OPERATING SYSTEMS
MODULE 3
CPU SCHEDULING

Misna VK
asst.professor
Dept of CS
Nasra college of arts and science,Tirurkad

SYNCHRONIZATION HARDWARE

- In operating systems, "synchronization hardware" refers to a set of specialized hardware instructions that enable efficient coordination between concurrent processes accessing shared resources, preventing data corruption <u>by ensuring only one process can modify a critical section of data at a time;</u>

- essentially acting as a locking mechanism to manage access to shared data.

There are three algorithms in the hardware approach of solving Process Synchronization problem:

1.Test and Set

2.Swap/disable-enable interrupts

3.Mutex

Hardware instructions in many operating systems help in the effective solution of critical section problems.

1.Test and set

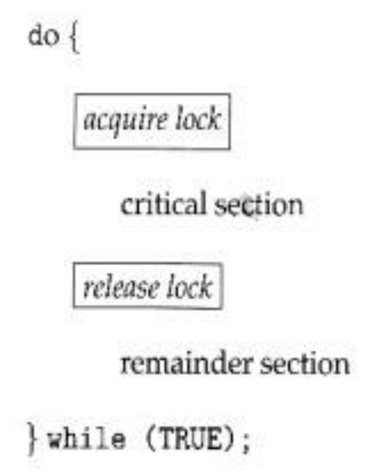
- Here, the shared variable is lock which is initialized to false.and when a process found this **lock=false**,it means critical section is free.so it will enter the critical section and then set **lock=true;**
- When another process comes,lock will be true.so the process should wait until lock becomes false again.
- The process in critical section will execute the critical section and when it exit it again set lock=false;so other waiting processes can enter the critical section.

2. **Swap/enable-disable interrupts**

- This is a simple solution for critical section problem.
- It simply prevents a process from being interrupted while it executes the critical section.
- Unfortunately, it does not work well for multiprocessor environments,due to the difficulty in enabling and disabling interrupts on all processors (time consuming)

3. Mutex

- A mutex lock in an operating system (OS) is a binary variable that prevents multiple threads from simultaneously accessing a shared resource. Mutex stands for mutual exclusion.
- A thread/process while entering to critical section it will lock the mutex.
- And after finishing execution it set mutex=unlock while leaving the cs.
- Any other process trying to access the critical section should wait if,the mutex found locked.



```
do {
    acquire lock
        critical section
    release lock
        remainder section
} while (TRUE);
```

CLASSICAL PROBLEMS OF SYNCHRONIZATION

Below are some of the classical problem depicting flaws of process synchronaization in systems where cooperating processes are present.

1.Bounded Buffer (Producer-Consumer) Problem
2.Dining Philosophers Problem
3.The Readers Writers Problem

Bounded Buffer (Producer-Consumer) Problem

Because the buffer pool has a maximum size, this problem is often called the **Bounded buffer problem**.

•This problem is generalised in terms of the **Producer Consumer problem**, where a **finite** buffer pool is used to exchange messages between producer and consumer processes.

•There will be 2 concurrent process ,one producer and one consumer and share a common buffer.

•In this Producers mainly produces a product and consumers consume the product, but both can use of one of the containers each time.

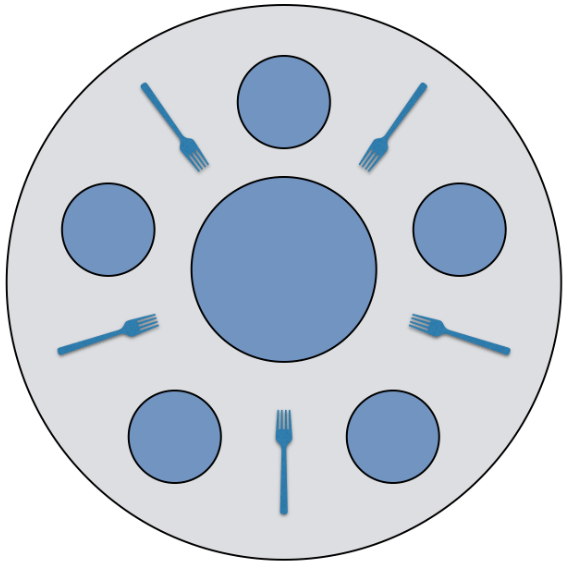
•The main complexity of this problem is that we must have to maintain the count for both empty and full containers that are available.

Dining Philosophers Problem

•The dining philosopher's problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.

•There are five philosophers sitting around a table, in which there are five chopsticks/forks kept beside them and a bowl of rice in the centre, When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

- The problem is to ensure that no philosopher will be allowed to starve because he cannot ever pick up both forks.
- There are 2 issues here:first, deadlock(*where each philosopher picks up one fork so none can get the second.*) must never occure.
- Second no set of philosophers should be able to prevent another philosopher fromever eating.



The Readers Writers Problem

- In this problem there are some processes(called **readers**) that only read the shared data, and never change it, and there are other processes(called **writers**) who may change the data in addition to reading, or instead of reading it.
- Here **multiple readers can access the object concurrently,**but **if a writer is accessing the object,no other processes(reader/writer) may access the object**

File And database system

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 Structure should be according to a required format that the operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.

File functions of organization

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**
- **Direct/Random access**
- **Indexed sequential access**

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. This access method is the most primitive one. Example: Compilers usually access files in this fashion.

Direct/Random access

• Random access file organization provides, accessing the records directly.

• Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.

• The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

Indexed sequential access

• This mechanism is built up on base of 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

Space Allocation

Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

- **Contiguous Allocation**
- **Linked Allocation**
- **Indexed Allocation**

Contiguous Allocation

- Each file occupies a contiguous address space on disk.
- Assigned disk address is in linear order.
- Easy to implement.
- External fragmentation is a major issue with this type of allocation technique.

Linked Allocation

- Each file carries a list of links to disk blocks.
- Directory contains link / pointer to first block of a file.
- No external fragmentation
- Effectively used in sequential access file.
- Inefficient in case of direct access file.

Indexed Allocation

- Provides solutions to problems of contiguous and linked allocation.
- A index block is created having all pointers to files.
- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.

Allocation and free space management

The system keeps tracks of the free disk blocks for allocating space to files when they are created. Also, to reuse the space released from deleting the files, free space management becomes crucial. The system maintains a free space list which keeps track of the disk blocks that are not allocated to some file or directory. The free space list can be implemented mainly as:

1. Bitmap or Bit vector

– A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: 0 indicates that the block is allocated and 1 indicates a free block. The given instance of disk blocks on the disk in Figure 1 (where green blocks are allocated) can be represented by a bitmap of 16 bits as:0000111000000110.

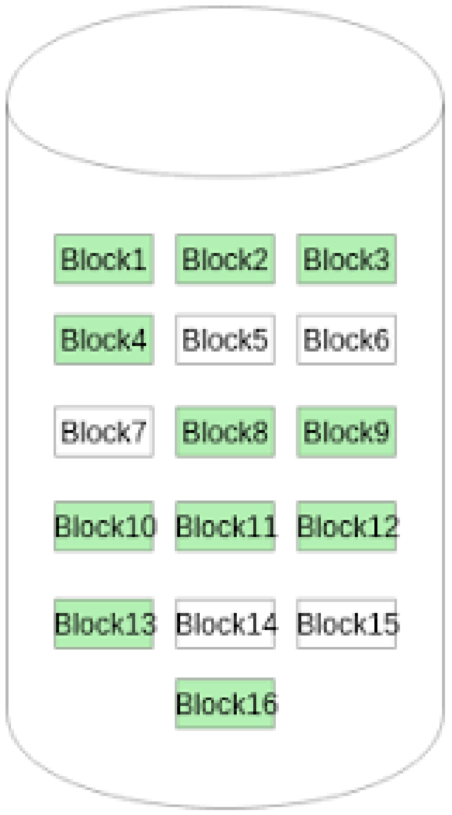


Figure - 1

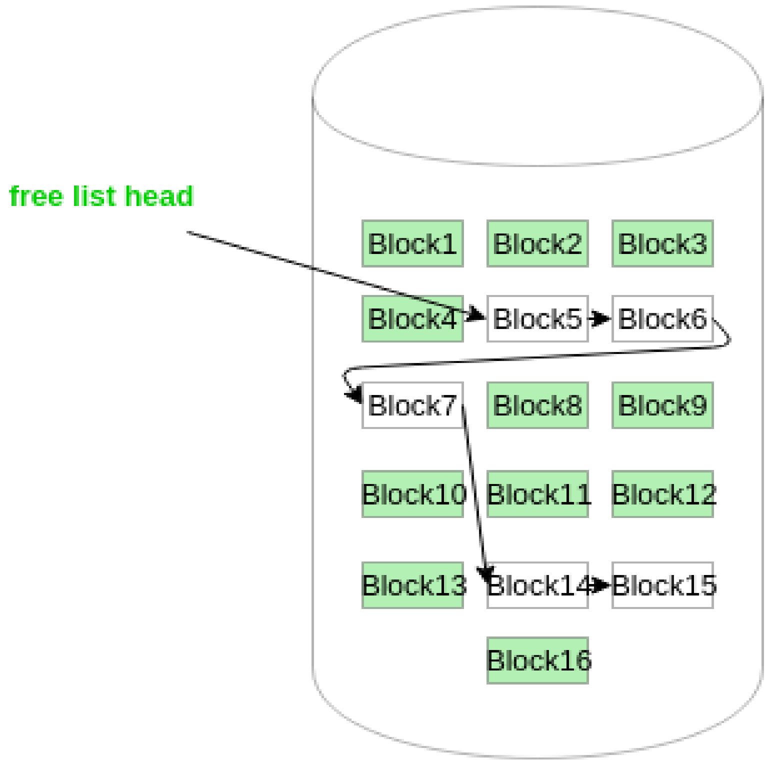


Figure - 2

2. Linked List

– In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory

In Figure-2, the free space list head points to Block 5 which points to Block 6, the next free block and so on. The last free block would contain a null pointer indicating the end of free list.

3. Grouping

– This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say n free blocks. Out of these n blocks, the first n-1 blocks are actually free and the last block contains the address of next free n blocks. An advantage of this approach is that the addresses of a group of free disk blocks can be found easily.

4. Counting

– This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block. Every entry in the list would contain: 1. Address of first free disk block 2. A number n For example, in Figure-1, the first entry of the free space list would be: ([Address of Block 5], 2), because 2 contiguous free blocks follow block 5.