Ch-3 - Inter-Process Communication

* Explain critical section with its section:

=> The Critical Section is a code segment where the shared variable can be accessed.

In Critical section one process can execute at a time.

Critical Section execute only one process critical section at a time.

All the other process have to wait to execute in their critical section.

There are four Critical section section.

1) Entry Section
2) Critical Section
3) Exit Section
4) Reminder Section

1 Entry Section: The entry section handles the entry into the critical section.

2 Critical Section: The Critical section acquires the resources needed for execution of Process.

3 Exit Section: The exit section handles the exit from critical section.

4 Reminder Section: Reminder section is informs the other processes that the critical Section is free.

- Drawback of Critical Section:

The Critical Section cannot be executed by more than one process at a same time.

Operating System faces the problem to allowing the process from entering the critical section.

* Explain Peterson's Solution:

=> Peterson's Solution is solve the critical section problem.

Peterson's solution is allow to execute ~~one~~ than more than one process at a same time in a critical section.

Peterson's solution can be implement only two process.

Peterson's solution is implemented in user mode and does not require hardware support.

- Algorithm:

```
#define N 2
#define True 1
#define False 0

int interested[N] = False
int turn;

Void Entry_Section (int process)
{
  int other;
```

```
other = 1 - process;
interested [process] = True;
turn = process;

while (interested [other] == True
        && turn = process);
}
void exit_section (int process)
{
  interested [process] = False;
}
```

Peterson's solution is generated mutual exclusion is for two process.

Mutual exclusion is allow two process at a same time in critical section.

- Drawback of Peterson's Solution:

Peterson's solution is only allowed two process at a time.

Peterson's Solution can not allow multiple process at a time.

* Explain Race Condition

=> A race condition occurs when a system perform more or two operation at a same time.

Race condition is a comman problem in multithread application.

Race condition also occur if instruction are process in incorrect order

Ex. If two people tried to turn on the light using two different switch at a same time.

Then one instruction might cancel or two action might trip the circuit breaker.

In computer memory, if commands to read and write a large amount of data are received at atmost same time.

Then machine overwrite some or all of the old data while data is read or write.

Race Condition can crashes the computer data.

* Explain Mutual Exclusion:

=> Mutual Exclusion is use in Concurrency Process.

Mutual Exclusion allow one process at a time to perform.

Mutual Exclusion used in Shared Memory and Shared Devices.

Mutual Exclusion is used in interuupt handler between two process.

Mutual Exclusion allow only one process can be inside the critical section at any time.

- Apporch of Mutual Exclusion:

1 Software : Process are done with using software and difficult to slove problem.

2 Hardware : Process are done with using Hardware devices.

3 Programming : If Problem occurs then OS is support.

- Requirement:

1 Freedom from Deadlocks

2 Freedom from Starvation

3 Algorithm breaks if processes are die or lost.

- Drawback:

It allows only one process to perform at a same time.

SMVS
Page No.
Date : / /

* Explain Strict Alternation Approach.

=> Strict Alternation is use to remove problem of Mutual Exclusion.

Strict Alternation appøroach is implemented at user mode.

Strict Alternation approach is use for only two process.

In this apporch when two process are perform mutual exculsion then Strict alternation approach lock one process and execute second process.

- Algorithm For ~~Process 2~~ : Two Process :

```
turn = 0;

while(true)
{
  while(turn != 0)
    critical_section();
  turn = 1;
    noncritical_section();
}
```

```
while(true)
{
  while(turn != 1)
    critical_section();
  turn = 0;
    noncritical_section();
}
```

* Explain Producer-Consumer Problem:

=> Producer-Consumer problem is a multi-process synchronization problem.

For Producer-Consumer problem we have to use fixed size buffer.

Producer is producing some items and Consumer is consuming the items that is produce by Producer.

The same memory buffer is shared by both Producer and consumer.

Producer is to produce the item and put it into the memory buffer.

Consumer is to ~~pr~~ consume the item into the memory buffer.

The Producer should produce data only when the buffer is not full.

If Buffer is full then producer does not produce the data.

The Consumen should consume data only when the buffer is not empty.

If Buffer is empty then consumer does not consume the data.

- Algorithm for Producer:

```
producer(void)
{
  int item;
  while(true)
  {
```

```
    produce_item(&item);
    if(counter == N)
        sleep();
    enter_item(item);
    counter = counter + 1;

    if(counter == 1)
        wakeup(consumer);
    }
}
```

- Algorithm for consumer:

```
consumer(void)
{
    int item;

    while(true)
    {
        if(counter == 0)
            sleep();

        consumer_item(&item);
        counter = counter - 1;

        if(counter = N-1)
            wakeup(Producer);
    }
}
```

* Explain Semaphores :

Semaphores are integer variable that are used to solve critical section Problem.

Semaphores are performed using two atomic operations.

1) Wait
2) Signal

1 Wait : The Wait operations decrements the value of its argument s If it is positive else no operation is performed.

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

2 Signal : The Signal operation increments the value of its argument s.

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

There are two types of Semaphores.

1) Counting Semaphores

2) Binary Semaphores

1 Counting Semaphores:

This Semaphores are used to coordinate the resource access.

IF the resources are added, Semaphore count automatically incremented, else count is decremented.

2 Binary Semaphores:

The Binary Semaphores are restricted to 0 and 1.

The wait operation only works when semaphores is 1 and signal operation succeeds when semaphores is 0.

- Drawback :

Semaphores are complicated. So, wait and signal operation perform in the correct order.

* Explain Reader's - Writer Problem.

=> When two types of process need to access a shared resoure such as file.

This condition is called Reader's - Writer Problem.

The Reader's - Writer Problem is used to manage synchronization

Ex. If two Person read and write file at a same time than it is not possible to manage.

If two Person access the file for read than it does not accures a problem.

To solve this situation, a writer should get exclusive access to an object.

Solution:

To solve this problem, when a writer is accessing the object than no reader or writer may access it.

At a time only one writer can access the object.

* Explain Dining Philosophers Problem.

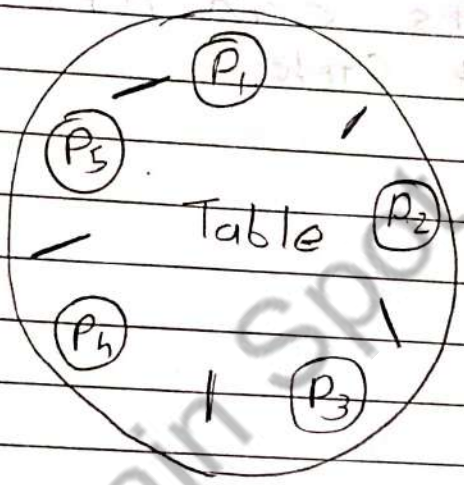
=> The dining Philosophers problem is the classical problem of synchronization.

There are 5 Philosophers sharing a circular table and they eat.

There are bowl of each Philosophers and 5 forks.

A Philosopher needs both their right and left ~~Fro~~ frook to eat.

When Philosopher think, they ~~igor th~~ ignore the eating and do not require ~~fo~~ frok.




```
void dinning_Philosopher ()
{
  while (true)
  {
    thinking();
    eating ();
  }
}
void eating()
{
  takeleftFork();
  eatfood(delay);
```

```
    putrightfork();
    putleftfork();
}
```

To solve this problem,

When philosophers put down the left fork it that philosophers can not obtain the right fork.