Assignment 4: Food Craft, Rust Galaxy

Dr. William Kreahling

April 21, 2021

Overview

For this assignment you will write a multi-threaded application that performs a deadlock free implementation of the following problem:

> You are part of the *Food Craft* mining expedition that has landed on the planet Banquatari in the Rust galaxy to mine food stuffs to sell to a galaxy teaming with explorers and colonists. The mining expedition is a large operation. The teams have been divided into different groups (all of which run concurrently):
>
> - Bread miners: This groups of miners have set up camp in the Wonder Valley and are mining bread. This group has an unlimited amount of bread.
> - Cheese miners: This group of miners have set up camp in the Colby Plains and mine cheese from the land. This group has an unlimited amount of cheese.
> - Bologna miners: This group of miners have set up camp in Os'cermayer Canyon and mine the naturally occurring deposits of bologna. This group has an unlimited supply of bologna.
> - The Foreman: This person coordinates supply drops at the central docks, a location equidistant from all three groups of miners.
> - The Messengers: These groups monitors the situation, move supplies, perform tallies, and send messages to the miners.
>
> *Food Craft* has set up the following supply structure so that every group of miners can make sandwiches to feed the hungry miners.
>
> - The foreman will, at random intervals, drop off two of the three types of supplies at the docks (cheese, bologna, and bread – the two sets of supplies will always distinct from one another). For example, the foreman might deliver bread and cheese during one supply drop. After delivery, the foreman *waits* to drop more supplies until a group of miners have signaled him that they have enough food to eat. The foreman sends up flares, visible at the docks, that indicate the type of supplies delivered. There is a different flare for each type of supply.
> - The group of miners that needs the complimentary ingredients will try and get them from the docks and make sandwiches for themselves to eat. Once they have all the supplies they need, they will signal the foreman that he can drop more supplies. This group *waits* until informed that supplies are ready. Supplies cannot be shared among different miner's groups.
> - The messengers can carry messages and supplies to the miners, but not to the foreman. Messengers *wait* until they are needed.

Your task is to set up a concurrent program that does not have deadlock, livelock, starvation, or explicit polling. **You may not change the logic of the foreman, and there can only be one foreman process.** You may use Mutexes and Condvars to accomplish this, processes should *wait, or wait_while* using the condition variables.

- All thread processes must be assigned a unique ID and run concurrently.
- There should be no race conditions, deadlock, livelock, starvation, or explicit polling!
- When not doing work, threads should be block/paused/asleep.
- Style and OO design will be graded.

- Processes should only be *awake* when they have useful work to do.

Your Driver:

- Your driver should accept two command line arguments.

  1. The amount of time before the driver ends the distribution operation If the amount is 0 or negative, the operation continues until the main thread is killed (ctrl-C), which in turn kills all the child threads.

  2. 'T' or 'F'. 'T' indicates to append output from the threads to a single file, 'F' means all output should be printed to the screen. If you are logging output to a file, then the time until the operation is stopped must be a positive value. Call the output file *log.txt*. **All threads are writing to the same file, concurrently.**

- There should be NO zombies!

Foreman class:

- The foreman should randomly pick two sets of supplies (they will never deliver two of the same type at one time), send a *distinct* 'signal' for each type of supply delivered to the docks. (i.e., there is one that indicates bread is delivered, another to indicate cheese is delivered, and a seperate signal to indicate bologna is delivered.)

  The foreman will not deliver any more supplies until all the previously delivered supplies have been picked up from the docks. The Foreman has an unlimited supply of bread, cheese, and bologna collected from all the miners on the planet (you are not modeling any type of delivery to the foreman).

- You may **NOT** change the logic of the foreman. **The MUST use a distinct signal for each type of supply**.

Miners class:

- Each miner class must wait until all supplies they need to make food are available, get the supplies and call a `make_food` method (that takes a random amount of time to complete), then calls a `eat_food` method (that takes a random amount of time to complete). Other processes are able to gather supplies and make and eat, while other groups of miners are making their sandwiches or eating. *The make and eat methods should indicate what is going on by printing.*

- Miners, when they have all the food they need to make food, they will signal the foreman that the docks are clear.

- Miners on this planet are a surly bunch and do not communicate with each other.

Messenger class:

- The `messenger` class listens for signals and sends signals, carrying supplies if needed.

- Messengers are not allowed to *send* messages or supplies to the foreman.

Shared Memory;

Docks class:

- Where the food is stored (this can be represented as strings).

- This class should follow correct design.

Mutex and Condvars:

- You will need to create and share signals (Mutexes and Convars) between processes. For example the miners signal the foreman when they have food, so there will need to be a shared signal(s) between the miners and the forman.

Other classes:

- You may design and add other classes/enums as long as they do not violate the rules of the program (or good OO design).

Details

Make sure you test your program in a way to prove to yourself (and me) that your food distribution program is running correctly and that the logging is happening in a thread safe manner.

You may not use atomics for any communication between miners, the foreman, and/or messengers.

Once threads have started they should not terminate until the time limit has expired or CTRL-C has been input.

You cannot redirect stdout to accomplish the file logging.

You are using Mutex and Condvar from the std::sync crates and threads from std::thread. You may not use any other crates for shared memory and threading.

main, miners, messsengers, docks, should all be in their own files.

1 Notes on Collaboration

You may work in teams of up to two on this assignment. Note that all members of a team will receive the same grade on the assignment. *I highly recommend teams for this assignment.*

2 Hand-In Instructions

This assignment is due by 11:59 PM on Friday May 7th. A single version version of the assignment is due from each team. Submit all source files associated with the program as well any instructions on compiling. To submit your files, use the *handin* command on agora. Handin works as follows:

```
handin.<course#>.<section#>  <assignment#>  <files>
```

Therefore, to submit this assignment, you must use the following command (assuming each of the files is in your current working directory):

```
handin.370.1  4  <list of files>
```

OR

```
handin.370.1  4  <rust_craft.tbz>
```

please remove your target directory before compressing and submitting