New York University Abu Dhabi
CS-UH 3010
Programming Assignment 3
Due: April 25th, 2021

Preamble:
In this assignment, you will write independent programs that run concurrently, get synchronized at various points, and their collaborative work empowers the operation of the Expo'21 Salad (E21S) production for visitors to the site of the event.

Each E21S salad consists of approximately 80 grams of tomatoes, 50 grams of green peppers and approximately 30 grams of onions.

You will have to implement the source code for 2–types of processes (or *templates*), each of which has a distinct role in the E21S salad creation process. The 2 types of processes are:

- chef: manages the salad creation for all *Expo'21* visitors. The chef supplies ingredients and coordinates the rapid and timely making of E21S salads with the help of saladmakers,
- saladmakers: await for the appropriate ingredients to be delivered on their workbench by the chef, receive items not available to them, weight all 3 ingredients before the latter are put on their cutting boards, proceed to chop the vegetables, and ultimately, create the E21S salad.

We assume that there are 1 chef and 3 saladmaker-s who work *concurrently*. The main goal of the assignment is to create independent processes that "carry out" (i.e., *simulate*) the work of salad-making at Expo'21.

Overall in this project, you will have to:

- using *(POSIX) Semaphores* to implement the required synchronization conditions and have clients, cashiers and server coordinate their work invoking only appropriate *P()* and *V()* primitives,
- creating pertinent structure(s) on a *shared memory segment* so that statistics for the salads produced can be compiled,
- having all participants, namely the chef and the 3 saladmaker processes *attach* to the above shared segment so that they can conveniently access and possibly manipulate the content of the main-memory resident objects(s).

Procedural Matters:
◇ Your program is to be written in `C/C++` and *must run* on `NYUAD's Ubuntu` server `bled.abudhabi.nyu.edu`.
◇ You have to first submit your project via `newclasses.nyu.edu` and subsequently, *demonstrate* your work.
◇ Dena Ahmed (`daa4-AT+nyu.edu`) will be responsible for answering questions as well as reviewing and marking the assignment in coordination and collaboration with the instructor.

Project Description:
Figure 1 depicts the overall process of creating salads at Expo'21. When the groups of the 4 processes is put into motion, a specific maximum number of salads is to be prepared. Every saladmaker has to have at his disposal all 3 ingredients in appropriate quantities: 2 to 3 small tomatoes weighting about 100 grams, 1 to 2 medium size green-peppers of about 80 grams, and 1 to 2 small onions of approximately 60 grams.

The chef has at her disposal all 3 key ingredients at all times and all 3 types of vegetables are placed in containers/baskets next to her. These containers are being continuously replenished by other employees not shown in the Figure 1 who work "behind the scenes". Every saladmaker has at his disposal only 1 of the 3 required ingredients in a basket placed next to him that is also continuously refilled by other workers. In this respect, every saladmaker awaits for his 2 *missing vegetables* to be placed on the bench so that he can have a chance to take them, do his independent weighting (the weight of the vegetables offered by chef do

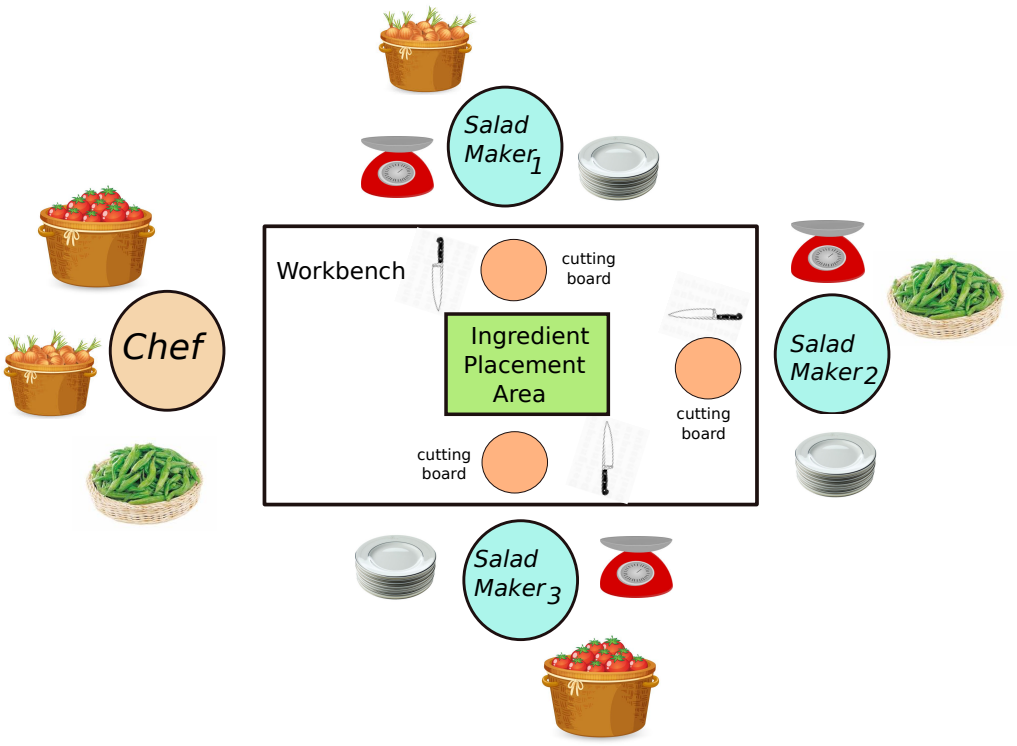

Figure 1: Collaborative Creation of the E21SSalads

not exactly represent the ideal weight specification!) and finally, chop along with the ingredient he has so as to fix a salad at a time. The entire preparation for a single salad takes a random number of seconds at the end of which the outcome is surrender to an always-available server for delivery to the next visitor.

Figure 1 shows how the entire process of E21S unfolds. There is a number of constraints that your solution should comply with:

1. The saladmaker-s always have at their disposal 1 type of vegetable as well as a plate to put on the salad as soon as the preparation completes.
2. Every saladmaker grabs the vegetable he has available in his basket and once weights it places it to his cutting-board.
3. He also waits so that the 2 ingredients he is missing become available on the workbench. As soon as these 2 become available, he picks then up, carries out weighting, places them on his board and carries on with the chopping and the making of a salad. This procedure takes some amount of time and once is complete, the saladmaker surrenders the plate to a server who always remains stand by and ready for delivery.
4. The chef has at her disposal 3 baskets each with a specific type of vegetable. However, every time the chef grabs only 2 types of required vegetables (for she has only two hands!). The specific combination of the 2 ingredients can assist only the saladmaker who is missing the vegetables in question. The chef appropriately "notifies" the corresponding saladmaker and waits until the materials have been picked. Only then, she goes on for another round of randomly grabbing of 2 ingredients to supply the next saladmaker and so on. For example, if chef picks tomatoes and onions, she then has to coordinate/wait for saladmaker_2.
5. The chef continues her work until the specific maximum number of salads have been prepared.

What needs to be done:
The independent programs chef and saladmaker-s have to be synchronized with *POSIX P()/V()* calls and have to have access to any objects/structures required in *shared memory*.

You have the freedom to adopt a (limited) number of semaphores as well as auxiliary objects that are to be placed in shared-memory so that your concurrent program can function correctly.

It would be probably a good idea that the chef program creates the *shared-memory segment* and appropriately initializes there pertinent objects/variables. The `ID` of this shared-memory segment has to become known to the executable of saladmaker-s as a command line parameter. The chef can also initialize the semaphores to appropriate initial values.

In shared memory, a number of variables pertinent to the work of each individual saladmaker has to be maintained. The variables store statistics such as the number of salads produced for each maker and the sum-weight of the individual ingredients that every saladmaker used in his work.

Your programs should create output that is *readily understood* and can help in verifying the correctness of the operations of all processes involved.

You can invoke your programs from either different `tty`s by typing in 4 specific command lines or write a shell-program that creates the respective processes with `fork()/exec*()`s.

Every time a saladmaker receives ingredients, there might be some deviation from the ideal weight specification for every ingredient in the E21S salad. For instance, if chef places on the workbench (whole) tomatoes their weight might be in the range of $[0.8 * 80..1.2 * 0.80]$ grams. To avoid food-waste, the saladmaker chaps all the provided quantity.

When your programs terminate, it is imperative that shared memory segments and semaphores are *purged*. The purging of such resources is a *must* for otherwise, these resources may get ultimately depleted and the kernel may not be able to provide for future needs.

The End Output of the Processes :
The following statistics have to be produced by the chef before this process terminates:

1. Number of Salads produced.
2. For each saladmaker:
   - The sum weight of each ingredient that used used overall by every saladmaker.
   - The total time that each saladmaker spent working on his salads.
   - The total time every saladmaker was waiting for pairs of ingredients to be delivered to him by the chef.
   - A temporal log that reveal the timeline operation of each saladmaker.
3. Listings of time periods that 2 or more saladmaker-s were busy at the same time (working in parallel).

How your executables should be invoked:
chef should be invoked as follows from a `tty`:
./chef -n numOfSalads -m cheftime
where

- chef is the executable for the manager
- the flag `-n numofSalads` indicates the number of salads that have to be created before the processes stop their execution, and
- the flag `-m cheftime` in the max time period that the chef takes a short break between serving successive pairs of ingredients. The actual time duration is randomly selected from the range: [0.50*`cheftime`..`cheftime`]

The program `saladmaker` can be invoked as follows:
`./saladmaker -m salmkrtime -s shmid`
where

- `saladmaker` is the executable that realizes the corresponding process,
- the flag `-m` designates the maximum time that a saladmaker spends to prepaper a salad once he/she has all material available. The actual time duration for a single salad preparation is randomly selected from the range: [0.80*`salmkrtime`..`salmkrtime`], and
- the flag `-s shmid` offers the identifier that the shared memory segments has and in which all necessary globally accessible objects have been placed. This identifier is likely produced by the chef program.

The order with the which the various flags appear is not predetermined. Obviously, you can use any additional flags and/or auxiliary variables you deem required for your programs to properly function.

In general, it would be a good idea to build a logging mechanism possibly in the form of an *append-only* file that records the work of each player in this group of processes as things develop over time. In this logging mechanism, the activities of each process so far are recorded in a way that is easily understood by anyone who wants to ascertain the *correct concurrent execution* of your programs.

What you Need to Submit:

1. A directory that contains all your work including source, header, Makefile, a readme file, etc.
2. A short write-up about the design choices you have taken in order to design your program(s); 1-2 pages in ASCII-text would be more than enough.
3. All the above should be submitted in the form of "flat" `tar` or `zip` file bearing your name (for instance `YaserFarhat-Proj3.tar`).
4. Submit the above tar/zip-ball using *NYUclasses*.

Grading Scheme:

Aspect of Programming Assignment Marked	Percentage of Grade (0–100)
Quality in Code Organization & Modularity	15%
Addressing All Synchronization Requirements	35%
Correct Concurrent Execution for Processes	30%
Use of Makefile & Separate Compilation	10%
Correct Use of In-line Parameters	5%
Well Commented Code	5%

Miscellaneous & Noteworthy Points:

1. You have to use *separate compilation* in the development of your program.
2. If you decide to use C++ instead of plain C, you *should not* use STL/templates.
3. Although it is understood that you may exchange ideas on how to make things work and seek advice from fellow students, sharing of code is *not allowed*.
4. If you use code that is not your own, you will have to provide appropriate citation (i.e., explicitly state where you found the code). Otherwise, plagiarism questions may ensue. Regardless, you have to fully understand what such pieces of code do and how they work.
5. The project is to be done individually as the syllabus indicates and should run on the LINUX server: `bled.abudhabi.nyu.edu`

6. You can access the above server through `ssh` using port `4410`; for example at prompt, you can issue: "`ssh yourNetID@bled.abudhabi.nyu.edu -p 4410`"

7. There is ongoing Unix support through the *Unix Lab* Saadiyat [Uni21].

References

[Uni21] NYUAD UnixLab. `https://unixlabnyuad.github.io/`. With Active Virtual Zoom Session, 2021.