RAMA UNIVERSITY

www.ramauniversity.ac.in

FACULTY OF ENGINEERING & TECHNOLOGY

BCA-307 Operating System

Lecturer-12

Manisha Verma
Assistant Professor
Computer Science & Engineering

Process Synchronization

- Mutex Locks
- Semaphores

Mutex Locks

- Previous solutions are complicated and generally inaccessible to application programmers
- OS designers build software tools to solve critical section problem
- Simplest is mutex lock
- Protect a critical section by first acquire() a lock then release() the lock
  - Boolean variable indicating if lock is available or not
- Calls to acquire() and release() must be atomic
  - Usually implemented via hardware atomic instructions
- But this solution requires busy waiting
  - This lock therefore called a spinlock

acquire() and release()

```
acquire() {
    while (!available)
        ; /* busy wait */
    available = false;
 }
release() {
    available = true;
 }
do {
 acquire lock
    critical section
 release lock
   remainder section
} while (true);
```

Semaphore

Synchronization tool that provides more sophisticated ways (than Mutex locks) for process to synchronize their activities.

Semaphore *S* – integer variable can only be accessed via two indivisible (atomic) operations.

wait() and signal().

Originally called P() and V()

Definition of the wait() operation

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

Definition of the signal() operation

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

Semaphore Usage

•Counting semaphore – integer value can range over an unrestricted domain

•Binary semaphore – integer value can range only between 0 and 1 same as a mutex lock can solve various synchronization problems.

Consider P_1 and P_2 that require S_1 to happen before S_2

Create a semaphore “synch” initialized to 0

```
P1:
   S1;

   signal(synch);

P2:

   wait(synch);

   S2;
```


Can implement a counting semaphore *S* as a binary semaphore.

Implementation

- Must guarantee that no two processes can execute the wait() and signal() on the same semaphore at the same time

Thus, the implementation becomes the critical section problem where the wait and signal code are placed in the critical section could now have busy waiting in critical section implementation

- But implementation code is short
- Little busy waiting if critical section rarely occupied

- Note that applications may spend lots of time in critical sections and therefore this is not a good solution

With each semaphore there is an associated waiting queue.
Each entry in a waiting queue has two data items:

- value (of type integer)
- pointer to next record in the list

Two operations:

- block – place the process invoking the operation on the appropriate waiting queue
- wakeup – remove one of processes in the waiting queue and place it in the ready queue

```
typedef struct{

   int value;

   struct process *list;

   } semaphore;
```

Implementation with no Busy waiting

```
wait(semaphore *S) {
   S->value--;
   if (S->value < 0) {
      add this process to S->list;
      block();
   }
}

signal(semaphore *S) {
   S->value++;
   if (S->value <= 0) {
      remove a process P from S->list;
      wakeup(P);
   }
}
```

Problems with Semaphores

Incorrect use of semaphore operations:

signal (mutex) …. wait (mutex)

wait (mutex) … wait (mutex)

Omitting of wait (mutex) or signal (mutex) (or both)

Deadlock and starvation are possible.

MCQ

Semaphore is a/an _____ to solve the critical section problem.
A. hardware for a system
B. special program for a system
C. integer variable
D. None of these

if the semaphore value is negative :
A. its magnitude is the number of processes waiting on that semaphore
B. it is invalid
C. no operation can be further performed on it until the signal operation is performed on it
D. All of these

The wait operation of the semaphore basically works on the basic _____ system call.
A. stop()
B. block()
C. hold()
D. wait()

What will happen if a non-recursive mutex is locked more than once ?

A. Starvation
B. Deadlock
C. Aging
D. Signaling

A semaphore …..

A. is a binary mutex
B. must be accessed from only one process
C. can be accessed from multiple processes
D. None of these