CS342 Operating Systems – Fall 2017
Project 1: Processes, IPC, Threads

Assigned: 05 Oct 2017, Thu
Due date: 19 Oct 2017, Thu, 23:55

You will do this project individually. You have to program in C under Linux. You are recommended to use the following distribution of Linux: Ubuntu 16.04 – 64 bit.

Part A: Processes and Pipes. [35 points]
Objective: Practicing process creation, process communication (by use of pipes), multi-process application development.

Write a program that uses the trapezoidal rule to approximate the integral of a function y=f(x) for a given domain interval. The program will be called integral and will take the following parameters from command line:

integral L U K N

L is the lower value of x and U is the upper value. We will take integral from x=L to x=U. N is the number of (child) processes to use to compute the integral. The program will find out the numeric integral of a function for the interval [L, U). Each child process will be responsible from an x-axis interval of length (U-L)/N. K is the number of subintervals that an interval (U-L)/N will be divided into. N can be at most 50. Minimum value is 1.

The program will create N child processes (1 to N) and each child process will compute a portion of the result. The parent will wait for the children to compute the partial results and will then add them up to find the numerical integral value. The parent will also create N pipes, one for each child, to enable the communication of children with the parent to send the results back.

*Note that when a child is created, its address space (memory) is populated from parent's (hence **initial** one way information passing can be done **once** from parent to child, including pipe descriptors, interval information child is responsible for, etc.). Since the address spaces are different, after a while later, they may contain different things in their memory and they can not access their variables. Therefore, in order to exchange information, they need to use an IPC mechanism like a pipe, or message queue, or shared memory. You will use pipes in this project.*

The child 1, for example, will compute the integral from for the interval [L, L+(U-L)/N). The child i will compute the integral for interval [L+(i-1)(U-L)/N, L+i(U-L)/N).

An interval [v,u), assigned to a child will be divided into K subintervals. The x-axis length of each subinterval will then be: (u-v)/k. The integral for a subinterval [a,b), will be found by forming a trapezoid among points a, b, f(a) and f(b) and finding the area of it. The child will do this for each subinterval of the interval assigned to it (that means it will do this K times). After a child process finds out its result, it will send the result to the parent via a pipe that was created by the parent.

The parent will wait for the children and collect the results computed by the children through reading from the respective pipes, add the results, and find out the integral. Then it will print the result to the screen.

Your program will be written in a file called integral.c. But the function for which we will compute the integral numerically will be specified in a separate file called function.c. You can not change the name of these files (function.c and

integral.c) We will provide you a Makefile to compile the program. The file function.c will just contain one function definition with the following prototype:

```
double compute_f (double x)
```

That means the name of the function is compute_f and the function will take one parameter of type double, that will be an x value. It will return a double value, which is the y value.

An example invocation of your program can be:

integral 5 35 1000 20

That means, integral will be taken form x=5 to x=25. 20 child processes will be created. Each will be assigned an x-axis interval of length 35-5/20 = 1.5 unit. Each child will divide its interval of length 1.5 into 1000 subintervals, each of length 1.5/1000 = 0.0015. For each such a trapezoid of base length 0.0015, the child will compute the area (i.e., it will do this 1000 times), and sum them up. Then it will send the result to the parent. The parent will collect such results from 20 children and will add them up and will print the result.

Part B: Threads. [35 points]
Objective: Practicing thread creation, thread communication and multi-threaded programming.

Develop the same program, but this time use threads instead of child processes. This time you will not need to use pipes. The maximum value of N can be 1000, this time. Call your program as tintegral. The rest is the same. You will use POSIX threads (Pthreads). You can find information about POSIX Pthreads in Internet.

Part C: Experiments. [30 points]

In this part you will design and conduct some experiments to evaluate the completion time of your programs. Measure the completion time of your programs for various values of N and for various functions. Report and plot the results. Try to draw conclusions. You will do this for both programs. Write a report for this part. Final report will be converted to PDF.

Submission:

Put the following files into a project directory named with your ID, tar the directory (using **tar xvf**), zip it (using **gzip**) and upload it to Moodle. For example, a student with ID 20140013 will create a directory named 20140013, will put the files there, tar and gzip the directory and upload the file. The uploaded file will be 20140013.tar.gz.

- integral.c
- tintegral.c
- function.c
- function.h: You don't need to modify this file. It is just including the prototype of the compute_f function.
- Makefile: Compiles the programs.
- report.pdf
- README.md file: Put your name and id here.

Additional Information and Clarifications:

- *Suggestion: work incrementally; step by step; implement something, test it, and when you are sure it is working move on with the next thing.*
- More **clarifications**, additional information and explanations that can be useful for you may be put to the **course website**, just near this project PDF. Check it regularly.
- The following web page contains a project skeleton. You can clone it into your local machine, if you wish. It can be a starting point.
  https://github.com/korpeoglu/cs342-fall2017-p1
  You can clone it with the following command:
  git clone https://github.com/korpeoglu/cs342fall2017_p1.git p1
  You will find the files in directory p1.