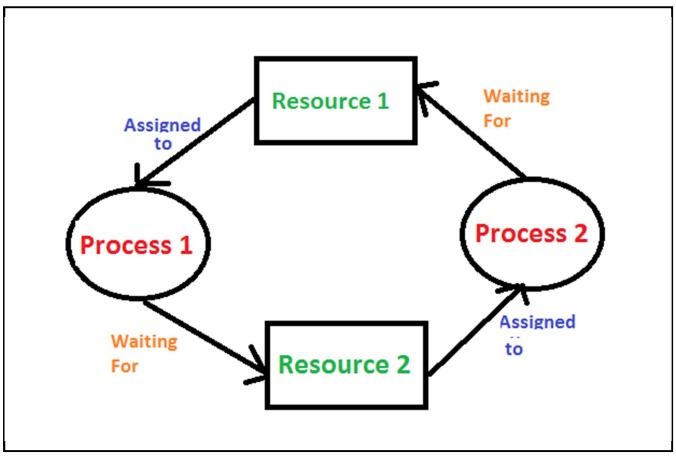
Deadlock



Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Deadlock Characterization

Deadlock can arise if following four conditions hold simultaneously (Necessary Conditions):

1. **Mutual Exclusion:** One or more than one resource are non-sharable (Only one process can use at a time)
2. **Hold and Wait:** A process is holding at least one resource and waiting for resources.
3. **No Preemption:** A resource cannot be taken from a process unless the process releases the resource.
4. **Circular Wait:** A set of processes are waiting for each other in circular form.

Methods for Handling Deadlocks

1. Deadlock Prevention

Deadlock prevention algorithms ensure that at least one of the necessary conditions does not hold true.

Mutual Exclusion: Not always possible to prevent deadlock by preventing mutual exclusion (making all resources shareable) as certain resources are cannot be shared safely.

Hold and Wait: We will see two approaches, but both have their disadvantages.

- A resource can get all required resources before it start execution.
- Second approach is to request for a resource only when it is not holding any other resource.

No preemption: We will see two approaches here.

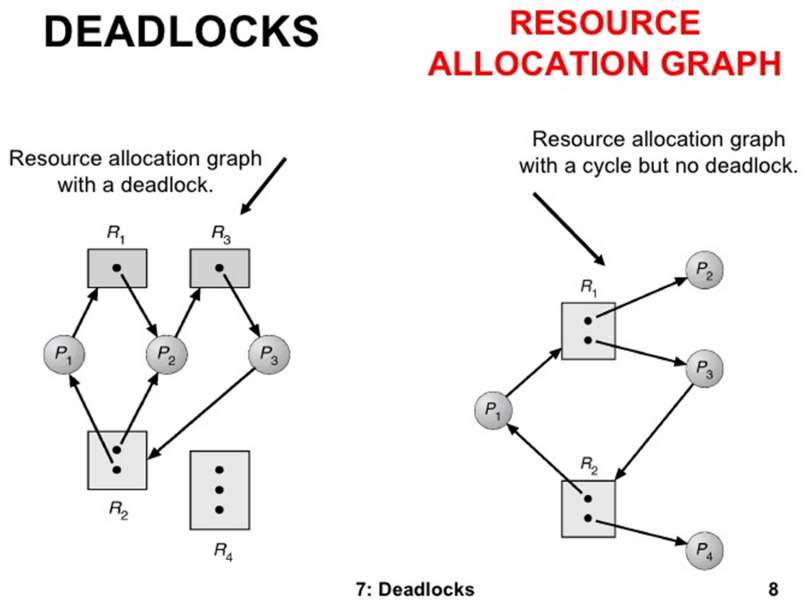
- If a process request for a resource which is held by another waiting resource, then the resource may be preempted from the other waiting resource.
- In the second approach, if a process request for a resource which are not readily available, all other resources that it holds are preempted.

Circular wait: To avoid circular wait, resources may be ordered and we can ensure that each process can request resources only in an increasing order of these numbers.

2. Deadlock avoidance

Safe State: If a system is already in a safe state, we can try to stay away from an unsafe state and avoid deadlock. Deadlocks cannot be avoided in an unsafe state. A safe sequence of processes and allocation of resources ensures a safe state.

Resource Allocation: A resource allocation graph is generally used to avoid deadlocks. If there are no cycles in the resource allocation graph, then there are no deadlocks. If there are cycles, there may be a deadlock.



The resource allocation graph is not much useful if there are multiple instances for a resource. In such a case, we can use Banker's algorithm.

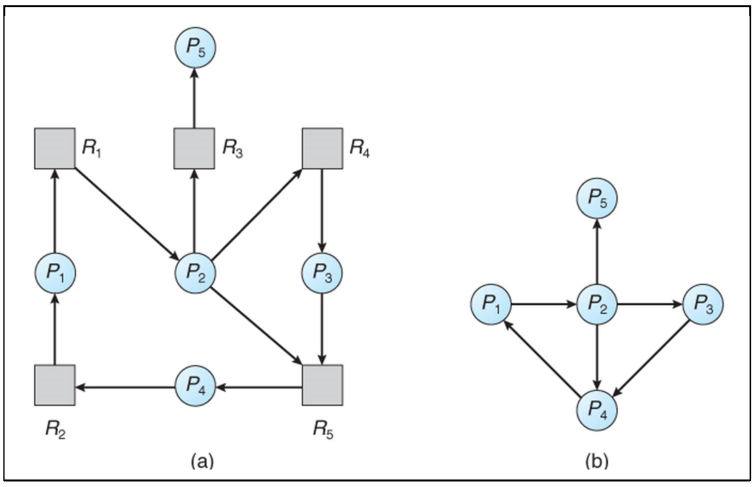
Banker's Algorithm

Banker's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it check for safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it don't allow the request made by the process.

Deadlock Detection

If deadlocks are not avoided, then another approach is to detect when they have occurred and recover somehow.

1. **If resources have single instance:** In this case for Deadlock detection we can run an algorithm to check for cycle in the Resource Allocation Graph. If each resource category has a single instance, then we can use a variation of the resource-allocation graph known as a wait-for graph.



(a) Resource allocation graph. (b) Corresponding wait-for graph

2. **If there are multiple instances of resources:** Detection of cycle is necessary but not sufficient condition for deadlock detection, in this case system may or may not be in deadlock varies according to different situations.

Recovery from Deadlock

There are three basic approaches to recovery from deadlock:

1. Inform the system operator, and allow him/her to take manual intervention.
2. Terminate one or more processes involved in the deadlock
3. Preempt resources.

Process Termination

Two basic approaches, both of which recover resources allocated to terminate processes:

- Terminate all processes involved in the deadlock.
- Terminate processes one by one until the deadlock is broken.

Resource Preemption

When preempting resources to relieve deadlock, there are three important issues to be addressed:

- **Selecting a victim -** Deciding which resources to preempt from which processes involves many of the same decision criteria outlined above.
- **Rollback -** Ideally one would like to roll back a preempted process to a safe state prior to the point at which that resource was originally allocated to the process.
- **Starvation -** This happens when the same process is chosen again and again as a victim, making sure that that process can never finish. To prevent starvation, an operating system usually limits the number of times a specific process can be chosen as a victim.

Process Concurrency

Concurrency is the tendency for things to happen at the same time in a system. Concurrency is also a programming design philosophy. In concurrent programming, programmers attempt to break down a complex problem into several simultaneous executing processes that can be addressed individually. Although concurrent programming offers better program structure than sequential programming, it is not always more practical.

Distribution is a form of concurrency where all communication between simultaneous threads is done exclusively via message passing. Distribution is useful because it employs a more lenient scaling of resource consumption, which economizes these resources. Whereas shared memory concurrency often requires a single processor per thread, distribution allows several threads to co-exist and communicate between one another.

Precedence Graphs

A PRECEDENCE GRAPH is a directed, acyclic graph where nodes represent sequential activities

A node in a precedence graph could be a software task, as recognized by the operating system, or could be statements within a program that can be executed concurrently with respect to each other, or could describe some activity taking place during the execution of a single machine instruction.

Process Synchronization

Process synchronization is the task of synchronizing the execution of processes in such a manner that no two processes have access to the same shared data and resource. Process Synchronization was introduced to handle problems that arose while multiple process executions.

Critical Section Problem

Informally, a critical section is a code segment that accesses shared variables and has to be executed as an atomic action. The critical section problem refers to the problem of how to ensure that at most one process is executing its critical section at a given time.

Solution to Critical Section Problem

A solution to the critical section problem must satisfy the following three conditions :

- **Mutual exclusion:** When a thread is executing in its critical section, no other threads can be executing in their critical sections.
- **Progress:** If no thread is executing in its critical section, and if there are some threads that wish to enter their critical sections, then one of these threads will get into the critical section.
- **Bounded waiting:** After a thread makes a request to enter its critical section, there is a bound on the number of times that other threads are allowed to enter their critical sections, before the request is granted.

Semaphores

A semaphore is a higher level mechanism used to control access to a common resource by multiple processes in a concurrent system such as a multiprogramming operating system. Semaphores are commonly used for two purposes: to share a common memory space and to share access to files. Semaphores are one of the techniques for Interpocess Communication (IPC).

Operations of Semaphores

Wait(): a process performs a wait operation to tell the semaphore that it wants exclusive access to the shared resource.

Signal(): a process performs a signal operation to inform the semaphore that it is finished using the shared resource.

Types of Semaphores

The two most common kinds of semaphores are:

Binary Semaphore: It is a special form of semaphore used for implementing mutual exclusion, hence it is often called Mutex. A binary semaphore is initialized to 1 and only takes the value 0 and 1 during execution of a program.

Counting Semaphores: These are used to implement bounded concurrency. Counting semaphore can take non-negative integer values.

Inter-Process Communication

A mechanism through which data is shared among the process in the system is referred to as Inter-process communication. Multiple processes communicate with each other to share data and resources. A set of functions is required for the communication of process with each other.

Various techniques can be used to implement the Inter-Process Communication. There are two fundamental models of Inter-Process communication that are commonly used, these are:

1. Shared Memory Model: In the shared-memory model, a region of memory which is shared by cooperating processes gets established. Processes can then able to exchange information by reading and writing all the data to the shared region.

2. Message Passing Model: In the message-passing form, communication takes place by way of messages exchanged among the cooperating processes.

File

A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

File Structure

A file has various kinds of structure. Some of them can be:

Simple Record Structure with lines of fixed or variable lengths.

Complex Structures like formatted document or reloadable load files.

No Definite Structure like sequence of words and bytes etc.

Attributes of a File

Following are some of the attributes of a file:

Name: It is the only information which is in human-readable form.

Identifier: The file is identified by a unique tag(number) within file system.

Type: It is needed for systems that support different types of files.

Location: Pointer to file location on device.

Size: The current size of the file.

Protection: This controls and assigns the power of reading, writing, executing.

Time, date, and user identification: This is the data for protection, security, and usage monitoring.

File Type

File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files −

Ordinary files

- These are the files that contain user information.
- These may have text, databases or executable program.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

Directory files

- These files contain list of file names and other information related to these files.

Special files

- These files are also known as device files.
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types −

Character special files – data is handled character by character as in case of terminals or printers.

Block special files – data is handled in blocks as in the case of disks and tapes.

File Access Methods

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files –

Sequential access: A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other.

Direct/Random access: Random access file organization provides, accessing the records directly. Each record has its own address on the file by which it can be directly accessed for reading or writing.

Indexed sequential access: An index is created for each file which contains pointers to various blocks. Index is searched sequentially and its pointer is used to access the file directly.

File Directories

Collection of files is a file directory. The directory contains information about the files, including attributes, location and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines.

Following is the information maintained in a directory:

- Name: The name visible to user.
- Type: Type of the directory.
- Location: Device and location on the device where the file header is located.
- Size: Number of bytes/words/blocks in the file.
- Position: Current next-read/next-write pointers.
- Protection: Access control on read/write/execute/delete.
- Usage: Time of creation, access, modification etc.
- Mounting: When the root of one file system is "grafted" into the existing tree of another file system is called Mounting.

Operation performed on directory are:

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Directory Structure

The directory structure is the organization of files into a hierarchy of folders.

There are many types of directory structure in Operating System. They are as follows :-

1) **Single Level Directory:** In Single Level Directory all files are in the same directory.
2) **Two – Level Directory:** In the two level directory structures, each user has her own user file directory (UFD). The system maintains a master block that has one entry for each user. When a user refers to a particular file, only his own UFD is searched.
3) **Tree Structured Directory:** A directory (or Sub directory) contains a set of files or sub directories. All directories have the same internal format.
4) **Acyclic Graph Directory:** Acyclic Graph is the graph with no cycles. It allows directories to share sub directories and files. With a shared file, only one actual file exists, so any changes made by one person are immediately visible to another.
5) **General graph directory:** The general graph directory is formed by adding links into an existing tree structure. It overcomes the problem of acyclic graph by allows the cycles in a directory. Thus it avoids the searching of a component twice in a subdirectory.

Directory Implementation

Directory efficiency, performance and reliability are totally dependent on the technique used to store them. They are saved on Disks with two well Known Techniques:

1) Linear List: It is the Simple method of directory implementation. It makes use of a file name and a pointer to data blocks. Whenever a new file is created, whole directory is checked to make sure that no existing file has the same name.

2) Hash Tables: In this technique Directories are saved in same linear fashion but a hash table is used. It takes a value that is returned from file name and returns a pointer to file name. This technique decreases the search time for files.

File Allocation Methods

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

1) Contiguous Allocation
- Each file occupies a contiguous address space on disk.
- Easy to implement.
- Allows random access.
- External fragmentation is a major issue with this type of allocation technique.

2) Linked Allocation
- Each file is a linked list of blocks
- NO external fragmentations
- Effective for sequential access.
- Problematic for direct access

3) Indexed Allocation
- Provides solutions to problems of contiguous and linked allocation.
- Both random and sequential access are easy
- Inefficient for small files
- Directory contains the addresses of index blocks of files.

The main idea behind these methods is to provide:

- Efficient disk space utilization.
- Fast access to the file blocks

Free Space Management

Since disk space is limited, we need to reuse the space from deleted files for new files, if possible. To keep track of free disk space, the system maintains a free-space list. The free-space list records all free disk blocks – those not allocated to some file or directory.

There are some methods or techniques to implement free space list. These are as follows:

1. **Bitmap:** When free space is list is implemented as bit map, each block of disk is represented by a bit. If the block is free, its bit is set to 1. If the block is allocated, the bit is 0.
2. **Linked List:** In this method, a linked list of all the free disk blocks is maintained. The first free block in the list can be pointed out by a head pointer, which is kept in a special location on the disk.
3. **Grouping:** A modification of the free-list approach is to store the addresses of n free blocks in the first free block. The first n-1 of these are actually free. The last one is the disk address of another block containing addresses of other n free blocks.
4. **Counting:** This method is based on the fact that we can allocate and free several contiguous blocks at the same time. This method keeps the address of the first free block and the number n of the free contiguous blocks that follows the first block. Each entry in the free-space list then consists of a disk address and a count.