Process Synchronization

There are two types of process:

1. Independent process: independent process is one that doesn't affect other processes executing in the system.
2. Cooperating process: cooperating process is one that can affect or affected by some other process executing in the system. They may share data by memory region, message passing or through files.

As cooperating process share resources so process synchronization problem may arise.

Race Condition:

This is a situation in which several processes access or manipulate the same data and the outcome of the execution depends on order in which the access takes place i.e. the outcome depends on the order of process executes.
So, to overcome this type of situation we need to ensure that only one process at a time can manipulate the shared data.

Critical section problem:

Critical section: it is a segment of code in which the process may change the variables, write a file etc. It is the part from where shared resources are accessed by processes. This section can be accessed by only one process at a time.
Note: If one process is executing in its critical section then no other process can execute in its critical section.

Entry Section: This section of code implementing the request of process to enter its critical section.
Exit Section: The critical section may be followed by this section.
Remaining Section: Remaining section contains the remaining code.

The solution to the critical section must satisfy the three requirements:
*Primary requirement: Mutual Exclusion, Progress.
*Secondary requirement: Bounded Waiting, portability.

1. Mutual exclusion: only one process can enter in critical section.
2. Progress: If any process doesn't want to enter in a critical section then that process doesn't stop other processes which wants to enter critical section.
3. Bounded waiting: There is a bound on the number of times that other processes can enter their critical section after a process has made a request to enter its critical section and before that request is granted.

Two approaches used to handle critical sections are preemptive kernel and non-preemptive kernel.
Preemptive kernel: It allows process to be preempted while it is running in kernel mode.
Non preemptive kernel: It doesn't allow to be preempted while it is running, process in kernel mode will run until it exits kernel mode. Therefore, non-preemptive kernel is free from race conditions as only one process is active in kernel at a time.

Please read all links in the given sequence –

1. Introduction: https://www.geeksforgeeks.org/process-synchronization-set-1/
2. Software and hardware approach: https://www.geeksforgeeks.org/operating-system-process-synchronization/
3. Critical section: https://www.geeksforgeeks.org/g-fact-70/

Peterson's solution:

1. This is software-based solution to the critical section problem.
2. Peterson's solution is restricted to two processes.

3. This is busy waiting solution as in busy waiting, process continuously checks the critical section for its chance.
4. It uses two shared variable – turn and interested (or flag).

```
Int turn;
Boolean flag [2];
```

Variable turn: indicates whose turn it is to enter its critical section.
Flag array: indicate if a process is ready or interested to enter its critical section.

Please read this link to learn more about Peterson's solution:

1. Algorithm: https://www.geeksforgeeks.org/process-synchronization-set-1/
2. Dekker's algorithm and programs: https://www.geeksforgeeks.org/petersons-algorithm-for-mutual-exclusion-set-1/
3. Dekker's algorithm: https://www.geeksforgeeks.org/operating-system-dekkers-algorithm/
4. CPU cycles and memory: https://www.geeksforgeeks.org/petersons-algorithm-for-mutual-exclusion-set-2-cpu-cycles-and-memory-fence/
5. Using processes and shared memory: https://www.geeksforgeeks.org/petersons-algorithm-using-processes-shared-memory/

Advantages:

1. Mutual exclusion is guaranteed.
2. Progress is satisfied.
3. Bounded waiting requirement is also satisfied.sy

Disadvantages:

1. Busy waiting (process continuously checks the critical section for its chance).
2. It is restricted to only two processes.

* **Busy waiting**: while a process is in critical section, then any other process that tries to enter critical section must loop continuously in the entry section. It wastes the CPU cycles. This type of semaphore also known as spinlock.

Test and Set:

1. It is a hardware solution.
2. It uses shared variable i.e. Lock
3. Lock variable can take either of two values 0 (unlock) or 1(lock).
4. Process must acquire lock before entering critical section and it releases the lock when it exits the critical section.
5. Process first check the status of the lock variable. If the value of lock variable is 1 that means critical section is locked so process must wait till it become free and if the value of lock is 0 then process takes the lock and executes the critical section.

Please read this link: https://www.geeksforgeeks.org/process-synchronization-set-1/
Advantage: Mutual exclusion and progress are guaranteed.
Disadvantage: Bounded waiting is not guaranteed.

Semaphores:

1. Semaphore is an integer variable that can be accessed through two operations: wait () and signal ().
2. Terminology for decrementing the semaphore variable - Down, P, wait ().
3. Terminology for incrementing the semaphore variable – Up, V, signal ().
4. If one process modifies the semaphore value, then other process cannot modify that same semaphore value.

5. There are two types of semaphores: binary semaphore and counting semaphore.
6. **Counting semaphore**:
   a. Counting semaphore can range over unrestricted domain.
   b. It has some integer value which shows how many processes can enter the critical section.
   c. Counting semaphore value initialized to number of instances of resource. If a process wants to access the resource, it first checks the counting semaphore value if its value is greater than 0 - this shows some instances of resources are free and then the process can enter its critical section thereby decreases the counting semaphore value by 1. Whenever the process completed, it can leave critical section thereby increment the counting semaphore value by 1 (i.e. number of available instances of resource is incremented by 1).
7. **Binary semaphore**:
   a. Binary semaphore can take either of two values 0 or 1.
   b. It is also known as mutex locks as it provides mutual exclusion.

Please read the following links in giver order-
1. Introduction: https://www.geeksforgeeks.org/process-synchronization-set-1/
2. Role of wait and signal + implementation: https://www.geeksforgeeks.org/semaphores-operating-system/
3. Mutex VS Semaphores: https://www.geeksforgeeks.org/mutex-vs-semaphore/
4. Producer Consumer Problem using semaphore: https://www.geeksforgeeks.org/producer-consumer-problem-using-semaphores-set-1/

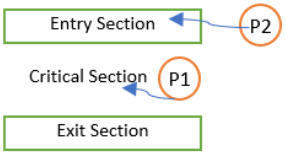
Advantages:
1. Only one process can enter in critical section.
2. Mutual exclusion, progress, bounded waiting are preserved.

Disadvantages:
1. Busy waiting (process continuously checks the critical section for its chance).
2. It may lead to priority inversion.

* **Busy waiting**: while a process is in critical section, then any other process that tries to enter critical section must loop continuously in the entry section. It wastes the CPU cycles. This type of semaphore also known as spinlock.
* **Priority Inversion**: when critical section first accessed by low priority process and later by high priority process.
Priority inversion explanation with example: https://www.geeksforgeeks.org/priority-inversion-what-the-heck/

Priority Inversion:

when critical section first accessed by low priority process and later by high priority process.
If a high priority task is indirectly preempted by lower priority task. This shows inversion of priorities of the two processes.



P1: low priority process.
P2: high priority process.

P2 request to enter critical section but P1 is executing in critical section. So, CPU is with P2 because of the priority but P1 is executing in critical section i.e. critical section is with P1.
As P2 try to enter critical section but P1 is in critical section so P2 doesn't allow to enter in critical section because of the process synchronization mechanism.
P1 wants the CPU to come out from the critical section as P1 is in critical section but can't execute because it is preempted by CPU.
P2 doesn't leave the CPU unless it gets the critical section. Now both P1 and P2 are in deadlock or spinlock.

Priority Inheritance:

It is the solution to priority inversion.
When low priority process blocks the high priority process, ignore its original priority assignment and executed its critical section at an elevated priority level. After executing its critical section and releasing its locks, the process returns to the original priority level.

Difference between priority Inversion and priority inheritance: https://www.geeksforgeeks.org/whats-difference-priority-inversion-priority-inheritance/

Monitors:

1. Monitor always ensures that only one process can enter in the monitor.
2. Monitor presents a set of programmer-defined operations that provide mutual exclusion in monitor.
3. It also contains shared variable declaration.

Syntax of Monitor:

```
Monitor pep      //name of the monitor
{
Variable declarations

procedure p1() {}
procedure p2() {}

initialization code () {}
}
```

Introduction to Monitors + syntax of Monitor + Condition Variables + operations of Monitors: https://www.geeksforgeeks.org/monitors/

Bakery Algorithm:

1. It is based on the criteria FCFS (First Come First Serve).
2. Before entering critical section, process received token or ticket number. holder of smallest ticket number enters the critical section.
3. Suppose, there is a bakery (critical section) with a numbering machine at its entrance so each customer (process) is given a number in an in incrementing way. Numbers increases by 1 as customer(process) enters the bakery (critical section). Global counter shows the total number of customers (process) that is being currently served.

Algorithm in detail: https://www.geeksforgeeks.org/operating-system-bakery-algorithm/

Dining philosopher problem:

It states that X philosophers spend their time on thinking and eating. These philosophers share a circular table surrounded by X chairs each belonging to one philosopher. There is bowl of noodles in the middle of table and X single chopsticks, each belongs to one philosopher. When a philosopher thinks, he doesn't interact with other

philosophers and after some time philosopher gets hungry and tries to pick the two (left and right) chopsticks that are closest to his. Philosopher may pick only one chopstick at a time (he cannot pick up a chopstick that is in the hand of neighbor philosopher). When hungry philosopher has both left and right chopsticks at the same time then he starts eating. When he finished eating, he puts down both left and right chopsticks and start thinking again.

***Explanation in details*:**

1. **Using semaphores:** https://www.geeksforgeeks.org/operating-system-dining-philosopher-problem-using-semaphores/
2. **Using Monitors:** https://www.geeksforgeeks.org/dining-philosophers-solution-using-monitors/

Reader Writer Problem:

Number of processes wants to share the data. Some of them only wants to read the data and others may want to modify the data. If two process(readers) only wants to read the shared data, then no adverse effect will result and if two process(one is writer and other may reader or writer) wants to share the data, then adverse effect will result as one process currently modifying the data and at the same time other process reads that data so dirty reading may be possible. For this, writers always have exclusive access to shared data.

Variations of Reader Writer problem-

1. No reader should wait for other readers to complete as writer is in waiting state. (writers may starve)
2. If writer is writing or modifying the shared data, then reader must wait. (readers may starve)

If process wants to read the shared data, then it requests the reader-writer lock in read mode.
If process wants to modify the shared data, then it requests the reader-writer lock in write mode.

1. Introduction and working: https://www.geeksforgeeks.org/readers-writers-problem-set-1-introduction-and-readers-preference-solution/
2. Reader writer problem using Monitors: https://www.geeksforgeeks.org/operating-system-reader-writers-solution-using-monitors/

Sleeping Barber problem:

Sleeping Barber problem states that there is one barber shop, one barber, one barber chair in cutting room and N waiting chairs for the waiting customer.

*When customer arrive, looks to see what the barber is doing. If the barber is sleeping on cutting chair, the customer wakes him up and sits on the cutting chair in cutting room.
*If the barber is cutting hair, the customer stays in the waiting room.
*If the waiting chairs in waiting room are free then customer sits in it and wait for his turn.
*If there is no free chair in waiting room then customer leaves the shop.

Introduction + problem with solution + sleeping barber problem with semaphores + diagrams:
https://www.geeksforgeeks.org/operating-system-sleeping-barber-problem/