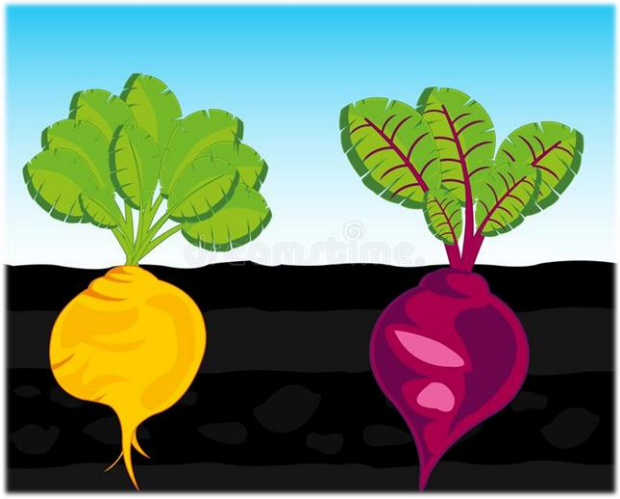
CS 3841 – Threads and Synchronization: The Root Vegetable Farm

1 Introduction



The purpose of this lab is to design, code, and test a program that uses multi-threading with synchronization mechanisms.

The root vegetable farm grows turnips and radishes for customers. Customers arrive at the farm and wait in line to be served by the farmer. Once they reach the farmer, they place their order for how many turnips and radishes they want to order and go home with their purchase. After a while they need to get more turnips and radishes, so they go back to the farm to purchase more. Customers always order the same amount of vegetables each time they reach the farmer.

The root vegetable farm has a set number of fields. Each turnip takes a set amount of time to grow. Once the turnip is done mature, the farmer's field hand puts it is put in a storage bin for the farmer to grab and sell to customers. The same is true for radishes. They have a number of fields and farm hands separated from the turnips and their own storage bin.

Fields can only grow a set amount of vegetables each season. Your job is to write a program to simulate the root vegetable farm for a single season (customers purchasing vegetables and fields growing vegetables). The simulation ends when all the fields have finished producing for the season. Any customers waiting to purchase vegetables are immediately sent home and the simulation ends.

2 References

You will be using threads for this lab. ***Each*** customer, turnip field, and radish field ***must*** be a ***separate thread***. You will also have to keep track of the turnips and radishes in the storage bins. There is one bin for each type of vegetable, however the farmer has extra-extra-large bins that never run out of space.

The following man pages might be helpful for review:

- `man pthreads`
- `man sem_overview`
- `man usleep`
- `man sem_wait`
- `man sem_post`
- `man pthread_mutex_lock`
- `man pthread_mutex_unlock`
- `man pthread_cond_wait`
- `man pthread_cond_signal`

In order to view the man pages for the pthread library, they have to be installed on your OS image. You can do that by running the command:

```
sudo apt-get install manpages-posix manpages-posix-dev
```

3 The Exercise

Your program must simulate the farm for a single season. Simulation parameters will be given to you in a text file on the command line like previous labs. The format for the text file will be as follows:

- The first line will contain three integers
  - The first is the number of turnip fields
  - The second is the amount of time it takes for a turnip to grow (in microseconds)
  - The third is the number of turnips each turnip field produces for the season
- The second line will contain three integers
  - The first is the number of radish fields
  - The second is the amount of time it takes for a radish to grow (in microseconds)
  - The third is the number of radishes each radish field produces for the season.
- The third line contains a single integer - the number of customers
- The rest of the file contains lines of 3 integers (one for each customer)
  - The first is the number of turnips the customer buys when they reach the farmer
  - The second is the number of radishes the customer buys when the reach the farmer
  - The third is the amount of time the customer waits at home after ordering before they get back in line to buy more turnips and radishes (in microseconds)

All numbers in the line will be separated by spaces. Here is an example:

```
4 1000 100
3 2000 125
3
1 2 500
3 4 1000
1 0 200
```

The parameters in this file says to simulate the root vegetable farm with 4 turnip fields where turnips take 1000 microseconds to grow and each field grows 100 turnips a season; 3 radish fields where radishes take 2000 microseconds to grow and each field grows 50 radishes a season. There are 3 customers: the first purchases 1 turnip and 2 radishes at a time and waits 500 microseconds between orders; the second purchases 3 turnips and 4 radishes at a time and waits for 1000 microseconds between orders; the third orders 1 turnip and radishes each time and waits 200 microseconds between orders. This simulation has a total of 10 threads (4 for the turnip fields, 3 for the radish fields, and 3 for customers).

NOTE: a single field can produce only a single vegetable at a time. In the example above, one turnip field produces a single turnip every 1000 microseconds.

The simulation will always have at least one turnip field, one radish field, and one customer. Furthermore, the customer will buy at least 1 turnip or 1 radish. They may by 0 of one, however (customer 3 above).

When turnips and radishes are done growing, the farm hand immediately places them into the storage bin (in zero time). Similarly, when customer orders their vegetables, they immediately receive from the storage bin (in zero time). Customers can travel between their home and the farm in zero time as well (they are quite speedy).

All customers initially arrive at the same time and form a line (in any order). After their order is filled and they are done waiting, customers must go to the end of the line and wait for all customers ahead of them to be served before they can order.

Your simulation must ensure that a customer waits when they reach the front of the line if there are not enough vegetables in the storage bins. This holds up the line, and nothing can be purchased until enough turnips and radishes are grown. Do not let customers skip others in line even if a customer later in line can have their order satisfied.

You will have to keep a record of how many times a customer had their order filled. This will need to be printed out at the end of your simulation.

You will have to keep track of how full each storage bin got. In other words, keep track of the largest number of turnips and radishes that were ever in the storage bin. Print this out at the end of your simulated day.

Customers can't purchase vegetables that haven't been grown. Make sure your output makes sense. For example, if your simulation only has a single turnip field that grows 100 turnips a season and a single customer that buys 2 turnips at a time, their order can't be filled more than 50 times!

Here is a sample output for the above example parameter file:

```
The Root Vegetable Farm
Turnip Fields - Number:4 Time:1000 Total:100
Radish Fields - Number:3 Time:2000 Total:125
Customer 0 - Turnips:1 Radishes:2 Wait:500
Customer 1 - Turnips:3 Radishes:4 Wait:1000
Customer 2 - Turnips:1 Radishes:0 Wait:200

Simulation finished:
Max turnips in the turnip bin: 217
Max radishes int the radish bin: 3
Customer 0 got their order filled 65 times
Customer 1 got their order filled 61 times
Customer 2 got their order filled 110 times
```

4 Development Tips

- The parameter files can be read in easily with fscanf, just like the previous labs
- As the number of threads is dynamic (specified in the parameter file), you will need to malloc a space for storing statistics. Don't forget to free your mallocs!
- You will need to use concurrency mechanisms for synchronization.
    - Semaphores store their count; you can use this to count items in a storage bin.
    - Mutex locks store their owner and allow only one thread to obtain the lock.
    - Condition variables combine a mutex lock with a condition that is either true or false.
    - Do NOT use empty 'while' loops to wait for vegetables.
- All threads for the simulation should start more or less at the same time. Think about how you can do this. Check out the concurrency examples from class for tips.
- Don't forget to synchronize access to the storage bins. All grown vegetables should end up in their bin and shouldn't be lost due to race conditions. Furthermore, only a single customer can buy from the farm at a time. Think mutual exclusion.
- You will need to figure out how to stop your simulation when the day is over. This may take some thought to figure out how to do it correctly to avoid memory leaks.
- Don't forget about using gdb to help with debugging (compile with -g to get additional debug symbols)
    - NOTE: when using gdb it is helpful to compile your program with additional debug symbols included. These allow gdb to show you more information when running commands like backtrace (bt). To compile with additional debug symbols use the -g flag on gcc. For example:

      ```
      gcc -Wall -g -o myprog mysourcefile1.c mysourcefile2.c mysourcefile3.c
      ```

- Using valgrind will be helpful in this lab to ensure you do not have any memory leaks.

- The parameter passed to a thread function is a void pointer. Make sure that if you are passing the address of variables on the stack that the value is still good when the thread needs it. Pointers to local variables declared on the stack will become invalid when the stack frame is freed when the function returns.
- When compiling your program you will need to include an additional compiler flag: -pthread. This tells the compiler to use the pthreas library. The in-class examples use this in the Makefile so you can see how to use it there as well. For example:

  ```
  gcc -Wall -g -o myprog -pthread mysourcefile1.c mysourcefile2.c mysourcefile3.c
  ```

  NOTE: the -pthread flag can be anywhere on the command line. It does NOT need to be the last thing on the command line.

5 Testing and Debugging

Use the example above to get you started with testing your lab. Creating a file with simpler parameters (fewer customers, turnip fields, and radish fields) might help you debug. You are also required to create some of your own parameter files to show that you adequately tested your program. Include those in your submission. Don't forget to test large simulations.

6 Deliverables

You will need to include all your source files, test case parameter files, and any other resources you used to complete lab. Please don't just google search for a solution, but if you do use google for any help, include a description and URL of what you used to help you.

You should ensure that this program compiles without warning (-Wall and -Wextra) prior to submitting. Using gnu90 standard is allowed (gcc default). If your implementation requires C99 or gnu99, please make that clear in the comments.

Also include a text, doc, or pdf file with a report that provides the following:

- Introduction – describe the lab in your own words
- Design – describe your thought processes in creating your solution
- Build Instructions – what must be executed (gcc, make, etc.) to build your program
- Analysis – Program needs to print out, for each customer, how many times they had their order filled. Use this information to evaluate the fairness of your implementation.
  - Is your implementation fair given the parameters for the amount of vegetables a customer orders and how long they wait between ordering?
  - What is "fair" in your own words?
  - In this implementation, when a customer arrives at the front of the line, if there are not enough vegetables, they hold up the entire line and wait until their vegetables have grown. What if instead, when a customer reached the front of the line if not enough vegetables were ready, they skip their ordering opportunity and immediately return to the end of the line without ordering? Given your definition of fair, is this behavior fair? Explain why or why not?
  - Deli counters, butcher stores, restaurants, etc. use a number system where a customer orders immediately, receives a number, and waits while their order is prepared. How would you change your implementation (in terms of how semaphores are used) to handle this scenario?
  - The farm storage bins have limitless capacity. Does this make sense in practice? How would your code need to be modified to handle a bin with a capacity limit?
- Conclusion – What specifically was challenging about this lab? What did you like about it? What could we do to improve it for others?

Prepare a zip file with all submitted files and upload the file to Canvas per your instructor’s instructions.