02/03/2021 CSPC43 - OS 106119100

Cycle Test - 1 Rajneesh Pandey

Question ① :

Virtual machine : They are virtual environment. They abstract the hardware of our personal computer such as CPU, disk, drives, memory NIC etc, into many different execution environments as per our requirements, hence giving the service of each executn in a single computer.

Performance :
Lets suppose Virtual Box' is running in system then,
virtual machine approach doesnot provide addition functionality (system calls and a file system) but it only provides interface that is same as basic hardware.

Question ② :

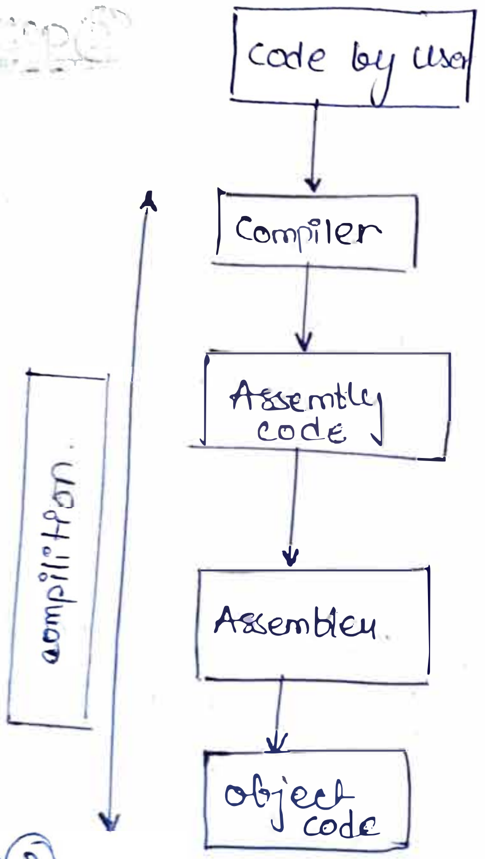
Formation of Processs from program.

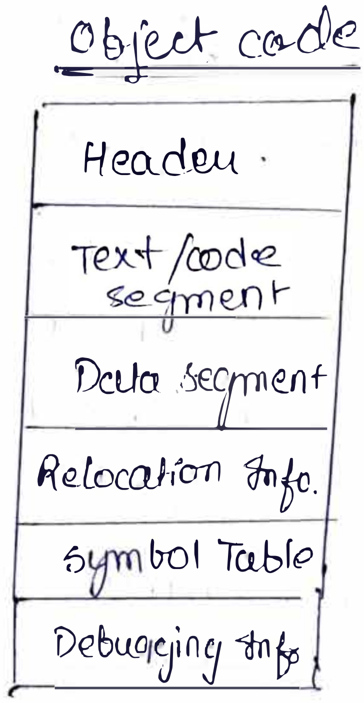
Process: A Process is a program in execution is called Program.

Step① : when we write code and save in main memory. and call compiler it convert into assembly code. Then assembler convert it to object code.

The object code produced after all the steps have,

1). Header
2). Text/code segment
3) Data segment
4) Relocation Information
5) Symbol table
6) Debugging Information.





Step ②

Now, Linkers work in 2 phases.

Phase - I

uses two

1. Segment Table
2. Symbol Table

Segments are loaded.

Phase - II

Here in this phase the linker actually resolves all the un-resolved symbol in symbol Table

Step ③

Now Loader program them into main memory in contigeous and/or non-contigeous memory allocation.

Question (3)

The Operating system maintains the following important process scheduling queues.

(1) Job queue:

This queue keeps all the process in the system.

(2) Ready queue:

This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue

(3) Device queue:

The process which are blocked due to unavailability of an I/O devices constitute this queue

Short term Scheduler:

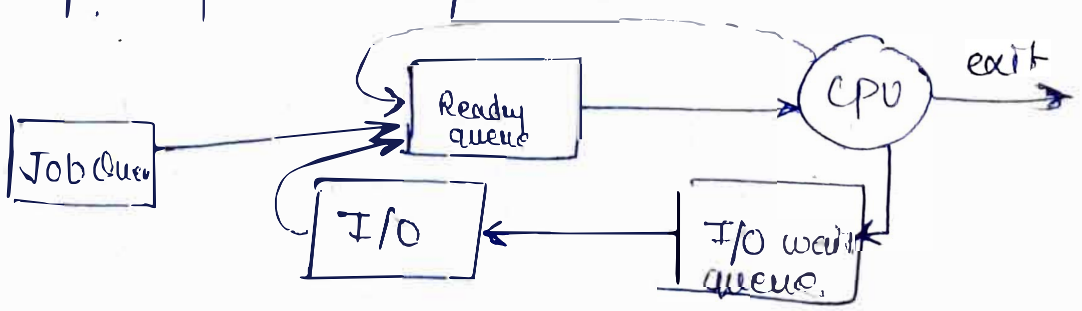
→ they are also known as dispatcher.

→ used for context switching

→ moves process from Ready to Running states

→ Assigned Picked process to processor.

→ can preempt some process.



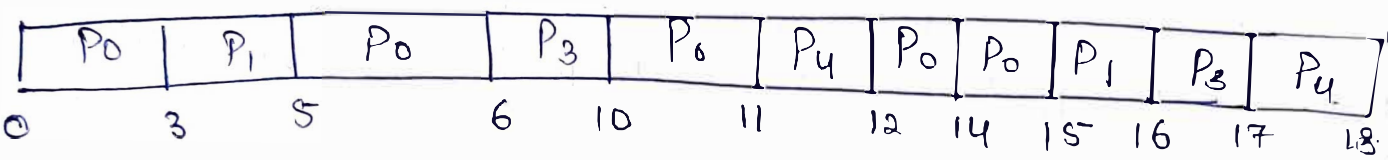
Question (4)

(i) Longest Remaining Time first (LRTF)

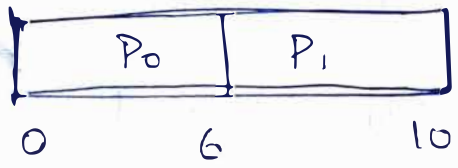
Process ID	Arrivi Time	Burst Tim
0	0	6
1	3	4
2	5	2
3	6	5
4	8	1

so, process · (Grant chart)

Po — ~~3~~ ~~2~~ 1
P_1 — ~~2~~ 1
P_2 — 2



ii) HRNN Ratio of $\left(\frac{\text{waite time + job.time}}{\text{Jobtime}}\right)$

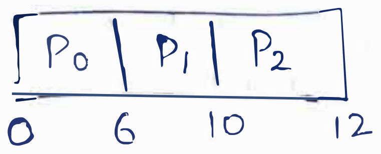


$P_1 = \frac{((6-3)+4)}{4} = 7/4 = 1.75$

$P_2 = \frac{((6-5)+2)}{2} = 3/2 = 1.5$

$P_3 = \frac{(6-6)+5}{5} = 1$

} P_1 includes



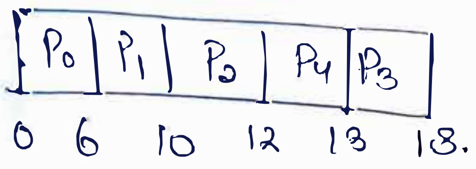
$P_2 = \frac{(10-5)+2}{2} = 7/2 = 3.5$

$P_3 = \frac{(10-6)+5}{5} = 1.8$

$P_4 = \frac{(10-8)+1}{1} = 3/1 = 3$

} P_2 add up

Naw, ans



Question (5)

Scheduler Activation attempts to provide the best of both kernel and user-level threads, while avoiding the disadvantage of both.

i) each application is provided with a virtual multiprocessor.

ii) Each application has its own thread schedular

(iii) This kernal is told how many threads an application would like to run.

(iv) The application needs is told when the kernel gives physical process from it.

Problemes with user-level threads:

→ user level threads are not directly mapped to physical processor.

→ A process that has many user-level threads running is only allocated one processor and the threads share that processor.

→ Operating system activity such as multiprogrammi I/O, and page faults.

-The equivalance between virtual and physical processors.

Question (6)

P1 → produce random number.

P2 → square of that random number if number is even.

P3 → display sum of the sq.

semaphore

```
Struct Semaphore {
        int count;
        Queue q;
    }
```

(P1) ⇒

```
int random_number( ){
    int randumNum = math.random( );
    return randumNum.
  }
```

P2 :

```
int square(
      int randumNo)
{   wait(s)
    if (randNum%2
          ==0)
  return (randNum *
             randNum)
  }
```

(P3) ⇒

```
int printsq.(int num)
{   wait(s)
    cout << num <<endl;

  }
```

now, using semaphore.

we have to ~~w~~write.

using semaphore as synchronization primitive.

```
void wait()
{ s.count --;

  if (s.count < 0)
  {
    ① Add process p
      into queue
    ② sleep(p);
  }
}
```

```
void signal()
{
  s.count++;
  if (s.count <= 0)
  {
    ① Remove a process
      p from q.
    ② wakeup(p);
  }
}.
```