gradeup

Process Management Part- 5

Content :-
1. Semaphores
2. Monitors

Semaphores :-
It is an operating system tools, used for process synchronization. Basically semaphores are of two types :-
1. Binary Semaphores
2. Counting Semaphores

Binary Semaphores :- It is an integer variable, which can accessed by a cooperating process , through the use two primitives viz. "wait" and "single". A binary semaphore is initialized by the Operating System to 1 . and it can assume only one of two values (either 1 or 0).
Int s = 1 /* Let S be a Binary semaphore, initialized to 1 */

Wait : The first process invoking will make the semaphore value 0 and proceed to enter in the critical section. If the subsequent process ,say P_1, is requesting to entry into its critical section, when another cooperating process is still executing in its critical section , then P_1 will be made to wait. So , at a time only one of the waiting processes is permitted to enter its critical section. A waiting process P_1 will repeatedly check the value of semaphore, till it is found to be 1.

```
Void wait(int *S)
{
 while(*S ==0 ) /* keep looping here as long as S is 0 */
 *S--;
}
```

Signal : This primitive is invoked by a cooperating process, when it is executing from the critical section. The operation comprises of incrementing the value of semaphore to 1 , to facilitate one of the waiting processes to enter its critical section.
Binary semaphores is also known as spin locks.
It meet the mutual exclusion and Progress requirement but not meet the Bounding waiting.

Limitation of Binary Semaphores:
1. Since the selection of waiting process in for entry into the critical section is absolutely random , it may happen that some of the cooperating processes may manage to sneak into their critical sections again and again , while others may keep waiting (in worst case may be forever) .So , a solution using binary semaphores , does not meet the requirement of Bounded Waiting
2. A process Waiting to enter its critical section will perform busy-waiting thus wasting CPU cycle. Due to this reason the binary semaphore are called spinlocks.

Advantage of using binary semaphore
The implementation of binary Semaphores of force is extremely simple.

Counting semaphore
The counting semaphore are free to inherit limitation of binary semaphore.
- When Semaphore count = 1 it implies that no process is executing in its critical section and no processes waiting in semaphore queue.

- when semaphore Count = 0 it implies that 81 processes executing in critical section but no process is waiting in semaphore queue .
- when semaphore Count = N it implies that one process is executing in the critical section and and N process are waiting in the in semaphore queue .
- when a process is waiting in a semaphore queue .,it is not performing any busy waiting. It is any off the statei.e. "Blocked" state or "Binary" state or "Waiting" state.
- When a waiting process/job is selected for entry into its critical section. It is transferred from "Blocked" stateto "Ready" state.
- When a waiting process is selected for entry into the critical section. It is shiftedfrom "Blocked" state to "Ready" state.

Advantage of Counting Semaphore visa-a-versa Binary Semaphores :-
a. Since ,the waiting processes will be permitted to enter their critical section in a FCFS order, the requirement of bounded waiting is fully met.
b. A waiting process does not perform busy waiting , thus saving some CPU cycles.

Disadvantages of Counting Semaphore visa-a-versa Binary Semaphores:-
a. A counting semaphore is more complex to implement, since it involves implementation of FCFS queue.
b. Additional context switches when a process is not able to enter its critical section, it would involve two context switches (a) from "Running" state to "Waiting" state and (b) from "waiting" state to "Ready" state. And then only, the process would enter its critical section. These context switches will involve certain overhead.

Classical Synchronization Problems:-
1. **The Bounded Buffer Problem :-** Suppose , there is a producer process ,which produce some data-time, that are being consumed by a consumer process. The producer is producing the data-items asynchronously. It is possible that, at time , the provider may produce data-time at a rate faster than the consumer can consume. So, we a limited(bounded) number of buffers to hold the data items ,which are awaiting consumption. So, the message communication is performed through shared memory .

2. **Dining Philosopher's Problem:-** Dining philosopher's problem is the classical problem of synchronization.

The problem stated as follows :-
a. There are N philosophers, sitting around a round dining problem
b. There are N plates, placed on the table each plate in front of a philosopher
c. There are N chopsticks, placed between the plates.
d. There is a bowl of spaghetti placed at the centre of the table .
e. Whenever, a philosopher feels hungry, he will attempt to pick up two chopsticks, which are shared with the nearest neighbours . If any of hisneighbourhappens to be eating at that time, the philosopher has to wait.
f. When, the hungry philosopher is able to get two chopsticks , he pours spaghetti in his plate and eats.
g. After, he finished to eat he places the chopsticks back on the table and starts to think. Now, the chopstick are available for his neighbours.

Limitation of this Algorithm :- It is possible that all philosophers may feel hungry almost at the same time. They all will pick up their perspective left chopsticks simultaneously. Now ,

all will keep waiting for their respective right chopstick forever . It is a deadlock condition and thus requirement of progress not meet.

A Deadlock free Algorithm :- The algorithm can be modified such that an even-numbered philosopher(i.e.0,2,4,6…) attempts to pick left chopstick first and then right one. Whereas , an odd numbered philosopher(1,3,5…) attempts to pick the right chopstick first and then left one.

3. **Sleeping Barber Problem :-** The problem can be stated as follows
a. A hair cutting saloon has n chairs in its waiting room and 1 chair in the barber's cabin for giving hair-cut.
b. If there is no customer arrives at the saloon , the barber goes to sleep.
c. When, a customer arrives at the saloon one of the following happens :-
i. If there is no customer in the saloon, the customer walks into the hair-cutting cabin, wakes up the barber and starts getting the hair cut
ii. If a customer is already getting hair-cut in the barber cabin, then the customer looks for an empty chair in the waiting room. If a chair is empty then the customer occupies an empty chair as per his order of arrival and waits for his turn. But if no chair is empty in waiting room, then the customer walks away, without getting a hair-cut.
d. When a customer, finishes getting the hair cut , he leaves the barber's cabin and sends the next waiting customer(as per FCFS order) to the barber's cabin and then leave the saloon.

4. **Readers and Writers Problem :-** The problem can be stated as follows:
a. A number of cooperating processes are accessing a shared file or database.
b. Some of the cooperating processes need to only read the data. These processes cab be referred as READERs.
c. Other cooperating processes needs to modify the data. These can be referred as WRITERS.
d. It is to be ensured that when the WRITER is in its critical section, no other READER or WRITER can execute in critical section.
e. When a READER is in its critical section then other READERs(not WRITERS) can also enter their critical sections.

Programming Error in the use of Semaphores:-
a. **Violation of mutual exclusion :-** consider the following programming error:
P_1 :
Signal(&S); /*Error*/
<Critical Section>

When P_1 performs signal on the semaphore, before its entry into the critical section, it clears the path for two more cooperative processes to enter their critical sections simultaneously, thus violating the requirement of mutual exclusion .

b. **Blocking critical section entry of cooperating processes forever :-** consider the following programming error :
P_2:
Wait(&S); /*Error*/
<Critical Section>
/* signal(&S) has been omitted by mistake */
<Remainder Section>

So, the use of semaphores is highly prone to programming errors, that may cause violation of mutual exclusion of may block the critical-section entry of cooperating processes forever.

Monitors:-

Monitor is a high level tool for process synchronization. The main limitation of semaphores is the rigidity of the wait and signal operations. In the case of monitors , the functions to access the global variables are programmer defined. The global variables, which need accessing by all cooperating processes, are encapsulated within the monitor , along with functions needed to access them. The global variables, defined within a monitor can be accessed only through the functions, defined within the monitor . The OS will ensure mutual exclusion and bounded waiting of the cooperating processes by ensuring that at a time only one of the cooperating processes can be active inside a monitor.

gradeup SUPER

Gradeup UGC NET
Super Superscription

Features:

1. 7+ Structured Courses for UGC NET Exam
2. 200+ Mock Tests for UGC NET & MHSET Exams
3. Separate Batches in Hindi & English
4. Mock Tests are available in Hindi & English
5. Available on Mobile & Desktop

Gradeup Super Subscription, Enroll Now