lecture-15
BDu

classmate
Date 22/02/18
Page ____

Monitors

- A high level abstraction that provides a convenient and effective mechanism for process synchronization.

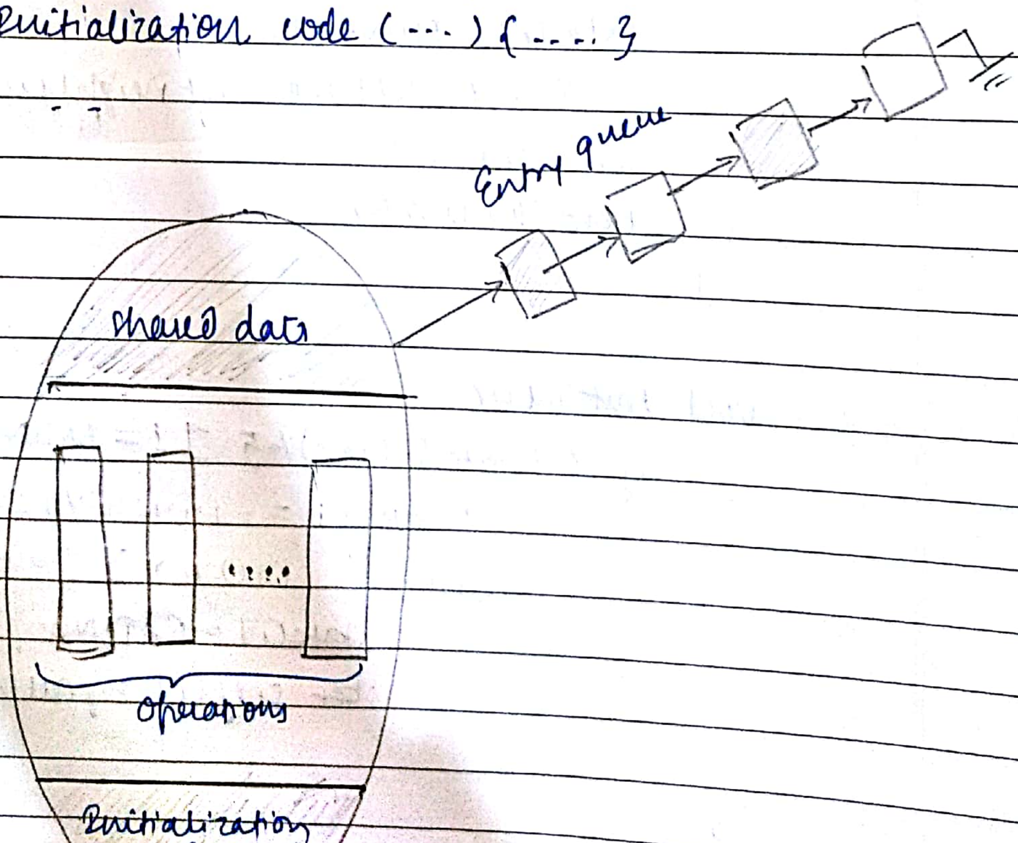
- Only one process may be active within the monitor at a time:

```
monitor monitor-name
{
        //shared variable dec
        ~~procedure~~ procedure P1(...) {....}

              ...

            procedure Pn(...) {...}
             initialization code (...) {....}

              ..
          }
 }
```



schematic view of a monitor

Solution to dining philosophers

```
monitor DP
{
    enum { THINKING, ~~EA~~ HUNGRY, EATING } state [5];
    condition self [5];

    void pickup (int i) {
        state[i] = HUNGRY;
        test (i);
        if (state[i] != EATING) self[i]. wait;
    }

     void putdown (int i) {
        state [i] = THINKING;
            // test left and right neighbours.
        test ((i+4) % 5) ;
        test ((i+1) %5);
     }

    void test(int i){
        if ( (state [ (i+4) %5 ] != EATING ) &&
                (state[i] = = HUNGRY) &&
               (state [ (i+1) % 5 ] != EATING) ) {
                    state[i] = EATING ;
                   ~~tes~~ self[i]. signal();
           }
    }
    initialization_code() {
        for (int i=0 ; i<5 ; i++)
            state [i] = THINKING;
    }

}
```

Q. Consider two concurrently running processes. Process P1 with a statement S_1 process P_2 has a statement S_2. Suppose that we require the statement S_2 is executed only after S_1 has completed. Now we put a semaphore "synch" in such a way so that there order is maintained and also mention its initial value.

Initially
sync=0

```
wait (synch)
{
   while (sync<=0);
   sync--;
}
```

```
do{
      S1;
      signal(sync);
}while(1);
```

```
do{
      wait(sync);
      S2;
}while(1);
```

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

System model

- System consists of resources : CPU cycles, memory space, I/O
- Each resource R_i has W_i instances.
- Each process utilizes a resource as follows:
  - request
  - use
  - release

Deadlock can arise if four conditions hold simultaneously:

- Mutual exclusion - only one process can use a resource at ~~at~~ a time.
- Hold and wait - a process holding at least one resource is waiting to acquire additional resources held by other
- 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 $\{P_0, P_1, \ldots P_n\}$ 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, ... P_{n-1} is waiting for P_n & P_n for P_0.