the input file `inputFileName.` (Input file whose name is specified as the third program argument should contain `n` lines and a single integer value at each line which must be written in the corresponding location of `array`.)

- Define a **`result`** array containing **`m`** elements in the shared memory which will be filled by processes **`1`** ... **`m`** and be read by a reporter process ***𝒫.*** Each element in the result array will have two floating point fields: **`average`** and **`stddev.`**

- **Behavior of each process:**

  The **`m`** child processes will wait until the parent/reporter process ***𝒫*** creates all the resources and prepares the **`array.`** Then each process will perform the following actions: The process having an index value of **x** will read the elements from the integer array whose index values are divisible by **x** in order to calculate the average and standard deviation values for these read elements. Process$_x$ is then responsible to store these values in the **`result`** array (in the corresponding index).

  Demonstrative Example: Process associated with index 3 (**x** = 3) will read elements: **`array[0]`**, **`array[3]`**, **`array[6]`**, **`array[9]`** (assume **`n`** = 11). Then, it will calculate the average and standard deviation of these four integer elements and store these results into **`result[2].average`** and **`result[2].stddev.`**

- **Behavior of process *𝒫*:**

  The parent/reporter process ***𝒫*** first creates the resources, prepares the **`array`** data and generates **`m`** child processes. Then it will wait until all of the **`m`** processes indexed with **`1`** ... **`m`** finish their jobs. Then ***𝒫*** reads the results of worker processes from the **`result`** array and writes the elements of this array to a file whose name is given by **`outputFileName.`** When finished, the ***𝒫*** process removes all resources, kills the child processes and ends.

Test: Your program will be tested in the form:

./program.exe <**`m`** > <**`n`** > <**`inputFileName`**> <**`outputFileName`**>

Please preserve the order and meaning of the program arguments.

Please test your program with different **`m`** and **`n`** values and different integer arrays and make sure to achieve expected results.

Please check that your program correctly removes all allocated resources (e.g. shared memory locations, semaphores, and any others you have used) when it exits.