SHANTHAN REDDY
CB.EN.U4CSE19459

THREAD SYNCHRONIZATION :-

It is concurrent execution of two (or) more threads that share critical resources. Synchronisation of threads help in avoiding conflicts regarding critical resources. otherwise, conflicts may arise, when parallel running threads attempts to modify a common variable at the same time.

* critical section :- The region of a program that try to access shared resources and cause race condition.

* Race condition :- It typically occurs when two (or) more threads try to read, write and possibly make the decisions based on the memory they are accessing concurrently.

Pseudocode :-

```
do
  {
  //entry section → wait( ); here request are processed for entry into
                              critical section.
     *critical section*
  //exit section → signal( ); here removes locks on critical section.
     -remaining section
  }
  while (True)
```

Solution for critical section:-

solution for a critical section Problem must satisfy following cond:-

(i) Mutual exclusion:- out of group of threads, only one thread can be in its critical section at a given point of time.

(ii) Progress:- If no thread is in the critical section, and if one or more thread wants to execute their critical section then only one of these threads must be allowed to get in.

(iii) Bounded wait:- After a thread makes reauests for getting into critical section, there is limit for how many process may get into the critical section, before threads reauest is granted.

widely used methods for critical section problem:-

* Petrsons solution.
* Mutex lock.
* semaphores.

SEMAPHORE:- It is a signaling mechanism, and a thread that is waiting on semaphore, can be signaled by another thread. There are two types of semaphores.

(i) Counting semaphore.

(ii) Binary semaphore.

All the semaphores make use of two atomic operations (i) wait and (ii) signal.

System calls in windows :-

(i) Create semaphore (

LPsecurity - attribute ,
lInitial Count,
lMaximum Count,
lPName

);

LPsecurity attribute :- it is a security attribute, if NULL handle can't be inherited by its child.

lInitial count :- must be greater than zero and less than or equal to max count. signaled state → greater than zero, Non-signaled state = 0

lMaximum count :- max count of semaphore object.

lPName :- name of semaphore object.

(ii) Wait for single object (

hHandle ;
dwMilliseconds

);

hHandle :- A handle to object.

dwMilliseconds :- time-out interval in milliseconds

(iii) WaitForMultipleObjects (

nCount,
lpHandle,
bWaitAll
dwMilliseconds,

);

nCount :- max Numbers of objects handles in array pointed to lpthreads

lpthread :- array of object handles.

bWaitAll :- If TRUE, returns when all objects are signaled, If FALSE, returns when any one objects is signaled.

dwMilliseond :- time-out interval in Millisecond.

(iv) ReleaseSemaphore (

hSemaphore,
lReleaseCount,
lpPreviousCount.

);