Differentiate between: (a) Context switch and process switch (b) Internal fragmentation and External fragmentation (c) Disk scheduling and CPU scheduling (d) thread and Process

| Thread Context Switch | Process Context Switch |
|---|---|
| TCS occurs when the CPU saves the current state of the thread and switches to another thread of the same process. | PCS occurs when the operating system's scheduler saves the current state of the running Program (including the state of PCB) and switches to another program. |
| TCS helps the CPU to handle multiple threads simultaneously. | PCS involves loading of the states of the new program for it's execution. |
| TCS does not involves switching of memory address spaces.All the memory addresses that the processor accounts remain saved. | PCS involves switching of memory address spaces.All the memory addresses that the processor accounts gets flushed. |
| Processor's cache and Translational Lookaside Buffer preserves their state. | Processor's cache and TLB gets flushed. |
| Though TCS involves switching of registers and stack pointers, it does not afford the cost of changing the address space.Hence it is more efficient. | PCS involves the heavy cost of changing the address space.Hence it is less efficient. |
| TCS is a bit faster and cheaper. | PCS is relatively slower and costlier. |

| Internal fragmentation | External fragmentation |
|---|---|
| In internal fragmentation fixed-sized memory, blocks square measure appointed to process. | In external fragmentation, variable-sized memory blocks square measure appointed to the method. |
| Internal fragmentation happens when the method or process is smaller than the memory. | External fragmentation happens when the method or process is removed. |
| The solution of internal fragmentation is the best-fit block. | The solution to external fragmentation is compaction and paging. |
| Internal fragmentation occurs when memory is divided into fixed-sized partitions. | External fragmentation occurs when memory is divided into variable size partitions based on the size of processes. |

| | |
|---|---|
| The difference between memory allocated and required space or memory is called Internal fragmentation. | The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, which is called External fragmentation. |
| Internal fragmentation occurs with paging and fixed partitioning. | External fragmentation occurs with segmentation and dynamic partitioning. |
| It occurs in worst fit memory allocation method. | It occurs in best fit and first fit memory allocation method. |

| Disk Scheduling | CPU Scheduling |
|---|---|
| Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. | CPU scheduling is the process of deciding which process will own the CPU to use while another process is suspended. |
| The purpose of Disk scheduling is to minimize the time it takes to access data on the disk and to minimize the time it takes to complete a disk access request. | The purpose of CPU Scheduling is to make the system more efficient, faster, and fairer. |
| Disk scheduling algorithms are more complex | CPU scheduling algorithms are less complex |
| Disk scheduling is performed more frequently | CPU scheduling is performed less frequently |
| Disk scheduling optimizes for seek time | CPU scheduling optimizes for turnaround time |
| e.g. FCFS, SSTF, SCAN, CSACN,LOOK | e.g. FCFS,SJF,LJF, Round Robin, SRTF, LRTF |

| Process | Thread |
|---|---|
| Process means any program is in execution. | Thread means a segment of a process. |
| The process takes more time to terminate. | The thread takes less time to terminate. |
| It takes more time for creation. | It takes less time for creation. |
| It also takes more time for context switching. | It takes less time for context switching. |
| The process is less efficient in terms of communication. | Thread is more efficient in terms of communication. |
| The process is isolated. | Threads share memory. |

| | |
|---|---|
| The process has its own Process Control Block, Stack, and Address Space. | Thread has Parents' PCB, its own Thread Control Block, and Stack and common Address space. |
| A system call is involved in it. | No system call is involved, it is created using APIs. |
| The process does not share data with each other. | Threads share data with each other. |

## Explain the various CPU scheduling algorithm with example.

https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/

## What is Demand paging? How can virtual memory concept be implemented using Demand paging? Describe the process of demand paging in OS.

Demand Paging is a method in which a page is only brought into main memory when the CPU requests it. At first, just those pages are loaded directly required by the operation. Pages that are never accessed are thus never loaded into physical memory.

Virtual memory can be implemented by dividing the program's address space into pages, and only loading the pages that are currently being used into memory. When a program requests a page that is not in memory, a page fault occurs. The operating system then brings the page into memory from secondary storage, and the program continues execution.

The process includes the following steps:

1. If the CPU tries to refer to a page that is currently not available in the main memory, it generates an interrupt indicating a memory access fault.
2. The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.
3. The OS will search for the required page in the logical address space.
4. The required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision-making of replacing the page in physical address space.
5. The page table will be updated accordingly.
6. The signal will be sent to the CPU to continue the program execution and it will place the process back into the ready state.

## Describe the ways of implementing Semaphores. How is Semaphores different from Critical Section?

### Implementation of Semaphore:

```
typedef struct {
     int value;
     struct process *list;
} semaphore;
```

```
wait(semaphore *S) {
          S->value--;
          if (S->value < 0) {
                add this process to S->list;
                block();
          }
}
```

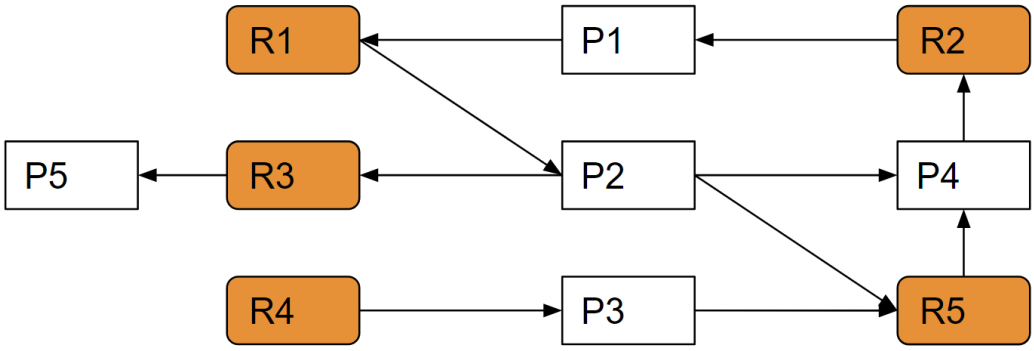
```
signal(semaphore *S) {
        S->value++;
        if (S->value <= 0) {
              remove a process P from S->list;
              wakeup(P);
        }
}
```

| Critical Section | Semaphore |
|---|---|
| Critical section are regions of code that must be executed by only one process at a time. | Semaphores are used to implement critical sections |
| Properties followed by code in critical section are: Mutual Exclusion, Progress, Bounded Waiting | Types: Binary Semaphore and Counting Semaphore |
| Problems caused by critical section are: Deadlock, Starvation and Overhead | Semaphores are used to enforce mutual exclusion, avoid race conditions and implement synchronization between processes. |

Discuss the various methods for Deadlock detection. Explain using examples.

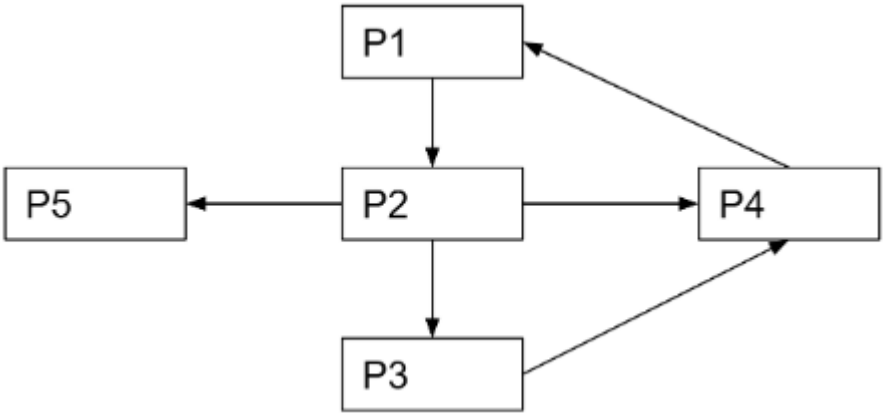
## Single Instance

We use the-Wait-for graph based on the resource allocation graph.
If the system has a cycle, it's experiencing Deadlock, and if there is no cycle, then there is no deadlock in the system.
Consider this RAG(Resource Allocation Graph):




To convert this into a Wait-for graph:

- First, remove all resource nodes and join the edges, i.e., the edge from P1 to R1, and the edge from R1 to P2, similarly doing this with all edges.
- After this, check if the wait-for graph has a cycle and if it's in a deadlock state.




P1 => P2 => P3 => P4, It is forming a cycle hence its in a deadlock state.
Some points to remember:

- Invoke an algorithm to detect cycles periodically.
- The complexity of such an algorithm is $n^2$, n=number of vertices.

## Multiple Instance

The resource allocation graph is converted into the request matrix and the resource distribution matrix by using **safety algorithm** on the system.

We can accomplish this by using the following three data structures.

- **Available[m]:** A vector of length m indicates the number of available resources of each type. Suppose Available[j]=k, then k instances of resource type Rj are available.
- **Allocation[n][m]:** This matrix indicates each process's n X m allocated resources. Suppose Allocation[i,j]=k; then Pi is allocated k instances of Rj.
- **Request[n][m]:** This matrix indicates the current Request of each process. If Request [i, j] = k, then process Pi requests k more instances of resource type Rj.

Pseudocode for Safety Algorithm is given below:

1. Let *Work* and *Finish* be vectors of length $m$ and $n$, respectively. Initialize $Work = Available$. For $i = 0, 1, ..., n\text{-}1$, if $Allocation_i \neq 0$, then $Finish[i] = false$; otherwise, $Finish[i] = true$.
2. Find an index $i$ such that both
   a. $Finish[i] == false$
   b. $Request_i \leq Work$

   If no such $i$ exists, go to step 4.
3. $Work = Work + Allocation_i$
   $Finish[i] = true$
   Go to step 2.
4. If $Finish[i] == false$ for some $i$, $0 \leq i < n$, then the system is in a deadlocked state. Moreover, if $Finish[i] == false$, then process $P_i$ is deadlocked.

Discuss the advantages and disadvantages of WINDOWS XP and WINDOWS 2000 operating system.

**Advantages of Windows XP:**

- **User-friendly Interface:** Windows XP introduced a more visually appealing and user-friendly interface compared to its predecessors.
- **Broad Hardware and Software Compatibility:** Windows XP had extensive hardware and software compatibility, making it compatible with a wide range of devices and applications.
- **Enhanced Multimedia Support:** Windows XP improved multimedia capabilities, offering better support for audio and video playback, as well as improved graphics performance.
  **System Stability**: Compared to its predecessor, Windows 98, Windows XP was more stable. It offered better system reliability and overall performance.

**Disadvantages of Windows XP:**

- **Security Vulnerabilities:** Microsoft stopped supporting Windows XP in 2014. This lack of support makes it a risky choice from a security standpoint.
- **Limited Support for Modern Hardware:** Users may face challenges when trying to use the latest peripherals or devices with Windows XP.

**Advantages of Windows 2000:**

- **Stability and Reliability:** It was designed with the robustness needed for business environments, making it a popular choice for organizations.

- **Improved Active Directory:** Windows 2000 introduced Active Directory. This feature made it easier to manage large-scale networks and improve security.
- **Enhanced Security:** Windows 2000 offered improved security features compared to previous Windows versions.

**Disadvantages of Windows 2000:**

- **Limited Hardware and Software Compatibility:** Some software applications were not fully compatible with Windows 2000, leading to potential compatibility issues.
- **Steeper Learning Curve:** Its interface and management tools were more complex compared to Windows XP, requiring a steeper learning curve for novice users.
- **Lack of Multimedia Features:** This is a disadvantage for users who required advanced multimedia capabilities.

Write short notes on: (a) File allocation method (b) Working Set model (c) Producer Consumer infinite buffer problem and its solution.

## (a) File allocation method

### 1. Contiguous Allocation

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,……b+n-1.* This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.

The directory entry for a file with contiguous allocation contains

- Address of starting block
- Length of the allocated portion.

**Advantages:**

- Both the Sequential and Direct Accesses are supported by this.
- This is extremely fast

**Disadvantages:**

- This method suffers from both internal and external fragmentation.
- Increasing file size is difficult

### 2. Linked List Allocation

In this scheme, each file is a linked list of disk blocks which **need not be** contiguous. The disk blocks can be scattered anywhere on the disk.

The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.

**Advantages:**

- This is very flexible in terms of file size.
- This method does not suffer from external fragmentation.

**Disadvantages:**

- It is slow.
- It does not support random or direct access.
- Pointers required in the linked allocation incur some extra overhead.

**3. Indexed Allocation**
In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block
**Advantages:**
- It provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

**Disadvantages:**
- The pointer overhead for indexed allocation is greater than linked allocation.
- It is inefficient in terms of memory utilization

## (b) Working Set model

This model is based on the above-stated concept of the Locality Model.
The basic principle states that if we allocate enough frames to a process to accommodate its current locality, it will only fault whenever it moves to some new locality. But if the allocated frames are lesser than the size of the current locality, the process is bound to thrash.

If D is the total demand for frames and $WSS_i$ is the working set size for process i,

$D=\sum WSSi$

Now, if 'm' is the number of frames available in the memory, there are 2 possibilities:

- (i) D>m i.e. total demand exceeds the number of frames, then thrashing will occur as some processes would not get enough frames.
- (ii) D<=m, then there would be no thrashing.

## (c) Producer Consumer infinite buffer problem and its solution.

The producer-consumer problem is a classic synchronization problem in operating systems. The problem arises when two processes, a producer and a consumer, share a common resource, such as a buffer. The producer produces data and puts it in the buffer, while the consumer consumes data from the buffer. The problem is to ensure that the producer does not produce data faster than the consumer can consume it, and that the consumer does not consume data faster than the producer can produce it.

In the case of an infinite buffer, the buffer can never be full, and the producer will never have to wait to put data in the buffer. However, the consumer may have to wait if the buffer is empty.

**Producer Process:**

```
do {
    . . .
   // produce an item in nextp
    . . .
   wait(empty);
   wait(mutex);
    . . .
   // add nextp to buffer
    . . .
   signal(mutex);
   signal(full);
} while (TRUE);
```

**Figure 6.10** The structure of the producer process.

In the above code, mutex, empty and full are semaphores. Here mutex is initialized to 1, empty is initialized to n (maximum size of the buffer) and full is initialized to 0.

The mutex semaphore ensures mutual exclusion. The empty and full semaphores count the number of empty and full spaces in the buffer.

After the item is produced, wait operation is carried out on empty. This indicates that the empty space in the buffer has decreased by 1. Then wait operation is carried out on mutex so that consumer process cannot interfere.

After the item is put in the buffer, signal operation is carried out on mutex and full. The former indicates that consumer process can now act and the latter shows that the buffer is full by 1.

**Consumer Process:**

```
do {
   wait(full);
   wait(mutex);
    . . .
   // remove an item from buffer to nextc
    . . .
   signal(mutex);
   signal(empty);
    . . .
   // consume the item in nextc
    . . .
} while (TRUE);
```

**Figure 6.11** The structure of the consumer process.

The wait operation is carried out on full. This indicates that items in the buffer have decreased by 1. Then wait operation is carried out on mutex so that producer process cannot interfere.

Then the item is removed from buffer. After that, signal operation is carried out on mutex and empty. The former indicates that consumer process can now act and the latter shows that the empty space in the buffer has increased by 1.