victoria university of Bangladesh

Final Assessment

student Name : Md. sohel Rana

student ID : 2119170011

course Title : operating system concepts

course code : CSI-231

program : B.Sc in CSE (Reg)

semester : summer-2022

Batch : 17th

Ans to the Qu:No: 1

(1) (a)

valid bit and invalid bit :-

* "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page.

* "invalid" indicates that the page is not in the process' logical address space.

(1) (b)

The architecture of segmentation :-

Logical address consists of a two tuple :

<segment-number, offset>,

segment table - maps two-dimensional physical addresses; each table entry has:

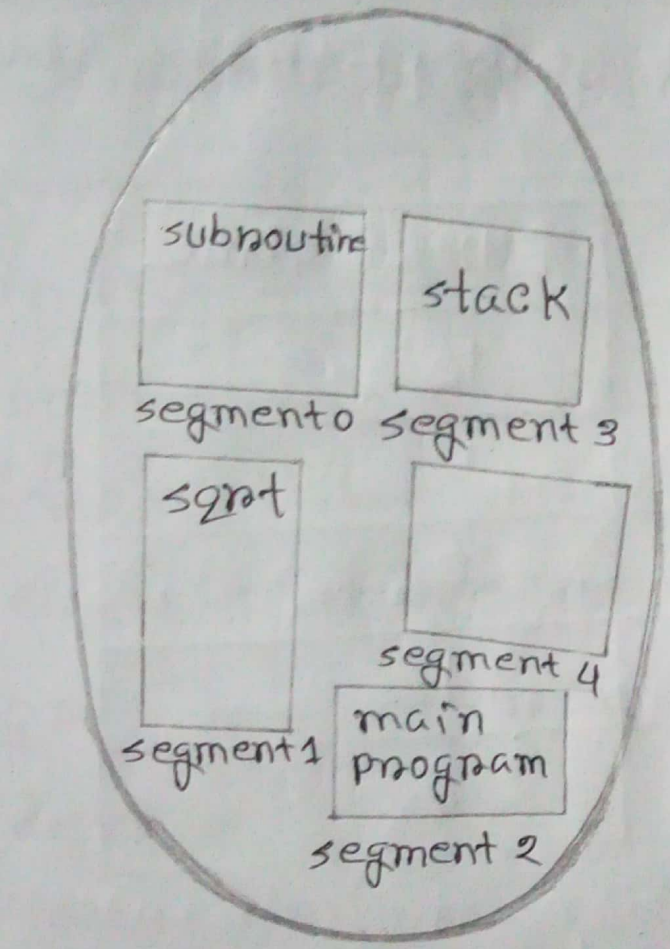
* base - contains the starting physical address where the segments reside in memory.

* limit - specifies the length of the segment.

segment-table base register (STBR) points to the segment table's location in memory.

segment-table length register (STLR) indicates number of segments used by a program; segment number s is legal if s < STLR

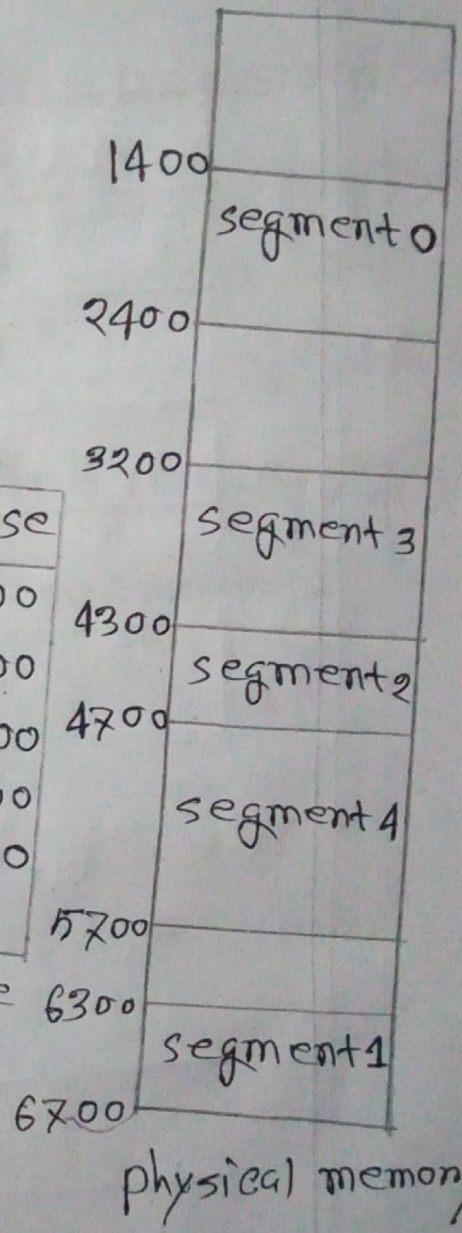
Example of segmentation



logical address space

	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



physical memory

(1) (c)

physical address :-

In computing, a physical 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.

Ans to the Qu: No : 3

(3) (a)

The characterization of deadlock :-

Mutual exclusion : only one process at a time can use a resource.

Hold and wait : A process holding at least one resource is waiting to acquire additional resources held by other processes.

No preemption : A resource can be released only voluntarily by the process holding it, after that process has completed its task.

circular wait : There exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by $P_2, \ldots, P_{n-1}$ is waiting for a resource that is held by P_n, and P_0 is waiting for a resource that is held by P_0.

(3) (b)

Below the steps to prevent deadlock:

Mutual Exclusion : not required for sharable resources; must hold for nonsharable resources.

Hold and wait : must guarantee that whenever a process requests a resource, it does not hold any other resources.

* Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.

* Low resource utilization; starvation possible.

No preemption :-

* If a process that is holding some resources requests another resource that cannot be immediatly allocated to it, then all resources currently being held are released.
* preempted resources are added to the list of resources for which the process is waiting.
* process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

circular wait :- Impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

(3) (0)

Safe state

#When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state

system is in safe state if there exists a safe sequence of all processes.

sequence $<P_1, P_2, \ldots, P_n>$ is safe if for each P_i, the resources that a P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j<i$.

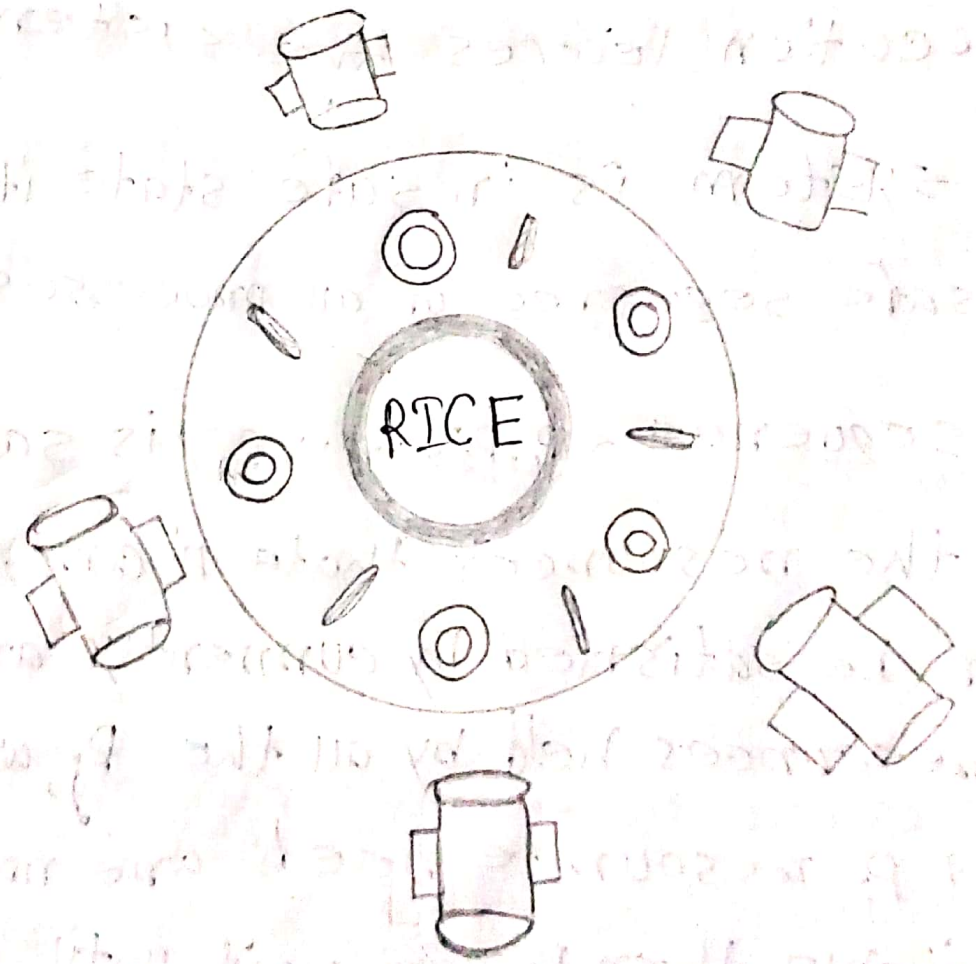
* If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.

* When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.

Ans to the Qu:No:4

(4) @

Dining-philosopher problem Now solve:



#shared data

* Bowl of nice (data set)
* semaphone chopstick[5] initialized to 1

The structure of philosopher i:

```
do {
  wait(chopstick[i]);
  wait(chopstick[(i+1) % 5]);
    // eat
  signal(chopstick[i]);
  signal(chopstick[(i+1) % 5]);
  // think
} while(TRUE);
```

(4)(b)

The Bounded-Buffer problem of synchronization:

N buffers, each can hold one item.

semaphore mutex initialized to the value 1.

semaphore full initialized to the value 0.

semaphore empty initialized to the value N.

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);
```

(4) (c)

starvation

starvation - indefinite blocking. A process may never be removed from the semaphore queue in which it is suspended.

Ans to the QUESTION: 5

(5)(a)

Multithreading Issues

Thread cancellation :-

Thread cancellation means terminating a thread before it has finished working. There can be two approaches for this, one is Asynchronous cancellation, which terminates the target thread immediately. The other is deferred cancellation allows the target thread to periodically check if it should be cancelled.

Signal Handling :-

signals are used in UNIX systems to notify a process that a particular event has occurred.

Now in when a multithreaded process receives a signal, to which thread it must be delivered. It can be delivered to all or a single thread.

fork() system call :-

fork() is a system call executed in the kernal through which a process creates a copy of itself. Now the problem in multithreaded process is, if one thread forks, will the entire process be copied or not?

security Issues :-

Yes, there can be security issues because of extensive sharing of resources between multiple threads. There are many other issues that you might face in a multithreaded proces, but there are appropriate solutions available for them.

(5) (b)

semaphore

In computer science, a semaphore is a variable or abstract data type used to control access to a common resource by multiple threads and avoid critical section problems in a concurrent system such as a multitasking operating system. semaphore are a type of synchronization primitive.

properties of semaphore

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

#works with many processes.

#can have many different critical sections with different semaphores.

Each critical section has unique access semaphores.

(5) (c)

Worst Fit

Allocate the largest hole; must also search entire list

* produces the largest leftover hole

First-fit and best-fit better than worst-fit in terms of speed and storage utilization (according to simulations).