Assignment 4
The Maach-Dal-Bhaat Problem

Introduction:

In this assignment you will be introduced with the concepts of shared memory, system V IPC, multiple process & thread management.

Climax (Objective):

To get acquainted with communication between processes and synchronization between them. To efficiently share resources between different processes and avoid deadlocks.

The Story:

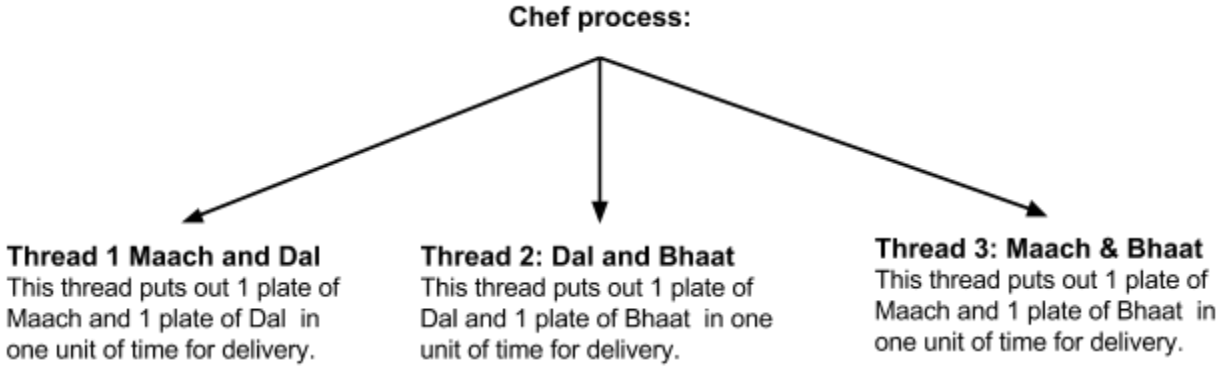
Hilbert's Grand hotel has opened its new branch in kolkata. It is well known that the Hilbert's hotel has countably infinite number of rooms, and each room has one guest checked-in. Everything was going merry, until all the guests have decided to try the special delicacy of the hotel, "The Maach-Dal-Bhaat". The dish essentially contains 3 items Rice, Fish Curry and Dal. Now this has put the restaurant manager in a dilemma, which is explained as follows. All the guests have exactly one among the three items delivered to them (i.e. one third of the guests have got Maach, another one third have got Dal and rest one third have got Bhaat.). But None of the guests will start eating the dish, until they get the other two dishes as well. As the hotel has recently started functioning, the kitchen hasn't come in a full swing. The chef prepares exactly two of the three items at once.

This is nothing less than the doom's day for the manager. The manager has decided to take up the matter into own hands and resolve it. The manager ordered to open up 3 service rooms to solve this issue. He has also created 3 separate lists of customers, such that they are divided based on what item they have in their hand (remember they have already been delivered one item among the three as in previous paragraph). Once the lists were made, each room had one list pasted on its wall such that Service Room 1 has names of guests who already have maach, Service room 2 has guests with only daal and so on. Depending on the two items that are ready, The items are delivered to the appropriate service room. The service room is then allocated to the first person whose name is in the list. It is guaranteed that the rooms will never be empty, as it will be occupied one or other customer as per the linear order in the list for a given room.

Actors in the play (Processes)

The chef process: The generator of items

The chef process is integral to the hotel (read as OS). So no modification in the design of chef process is allowed. The role of this process is to cook and then put the three resources, macch, dal and bhaat. But the chef process can cook and deliver only two items at once. We model the scenario with the help of three threads.



Note that, the three threads use a binary semaphore called ‘chef_mutex’ so as to avoid race conditions among them. We model the items for the dish as **3 separate binary semaphores namely maachSem, daalSem, bhaatSem**. Each thread will use signal operation to notify that the corresponding item is already cooked.

The waiter process: Serving the three items.

The waiter process will make sure that the items are served to the proper service room such that, one customer can have his full meal for any given allocation. This process also will have three threads. Each of the thread will be waiting by the semaphore wait operation corresponding to one of the three items. If chef prepares one particular item (semaphore signalled), then a corresponding thread can resume its operation which was waiting by virtue of the semaphore

The Side actors (Variables in the process)
WaiterMutex - A binary semaphore to avoid race condition between the three waiter threads.

Three counters, one for each item to keep track of how many items are already prepared by chef and yet to be delivered.

Three semaphores maachSem, daalSem, bhaatSem shared with the chef process.

Three new semaphores, maachEnable, bhaatEnable, dalEnable one each for synchronizing with each of the service room processes. Now whenever two resources are available, we can allocate those resources to one of the service room processes, as it already has the other resource. We use these 3 semaphores to perform the same.

Reception process:

The reception process is responsible for the following

- Creates 3 service room processes - Fork and then execlp to call the binary for service room (mandatory)
- Creates a shared Memory for “Service_Room_Occupancy_Table”, and then share a portion to each of the service room processes. (Described in the data structure description)
- On user request, prints the “Guestbook” as a formatted table

Service Room process:

We assume that there are three rooms where each room is always occupied by one guest who has a particular item with him/her. Please note that a room will contain all the names of people who have a particular item say, service room 1 will have all the names of people who have maach with them.

The side actors:

guestList - An array to contain names of guests. Fill it with a fixed number of names in the program code itself (or read from a file). If all the names are used up, reuse the list. consider the list to be cyclic.

Pointer to relevant part of “Service_Room_Occupancy_Table” - Pointer to the relevant row allocated to store the information about the room’s current occupancy

Pointer to “Guestbook” shared Memory

Semaphore - One among the three semaphores maachEnable, bhaatEnable, dalEnable in each of the processes, shared with the waiter process.

The process should perform the following actions

- The process will be in waiting, by virtue of one of the semaphores that it shares with waiter process, and when signalled by the appropriate waiter thread it resumes its eat method
- Whenever a guest finishes eat method, the details should be added in the ‘Guestbook’ and then the row in “Service_Room_Occupancy_Table” should be updated with the next name and arrival time.

Art Direction (Data structure):

To see the status of the problem.

Service_Room_Occupancy_Table:

This is shared between the three service room processes and the waiter process. This table has 3 rows. The beginning address for each row is shared with each service room process pointing. Whenever a new person is allotted the room, the table is updated. The waiter process should have a method to print and display the formatted table. The attributes of the table are as shown in the table sample

Guest Name	Room Entry time	Room_ID
Rajan	1.00 pm	1
Yadav	1.02 PM	3
Rohit	2.00pm	2

Guest_Book

This is a table that is updated by the service room process when the corresponding guest leaves the service room. This should be retrievable by the reception process and shown to the user on display as a formatted table. The attributes of the table are as shown in the sample

Guest Name	Room Entry time	Room Exit time	Comment	Room_ID
Rajan	1.00 pm	2.00 PM	Good	1
Yadav	1.02 PM	3.00 PM	Delicious	3
Rohit	2.00pm	2.30 PM	Not good	2

Script (Pun intended)

Chef Process

Chef Semaphores initialisation

```
1. chefMem  = Semaphore(1)  // 1 ⇒ initialized to up.
2. maachSem = Semaphore(0)  // 0 ⇒ initialized to down.
3. dalSem   = Semaphore(0)
4. bhaatSem = Semaphore(0)
```

Sample for Thread 1

```
chefMutex.wait()
     maachSem.signal()
     dalSem.signal() /* 1. A unit of each maach and dal has
          * been cooked.
          * 2. The Chef doesn't maintain any counting of the
          * of number of units cooked. */
chefMutex.signal()
```

Waiter Process

```
int countDal, countBhaat, countMaach; /* Each one of them
     *  represents count of an item. Note that they are not
     *  touched by the Chef process. */
```

sample for thread 1 in waiter process:

```
maachSem.wait() // maach was cooked and picked up by this
waiter.
WaiterMutex.wait() // For among there are three competing
threads.

if countDal > 0:
  /* Both Dal and maach are available.
   * Please analyse why the maach counter is not decremented. */
  countDal--
  BhaatEnable.signal()
else if countBhaat > 0:
  countBhaat--
  DalEnable.signal()
else:
  countMaach++
WaiterMutex.signal()
```

Service room (having Bhaat in abundance):

```
while(1):
     bhaatEnable.wait() //signaled by the waiter process
     guest_eat()    // guest is eating
     updateGuest_Book()// log the current guest
     updateService_Room_Occupancy_Table(guestList)/* pick new
                                        guest from the list*/
     bhaatEnable.singal()
```

NOTE: Use suitable semaphores (not mentioned in the assignment) wherever required for synchronization in accessing shared memory.

Anti-Climax (Deliverables)

For the submission you are required to submit your codes for:

1) Chef Process
    a) Chef Process with mutex

b) Chef Process without mutex

2) Waiter Process
3) Service Room Process

For 1a and 1b, the same code for 2 and 3 should be used.

4) A makefile to compile and run the codes.
5) Doxy File to generate doxygen documentation
6) A log file should be generated, about which is detailed below. You should use the log file yourself to analyse the problems that can happen with the current program design in both the scenarios.
    a) From the log, you should report the number of resources that were generated by the chef, but not properly collected by the waiter process.
    b) You should separately report for the scenarios when mutex is used in chef and when mutex is not used as well.

7) The report file, of which the details to be reported are given in “report.txt”. For obtaining the observations as required in report.txt, you may do the same manually or use a programming language of your choice.

The Chef process (both 1a and 1b, whichever is running) should share a common log file, as mentioned in point number 6. with the waiter process. The file should be used to update the resources generated (in case of chef), and the resources that are received by the waiter. The log file should essentially display the following contents, as shown in the following table. You may save the file as a “CSV” file, where each column is separated by a comma, and each row is separated by a newline. A sample of desired log file is uploaded in moodle.

Process Name	Thread number	Item	Time (micorseconds)	Service Room invoked
Chef	2	Maach	20	0
Chef	2	Dal	22	0
Waiter	1	Maach	24	100
Waiter	2	Dal	25	1

In the last column, Chef will always output a zero as chef never calls a guest. In case of waiter, if it is not able to invoke any service room, then it should output a 100, else it

should output the corresponding service room number, which is the ROOM_ID, used in GUEST_ROOM data structure. For time you may give the time in nanoseconds or microseconds relative to the time the chef process has started.

The value to be put in the corresponding sleep methods will be given as command line input. The entire program should be compiled and run with a makefile

make withMutex INPUT=5 6 7 10 11 12

Here, you should compile and run 1a, 2 and 3, where each thread of the chef (in 1a) will sleep for 5 , 6 and 7 seconds respectively, after each time the corresponding thread is invoked. The Waiter process threads will sleep for 10, 11 and 12 seconds respectively after each time a thread is invoked

make withoutMutex INPUT =5 6 7 10 11 12

Here, you should compile and run 1b, 2 and 3 and the duration of sleep is as discussed above.

Console Output

When your code executes, the following options should be available on the screen:

1. Print Guest_Book
2. Print Service_Room_Occupancy_Table
3. Exit

Option 1 and 2 will print the current status of the respective tables on pressing respective keys. Option 3 will make the program “gracefully exit”, i.e. all the shared memory should be cleared and deallocated and all the threads should be killed (THIS OPTION HAS SIGNIFICANT MARKS).

Runtime

You should analyze and produce the reports for the following 3 different scenarios:

1. 2000 entries in the Service_Room_Occupancy_Table
2. 10,000 entries in the Service_Room_Occupancy_Table
3. 10 minutes of runtime