2nd Sessional Schedule

Subject - Operating System

{ Section - C }

[Answer - 3 (b)]

Semaphore – A Semaphore is a variable that indicates the number of resources that are available in a system at a particular time & this Semaphore variable is generally used to achieve the process synchronization. It is denoted by 'S'.

It has uses two function

i) wait()
ii) Signal()

wait() fun" is used to decrement of the value semaphore variable S.

```
wait (S) {
  while (S==0);
    S--;
  }
```

Signal() fun" is used to increment the value of a semaphore.

```
signal (S) {
      S++;
  }
```

Types of Semaphore – Two types of Semaphore:

i) Binary Semaphore – The value of the semaphore variable will be 0 or 1. Initially, the value of Semaphore variable is set to 1 & if some process wants to use some resources then the wait() is called & the value of the Semaphore is changed to 0 from 1.

ii) Counting Semaphore – The Semaphore variable is initiallized with a num of resources available. After whenever a process need some resources, then the wait() is called & the value of the 'S' is decreased by 1.

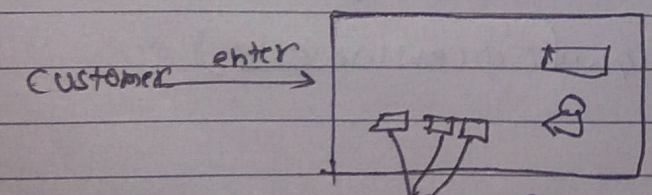
Answer - 3 (c)

Sleeping Barber Problem

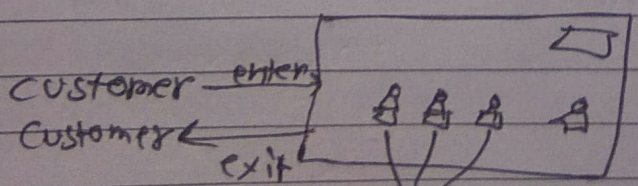
- It is a classical inter process communication & synchronization problem b/w multiple O.S. processes.
- The problem is analogous to that of keeping a barber working when there are customers resting when there are none, & doing so in an orderly manner.

Solution - Many possible solution are available. The key element of each is a mutex, which insures that only 1 of the participants can change state at once.

The barber must acquire this mutual exclusion before checking for customers & release it when they begin either to sleep or haircut.



Chair is available then customer wait



Chair is not available for waiting

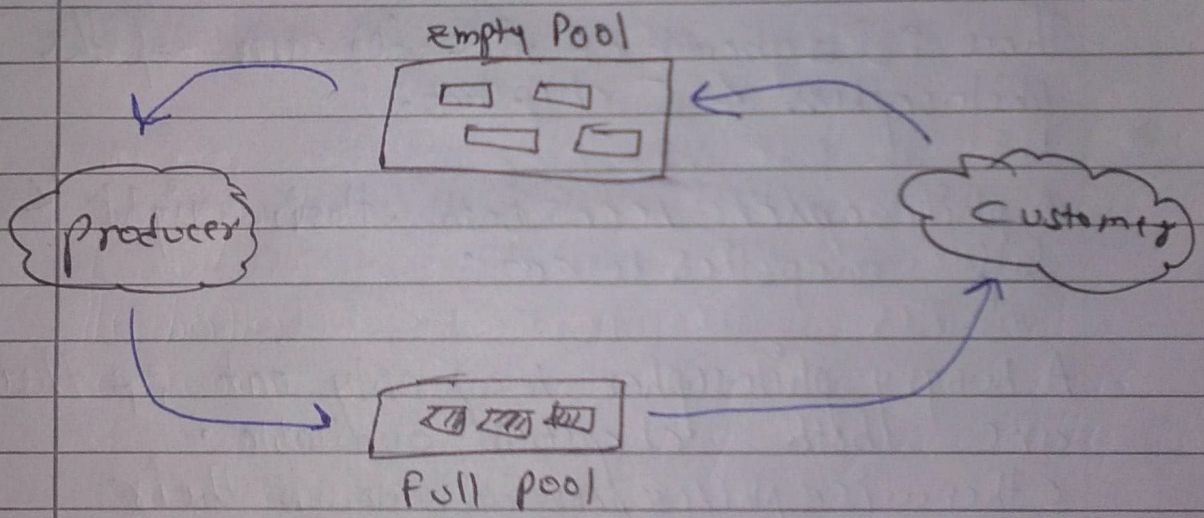
```
Semaphore barberReady = 0 ;
Semaphore accesWRSeats = 1 ;

Semaphore custReady = 0;
int numOffreeWRSeats = N;
def Barber();
    while true;
        wait(custReady)
        wait(accesWRSeats)
        numoffreeWRSeats += 1

    Signal(barberReady)
    Signal(accesWRSeats)

def customer();
    while true;
        wait(accesWRSeats);
        if numoffreeWRSeat > 0;
        numoffree WRSeats -= 1
        Signal(custReady)
        Signal(accesWRSeats)

        wait(barberReady)
        else:
            Signal(accesWRSeats)
```

(Section-B)

[Answer – 2(b)]

Bounded - Buffer Problems

It is a classic example of concurrent access to a shared resources.
A bounded buffer lets multiple producers & multiple consumers share a single buffer.
Producers write data to the buffer & consumers read data from the buffer.



[Answer – 2(C)]

Two basic models of inter process communication are there –

i) Shared memory
ii) Message passing

A region of memory that is shared by co-operating processes is established, in

shared memory model. process can then exchange their information by reading & writing data to the shared region.

[Answer - 2 d]

Dining Philosophers Problem

It states that there are 5 philosophers sharing a circular table & they eat & think alternatively.

There is a bowl of rice for each of the philosophers & 5 chopstics.

A philosopher needs both their right & left chopstics to eat.

A hungry philosopher may only eat if there are both chopstics available.
Otherwise philosopher puts down their chopstics & begin thinking again.

[Answer - 2 (e)]

Peterson's Solution

- It is a classical software based solution to the critical section problem.
- In peterson's solution, we have 2 shared variables.
- boolean flag[i] : initialized to FALSE, initially no one is interested in entering the critical section.
- int turn : The process whose turn is to enter the critical section.

```
do {
    flag[i] = TRUE;
    turn = j;
    while ( flag[j] && turn == j);

        [critical section]

    flag[i] = FALSE;
        [remainder section]
}
while (TRUE);
```

The critical section is a code segment where the shared variables can be accessed.
An atomic action is required in a critical section i.e. only one process can execute in its C.S at a time.

[Section - A]

[Answer - 1 (a)]

Communication of process is important because provide environment that allows proces operation.

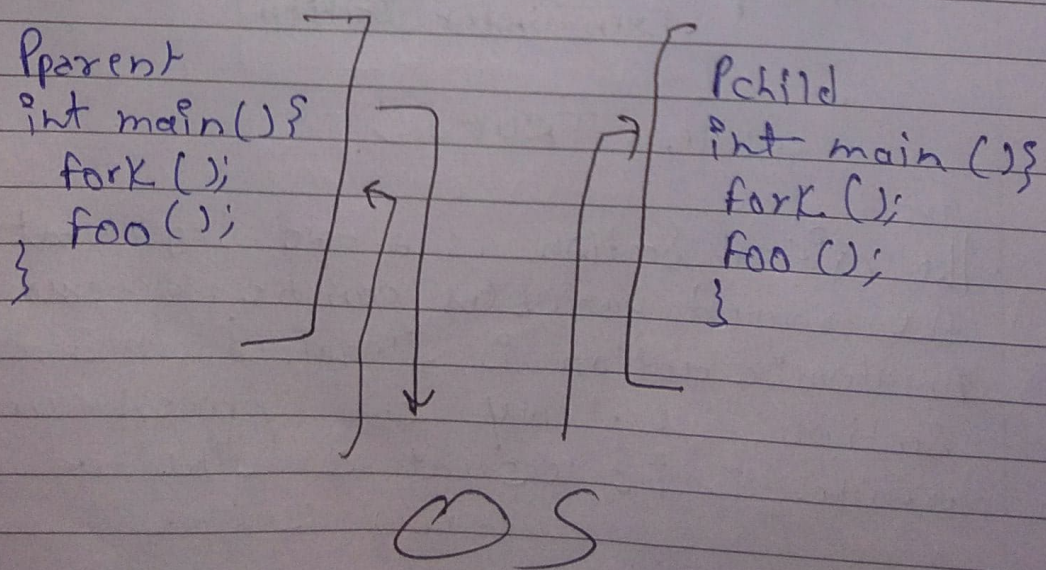
(i) Information sharing
(ii) Computation speedup
(iii) Modularity
(iv) Convenience

[Answer - 1 (b)]

working of fork() system call

* It is use for creating new process which called child process which run concurrently with the process that makes the fork() call.

[Answer - 1 (c)]



[Answer - 1 (d)]

When a process creates a new process, the identify to the newly created process is passed to its parent.

When a parent process is terminating, then all of its children process is also terminated.

Answer - 1 (e)

Concurrency is a way to execute more than 1 thing at the same time. In an application concurrency refers to multiple tasks running at the same time Concurrency.

critical Section

```
do {
    entry section
    critical section
    exit section
    remainder section
} while (TRUE);
```

Solution

- Mutual Exclusion
- Progress
- Bounded waiting.