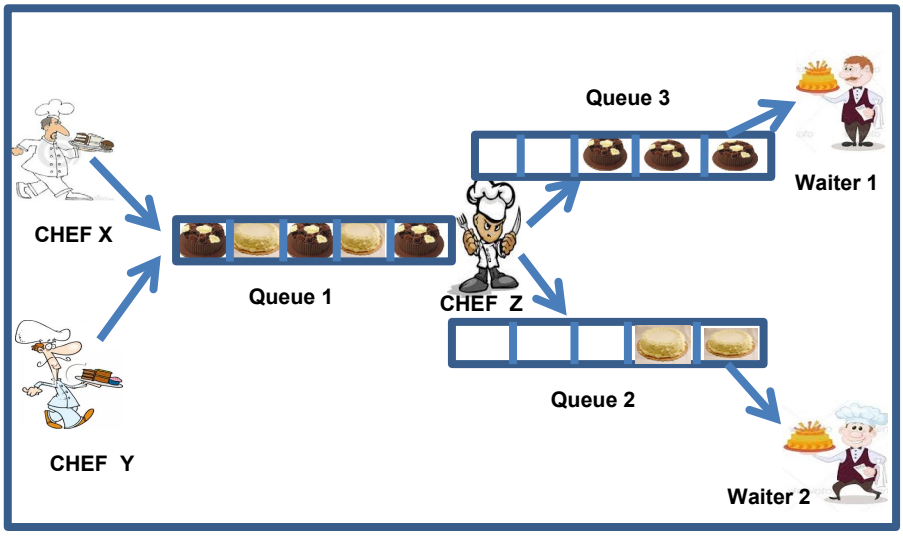
CSE 314: OS Sessional

IPC Offline

Cake factory

Suppose you own a cake shop named “Delicious cakes”. You have two pastry chefs, Chef X and Chef Y. Chef X is an expert chef for producing chocolate cakes, where Chef Y produces delicious vanilla cakes. Both types of cakes are stored in Queue 1. Queue 1 can hold at most 5 cakes. Next Chef Z picks up the cakes from queue 1, one at a time and decorates it (ignore the decoration time). After decoration Chef Z sends the cake to Queue 2 if it is a vanilla cake or sends the cake to Queue 3 if it is a chocolate cake. Afterwards, Waiter 2 takes the vanilla cake from the Queue 2 one at a time and serves it to the customer. On the other hand, Waiter 1 serves the customer by taking the chocolate cake from Queue 3 one at a time. If Chef Z and both the waiters find out that their corresponding queues are empty they take a coffee break. Moreover, if Chef X and Y find out that the Queue 1 is full they stop producing the cakes and take a tea break. Now you need to simulate the system using threads and synchronization primitives and show different states of all the chefs and waiters on the screen.



Technical Specification:

- You have to use C or C++ as language and Linux as operating system.
- Create separate process for this system i.e., chefs and waiters.
- It is clear that the processes here are not independent, rather they are cooperating processes. i.e. two or more processes working together for a specific goal. Chef Z can't pick up a cake from Queue 1 and decorate it, if Chef X and Chef Y haven't produced it yet. Therefore, some sort of Inter-Process-Communication (IPC) is necessary here.
- There are two paradigms of IPC – Shared Memory and Message Passing. For this assignment, we will use Shared memory.
- Chef X and Y push in Queue 1, Chef Z pops from Queue 1 and pushes chocolate cakes to Queue 3, and pops from Queue 1 and pushes to Queue 2 for vanilla cakes, Waiter 1 pops from Queue 3 and Waiter 2 pops from Queue 2. These queues contain the data which we need to communicate over processes, which means they should reside in a shared memory area.
- Queues need to be accessed from more than one process. You should use IPC primitives like **semaphores and mutex locks** to synchronize the critical zones.
- Whenever any queue is full or empty, corresponding processes should wait or be signaled. Please study the Producer-Consumer / Bounded Buffer problem.
- Semaphore and Shared Memory: Beginning Linux Programming, 3rd Ed, Ch-14
- IPC Primitives: Operating System Concepts (7thEd) Sec 6.1, 6.2, 6.5, 6.5.1, 6.6.1