

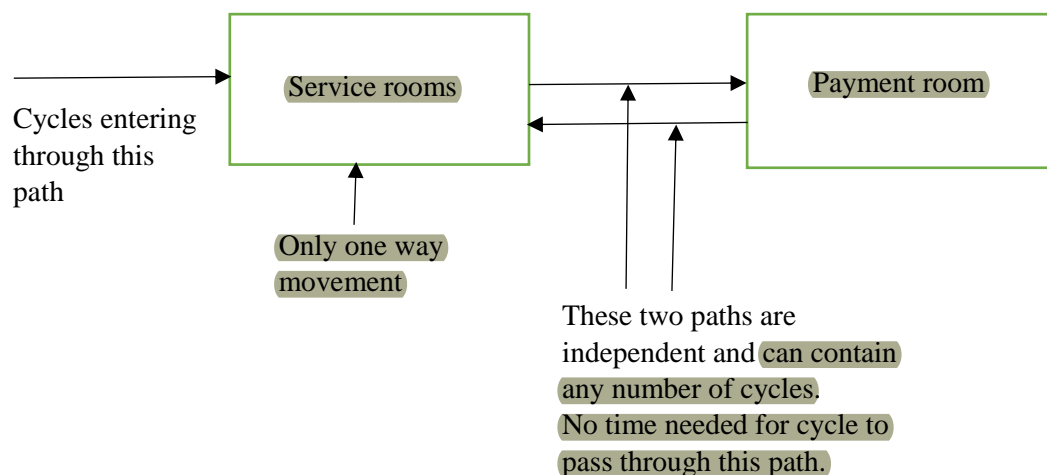
CSE 314: OS Sessional

Offline on IPC



Imagine there exists a cycle repairing shop. This shop has some weird rules for the cycles coming here to take service. All the cycles that want servicing from this shop **must be present** at the time of opening of the shop. There are S number of servicemen in the shop. They work sequentially one after another; i.e., serviceman 1 first inspects a cycle, then the cycle is passed to serviceman 2 and so on. After serviceman 1 finishes inspecting a cycle and *this cycle is passed to serviceman 2*; another cycle from the remaining cycles starts taking service from serviceman 1 and so on.

After a cycle's servicing is finished (i.e., it took service from all S servicemen), owner of the cycle must go to a room for the payment. However, that room can only contain C number of customers. After a cyclist is done with his payment, he will go out of the shop. However, he must return **through the path that is being used by the servicemen for servicing the cycles**. But, this path is so narrow that only one directional movement is possible; i.e., when any cycle is being serviced by any one of the servicemen, no cyclist can return through this path. Similarly, while a cyclist is returning through this path for departure, no new cycle can enter this path for starting its service.

Your task is to simulate the mentioned scenario.



Specifications

1. You have to use C or C++ as language and **Linux as operating system.**
2. It is clear that **the processes here are not independent.** Rather, **they are cooperating one another.** For example, a cycle can't start taking service while another cycle is being serviced by serviceman 1. Therefore, **some sort of Inter Process Communication is needed here.**
3. **It is not desired that a cyclist has completed his payment and is waiting for departure for too long.** So, **whenever a cyclist is waiting for departure, you need to temporarily block new cycles from entering into service to let that cyclist depart.**
4. **You need to use Shared Memory paradigm for accomplishing IPC in this offline;** i.e., different cyclists will take information (e.g., how many customers are already in the room for payment) from a shared memory which is accessible by all the cyclists. **As this memory needs to be accessed from more than one process, you need to synchronize those using semaphores and locks.** 
5. Each serviceman takes **random amount of time** to finish servicing of each cycle.
6. Each cyclist takes **random amount of time** for finishing the payment.
7. Each cyclist takes **random amount of time** for departure after payment.
8. [This script](#) will be used for evaluating your offline. **You must change only the first line of the script.** 
9. See sample output [here](#). **The output is produced for 10 cycles, 3 servicemen and the capacity of the payment room is 2.** Yours may not exactly match this. **But beware of sending one cyclist to a serviceman before sending the previous cyclist to the next serviceman.**

Resources

1. [Recording](#) of the class
2. Read **chapter 12** of the [book](#)
3. [Demo code](#) taught in the class

Submission Guideline

Submit only the cpp file named by your roll number e.g., 1605130.cpp after zipping. So, you will submit 1605130.zip and after extracting it, we will get 1605130.cpp.

Marks Distribution

Task	Percentage
Handling threads and semaphores	15%
Synchronization in taking service	20%
Synchronization in payment	15%
Prioritizing departure over taking service	20%
Avoiding any race condition and deadlock	15%
Generating given output	10%
Proper submission	5%

Deadline

October 2, 11:15PM.