CSE344 Homework4 Report

Yeşim Yalçın-200104004094

1-) My Solution Approach

Command Line Argument Analysis
My first action was to check if the command line arguments were inputed properly.
The data file name can be relative or absolute. N must be bigger than 1, C must be bigger than 4.
Note: It is assumed the content of the data file is valid meaning it has total of NxC 1s and 2s. However, if the data file includes anything other than 1 or 2, they are ignored.

Buffering Problem
In my main process I set the buffering of stdout to NULL to print out without buffering. Moreover, I only use write & perror to print out outputs. All of these result in proper output. No output of a thread is mixed with another one.

Creating and Using Semaphores
I create a semaphore key with ftok and use this key to create a semaphore set which has 2 semaphores. The first one represents the semaphore to track down 1s, the second one represents the semaphore to track down 2s.
In the supplier thread, I create semaphore operations called upS0 and upS1. The first one is used if a 1 is read. The second one is used if a 2 is read. These are for posting.
In consumer threads, I create a semaphore operation called waitIngreadients. This operation waits on both semaphores.

The customers wait on both semaphores atomicaly. The supplier posts the related semaphore when it reads an item. A customer wakes up if it receives enough items. Does it's operations then waits again. This is done for all items. N times for each customer.

Threads
I firstly create the supplier thread with detached state. Then I create the customer threads. Afterwards, the main process joins on all customer threads. When all items are read and delivered to the customers, the customers return and the main process wakes up from all joins.

Eventhough the main process wakes up only after all customers finish their operations, it still cannot know when the supplier thread exits. The supplier thread's job is finished after delivering the items, it will not do any crucial operations afterwards. It will only free it's resources and exit. However, it is not guaranteed that the supplier thread will exit by itself before the main process. To achieve that, I exit from the main process with pthread_exit function. This lets the supplier thread do all it's cleaning and proper exiting before the main process actually exits.

SIGINT
When SIGINT is sent to the process, a random thread catches it. I have a sigint handler which sets a sigintFlag as 1. If any process sees this flag as 1, it posts both of the semaphores then exit properly. The posting is done to let another thread know about this sigint. Each thread posts the semaphores for another one to let it know. Eventually, all threads know that sigint is

received and all of them exit. In the end main process is unblocked from join and it exits as well.

2-) The Parts I Have Completed

I have completed all the requirements of the homework and tested in various ways.