INTER-PROCESS COMMUNICATION AND SYNCHRONISATION:

Lesson-10: Sharing Data between Processes using the Inter process Communications

1. Sharing Data between the Processes

Sharing Data between the Processes

- Some data is common to different processes or tasks. Examples are as follows:

- **Time**, which is updated continuously by a process, is also used by display process in a system
- **Port input data**, which is received by one process and further processed and analysed by another process.
- **Memory Buffer data** which is inserted by one process and further read (deleted), processed and analysed by another process

Shared Data Problem…

- Assume that at an instant when the value of variable operates and during the operations on it, only a part of the operation is completed and another part remains incomplete.
- At that moment, assume that there is an interrupt.

Shared Data Problem Arising on Interrupt

- Assume that there is another function. It also shares the same variable. The value of the variable *may* differ from the one expected if the earlier operation had been completed. .

Shared data problem...

Whenever another process sharing the same partly operated data , then shared data problem arises.

Example

- Consider x, a 128-bit variable, $b_{127}.....b_0$.
- Assume that the operation OP_{sl} is shift left by 2 bits to multiply it by 4 and find $y = 4 \times x$.

Example…

- Let OP_{sl} be done non-atomically in four sub-operations, OPA_{sl}, OPB_{sl}, OPC_{sl} and OPD_{sl} for $b_{31}…….b_0$, $b_{63}…….b_{32}$, $b_{95}…….b_{64}$ and $b_{127}…….b_{96}$, respectively.
- Assuming at an instance– OPA_{sl}, OPB_{sl} and OPC_{sl} completed and OPD_{sl} remained incomplete

Example…

- Now interrupt *I* occurs at that instance.
- *I* calls some function which uses *x* if *x* is the global variable.
- It modifies *x* to b′127…….b′0.
- On return from interrupt, since OPD_{sl} did not complete, OPD_{sl} operates on $b'_{127}…….b'_{96}$.

Example…

- Resulting value of x is different due to the problem of incomplete operation before *I* occurred

Example of date-time

- Consider date *d* and time *t*.
- Let *d* and *t* are taken in the program as global variables.
- Assume that a thread *Update_Time_Date* is for updating *t* and *d* information on system clock tick interrupt *I*S.
- The thread *Display_Time_Date* is for displaying that *t* and *d* information

Example of date-time…

- Assume that when *Update_Time_Date* ran the *t* = 23:59:59 and date *d* = 17 Jul 2015.
- The *Display_Time_Date* gets interrupted and assume that displaying *d* and operation *t* operations are non-atomic.
- Display of *d* was completed but display of *t* was incomplete when interrupt I_S occurs

Example of date-time…

- After a while, the *t* changes to *t* = 00:00:00 and date *d* = 18 Jul 2007 when the thread *Update_Time_Date* runs.
- But the display will show *t* = 00:00:00 and date *d* = 17 Jul 2007 on re-starting of blocked thread *Display_Time_Date* on return from the interrupt.

2. Steps for the Elimination of Shared Data Problem

Solution of Shared Data Problem

- Use reentrant function with atomic instructions in that section of a function that needs its complete execution before it can be interrupted. This section is called the critical section.

Solution of Shared Data Problem

- Put a shared variable in a circular queue. A function that requires the value of this variable always deletes (takes) it from the queue *front*, and another function, which inserts (writes) the value of this variable, always does so at the queue *back*.

Solution of Shared Data Problem

- Disable the interrupts (DI) before a critical section starts executing and enable the interrupts (EI) on its completion.
- DI─ powerful but a drastic option. An interrupt, even if of higher priority than the critical function, gets disabled.

Solution of Shared Data Problem

- Use lock ( ) a critical section starts executing and use unlock ( ) on its completion.

Solution of Shared Data Problem

- A software designer usually not use the drastic option of disabling interrupts in all the critical sections, except in automobile system like software

Solution of Shared Data Problem

- Use IPC (Inter-Process Communication)
- Using semaphore as mutex for the shared data problem.

3. Use of Mutex for the Elimination of Shared Data Problem

Use of a mutex semaphore

- Facilitates mutually exclusive access by two or more processes to the resource (CPU).
- The same variable, sem_m, is shared between the various processes.
- Let process 1 and process 2 share sem_m and its initial value is set = 1

Use of mutex…

- Process 1 proceeds after sem_m decreases and equals 0 and gets the exclusive access to the CPU.
- Process 1 ends after sem_m increases and equals 1; process 2 now gets exclusive access to the CPU.

Use of mutex…

- Process 2 proceeds after sem_m decreases and equals 0 and gets exclusive access to CPU.
- Process 2 ends after sem_m increases and equals 1; process 1 now gets the exclusive access to the CPU

Use of mutex…

- sem_m is like a resource-key and shared data within the processes 1 and 2 is the resource.
- Whosoever first decreases it to 0 at the start gets the access to and prevents other to run with whom the key shares

4. Difficulties in Elimination of Shared Data Problem Using Mutex

Use of semaphores

- Use of semaphores does not eliminate the shared data problem completely
- Solution of deadlock and priority inversion must also be looked when using semaphores

Summary

We learnt

- Shared data problem can arise in a system when another higher priority task finishes an operation and modifies the data or a variable before the completion of previous task operations.
- Disabling interrupt mechanism, using semaphores and using reentrant functions are some solutions.

We learnt

- Mutex can be used for solving shared data problem
- Mutex is a semaphore that gives at an instance two tasks mutually exclusive access to resources and is used in solving shared data problem.

End of Lesson-10 of Chapter 9 on Sharing Data between Processes using Inter process Communications