Devanshy Surana
1032210755, PC-12

Page No.
Date

Operating System
Lab Assignment 5
Title: Process Synchronization

FAQ's.

Ans 1] A semaphore is simply an integer variable that is shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in multiprocessing environment.

Semaphore are of 2 types: Binary (variable takes 0 or 1) and General/counting: Variable takes integer value.

wait: Decrement value of semaphore variable (P)
signal: Increment value of semaphore variable (v)

When a process executes wait operation and finds semaphore value not greater than 0, it must wait.
Instead of process doing a busy wait, process is placed into a waiting queue associated with the semaphore and CPU selects another process to execute.
Waiting process is restarted, when some other process executes a signal operation. Process moves from waiting state to ready state and process is placed in ready queue.
For both counting and binary semaphores, a queue is used to hold processes waiting on semaphore.
The fairest policy is First-In-First-Out (FIFO): The process that has been blocked the longest is released from the queue.

Page No.
Date

Ans 2) Producer-Consumer problem:
- One of the classic problems of synchronization
- Producer produces an item and adds to buffer of limited size
- Consumer takes out an item from buffer and consumes it.
- Buffer is a shared resource and must be used in mutual exclusion manner by both processes.

The problem:
Only one producer or consumer may access the buffer at any one time.
Ensure that the producer can't add data into full buffer and consumer can't remove data from an empty buffer.

Solution using semaphores:
Initialization

```
char item; // could be any data type.
char buffer[n];
Semaphore full=0; // counting semaphore for full slots.
Semaphore empty=n; // counting semaphore for empty slots.
Semaphore mutex = 1; // binary semaphore for mutual exclusion of
buffer.
```

Producer Process:

```
 do
{
   produce an empty in next p
    wait (empty);
    wait (mutex);
add next p to buffer
    signal (mutex);
    signal (full);
 } while (true)
```

Consumer Process.

```
do
{
 wait(full);
 wait(mutex);
 remove an item from buffer to next c
  signal(mutex);
  signal(empty);
 consume the item in next c;
 } while(true)
```

Ans 3) Different process synchronization mechanisms are :
Semaphores :
A semaphore is a signalling mechanism and a thread that is waiting on a semaphore can be signaled by another thread
A semaphore uses 2 atomic operations, wait and signal for process synchronization.
2 types of semaphore are : Binary and Counting semaphores.

Mutex.
Mutex allows the programmer to 'lock' an object so that only one thread can access it.
To control access to a critical section of the code, programmer is required to lock a mutex before entering into a CS and then unlock the mutex while leaving the CS.
Mutex is like a binary semaphore, but the thread which locks the mutex can~~to~~ only unlock the mutex.
Mutex is a locking mechanism used to synchronize acces to a resource.

Page No.
Date

Monitors.
Monitors are a synchronization construct
They ~~can~~ contain data variable and procedures
Data variables cannot be directly accessed by a process.
Monitors allow only a single process to access the variable at a time.
Monitors ensures mutual exclusion: no need to program this constraint emplicitly. Hence shared data are protected by placing them in monitor.
The monitor locks the shared data on process entry.

[illegible]
25/11/22