Marc-Andre Fichtel - 30014709
CPSC 457 - Fall 2017

Assignment 3

Due Friday, Nov 5, 2017 at 11:59pm.
Individual assignment. No group work allowed.
Weight: 6% of the final grade.

Question 1:

The formula for CPU utilization with n similar processes, each spending p% of its time performing I/O, is $1 - p^n$. So for 3 similar processes each spending 80% of its time performing I/O, the CPU utilization is $1 - 0.8^3 = 1 - 0.512 = 0.488 = 48.80\%$.

Question 2:

The Gantt chart is as follows:

1	2	1	3	3	4	1	1	1	1	1	1	1	1	5	5	5	5	5

0 1 2 3 5 6 14
19

The table can be completed as follows:

Process	Arrival	Burst	Start	Finish	Turnaround	Waiting
P1	0	10	0	14	14	4
P2	1	1	1	2	1	0
P3	3	2	3	5	2	0
P4	5	1	5	6	1	0
P5	9	5	14	19	10	5

Thus, the average wait time is $(4 + 0 + 0 + 0 + 5)/5 = 9/5 = 1.8$ seconds.

Question 3:

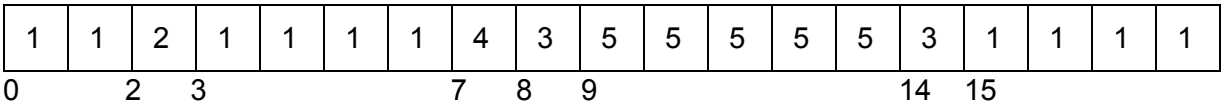
The Gantt chart is as follows:

1	1	2	1	3	1	3	4	1	1	5	1	5	1	5	1	5	1	5

0 2 3 4 5 6 7 8 10 11 12 13 14 15 16 17 18
With a quantum of 1sec, there are 17 context switches.

Question 4:

The Gantt chart is as follows:



Question 6:

The following tables were created with the time recorded when executing the programs on my personal laptop running Ubuntu.

Test file: **`medium.txt`**			
# threads	**Observed timing**	**Observed speedup compared to original**	**Expected speedup**
original program	29.879s	1.0	1.0
1	28.457s	1.05	1.0
2	22.420s	1.33	2.0
3	19.650s	1.52	3.0
4	18.006s	1.66	4.0
8	17.144s	1.74	8.0
16	16.221s	1.84	16.0

The results are as expected. The program I wrote (my **`countPrimes`** version) lets each number be be checked for primality by one dedicated thread. The workload of checking the primality of very large numbers is *not* shared between several available threads. If this was the case, the observed speedup would be closer to the expected speedup, because each thread would be doing a more equal share of the overall workload. Because **`medium.txt`** includes the most numbers out of any test input file, the speedup results are the most pronounced here.

Test file: **hard.txt**			
# threads	**Observed timing**	**Observed speedup compared to original**	**Expected speedup**
original program	9.973s	1.0	1.0
1	9.864s	1.01	1.0
2	9.794s	1.02	2.0
3	9.563s	1.04	3.0
4	9.585s	1.04	4.0
8	9.583s	1.04	8.0
16	9.883s	1.01	16.0

Test file: **hard2.txt**			
# threads	**Observed timing**	**Observed speedup compared to original**	**Expected speedup**
original program	9.732s	1.0	1.0
1	9.672s	1.0	1.0
2	9.878s	0.99	2.0
3	9.554s	1.02	3.0
4	9.751s	0.99	4.0
8	9.587s	1.02	8.0
16	9.955s	0.98	16.0

No significant speedup changed were recorded with **hard.txt** or **hard2.txt**. This is because of the already mentioned shortcoming mentioned above: My version of the program does not split the workload of checking the primality of large numbers between multiple available threads. Every number is handled by one single thread. Due to that, the run times for the two harder test input files are virtually identical to that of the original program. In so far, these outcomes are,

also, as expected. If my program was altered to let multiple available threads handle singular large numbers, then more significant speedups should theoretically be recorded.