

* Assignment No: 01 *

1) Date:

Assignment No: 01

Title: White a program to solve classical problem of synchroniz-

ation using semaphore & muter.

Problem stedement: Implement program in C/C++/ Java/Python to solve classical prioblems of synchronization using reutex and Semaphore

5] software and torchware requirement.

Software Requirement: 1) IDE (Integrated Development Environment)

2) Compiler.

Hardwhere Requirement:

1) Windows 10 PC, keyboard, Mouse, (Is 9gen) (84B Ram)

Learning Objective:
1) To solve the given problem using programming

2) Apply oppropriate data structure to solve the given problem.

3) understand the use of demaphore and muter

Learning Outcome:

1) Understood the clossical problem and ways to some it using semaphore and mutex. 2) gained problem solving skills.



3) Applied suitable dats structure to solve the given problem.

4) Understood the concept of Semaphore and meeter.

8) Theory-Concept in builty:

There are carious problem related to synchroni-

1) Bounded beiffer problem.
2) Dining philosophers problem.

3) The readers writer problem

Lets ducuss the 1) Bounded buffer problem. In computing, the producer-consumer problem (valso known is the bounded-buffer problem) is a problem. The problem discribes two processes, the producer and the consumer which share Common, fixed-size buffer used as oriene.

· The producer's job is to generate item (data)

put it into the buffer, and start again.

the item (i.e reemoving it from buffer), one piece at a time.

Problem: Jo make sure their the produces wor't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty-buffer. an empty-buffer.

Solution: The presclucer is to either go to sleep or discord data if the buffer is full. the next

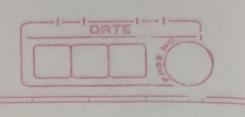
time the consumer removes an item from the buffer, it wakes up the sleeping could result in deadlock where processes are waiting to be awakeened Lutex:

A mutex provide mutual exclusion either In producer or consumer can have the key (mutex) and friceed with their work. As long as the buffer is filled by the proclucer, the consumer needs to wast and vice. Versa. At any point of time only one thread can work

Semaphore:

A semaphore is a generalized mutex. In lieu of a single buffer, we can splid the 4kb buffer Into 1 kB buffer (identical resources). A semaphore four I kB buffer (identical resources). A semaphore tour buffers. The consumer and producer can work

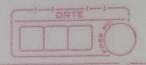
Structly speaking, a mutex is locking mechanism used to synchronize access to a resource. Only one task (can be a threed process based on as abstraction) can acquire the muter. it means there is ownership associated with a meetex, and only the owner con wellease the lock (mentex).



Semaphare is singnaling mechanisms Cim done you can carry on " kind of signal) For example it we are listening to song (one task) on your mobile and at the same time, your friend calls your, an interrupt is triggered upon which an inferrupt service routine (ISR) signals the call processing task to wakeup.



9]	Agoriethm.
*	Algorithm for purducer's Thread.
4	The oritical section is a code segment when
	Showed variables can be accessed. In absorb
	While true repeat 8 teps 5 to 8
3)	Procluce value 'item' to be insuded in buffer.
4)	Make Semophore wait until the shalled buffer
	Nake Semophore wait lentil the shalled buffer is empty (on atteast there is I item in the buffer)
5)	Lock the medex
6)	The 9 tem earlier produced, add it in the buffer.
7)	Now free the locked mutex (unlock it)
(8	Now, free the locked mutex (Unlock it) Notify the threads about the item inscreted in Buffer
a).	End some silvention sof been in when in
	Justinece.
de	Algorithm for Consume's Thread
	: Marine Corner
hi	Startition ni si ano an aertes bases si te
2)	While ik True upport steps 5 to 7
3)	Wait until there is atteast a item produced in Buffer
4)	Jock the mutex.
(2	Buffer is available now for consumption, consume item
454	I on the bulber.
6)	VIIII in the morte very
7)	Acknowledge all other threads asout consumption of
	one item from buffers.
8)	End.



* Additional Theory.

Witical Section:

The oritical section is a cocle segment where the shared variobles can be accessed. An atomic action is required in a critical section i.e only one process can execute in its writical section at a time. At other processes have to wait to execute in this critical sections.

Ruly of cuitical Section:

1) Lutual Exclusion

It is a special type of binary Semaphore which is used for controlling access to showed resource.

2) Process solution:

9t is used when no one is in cuitical section & some one wants in.

3) Bound waiting.

After a purcus makes a request for getting into its witical section, there is a limit for how many other purcusses can get into their witical section.



Application of Assignment:

This Assignment can be used to understand
the classical problem of synchronization
and the possible ways to some it or find the
solution using bemaphore and mutex. Applications: fele management, memory management, persons management. Conclusion Understood the concept of Synchronization, Semaphore and mutex and the optimal way to witness the classical problem of consumer and producer. Within thoducers anthuctor, assign 16] Rejuences. formal variable 1) Geeks for Geeks. 2) www. java made 80 easy. com. Ocale new trued with havemeter as "huducar" Brunke stante) tojouroly cale. method on it a residence of

Code:

```
#include <unistd.h>
#include <iostream>
using namespace std;
#define MAX SIZE 10
pthread mutex t mutexBuffer;
    long long queue array[MAX SIZE];
        return (qRear + 1) % MAX_SIZE == qFront ? true : false;
            qRear = (qRear + 1) % MAX SIZE;
```

```
qFront = (qFront + 1) % MAX SIZE;
            cout << "Buffer Holds Nothing\n\n";</pre>
            for (int i = 0; i < MAX SIZE; i++)</pre>
Queue buffer;
    long tid = reinterpret cast<std::uintptr t>(args);
```

```
pthread mutex init(&mutexBuffer, NULL);
    if (pthread create(&th[i], NULL, &producer, (void *)(i + 1)) != 0)
    if (pthread create(&th[j + i], NULL, &consumer, (void *)(i + 1)) !=
```

```
{
    cerr << "Failed to create thread";
}

for (i = 0; i < producerCount + consumerCount; i++)

{
    if (pthread_join(th[i], NULL) != 0)
    {
        cerr << "Failed to join thread";
    }
}
sem_destroy(&semEmpty);
sem_destroy(&semFull);
pthread_mutex_destroy(&mutexBuffer);
return 0;
}</pre>
```

Output:

```
Consumer 6 is Waiting for a producer to produce
Consumer 6 is Waiting for a producer to produce
Consumer 6 is Waiting for a producer to produce
Consumer 6 is Waiting for a producer to produce
Consumer 6 is Waiting for a producer to produce
producer 4 produced : 1
Current State of the buffer : 1 0 0 0 0 0 0 0 0
```

```
Current State of the buffer : 1 2 3 4 0 0 0 0 0
producer 2 produced : 5
Current State of the buffer : 1
producer 5 produced : 6
Current State of the buffer : 1 2 3 4 5 6 0 0 0
producer 1 produced : 7
Current State of the buffer : 1 2 3 4 5 6 7 0 0 0
Consumer 6 Consumed 1
Current State of the buffer : 0 2 3 4 5 6 7 0 0 0
Consumer 6 Consumed 2
Current State of the buffer : 0 0 3 4 5 6 7 0 0 0
Consumer 6 Consumed 3
Current State of the buffer : 0 0 0 4 5 6 7 0 0 0
producer 4 produced : 8
Current State of the buffer : 0 0 0 4 5 6 7 8 0 0
producer 3 produced : 9
Current State of the buffer : 0 0 0 4 5 6 7 8 9 0
Consumer 6 Consumed 4
```