UNIT 1 OPERATING SYSTEM FOR PARALLEL COMPUTER

Structure **Page Nos.**

1.0 INTRODUCTION

In Blocks 1 and 2, we have discussed parallel computing architectures and parallel algorithms. This unit discusses the additional requirements at operating system and software levels which will make the parallel programs run on parallel hardware. Collectively, these requirements define the parallel program development environment. A parallel programming environment consists of available hardware, supporting languages, operating system along with software tools and application programs. The hardware platforms have already been discussed in the earlier units. These topics include discussion of shared memory systems, message passing systems, vector processing; scalar, superscalar, array and pipeline processors and dataflow computers. This unit also presents a case study regarding operating systems for parallel computers.

1.1 OBJECTIVES

After studying this unit you will be able to describe the features of software and operating systems for parallel computers.

In particular you should be able to explain the following:

- various additional requirements imposed at OS level for parallel computer systems;
- parallel programming environment characteristics;
- multitasking environment, and
- features of Parallel UNIX.

1.2 PARALLEL PROGRAMMING ENVIRONMENT CHARACTERISTICS

The parallel programming environment consists of an editor, a debugger, performance evaluator and programme visualizer for enhancing the output of parallel computation. All programming environments have these tools in one form or the other. Based on the features of the available tool sets the programming environments are classified as basic, limited, and well developed. The Basic environment provides simple facilities for program tracing and debugging. The limited integration facilities provide some additional tools for parallel debugging and performance evaluation. Well-developed environments provide most advanced tools of debugging programs, for textual graphics interaction and for parallel graphics handling.

There are certain parallel overheads associated with parallel computing. The parallel overhead is the amount of time required to coordinate parallel tasks, as opposed to doing useful work. These include the following factors:

i) Task start up time
ii) Synchronisations
iii) Data communications.

Besides these hardware overheads, there are certain software overheads imposed by parallel compilers, libraries, tools and operating systems.

The parallel programming languages are developed for parallel computer environments. These are developed either by introducing new languages (e.g. occam) or by modifying existing languages like, (FORTRAN and C). Normally the language extension approach is preferred by most computer designs. This reduces compatibility problem. High-level parallel constructs were added to FORTRAN and C to make these languages suitable for parallel computers. Besides these, optimizing compilers are designed to automatically detect the parallelism in program code and convert the code to parallel code.

In addition to development of languages and compilers for parallel programming, a parallel programming environment should also have supporting tools for development and text editing of parallel programmes.

Let us now discuss the examples of parallel programming environments of Cray Y-MP software and Intel paragaon XP/S.

The Cray Y-MP system works with UNICOS operating system. It has two FORTRAN compilers CFT 77 and CFT for automatic vector code generation. The system software has large library of routines, program management utilities, debugging aids and assembler UNICOS written in C. It supports optimizing, vectorizing, concurrentising facilities for FORTRAN compilers and also has optimizing and vetorizing C compiler. The Cray Y-MP has three multiprocessing/multitasking methods namely, (i) macrotasking, (ii) microtasking, (iii) autotasking. Also, it has a subroutine library, containing various utilities, high performance subroutines along with math and scientific routines.

The Intel Paragaon XP/S system is an extension of Intel iPSC/860 and Delta systems, and is a scalable and mesh connected multicompiler which is implemented in a distributed memory system.

The processors that for nodes of the system are 50 MHz i860 XP Processors. Further, it uses distributed UNIX based OS technology. The languages supported by Paragaon include C, C++, Data Parallel Fortran and ADA. The tools for integration include FORGE and Cast parallelisation tools. The programming environment includes an Interactive Parallel Debugger (IPD).

1.3 SYNCHRONIZATION PRINCIPLES

In multiprocessing, various processors need to communicate with each other. Thus, synchronisation is required between them. The performance and correctness of parallel execution depends upon efficient synchronisation among concurrent computations in multiple processes. The synchronisation problem may arise because of sharing of writable data objects among processes. Synchronisation includes implementing the order of operations in an algorithm by finding the dependencies in writable data. Shared object access in an MIMD architecture requires dynamic management at run time, which is much more complex as compared to that of SIMD architecture. Low-level synchronization primitives are implemented directly in hardware. Other resources like CPU, Bus and memory unit also need synchronisation in Parallel computers.

To study the synchronization, the following dependencies are identified:

i) **Data Dependency:** These are WAR, RAW, and WAW dependency.

ii) **Control dependency:** These depend upon control statements like GO TO, IF THEN, etc.

iii) **Side Effect Dependencies:** These arise due to exceptions, Traps, I/O accesses.

For the proper execution order as enforced by correct synchronization, program dependencies must be analysed properly. Protocols like wait protocol, and sole access protocol are used for doing synchronisation.

1.3.1 Wait protocol

The wait protocol is used for resolving the conflicts, which arise because of a number of multiprocessors demanding the same resource. There are two types of wait protocols: busy-wait and sleep-wait. In busy-wait protocol, process stays in the process context register, which continuously tries for processor availability. In sleep-wait protocol, wait protocol process is removed from the processor and is kept in the wait queue. The hardware complexity of this protocol is more than busy-wait in multiprocessor system; if locks are used for synchronization then busy-wait is used more than sleep-wait.

Execution modes of a multiprocessor: various modes of multiprocessing include parallel execution of programs at (i) Fine Grain Level (Process Level), (ii) Medium Grain Level (Task Level), (iii) Coarse Grain Level (Program Level).

For executing the programs in these modes, the following actions/conditions are required at OS level.

i) Context switching between multiple processes should be fast. In order to make context switching easy multiple sets should be present.

ii) The memory allocation to various processes should be fast and context free.

iii) The Synchronization mechanism among multiple processes should be effective.

iv) OS should provide software tools for performance monitoring.

1.3.2 Sole Access Protocol

The atomic operations, which have conflicts, are handled using sole access protocol. The method used for synchronization in this protocol is described below:

1) **Lock Synchronization:** In this method contents of an atom are updated by requester process and sole access is granted before the atomic operation. This method can be applied for shared read-only access.

2) **Optimistic Synchronization:** This method also updates the atom by requester process, but sole access is granted after atomic operation via abortion. This technique is also called post synchronisation. In this method, any process may secure sole access after first completing an atomic operation on a local version of the atom, and then executing the global version of the atom. The second operation ensures the concurrent update of the first atom with the updation of second atom.

3) **Server synchronization:** It updates the atom by the server process of requesting process. In this method, an atom behaves as a unique update server. A process requesting an atomic operation on atom sends the request to the atom's update server.

Check Your Progress 1

1) In sleep-wait synchronization mechanism, we know a process is removed/suspended from the processor and put in a wait queue. Suggest some fairness policies for reviving removed/suspended process
……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

1.4 MULTI TASKING ENVIRONMENT

Multi tasking exploits parallelism by:

1) Pipelining functional units are pipe line together
2) Concurrently using the multiple functional units
3) Overlapping CPU and I/O activities.

In multitasking environment, there should be a proper mix between task and data structures of a job, in order to ensure their proper parallel execution.

In multitasking, the useful code of a programme can be reused. The property of allowing one copy of a programme module to be used by more than task in parallel is called reentrancy. Non-reentrant code can be used only once during lifetime of the programme. The reentrant codes, which may be called many times by different tasks, are assigned with local variables.

Shared variable programme structures

The concept of shared variable has already been discussed above. In this section, we discuss some more concepts related to the shared programme.

1.4.1 Concept of Lock

Locks are used for protected access of data in a shared variable system. There are various types of locks:

1) **Binary Locks:** These locks are used globally among multiple processes.

2) **Deckkers Locks:** These locks are based on distributed requests, to ensure mutual exclusion without unnecessary waiting.

1.4.2 System Deadlock

A deadlock refers to the situation when concurrent processes are holding resources and preventing each other from completing their execution.

The following conditions can avoid the deadlock from occurring:

1) **Mutual exclusion:** Each process is given exclusive control of the resources allotted to it.

2) **Non-preemption:** A process is not allowed to release its resources till task is completed.

3) **Wait for:** A process can hold resources while waiting for additional resources.

4) **Circular wait:** Multiple processes wait for resources from the other processes in a circularly dependent situation.

1.4.3 Deadlock Avoidance

To avoid deadlocks two types of strategies are used:

1) **Static prevention:** It uses P and V operators and Semaphores to allocate and deallocate shared resources in a multiprocessor. Semaphores are developed based on sleep wait protocol. The section of programme, where a deadlock may occur is called critical section. Semaphores are control signals used for avoiding collision between processes.

   P and V technique of Deadlock prevention associates a Boolean value 0 or 1 to each semaphore. Two atomic operators, P and V are used to access the critical section represented by the semaphore. The P(s) operation causes value of semaphore s to be increased by one if s is already at non-zero value.

   The V(s) operation increases the value of s by one if it is not already one. The equation s=1 indicates the availability of the resource and s=0 indicates non-availability of the resource.

   During execution, various processes can submit their requests for resources asynchronously. The resources are allocated to various processors in such a way that they do not create circular wait. The shortcoming of the static prevention is poor resource utilisation.

2) Another method of deadlock prevention is **dynamic deadlock avoidance**. It checks deadlocks on runtime condition, which may introduce heavy overhead in detecting potential existence of deadlocks.

Check Your Progress 2

1) Explain the spin lock mechanism for synchronisation among concurrent processes. Then define a binary spin lock.

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

1.5 MESSAGE PASSING PROGRAMME DEVELOPMENT ENVIRONMENT

In a multicomputer system, the computational load between various processors must be balanced. To pass information between various nodes, message-passing technique is used. The programming environment of a multicomputer includes a host runtime system and resident operating system called Kernel in all the node computers. The host system provides uniform communication between processes without intervention of the nodes, host workstation and network connection. Host processes are located outsides the nodes. The host environment for hypercube computer at Caltech is a UNIX processor and uses UNIX and language processor utilities to communicate with node processes. The Kernel is separately located in each node computer that supports multiprogramming with an address space confined by local memory. Many node processes can be created at each node. All node processes execute concurrently in different physical node or interleaved through multiprogramming within the same node. Node processes communicate with each other by sending or receiving messages.

The messages can be of various types. A particular field of all messages can be reserved to represent message type. The message passing primitives are as follows:

- Send (type, buffer, length, node process)
- Receive (type, buffer, length)

Where type identifies the message type, buffer indicates location of the message, length specifies the length of the message, node designates the destination node and process specifies process ID at destination node. Send and receive primitives are used for denoting the sending and receiving processes respectively. The buffer field of send specifies the memory location from which messages are sent and the buffer field of receive indicates the space where arriving messages will be stored.
The following issues are decided by the system in the process of message passing:

1) Whether the receiver is ready to receive the message
2) Whether the communication link is established or not
3) One or more messages can be sent to the same destination node

The message passing can be of two types: Synchronous and Asynchronous

In **Synchronous**, the message passing is implemented on synchronous communication network. In this case, the sender and receiver processes must be synchronized in time and space. Time synchronization means both processes must be ready before message transmission takes place. The space synchronization demands the availability of interconnection link.

In **Asynchronous** message passing, message-sending ands receiving are not synchronized in time and space. Here, the store and forward technology may be used. The blocked messages are buffered for later transmission. Because of finite size buffer the system may be blocked even in buffered Asynchronous non-blocking system.

1.6 UNIX FOR MULTIPROCESSOR SYSTEM

The UNIX operating system for a multiprocessor system has some additional features as compared to the normal UNIX operating system. Let us first discuss the design goals of the multiprocessor UNIX. The original UNIX developed by Brian Kernighan and Dennis Ritchie was developed as portable, general purpose, time-sharing uniprocessor operating system.

The OS functions including processor scheduling, virtual memory management, I/O devices etc, are implemented with a large amount of system software. Normally the size of the OS is greater than the size of the main memory. The portion of OS that resides in the main memory is called kernel. For a multiprocessor, OS is developed on three models viz: Master slave model, floating executive model, multithreaded kernel. These UNIX kernels are implemented with locks semaphores and monitors.
Let us discuss these models in brief.

1) **Master slave kernel:** In this model, only one of the processors is designated as Master.

   The master is responsible for the following activities:

   i) running the kernel code

   ii) handling the system calls

   iii) handling the interrupts.

   The rest of the processors in the system run only the user code and are called slaves.

2) **Floating-Executive model:** The master-slave kernel model is too restrictive in the sense that only one of the processors viz the designated master can run the kernel. This restriction may be relaxed by having more than one processors capable of running the kernel and allowing additional capability by which the master may float among the various processors capable of running the kernel.

3) **Multi-threaded UNIX kernel:** We know that threads are light-weight processors requiring minimal state information comprising the processor state and contents of relevant registers. A thread being a (light weight) process is capable of executing alone. In a multiprocessor system, more than one processors may execute simultaneously with each processor possibly executing more than one threads, with

the restriction that those threads which share resources must be allotted to one processor. However, the threads which do not share resources may be allotted to different processors. In this model, in order to separate multiple threads requiring different sets of kernel resources spin locks or semaphores are used.

1.7 SUMMARY

In this unit, various issues relating to operating systems and other software requirements for parallel computers are discussed.

In parallel systems, the issue of synchronisation becomes very important, specially in view of the fact more than one processes may require the same resource at same point of time. For the purpose, synchronisation principles are discussed in section 1.3. For the purpose of concurrent sharing of the memory in multitasking environment, the concepts of lock, deadlock etc are discussed in section 1.4. Finally, extensions of UNIX for multiprocessor systems are discussed in section 1.6.

1.8 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Out of large number of fairness policies, the following three policies are quite well known:

   1) First-In-First-Out (FIFO) policy
   2) Upper-bounded misses policy
   3) Livelock-free policy

   First-In-First-Out (FIFO) as the name suggests that no process will be served out of turn. The disadvantage is that in some cases cost of computation and memory requirement may be too high.

   The second fairness policy may be implemented in a number of ways. Depending upon the implementation, the implementation costs and memory requirements may be reduced.

Check Your Progress 2

1) Under the centralized shared (CS) memory, the gate for entry and exist to CS is controlled by a single binary variable say y, which is then shared by all processes attempting to access CS.

   For defining the binary spin lock
   Let y be the single variable for entry to the gate. Initially the value of y is set to 0 (zero) indicating that entry by any one process is allowed. Then spin lock controlling mechanism continuously checks one by one all processes in turn, whether any one of these processes needs to access the memory. Once, any of the processes is found need to access CS, that process say Pp is allowed to access the CS and the entry-allowing variable y is assigned 0 (zero) indicating next process in queue requiring access to CS is allowed to access CS.

1.9 FURTHER READINGS

1) Kai Hwang: *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, Tata-McGraw-Hill (2001)

2) Thomas L. Casavant, Pavel Tvrdik, Frantisck Plasil, *Parallel Computers: Theory and Practice*, IEEE