Explain the Concurrent OOP and an Actor model in object oriented model

4. Object Oriented Model

- Objects dynamically created and manipulated.
- Processing is performed by sending and receiving messages among objects.

Concurrent OOP

- Need of OOP because of abstraction and reusability concept.
- Objects are program entities which encapsulate data and operations in single unit.
- Concurrent manipulation of objects in OOP.

Actor Model

- This is a framework for Concurrent OOP.
- Actors -> independent component
- Communicate via asynchronous message passing.
- 3 primitives -> create, send-to and become.

Parallelism in COOP

3 common patterns for parallelism-:

1) Pipeline concurrency
2) Divide and conquer
3) Cooperative Problem Solving

What are the principles of synchronization mechanisms? Explain them.

11.2.1 Principles of Synchronization

- The performance and correctness of a parallel program execution rely heavily on efficient synchronization among concurrent computations in multiple processors.
- The source of synchronization problem is the sharing of writable objects (data structures) among processes. Once a writable object permanently becomes read-only, the synchronization problem vanishes at that point.
- Synchronization consists of implementing the order of operations in an algorithm by observing the dependences for writable data.
- Lowe-level synchronization primitives are often implemented directly in hardware. Resources such as the CPU, bus or network and memory units may also be involved in synchronization of parallel computations.

  The following methods are used for implementing efficient synchronization schemes.

  - Atomic Operations
  - Wait Protocols
  - Fairness policies'
  - Access order
  - Sole access protocols

Define parallel programming model. Explain any to models

10.1 Parallel Programming Models

Programming model -> simplified and transparent view of computer hardware/software system.

- Parallel Programming Model are specifically designed for multiprocessors, multicomputer or vector/SIMD computers.

We have 5 programming models-:

1. Shared-Variable Model
2. Message-Passing Model
3. Data-Parallel Model
4. Object Oriented Model
5. Functional and Logic Model

1. Shared-Variable Model

- In all programming system, processors are **active resources** and memory & IO devices are **passive resources**. Program is a collection of processes. Parallelism depends on how IPC(Interprocess Communication) is implemented. Process address space is shared.
- To ensure orderly IPC, a mutual exclusion property requires that shared object must be shared by only 1 process at a time.

Shared Variable communication

- Used in multiprocessor programming
- Shared variable IPC demands use of shared memory and mutual exclusion among multiple processes accessing the same set of variables.

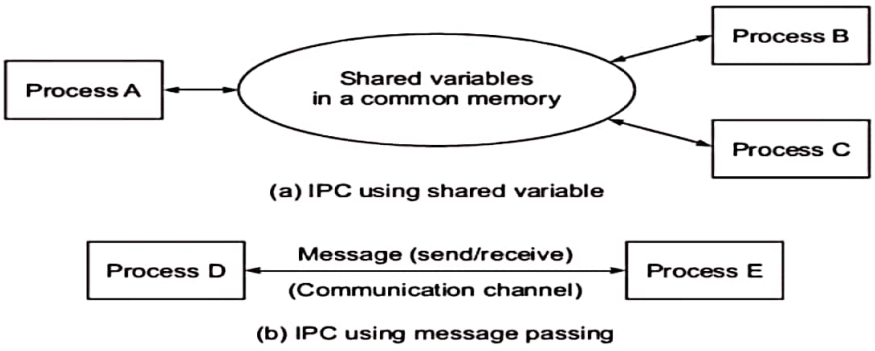


Fig. 10.1 Two basic mechanisms for interprocess communication (IPC).

Critical Section

- Critical Section(CS) is a code segment accessing shared variable, which must be executed by only one process at a time and which once started must be completed without interruption.
- It should satisfy following requirements-:

✓ **Mutual Exclusion:** At most one process executing CS at a time.
✓ **No deadlock in waiting**: No circular wait by 2 or more process.
✓ **No preemption:** No interrupt until completion.
✓ **Eventual Entry**: Once entered CS,must be out after completion.

Protected Access

- Granularity of CS affects the performance.
- If CS is too large,it may limit parallism due to excessive waiting by process.
- When CS is too small,it may add unnecessary code complexity/Software overhead.

4 operational Modes

- Multiprogramming
- Multiprocessing
- Multitasking

Multithreading

2. Message Passing Model

Two processes D and E residing at different processor nodes may communicate wit each other by passing messages through a direct network. The messages may be instructions,

data,synchronization or interrupt signals etc. Multicomputers are considered loosely coupled multiprocessors.

Synchronous Message Passing

- No shared Memory
- No mutual Exclusion
- Synchronization of sender and reciever process just like telephone call.
- No buffer used.
- If one process is ready to cummunicate and other is not,the one that is ready must be blocked.

Asynchronous Message Passing

- Does not require that message sending and receiving be synchronised in time and space.
- Arbitrary communication delay may be experienced because sender may not know if and when the message has been received until acknowledgement is received from receiver.
- This scheme is like a postal service using mailbox with no synchronization between senders and receivers.

3. Data Parallel Model

- Used in SIMD computers. Parallelism handled by hardware synchronization and flow control.
- Fortran 90 ->data parallel lang.
- Require predistrubuted data sets.

Data Parallelism

- This technique used in array processors(SIMD)
- Issue->match problem size with machine size.

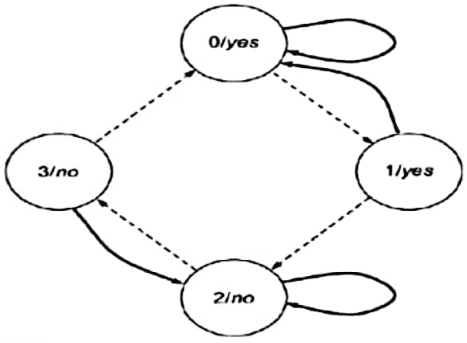
Array Language Extensions

- Various data parallel language used
- Represented by high level data types
- CFD for Illiac 4,DAP fortran for Distributed array processor,C* for Connection machine
- Target to make the number of PE's of problem size.

Mention branch prediction methods and explain

12.10 Branch Prediction

- About 15% to 20% of instructions in a typical program are branch and jump instructions, including procedure returns.
- Therefore—if hardware resources are to be fully utilized in a superscalar processor—the processor must start working on instructions beyond a branch, even before the branch instruction itself has completed. This is only possible through some form of branch prediction.
- What can be the logical basis for branch prediction? To understand this, we consider first the reasoning which is involved if one wishes to predict the result of a tossed coin.



In states 0 & 1: Branch taken
In states 2 & 3 Branch no taken

Solid line: Correct predication.
Broken line: incorrect prediction

Fig. 12.15 State transition diagram of 2-bit branch predictor[8]

- A basic branch prediction technique uses a so-called *two-bit predictor*. A two-bit counter is maintained for every conditional branch instruction in the program. The two-bit counter has four possible states; these four states and the possible transitions between these states are shown in Fig. 12.15.
- When the counter state is 0 or 1, the respective branch is predicted as *taken*; when the counter state is 2 or 3, the branch is predicted as *not taken*.
- When the conditional branch instruction is executed and the actual branch outcome is known, the state of the respective two-bit counter is changed as shown in the figure using solid and broken line arrows.
- When two successive predictions come out wrong, the prediction is changed from *branch taken* to *branch not taken*, and *vice versa*.

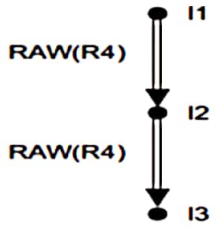


Fig. 12.14 Example of RAW & WAR dependences

- In Fig. 12.14, state transitions made on mis-predictions are shown using broken line arrows, while solid line arrows show state transitions made on predictions which come out right.
- This scheme uses a two-bit counter for every conditional branch, and there are many conditional branches in the program.

- Overall, therefore, this branch prediction logic needs a few kilobytes or more of fast memory.
- One possible organization for this branch prediction memory is in the form of an array which is indexed by low order bits of the instruction address.
- If twelve low order bits are used to define the array index, for example, then the number of entries in the array is 4096.
- To be effective, branch prediction should be carried out as early as possible in the instruction pipeline.
- As soon as a conditional branch instruction is decoded, branch prediction logic should predict whether the branch is taken.

- Accordingly, the next instruction address should be taken either as the branch target address (i.e. branch is taken), or the sequentially next address in the program (i.e. branch is not taken).