Processes & Message Passing

Lecture-7

Parallel & Distributed Computing

Course Outlines

Course Name: Parallel and Distributed Computing
Credit Hours: 3(3-0)
Prerequisites: Data Communications and Computer Networks

Course Outlines:

Why use parallel and distributed systems? Why not use them? Speedup and Amdahl's Law, Hardware architectures: multiprocessors (shared memory), networks of workstations (distributed memory), clusters (latest variation). Software architectures: threads and shared memory, processes and message passing, distributed shared memory (DSM), distributed shared data (DSD). Possible research and project topics, Parallel Algorithms, Concurrency and synchronization, Data and work partitioning, Common parallelization strategies, Granularity, Load balancing, Examples: parallel search, parallel sorting, etc. Shared-Memory Programming: Threads, Pthreads, Locks and semaphores, Distributed-Memory Programming: Message Passing, MPI, PVM. Other Parallel Programming Systems, Distributed shared memory, Aurora: Scoped behaviour and abstract data types, Enterprise: Process templates. Research Topics.

Process and Message Passing

- Numerous programming languages (message passing paradigm) and libraries have been developed for ***explicit parallel programming***.
- The message-passing programming paradigm is one of the oldest and most widely used approaches for programming parallel computers.

Process and Message Passing cont…

- There are two key attributes that characterize the message-passing programming paradigm.
- The first is that it assumes a ***partitioned address*** space and the second is that it supports only ***explicit parallelization***.

Process and Message Passing cont…

- The logical view of a machine supporting the message-passing paradigm consists of ***p*** processes, each with its own address space.
- Instances of such a view come naturally from clustered workstations and non-shared address space multicomputers.

Process and Message Passing cont…
Structure of Message-Passing Programs

- Message-passing programs are often written using the:
- **Asynchronous** paradigm.
- **Loosely synchronous** paradigm.

Process and Message Passing cont…
Structure of Message-Passing Programs

- In the asynchronous paradigm, all concurrent tasks execute asynchronously (no coordination).
- This makes it possible to implement any parallel algorithm.
- However, such programs can be harder to understand, and can have ***nondeterministic behavior*** due to ***race conditions***.

Process and Message Passing cont…
Structure of Message-Passing Programs

- Loosely synchronous programs are a ***good compromise***.
- In such programs, tasks or subsets of tasks ***synchronized to perform interactions***.
- However, between these interactions, ***tasks execute completely asynchronously***.

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations

- Since **interactions** are accomplished by ***sending and receiving messages***, the basic operations in the message-passing programming paradigm are send and receive.
- In their simplest form, the prototypes of these operations are defined as follows:

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations

- send(void *sendbuf, int nelems, int dest)
- receive(void *recvbuf, int nelems, int source)

The **sendbuf** points to a buffer that ***stores the data to be sent***

The **recvbuf** points to a buffer that ***stores the data to be received***

The **dest** is the ***identifier of the process that receives the data***

The **source** is the ***identifier of the process that sends the data***

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations

```
P0                          |  P1
                            |
a = 100;                    |  receive(&a, 1, 0)
send(&a, 1, 1);             |  printf("%d\n", a);
a=0;                        |
```

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations

- The important thing to note is that process P0 changes the **value of a** to 0 immediately following the send.
- The semantics of the send operation require that the value received by process P1 must be 100 as opposed to 0.

```
P0                      | P1
                        |
a = 100;                | receive(&a, 1, 0)
send(&a, 1, 1);         | printf("%d\n", a);
a=0;                    |
```

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations

- Most message passing platforms have additional hardware support for sending and receiving messages.

1. They may support **DMA** (direct memory access) and
2. Asynchronous message transfer using **network interface hardware**.

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations

- Network interfaces allow the transfer of messages from buffer memory to desired location without CPU intervention.
- Similarly, DMA allows copying of data from one memory location to another (e.g., communication buffers) without CPU support (once they have been programmed).

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations

- As a result, if the send operation programs the communication hardware and returns before the communication operation has been accomplished, process P1 might receive the value 0 instead of 100!

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations
Blocking Message Passing Operations

- A simple solution to the dilemma presented in the code fragment above is for the send operation to return only when it is semantically safe to do so.
- It simply means that the sending operation blocks until it can guarantee that the semantics will not be violated on return irrespective of what happens in the program subsequently. There are two mechanisms by which this can be achieved.

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations
Blocking Message Passing Operations

- There are two mechanisms by which this can be achieved:

1. **Blocking Non-Buffered Send/Receive**

2. **Blocking Buffered Send/Receive**

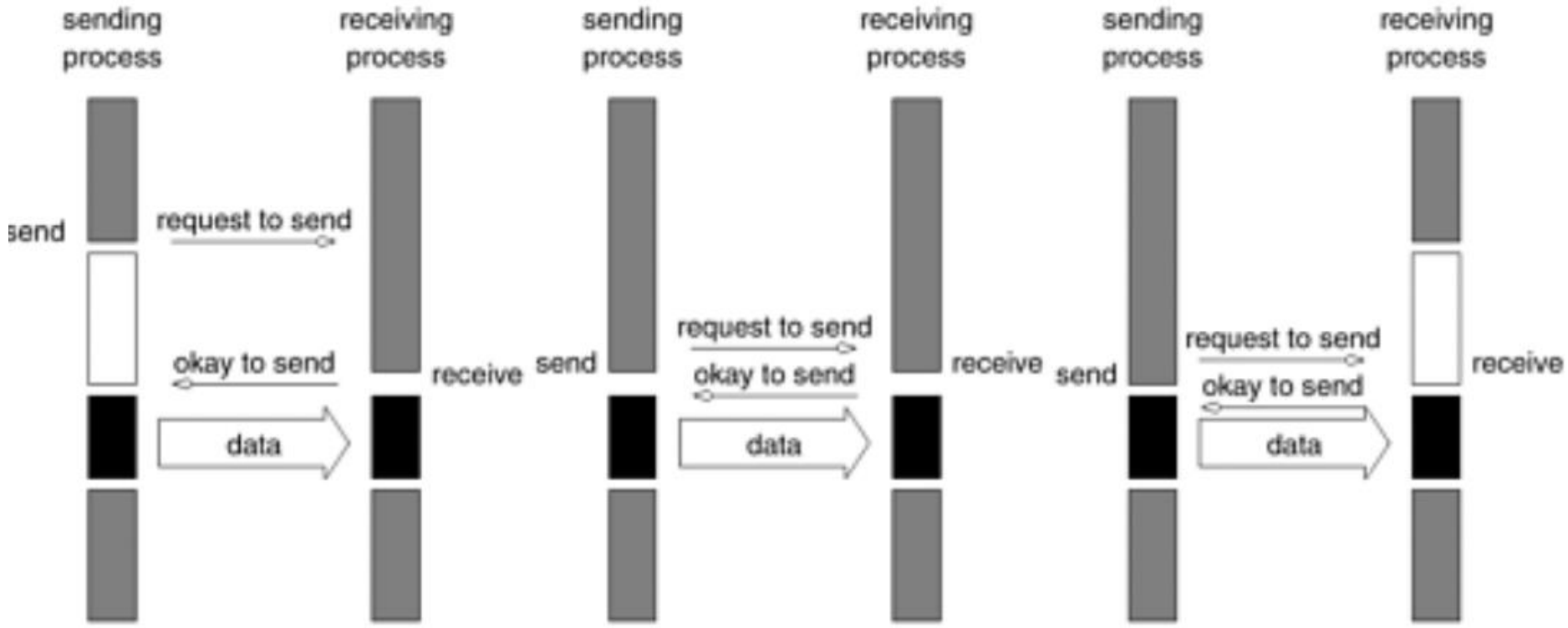
Process and Message Passing cont…
The Building Blocks: Send and Receive Operations
Blocking Message Passing Operations

- **Blocking Non-Buffered Send/Receive:** In the first case, the send operation does not return until the matching receive has been encountered at the receiving process.
- When this happens, the message is sent and the send operation returns upon completion of the communication operation.

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations
Blocking Message Passing Operations

- Typically, this process involves a handshake between the sending and receiving processes. The sending process sends a request to communicate to the receiving process.
- Since there are no buffers used at either sending or receiving ends, this is also referred to as a ***non-buffered blocking operation*** .

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations
Blocking Message Passing Operations



(a) Sender comes first; idling at sender

(b) Sender and receiver come at about the same time; idling minimized

(c) Receiver comes first; idling at receiver

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations
Blocking Message Passing Operations

- In cases (a) and (c), we notice that there is considerable idling at the sending and receiving process.
- It is also clear from the figures that a blocking non-buffered protocol is suitable when the send and receive are posted at roughly the same time.
- However, in an asynchronous environment, this may be impossible to predict. This idling overhead is one of the major drawbacks of this protocol.

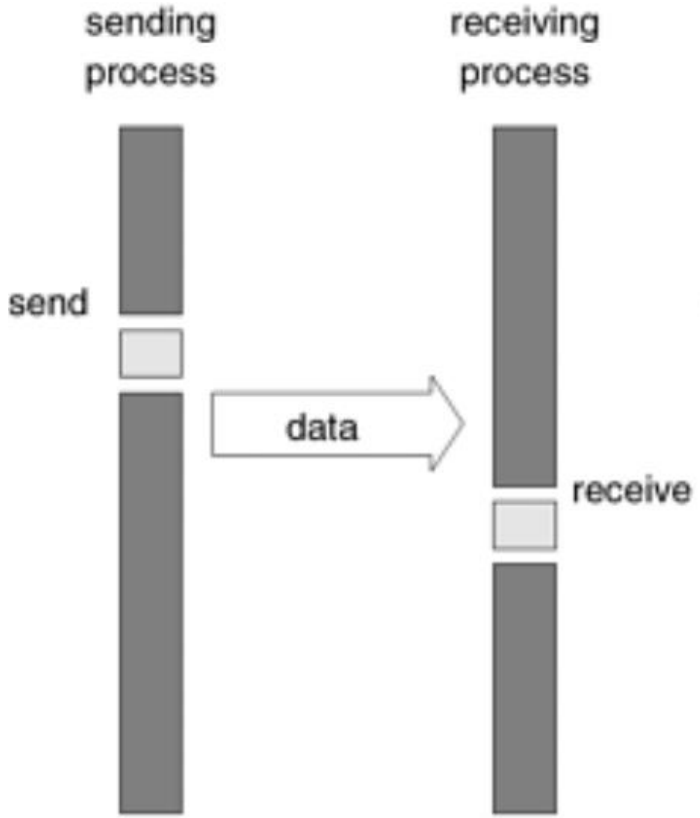
Process and Message Passing cont…
The Building Blocks: Send and Receive Operations
Blocking Message Passing Operations

- **Blocking Buffered Send/Receive:** A simple solution to the idling and deadlocking problem outlined above is to rely on buffers at the sending and receiving ends.
- We start with a simple case in which the sender has a buffer pre-allocated for communicating messages.
- On encountering a send operation, the sender simply copies the data into the designated buffer and returns after the copy operation has been completed.

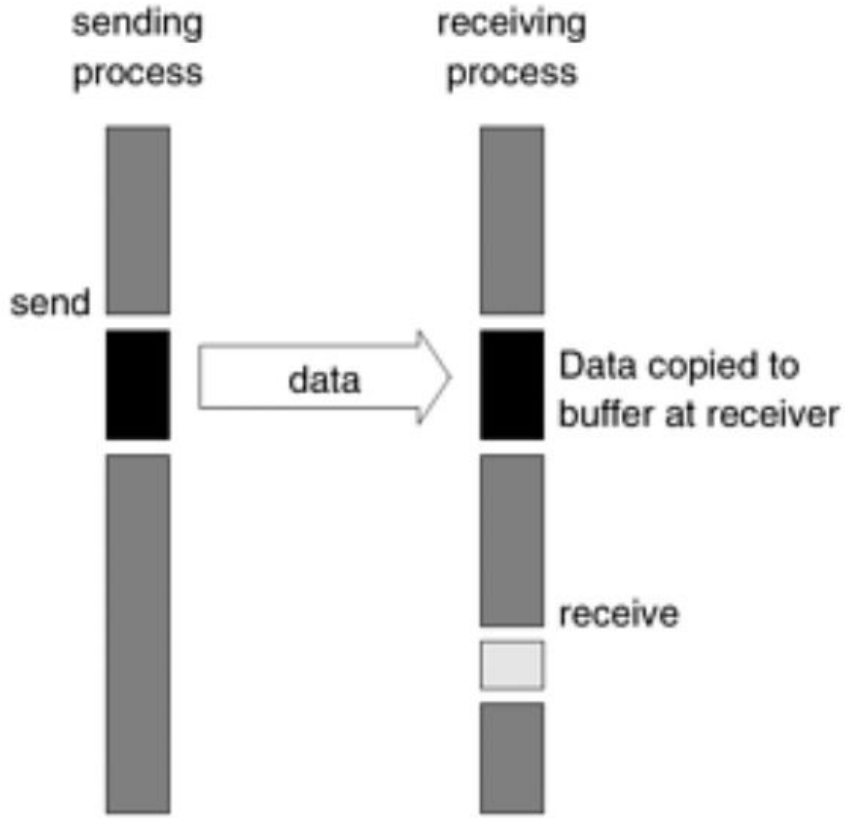
Process and Message Passing cont…
The Building Blocks: Send and Receive Operations
Blocking Message Passing Operations

- The sender process can now continue with the program knowing that any changes to the data will not impact program semantics.
- Note that at the receiving end, the data cannot be stored directly at the target location since this would violate program semantics.
- Instead, the data is copied into a buffer at the receiver as well.

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations
Blocking Message Passing Operations



(a) presence of communication hardware with buffers at send and receive ends



(b) absence of communication hardware, sender interrupts receiver & deposits data in buffer at receiver end

Process and Message Passing cont…
The Building Blocks: Send and Receive Operations
Blocking Message Passing Operations

- The sender process can now continue with the program knowing that any changes to the data will not impact program semantics.
- Note that at the receiving end, the data cannot be stored directly at the target location since this would violate program semantics.
- Instead, the data is copied into a buffer at the receiver as well.