Problem Set 7

We shall multiply, virtual shared memory style!

The handout contains a Makefile and minimal starter code. Your task is to write a program that reads and multiply matrices. Your program takes, on the command line, 3 arguments denoting the names of files containing two matrices and the number of *processes* you are expected to use to carry out your task. For instance, running

```
./matrix m1.txt m2.txt 2
```

Will execute the matrix multiplication on the matrices in the files `m1.txt` and `m2.txt` and use two sub *processes* to carry out this task.

Matrix Refresher

Remember that a matrix is a *n* by *m* 2D array with *n* rows and *m* columns. For instance, a 2x3 matrix is

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

while a 3x3 identity matrix is:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore computing A * I yields a 2x3 matrix R

$$R = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

In general, we have R = A * B with

$$A \in \mathbb{R}^{n \times m}, B \in \mathbb{R}^{m \times p}$$

$$\forall i \in 1..n, j \in 1..p : R_{ij} = \sum_{k \in 1..m} A_{ik} \cdot B_{kj}$$

Which requires $O(n^3)$ runtime (when *n* =*m* = *p*).

A multi-process approach

Given two matrices A and B, where A has *n* rows, to speed up the task with *k* processes, it is natural to *slice* the first matrix into *k* bands, in which each band contains (approximately)

$$\lfloor \frac{n}{k} \rfloor$$

rows. (If *n* does not divides equally into *k*, spread the *r* remaining rows over the first *r* bands. For instance, with n=10 and k=4, 10/4=2 with a remainder of 2 and therefore we would have 4 bands with [3,3,2,2] rows each. Then each child process is tasked with computing his part of the product for the rows entrusted to it. In this example, the first worker would multiply the first 3 rows of the first matrix by the second to obtain the first 3 rows of the result while the second worker would work on the *next* 3 rows of the first matrix and multiply them by the second to obtain the second block of three rows of the result.

This division of labor technique splits the work as evenly as possible among the *k* workers involved and leaves the multiplication algorithms essentially untouched.

Putting it together

Your program must read the two input matrices, compute the product with *k* sub processes (as specified on the command line) and print the resulting matrix on its standard output (in row major mode), one row of the matrix per line of output (do not output the size of the result, only its content). Your code should be able to handle arbitrarily sized matrices and should experience a speedup once we use more than one process.

It might be *simpler* to consider Shared Virtual Memory to do this task. Remember that you need to coordinate to know that the computation is complete. There are several ways to do this, but we would like you to make use of a **Semaphore** to carry out this task (just pretend that the wait system call is not available. How to use the semaphore beyond that is up to you!