CPSC 457 Fall 2025 TUT14 – TA Zahra Arabi
Timothée Chuat UCID 30301255
timothee.chuat@ucalgary.ca

Update: GIT

GitHub

I started the first assignment on this public GitHub repository: https://github.com/Daftox/CPSC-457/
However, I forgot it was supposed to be done in a private GitLab repository and that I needed to provide you with access to it. The above GitHub repository will remain available until the assignment is graded so that you can still review the original version history, but it will be deleted afterwards.

GitLab

I have recreated the repository on GitLab with an almost identical version history (the original commit dates from the GitHub repository are included in parentheses in each commit message): https://gitlab.com/timothee.chuat/cpsc-457
The other member with Developer access is @zahra.arab
I will also use 1 flex day in order to properly resubmit on D2L and to push this updated PDF report file and a fixed version of part 1 that doesn't use pipes to both repositories.

Sorry for the inconvenience!

Assignment 1

Part 1 – Treasure

Reflection

Challenge(s)?

Instead of shared memory, I had to find another way to communicate from child to parent, because a child can't communicate a column index greater than 255. So, I chose to use a pipe, which efficiently sent the treasure's column index to the parent. I had to learn how to read from it, write into it and close it.
Update: After joining the discord server, I found out that it seems like we are not allowed to use pipes either, but the TAs hinted us that we can simply search the treasure in the parent's process starting from the row of the child that found the treasure. I used a 2D array of char to save the lines retrieved by a call to fgets(). The parent now looks for the treasure itself. I also added the verification of lines length of the text file.

Part 2 – Primality tests

Reflection

How you made sure the program does not create more than the required number of processes.

If the number of children specified on the command line is larger than the range of numbers being tested, it is truncated to match the range. As a result, the parent's for loop, which calls fork() once per iteration, will not iterate more than the allowed number of children. Each child process exits immediately after completing its task and does not create any further children.

How the work was divided among child processes.

Each child tests the same range of numbers (it is the floor of this division: ***int range_size = numbers / children***) except for the last one, which tests all the remaining numbers (in the child's code there is ***if (i == children - 1) { end_range = upper_bound; }*** to ensure it). Hence, its workload is greater than or equal to that of the other children.

How you made access to shared memory by child processes safe.

I respected the suggested memory layout for this assignment: each child process writes to a non-overlapping block of memory ***(Child i writes to indices [i * MAX_PRIMES_PER_CHILD ... (i+1)*MAX_PRIMES_PER_CHILD - 1]).*** I used a pointer with an offset for each child to write into its own chunk of memory. The max_primes_per_child variable is set to half of the child's range, except when the range is 1, in which case it is also 1. This is because we can already ignore the even numbers, which optimizes the required shared-memory size (even though this optimization was not explicitly required, it was too logically tempting to skip). As a side note, I filled the shared memory with -1 beforehand so the parent knows where to stop printing numbers before moving on to the next child's reserved memory chunk.