Victoria University of Bangladesh

Final -Term Examination– Summer 2022

Submitted To:

Renea Chowdhury Shormi

Lecturer

Victoria University of Bangladesh

Submitted By:

Name: Anny Konika Das,

ID: 2120190011

Program: B.Sc in CSE.

Semester: Summer-2022

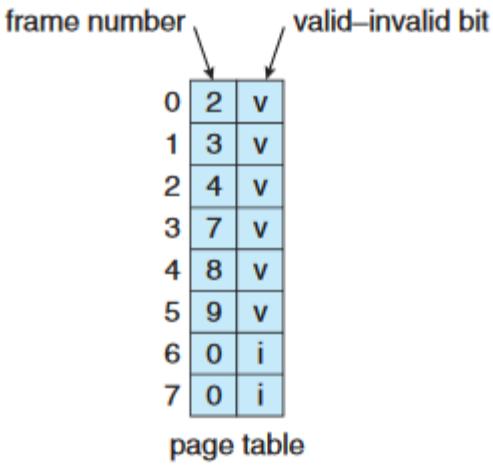
Course Title: Operating System Concepts

Code: CSI-231.

1.No.Qus.Ans.

a)ans: Valid and Invalid:

Valid indicates that the associated page is in the logical address space. Invalid indicates that the associated page is not in logical address space. One additional bit is generally attached to each entry in the page table: a valid–invalid bit. When this bit is set to *valid*, the associated page is in the process's logical address space and is thus a legal (or valid) page. When the bit is set to *invalid*, the page is not in the process's logical address space.



t might be that the system has no way of knowing which pages are valid (0-5) and which ones are invalid (6 and 7) other than *marking* the entries for the invalid pages with an invalid flag.

b)ans: Segmentation Architecture:

Segment Table

A Table that is used to store the information of all segments of the process is commonly known as Segment Table. Generally, there is

no simple relationship between logical addresses and physical addresses in this scheme.

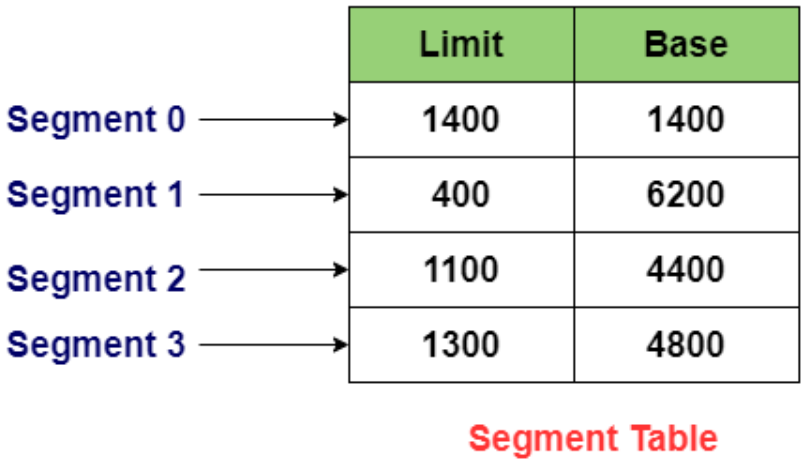
- The mapping of a two-dimensional Logical address into a one-dimensional Physical address is done using the segment table.
- This table is mainly stored as a separate segment in the main memory.
- The table that stores the base address of the segment table is commonly known as the Segment table base register (STBR)

In the segment table each entry has :

1. **Segment Base/base address:** The segment base mainly contain**s** the starting physical address where the segments reside in the memory.
2. **Segment Limit:** The segment limit is mainly used to specify the length of the segment.

Segment Table Base Register(STBR) The STBR register is used to point the segment table's location in the memory.

Segment Table Length Register(STLR) This register indicates the number of segments used by a program. The segment number s is legal if **s<STLR**



- Protection

- With each entry in segment table associate:

- validation bit = 0 - illegal segment

- read/write/execute privileges

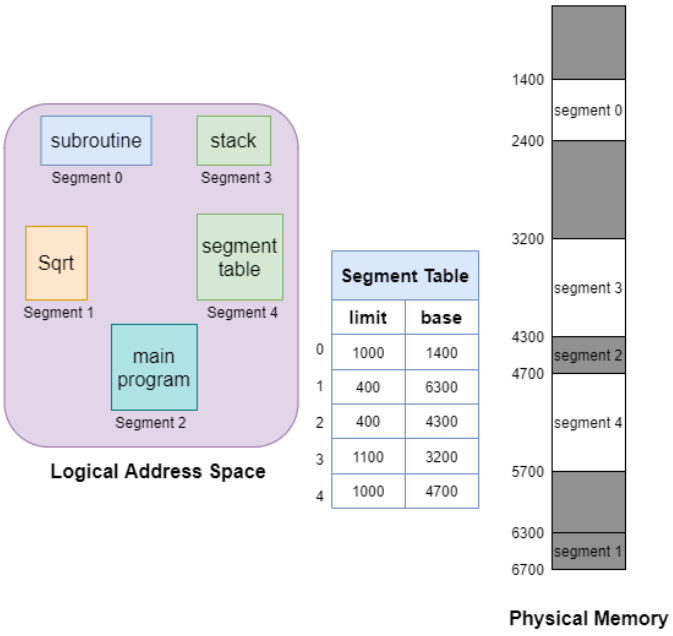
- Protection bits associated with segments; code sharing occurs at segment level

- Since segments vary in length, memory allocation is a dynamic storage-allocation problem

- A segmentation example is shown in the following diagram.

Example of Segmentation

Given below is the example of the segmentation, There are five segments numbered from 0 to 4. These segments will be stored in Physical memory as shown. There is a separate entry for each segment in the segment table which contains the beginning entry address of the segment in the physical memory(denoted as the base) and also contains the length of the segment(denoted as limit).



Segment 2 is 400 bytes long and begins at location 4300. Thus in this case a reference to byte 53 of segment 2 is mapped onto the location 4300 (4300+53=4353). A reference to segment 3, byte 85 is mapped to 3200(the base of segment 3)+852=4052. A reference to byte 1222 of segment 0 would result in the trap to the OS, as the length of this segment is 1000 bytes.

c)ans: Physical Address: In computing, a physical address (also real address, or binary address), is a memory address that is represented in the form of a binary number on the address bus circuitry in order to enable the data bus to access a particular storage cell of main memory, or a register of memory-mapped I/O device. Physical Address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used. The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space. User can never view physical address of program. The user can indirectly access physical address but not directly. Physical address will not change.

3.No.Qus.Ans:

a)ans: Deadlock characterization describes the distinctive features that are the cause of deadlock occurrence

Deadlock is a condition in the multiprogramming environment where the executing processes get stuck in the middle of execution waiting for the resources that have been held by the other waiting processes thereby preventing the execution of the processes. In this content, we will discuss the characteristics that are essential for the occurrence of deadlock. The four conditions that must sustain at the same time to eventuate a deadlock are: mutual experience, hold and wait, no preemption, circular wait.

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

1. Mutual exclusion: In a multiprogramming environment, there may be several processes requesting the same resource at a time. The mutual exclusion condition, allow only a single process to access the resource at a time. While the other processes requesting the same resource must wait and delay their execution until it has been released. The mutual exclusion condition prevents two or more processes to access the same resource at a time. only one process at a time can use a resource.

2. Hold and wait: The hold and wait condition simply means that the process must be holding access to one resource and must be waiting to get hold of other resources that have been acquired by the other processes.

3. No preemption: A process acquiring a resource, cannot be preempted in between, to release the acquired resource. Instead, the process must voluntarily release the resource it has acquired when the task of the process has been completed.

4. Circular wait: The processes must be waiting in a circular pattern to acquire the resource. This is similar to hold and

waits the only difference is that the processes are waiting in a circular pattern. There exists a set {P0, P1, ..., P0} of

waiting processes such that P0 is waiting for a resource

that is held by P1, P1 is waiting for a resource that is held by P2, ..., Pn–1 is waiting for a resource that is held by

Pn, and P0 is waiting for a resource that is held by P0.

b)ans: Introduction:

A deadlock in the operating system is a situation of indefinite blocking of one or more processes that compete for resources. Deadlock involves resources needed by two or more processes at the same time that cannot be shared. We can understand this from the above example, two cars require the road at the same time but it cannot be shared as it is one way. There are four necessary conditions for deadlock. Deadlock happens only when all four conditions occur simultaneously for unshareable single instance resources.

The conditions for deadlock are:

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait.

There are three ways to handle deadlock:

1. **Deadlock prevention:** The possibility of deadlock is excluded before making requests, by eliminating one of the necessary conditions for deadlock. **Example:** Only allowing traffic from one direction, will exclude the possibility of blocking the road.
2. **Deadlock avoidance:** Operating system runs an algorithm on requests to check for a safe state. Any request that may result in a deadlock is not granted.

   Example: Checking each car and not allowing any car that can block the road. If there is already traffic on road, then a car coming from the opposite direction can cause blockage.

3. **Deadlock detection & recovery:** OS detects deadlock by regularly checking the system state, and recovers to a safe state using recovery techniques. **Example:** Unblocking the road by backing cars from one side. Deadlock prevention and deadlock avoidance are carried out before deadlock occurs.

In this article, we will learn about deadlock prevention in OS.

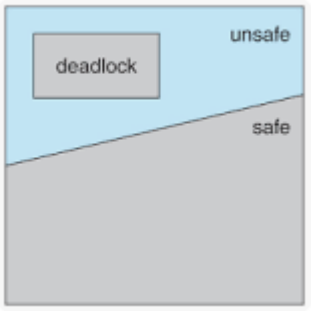
Deadlock prevention is a set of methods used to ensure that all requests are safe, by eliminating at least one of the four necessary conditions for deadlock. Deadlock Prevention in Operating System. A process is a set of instructions. When a process runs, it needs resources like CPU cycles, Files, or Peripheral device access. Some of the requests for resources can lead to deadlock.

Deadlock prevention is eliminating one of the necessary conditions of deadlock so that only safe requests are made to OS and the possibility of deadlock is excluded before making requests. As now requests are made carefully, the operating system can grant all requests safely. Here OS does not need to do any additional tasks as it does in deadlock avoidance by running an algorithm on requests checking for the possibility of deadlock.

C) ans: Safe State:

- A state is **safe** if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock. A state is safe if the system can allocate all resources requested by all processes (up to their stated maximums) without entering a deadlock state.
- More formally, a system is in a safe state only if there exists a **safe sequence**.
    - A sequence of processes $< P_1, P_2, \ldots, P_n >$ is a safe sequence for the current allocation state if, for each P_i the resource requests that P_i can still make can be satisfied by the currently available resources plus the resources held by all P_j, with $j < i$.

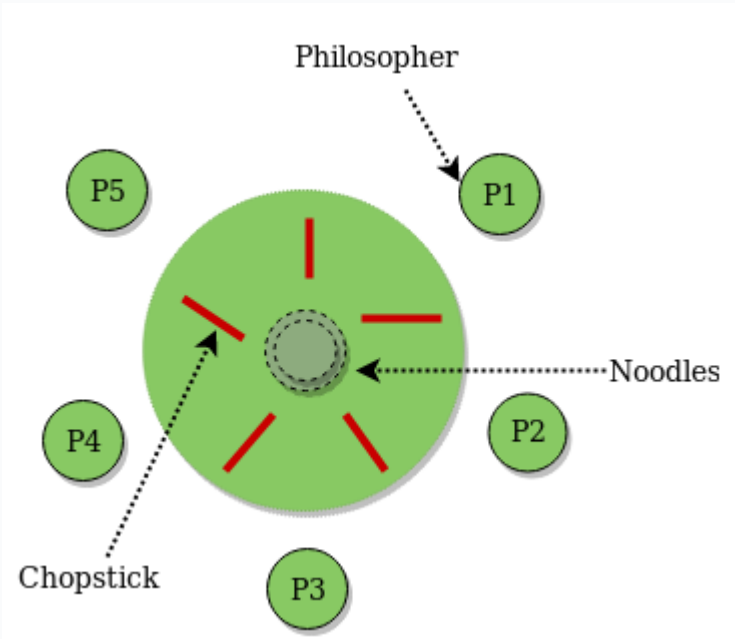
- In this situation, if the resources that P_i needs are not immediately available, then P_i can wait until all P_j have finished.
- When they have finished, P_i can obtain all of its needed resources, complete its designated task, return its allocated resources, and terminate.
- When P_i terminates, P_{i+1} can obtain its needed resources, and so on.



4.No.Qus.Ans:

A) ans:

A solution of the Dining Philosophers Problem is to use a semaphore to represent a chopstick. A chopstick can be picked up by executing a wait operation on the semaphore and released by executing a signal semaphore.**The Dining Philosopher Problem –** The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.



Semaphore Solution to Dining Philosopher –
Each philosopher is represented by the following pseudocode:

```
process P[i]
 while true do
   {  THINK;
      PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);
      EAT;
      PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])
   }
```

There are three states of the philosopher: **THINKING, HUNGRY, and EATING**. Here there are two semaphores: Mutex and a semaphore array for the philosophers. Mutex is used such that no two philosophers may access the pickup or putdown at the same time. The array is used to control the behavior of each philosopher. But, semaphores can result in deadlock due to programming errors.

B) ans: Introduction. Bounded buffer problem, which is also called **producer consumer problem**, is one of the classic problems of synchronization. The bounded-buffer problems (aka the producer-consumer problem) is a classic example of concurrent access to a shared resource. A bounded buffer lets multiple producers and multiple consumers share a single buffer. Producers write data to the buffer and consumers read data from the buffer. In Bounded Buffer Problem there are three entities storage buffer slots, consumer and producer. The producer tries to store data in the storage slots while the consumer tries to remove the data from the buffer storage. It is one of the most important process synchronizing problem let us understand more about the same.

Problem:

The bounded buffer problem uses Semaphore. Please read more about Semaphores here before proceeding with this post here. We need to make sure that the access to data buffer is only either to producer or consumer, i.e. when producer is placing the item in the buffer the consumer shouldn't consume. We do that via three entities –

- Mutex mutex – used to lock and release critical section
- empty – Keeps tab on number empty slots in the buffer at any given time
    - Initialised as n as all slots are empty.
- full – Keeps tab on number of entities in buffer at any given time.
    - Initialised as 0

 The structure of the producer process

do {

// produce an item in nextp

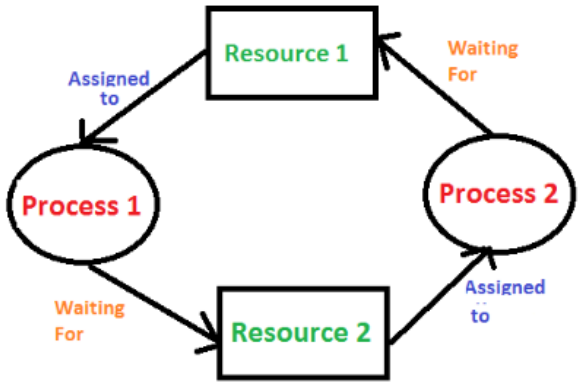
```
wait (empty);

wait (mutex);

// add the item to the buffer

signal (mutex);

signal (full);

} while (TRUE);
```

 The structure of the 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);
```

C) ans: Starvation:

Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time. In heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU. In starvation resources are continuously utilized by high priority processes. Problem of starvation can be resolved using Aging. In Aging priority of long waiting processes is gradually increased.



5.No.Qus.Ans:

A) ans: Multithreading Issues:

Multithreaded programs can sometimes lead to unpredictable results as they are essentially multiple parts of a program that are running at the same time. Complications for Porting Existing Code − A lot of testing is required for porting existing code in multithreading.

Below we have mentioned a few issues related to multithreading. Well, it's an old saying, All good things, come at a price. There are several threading issues when we are in a multithreading environment. In this section, we will discuss the threading issues with system calls, cancellation of thread, signal handling, thread pool and thread-specific data.

Multithreading Issues

1. System Calls
2. Thread Cancellation
3. Signal Handling
4. Thread Pool
5. Thread Specific Data

1. fork() and exec() System Calls:

The fork() and exec() are the system calls. The fork() call creates a duplicate process of the process that invokes fork(). The new duplicate process is called child process and process

invoking the fork() is called the parent process. Both the parent process and the child process continue their execution from the instruction that is just after the fork(). Consider that a thread of the multithreaded program has invoked the fork(). So, the fork() would create a new duplicate process. Here the issue is whether the new duplicate process created by fork() will duplicate all the threads of the parent process or the duplicate process would be single-threaded.

2. Thread cancellation

Termination of the thread in the middle of its execution it is termed as ‘thread cancellation’. Let us understand this with the help of an example. Consider that there is a multithreaded program which has let its multiple threads to search through a database for some information. However, if one of the thread returns with the desired result the remaining threads will be cancelled. Now a thread which we want to cancel is termed as target thread. Thread cancellation can be performed in two ways:

Asynchronous Cancellation: In asynchronous cancellation, a thread is employed to terminate the target thread instantly.

Deferred Cancellation: In deferred cancellation, the target thread is scheduled to check itself at regular interval whether it can terminate itself or not.

3. Signal Handling

Signal handling is more convenient in the single-threaded program as the signal would be directly forwarded to the process. But when it comes to multithreaded program, the issue arrives to which thread of the program the signal should be delivered. The issue of an asynchronous signal is resolved up to some extent in most of the multithreaded UNIX system. Here the thread is allowed to specify which

signal it can accept and which it cannot. However, the Window operating system does not support the concept of the signal instead it uses asynchronous procedure call (ACP) which is similar to the asynchronous signal of the UNIX system. UNIX allow the thread to specify which signal it can accept and which it will not whereas the ACP is forwarded to the specific thread.

4. Thread Pool

When a user requests for a webpage to the server, the server creates a separate thread to service the request. Although the server also has some potential issues. Consider if we do not have a bound on the number of actives thread in a system and would create a new thread for every new request then it would finally result in exhaustion of system resources. We are also concerned about the time it will take to create a new thread. It must not be that case that the time require to create a new thread is more than the time required by the thread to service the request and then getting discarded as it would result in wastage of CPU time.

5. Thread Specific data

We all are aware of the fact that the threads belonging to the same process share the data of that process. Here the issue is what if each particular thread of the process needs its own copy of data. So the specific data associated with the specific thread is referred to as **thread-specific data.** Consider a transaction processing system, here we can process each transaction in a different thread. To determine each transaction uniquely we will associate a unique identifier with it. Which will help the system to identify each transaction uniquely.

6. Security Issues

Yes, there can be security issues because of extensive sharing of resources between multiple threads.

B)ans: Semaphore:

Semaphore is simply a variable that is non-negative and shared between threads. A semaphore is a signaling mechanism, and a thread that is waiting on a semaphore can be signaled by another thread. It uses two atomic operations, 1) Wait, and 2) Signal for the process synchronization.

Properties of Semaphore:

1. It's simple and always have a non-negative Integer value.

2. Works with many processes.

3. Can have many different critical sections with different semaphores.

4. Each critical section has unique access semaphores.

5. Can permit multiple processes into the critical section atonce, if desirable.

C)ans: Worst Fit:

Worst Fit allocates a process to the partition which is largest sufficient among the freely available partitions available in the main memory. If a large process comes at a later stage, then memory will not have space to accommodate it. In this allocation technique, the process traverses the whole memory and always search for the largest hole/partition, and then the process is placed in that hole/partition. It is a slow process because it has to traverse the entire memory to search the largest hole. Since this process chooses the largest hole/partition, therefore there will be large internal fragmentation. Now, this internal fragmentation will be quite big so that other small processes can also be placed in that leftover partition.

“THE END”