Process synchronization

Chapter - 4

Process Synchronization

- **Process**

  Program in execution

- **Synchronization**

  The concurrent processes are sharing the same resources ,they need to coordinate and synchronize among themselves.

Race condition

- Race condition is a situation ,where several processes access and manipulate the same data concurrently, and the result of the execution depends on the particular order in which the access takes place.

The general structure of a process

```
do
{
    entry section
    critical section
    exit section
    remainder section
}
While (true)
```

- **Entry section**

  Each process request the permission to enter the critical section.

- **Critical section**

  It is a section of code where the process may be changing common variables, common tables and files.

  If one process is executing in its critical section, no other process is allowed to execute its critical section

- **Exit section** : This code releases the access.
- **Remainder section** : The remainder section refers to the rest of the code.

Two commonly used instructions in process synchronization are

- `Test and set instruction
- swap instruction

Test and set instruction

- It is used to avoid the interruption.

```
boolean TestAndSet(int lock)
{
 if(lock==0)
 {
  lock=1;
  return false;
 }
else
  return true;
 }
```

SWAP Instruction

- The swap() instruction is executed automatically and it operates on the contents of two words.
- This instruction exchanges the contents of a register with that of a memory location.

```
void swap ( int register, int memory)
{
   int temp;
   temp = memory;
   memory = register;
    register = temp;
}
```

Semaphores

- A semaphore is an integer variable with non-negative values which can be accessed only through two standard atomic functions :

  wait() and signal()

- Wait(S)

It decrements the semaphore value .If the semaphore value becomes negative, then process executing the wait() is blocked.

```
Wait(S)
{
S--;
While (s<=0);
// do nothing
}
```

- Signal(S)

This operation increments the value of its semaphore S. If the value is not positive, then a process blocked by a wait() operation is unblocked.

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

Types of semaphores

- Binary Semaphores
- Counting Semaphores

Binary semaphore

- It is a semaphore which can only take of 0 and 1.
- It is also known as mutex locks (mutual exclusion locks).
- Binary semaphores can be used to solve the critical section problem.

Counting semaphores

- First the semaphore is initialized equal to the number of resources available.
- When the count for semaphore becomes 0, all the resources are in use.

Classical problems(IPC problems) of synchronization

- Producer-consumer problem(or bounded-buffer problem)
- Readers-writers problem.
- Dining Philosophers problem.

Producer-consumer problem

- It is also known as bounded-buffer problem.
- The problem describes two processes, the producer and the consumer who share a common fixed size buffer.
- The producer’s job is to generate a piece of code ,put it into the buffer.
- The consumer’s job is to remove one piece of code at a time from the buffer

- The producer won’t try to add data into the buffer if it is full.
- The consumer won’t try to remove the data from the buffer if it is empty.
- The producer is to go to sleep if the buffer is full and the next time consumer removes an item from buffer ,it wakes up the producer.
- The consumer is go to sleep if the buffer is empty and the next time the producer puts the data into the buffer, it wakes up the sleeping consumer.

Producer-consumer Algorithm

```
Int itemcount=0;
Void producer()
{
While(true)
{
        Item = produce_item();
        If(itemcount==buffer_size)
        Sleep();
        Putitemintothebuffer(item);
```

```
Itemcount=itemcount+1;
If(itemcount==1)
Wakeup(consumer);
}
}
```

```
Void consumer()
{
While(true)
{
If(itemcount==0)
Sleep();
Item=removeitemfrombuffer();
Itemcount--;
If(itemcount==buffersize-1)
Wakeup(producer);
}
```

Dining Philosophers problem

- Five philosophers are sitting around a circular table.
- Dining table has five chop sticks and bowl of rice in the middle.
- Philosopher either can eat or think
- When a philosopher wants to eat, he uses two chop sticks.
- When philosopher wants to think, he keeps down both chop sticks.
- It is assumed that no philosopher can know when others wants to eat.

Dining Philosophers problem

```
Semaphore fork[5];
 int  i;
Void philosopher(int  i)
{
do
Wait (fork[i]);
Wait(fork[(i+1)%5]);
eat();
signal(fork[i]);
signal(fork[(i+1)%5]);
think();
} while (true);
}
```

Readers – writers Problem

1) There is a data area shared by a number of processes. The data area could be a file, a block of main memory etc.

2) There are number of processes that only read the data area and are referred to as **readers**. The processes that only write to the data area are known as **writers**. The conditions to be satisfied are as below.

3) Any number of readers may simultaneously read the data.

4) Only one writer at a time may write into the data area.

5) If a writer is writing, no reader may read it.

```
int readcount;
Semaphore mutex=1,wrt=1;
void reader()
{
do
{
Wait(mutex);
Readunit();
Readcount--;
If (readcount==0)
Signal(wrt);
Signal(mutex);
}
While(true);
}
```

```
Void writer()
{
do
{
Wait(wrt);
Writeunit();
Signal(wrt);
}
While(true);
}
```

```
void main()
readcount = 0;
//call procedures reader, writer
}
```

Monitor

- Monitor is a high level synchronization construct.
- A monitor is an abstract data type that encapsulates private data with public methods.

A monitor has four components

- Initialization
- Local data
- Monitor procedures
- Monitor entry queue

- **Initialization**

  It contains the code that is used exactly once when the monitor is created.

- **Local or shared data**

  It contains the private data which can be only use within the monitor by the local procedures.

- **Monitor procedures**

  A process enters the monitor by involving one of these procedures.

- **Monitor entry queue**

  It contains all the threads waiting to enter into the monitor.