Inter-process Communication

IPC

- A process contains everything needed for execution
  - An address space (defining all the code and data pages)
  - OS resources (e.g., open files) and accounting information
  - Execution state (PC, SP, regs, etc.)
  - Each of these resources is exclusive to the process

- Processes need to interact with each other
- inter-process communication (IPC) is the term used to refer to the activity of sharing data across multiple processes, via communication protocols

Inter-process communication Methods:

- Shared memory - permits processes to communicate by simply reading and writing to a specified memory location.
- Mapped memory is similar to shared memory, except that it is associated with a file in the filesystem.
- File - A record stored on disk, or a record synthesized on demand by a file server, which can be accessed by multiple processes.
- Pipe -A two-way data stream between two processes interfaced through standard input and output and read in one character at a time. Pipes permit sequential communication from one process to a related process.
- Message passing - Allows multiple programs to communicate using channels, commonly used in concurrency models.
- Sockets support communication between unrelated processes even on different computers.

Area of Essential IPC

- Modern operating systems execute processes *concurrently*
- Processes need cooperate and Compete with each other

Process Cooperation

- Processes are cooperating if they can affect each other. The simplest example of how this can happen is where two processes are using the same file. One process may be writing to a file, while another process is reading from the file; so, what is being read may be affected by what is being written. Processes cooperate by sharing data. Cooperation is important for several reasons:
  - **Information sharing**: Several processes may need to access the same data (such as stored in a file.)

.

- **Computation speedup**: A task can often be run faster if it is broken into subtasks and distributed among different processes. This depends upon the processes sharing data. (Of course, real speedup also required having multiple CPUs that can be shared as well. For example, consider a webserver which may be serving many clients. Each client can have their own process or thread helping them. This allows the server to use the operating system to distribute the computer's resources, including CPU time, among the many clients.

- **Convenience**: An individual user can run several programs at the same time, to perform some task. For example, a network browser is open, while the user has a remote terminal program running (such as telnet), and a word processing program editing data

Process Competition

- Computer resources are too few compared to process
- Proccess are always in a race condition where each process attempts to grab the resource it requires

- Areas of essential IPC include
  - Mutual Exclusion
  - Synchronization
  - Deadlock prevention

Mutual Exclusion

- Computer resources are either sharable or non-sharable
- mutual exclusion refers to the act of ensuring that no two concurrent processes access a non-sharable resource at the same time
- it is a basic requirement in concurrency control, to prevent race conditions

Process Synchronization

- Process synchronization refers to the idea that multiple processes are to join up or handshake at a certain point, in order to reach an agreement or commit to a certain sequence of action

Resolving IPC Issues

- IPC is facilitated through the use of semaphores and monitors

Semaphore

- A semaphore is an abstract data type that is used for controlling access, by multiple processes, to a common resource in a parallel programming or a multi user environment
- A semaphore is a protected variable whose value can be accessed and altered only by the operations W and S and initialization operation called 'Semaphoiinitislize'
- Semaphore concept introduced by E. W. Dijkstra (1965) after studying rail line communication

Semaphore Features

- Non – Negative integer
- Its value is initialized – set to a certain value
- Acted upon by two operation of WAIT and SIGNAL
- Operations are indivisible - The operations of Wait or Signal are atomic.

WAIT and SIGNAL operations

- WAIT decrements semaphore value by 1
  - S=s-1
- SIGNAL operation increments the semaphore value by 1
  - S=s+1
- A wait operation signifies that a resource which was free has been taken over by a proces
- A signal operation signifies that a resource has been freed by a process

- When **Wait** is executed by a thread, we have two possibilities:
  - **The counter of *S* is positive**
    In this case, the counter is decreased by 1 and the thread resumes its execution.
  - **The counter of *S* is zero**
    In this case, the thread is suspended and put into the private queue of ***S***.
- When **Signal** is executed by a thread, we also have two possibilities:
  - **The queue of *S* has no waiting thread**
    The counter of ***S*** is increased by one and the thread resumes its execution.
  - **The queue of *S* has waiting threads**
    In this case, the counter of ***S*** must be. One of the waiting threads will be allowed to leave the queue and resume its execution. The thread that executes **Signal** also continues.

Critical Section Problem

- In concurrent programming, a critical section is a part of a multi-process program that may not not be concurrently executed by more than one of the program's processes/threads
- The critical section of a resource is the point at which access is made to the resource

- The protocol developed for solving the Critical Section Problem involved three steps:
  - **Entry section**: Before entering the critical section, a process must request permission to enter
  - **Critical section**: After permission is granted, a process may execute the code in the critical section. Other processes respect the request, and keep out of their critical sections.
  - **Exit section**: The process acknowledges it has left its critical section.

Resolving Mutual Exclusion

- How do we ensure that a non-sharable resource is only accessed by one process at a time.
  - Identify the critical section of the resource
  - Enclose the critical section with a single semaphore and initialize its value to 1

Resolving Process synchronization

- Identify the points at which the processes need to be synchronized
  - Enclose the two points with a single semaphore and initialize its value to 0
  - Let the dependent process perform a wait operation and the other a signal operation

Process Deadlock

- A deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function.

Conditions Necessary and Sufficient for a process Deadlock

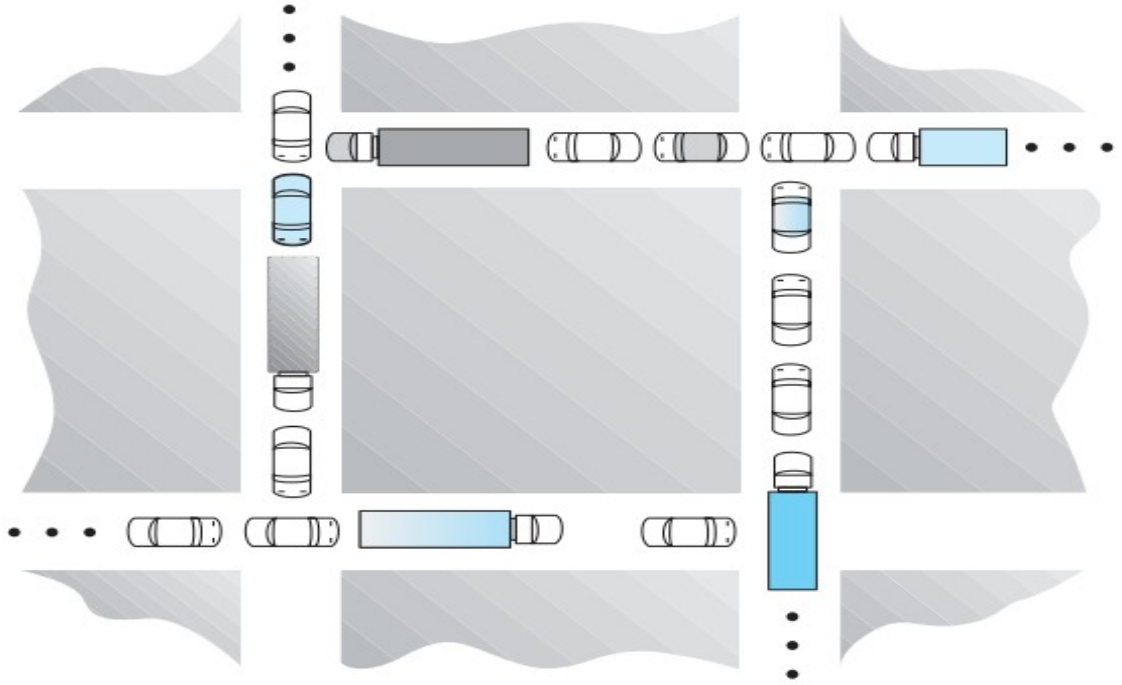
1. Mutual Exclusion Condition The resources involved are non-shareable. **Explanation:** At least one resource (thread) must be held in a non-shareable mode, that is, only one process at a time claims exclusive control of the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

2. Hold and Wait Condition Requesting process hold already resources while waiting for requested resources. **Explanation:** There must exist a process that is holding a resource already allocated to it while waiting for additional resource that are currently being held by other processes.

3. No-Preemptive Condition Resources already allocated to a process cannot be preempted. **Explanation:** Resources cannot be removed from the processes are used to completion or released voluntarily by the process holding it.

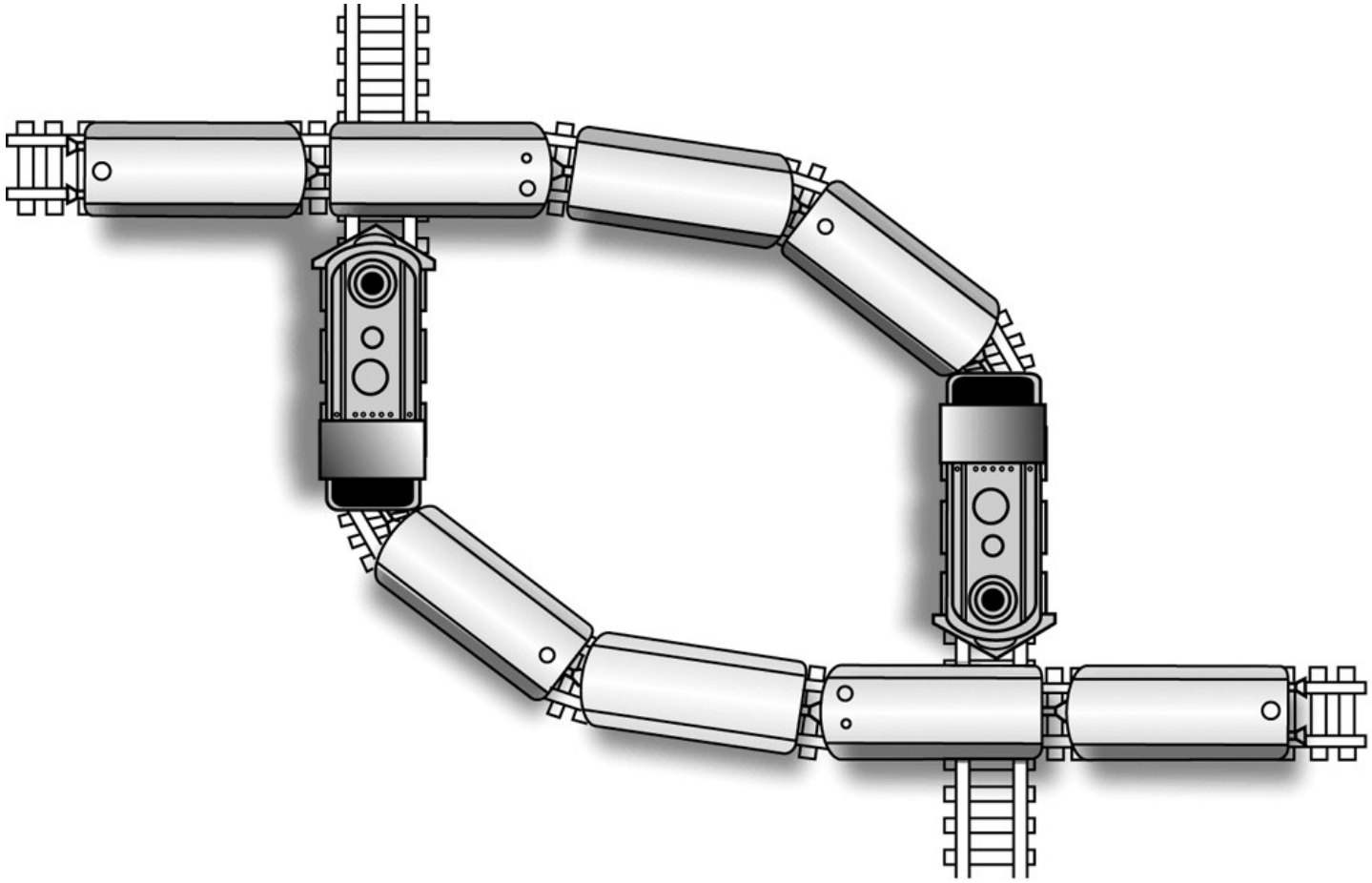
4. Circular Wait Condition The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list

- As an example, consider the traffic deadlock in the following figure



- Consider each section of the street as a resource.
  - Mutual exclusion condition applies, since only one vehicle can be on a section of the street at a time.
  - Hold-and-wait condition applies, since each vehicle is occupying a section of the street, and waiting to move on to the next section of the street.
  - No-preemptive condition applies, since a section of the street that is a section of the street that is occupied by a vehicle cannot be taken away from it.
  - Circular wait condition applies, since each vehicle is waiting on the next vehicle to move. That is, each vehicle in the traffic is waiting for a section of street held by the next vehicle in the traffic.

A deadlock resulting from competition for nonshareable railroad intersections



Dealing With Deadlocks

- Prevention
- Avoidance
- Detection and recovery
- Ignorance