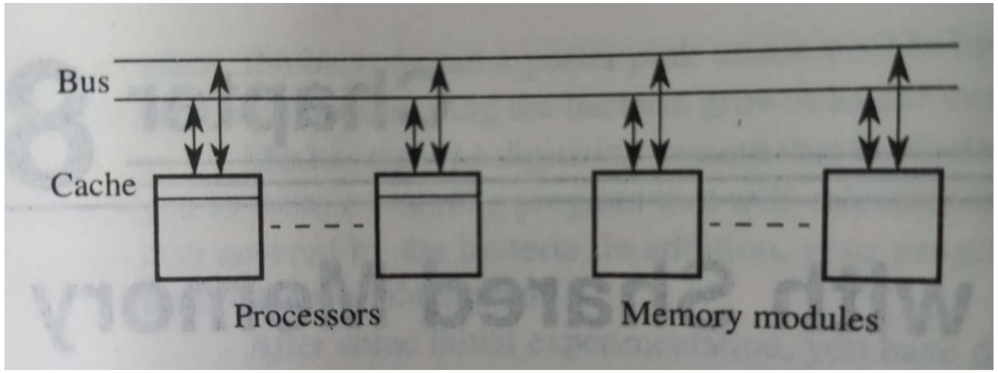
7 Programming with Shared Memory

[Weightage(15%): Approx. 10-11 Marks out of 70 Marks]

- Shared Memory Multiprocessors
- Sharing Data
- Parallel Programming Languages and Constructs
- OpenMP
- Performance Issues

7.1 Shared Memory Multiprocessors



- Any memory location can be accessed by any processor.
- Each memory location is given a unique address within a range of addresses (*Single address space*)
- All processors and memory modules are attached to the same set of wires/bus.
- This set up is suitable for small number of processors (E.g. 8)
- For complex interconnections with more number of processors, we need to bring the memory closer to the processors. High-speed cache is present with the processors.

- **What are the options for programming multicomputer?**

1. Using completely a new programming language
2. Modifying an existing sequential programming language
3. Using library routines with an existing sequential programming language

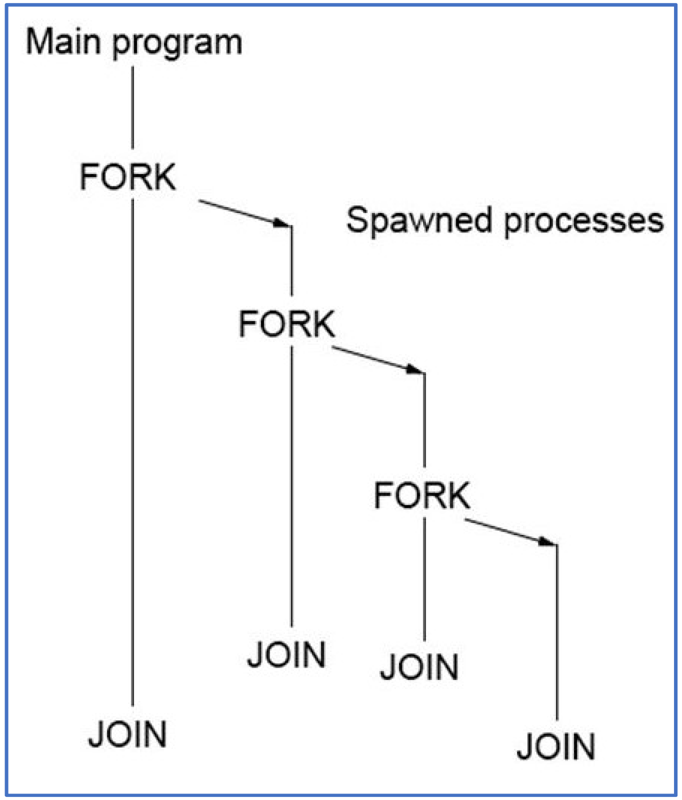
7.1 Shared Memory Multiprocessors

- Some early Parallel Programming Languages

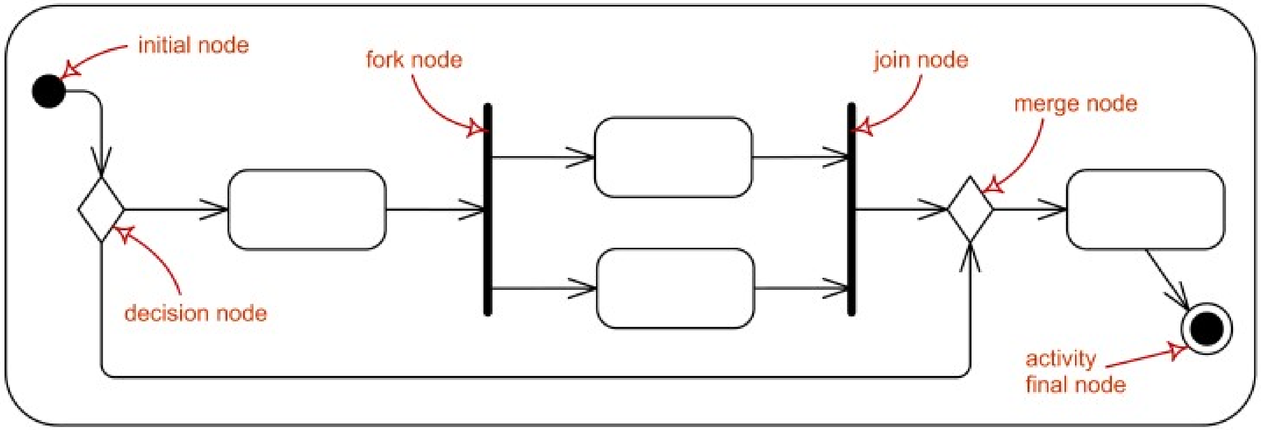
Language	Originator/date	Comments
Concurrent Pascal	Brinch Hansen, 1975	Extension to Pascal
Ada	U.S. Dept. of Defense, 1979	Completely new language
Modula-P	Bräunl, 1986	Extension to Modula 2
C*	Thinking Machines, 1987	Extension to C for SIMD systems
Concurrent C	Gehani and Roome, 1989	Extension to C
Fortran D	Fox et al., 1990	Extension to Fortran for data parallel programming

7.2 Constructs for Specifying Parallelism

7.2.1 Creating Concurrent Processes



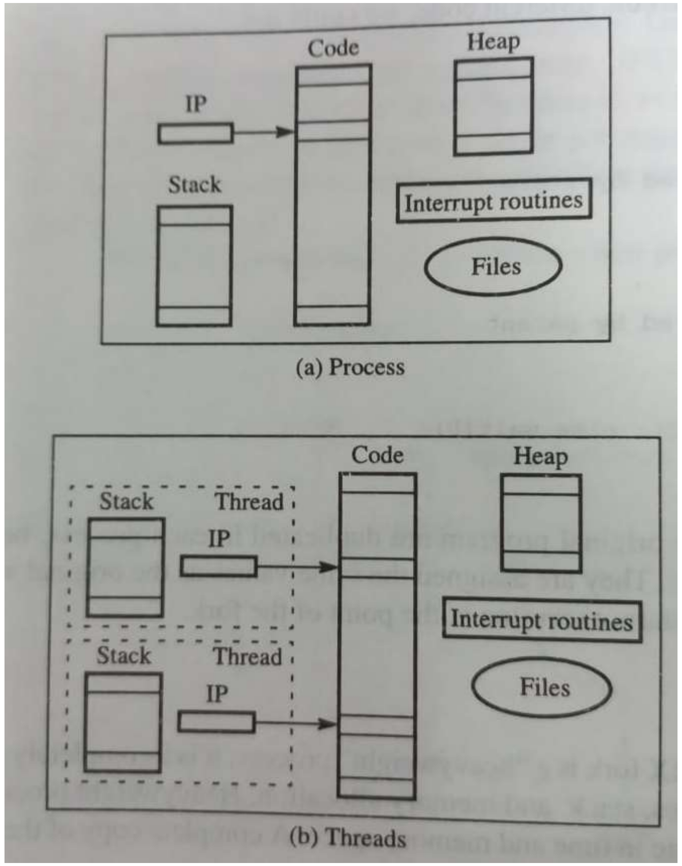
FORK-JOIN Construct



FORK-JOIN Activity Diagram

7.2 Constructs for Specifying Parallelism

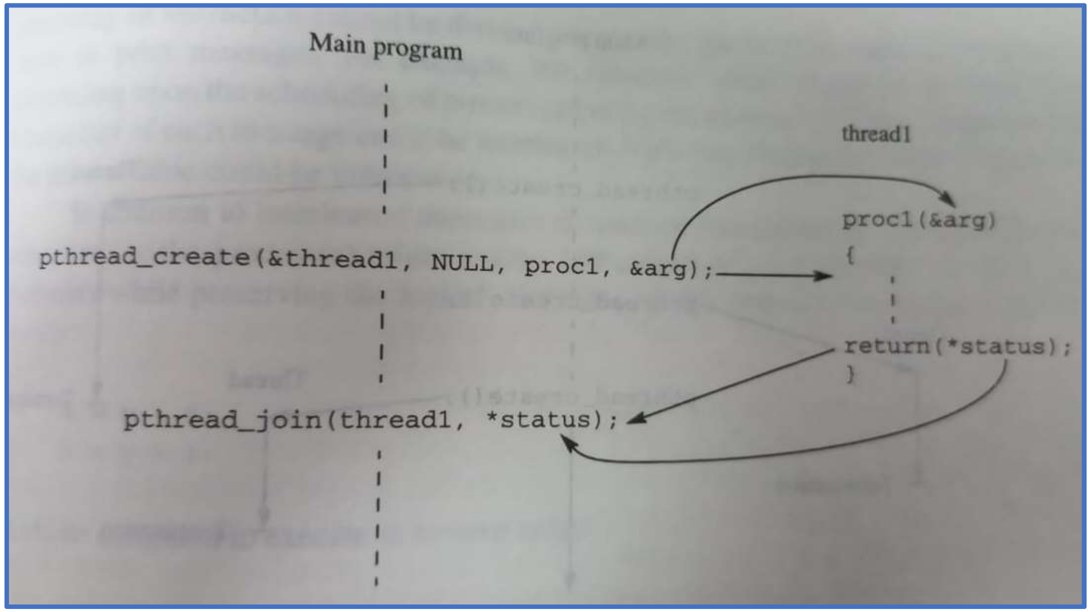
7.2.2 Process Vs. Threads

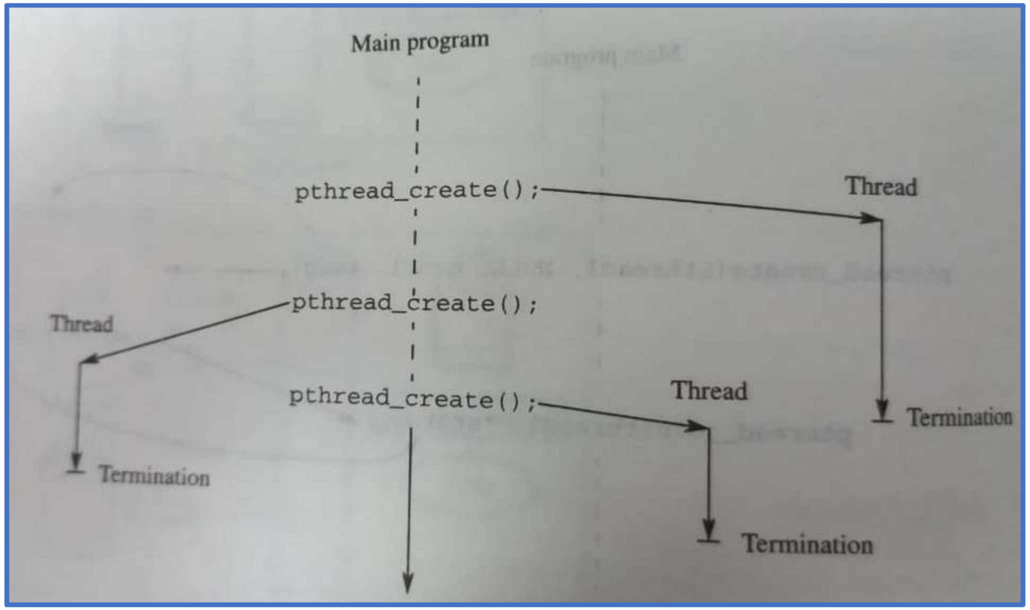


Processes	***Threads***
Processes are used when the tasks are essentially unrelated	Threads are used when tasks are actually part of the same job
Each process has its own address space and PCB	Threads belonging to the same process share the process' address space, code data, and files, but not the registers and stack
Address spaces are protected from each other	No address space protections
Switching between processes is done at the kernel level	Switching between threads can be done at either the user level or the kernel level
Owned by one or more users	Usually own by a single user

7.2 Constructs for Specifying Parallelism

7.2.2 Thread (create, join, detached)





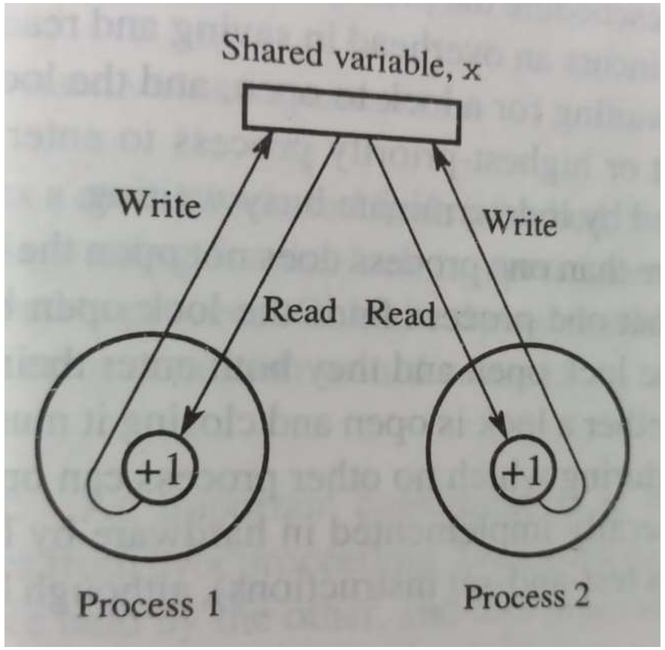
7.3 Sharing Data

- Creating Shared Data
- Accessing Shared Data
  - Locks
  - Deadlock
  - Semaphores
  - Monitor
  - Condition Variables

- **Creating Shared Data:**
  - Each Process has its own virtual address space
  - The shared memory system calls (E.g. `shmget()`)allow processes to attach a segment of physical memory to their virtual address space
  - Variables declared at the top of the program are *global* and that declared within routines are *local*.

7.3 Sharing Data

- Creating Shared Data
- Accessing Shared Data
  - Locks
  - Deadlock
  - Semaphores
  - Monitor
  - Condition Variables



- **Accessing Shared Data:**
  - Reading the variable by different processes does not cause conflicts, but writing new values may do so.
  - For instance, consider two processes, each of them trying to increment the value of the variable x, almost at the same time.
  - A mechanism to ensure that only one process accesses a particular resource at a time is called ***critical section*** and arrange that only one such critical section is executed at a time. The process in the critical section prevents all other processes to enter into the critical section for the same resource.
  - Once the process is finished its critical section, other process is allowed to enter a critical section. This mechanism is called ***mutual exclusion***.

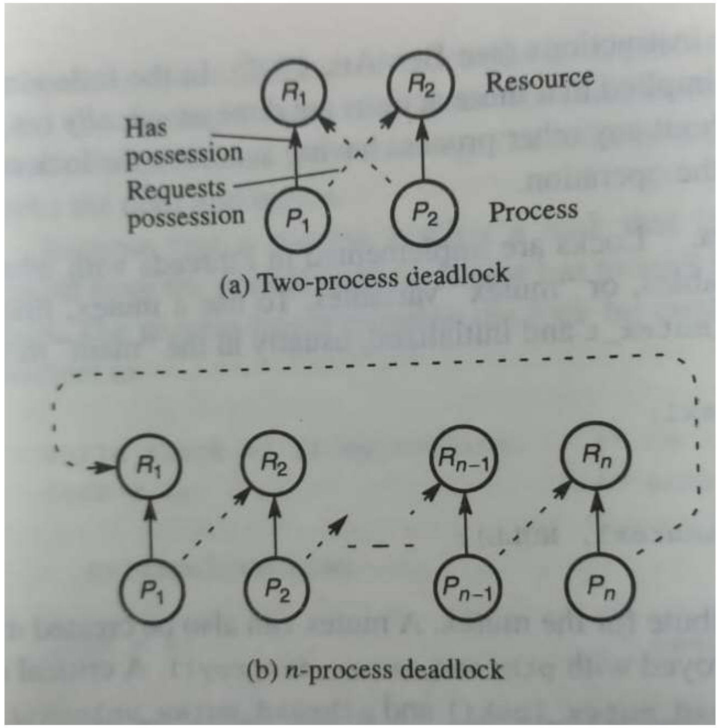
7.3 Sharing Data

- Creating Shared Data
- Accessing Shared Data
  - Locks
  - Deadlock
  - Semaphores
  - Monitor
  - Condition Variables

- **Locks:**
  - The simplest mechanism to ensure mutual exclusion for critical section.
  - A lock is a 1-bit variable that is 1 (process is in critical section) or 0 (process is NOT in critical section)
  - The process comes to a critical section and finds it open, locks it (to prevent others to enter).
  - Once the process is finishes, it unlocks the critical section and leaves.

7.3 Sharing Data

- Creating Shared Data
- Accessing Shared Data
  - Locks
  - Deadlock
  - Semaphores
  - Monitor
  - Condition Variables



- **Deadlock:**
  - Deadlock prevents processes from ever proceeding.
  - It occurs with two processes when one requires a resource held by other and vice-versa.
  - Deadlock can also occur in a multiprocessor system (*deadly embrace*)
  - Deadlock can be eliminated if both processes make requests first for one resource and then for the other.

7.3 Sharing Data

- Creating Shared Data
- Accessing Shared Data
  - Locks
  - Deadlock
  - Semaphores
  - Monitor
  - Condition Variables

- **Semaphores:**
  - Semaphore is a positive integer (including zero) operated upon by two operations viz. `P` and `V`.
  - `P` operation on semaphore `s` is written as `P(s)`, waits until `s` is greater than zero and then decrements `s` by one and allows the process to continue.
  - `V` operation increments `s` by one to release one of the waiting process (if any).
  - Processes delayed by `P(s)` are kept in temporary state until released by a `V(s)` on the same semaphore.

7.3 Sharing Data

- Creating Shared Data
- Accessing Shared Data
  - Locks
  - Deadlock
  - Semaphores
  - Monitor
  - Condition Variables

- **Monitor:**
  - Semaphores are open to human errors.
  - Monitor is a suit of procedures that provides the only method to access a shared resource.
  - Essentially, the data and the operations that can operate upon the data are encapsulated into one structure.
  - Reading and writing can only be done by using a monitor procedure, and only one process can use a monitory procedure at any instant.
  - If a process requests a monitor procedure while another process is using one, the requesting process is suspended and placed on a queue. When the active process has finished using the monitor, the first process in the queue is allowed to use a monitor procedure.

7.3 Sharing Data

- Creating Shared Data
- Accessing Shared Data
  - Locks
  - Deadlock
  - Semaphores
  - Monitor
  - Condition Variables

- **Condition Variable:**
  - Often a critical section is to be executed if a specific global condition exists; for example, if a certain value of a variable has been reached.
  - Three operations are defined for a condition variable:

  `Wait (cond_var)` – wait for the condition to occur

  `Signal (cond_var)` – signal that the condition has occur

  `Status (cond_var)` – return the number of processes waiting for the condition to occur

OpenMP

https://engineering.purdue.edu/~smidkiff/ece563/files/ECE563OpenMPTutorial.pdf

OR

https://www.cse.iitk.ac.in/users/pmalakar/acmws/OpenMP_Intro.pdf

7 Programming with Shared Memory

[Weightage(15%): Approx. 10-11 Marks out of 70 Marks]

1. Draw a typical shared memory multiprocessor interconnection/architecture and explain the same in brief.
2. What are the options for programming multicomputer?
3. List some early parallel programming languages.
4. Explain the FORK-JOIN constructs to create concurrent processes, with neat diagram.
5. Differentiate between Processes and Threads.
6. Explain Thread create, join and detached with sample syntax.
7. What are the issues with accessing shared data? How the issues can be addressed with the idea of critical section and mutual exclusion?
8. Explain following concepts with respect to accessing shared data
    1. (a) Locks (b) Deadlock (c) Semaphores (d) Monitor (e) Condition Variables
9. Explain OpenMP in detail.