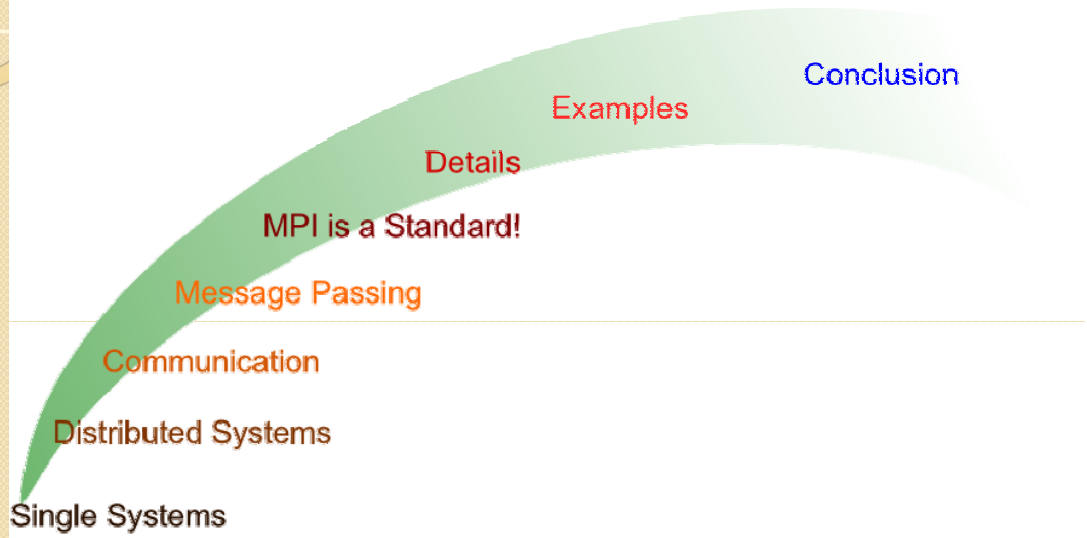
Concurrent Programming

Session 17: An Introduction to Message Passing Interface (MPI)

Computer Engineering Department
Iran University of Science and Technology
Tehran, Iran

Lecturer: Reza Bakhshi
Distributed Systems Lab.
Computer Engineering Department,
Iran University of Science and Technology,
mmfrezabakhshi@yahoo.co.uk

Agenda



Single Systems

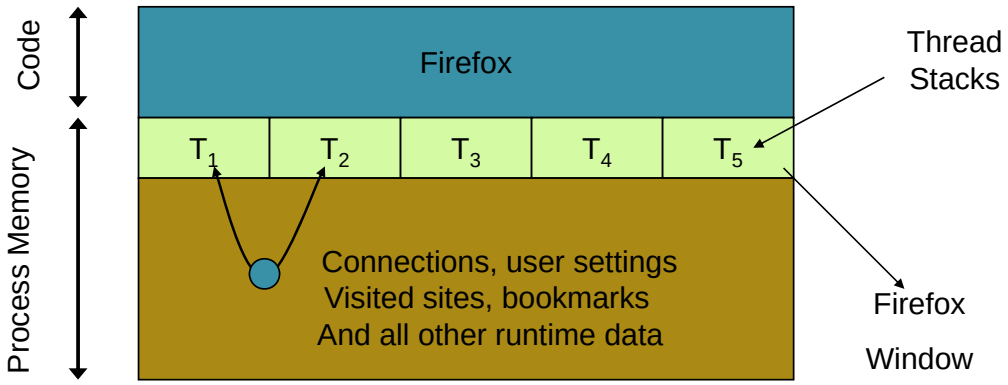
- By single we mean:
  - One or more processors
  - Sharing a single memory address space
- Programming model
  - Shared memory (through variables/objects)

Distributed Systems

- One or more processors
- Connected via a network
- Memory address
  - Different/separated /no direct access
- Process
  - Different Memory Addresses
  - Normally no SHM! + No support!

Communication(1)

- Single Systems, thread communication



Communication(2)

- Single Systems, Process Communication
  - Pipes
  - TCP/UDP
  - (if possible) Shared Memory
    - Gaming, Database
  - Semaphores
  - File & File locks

Communication(3)

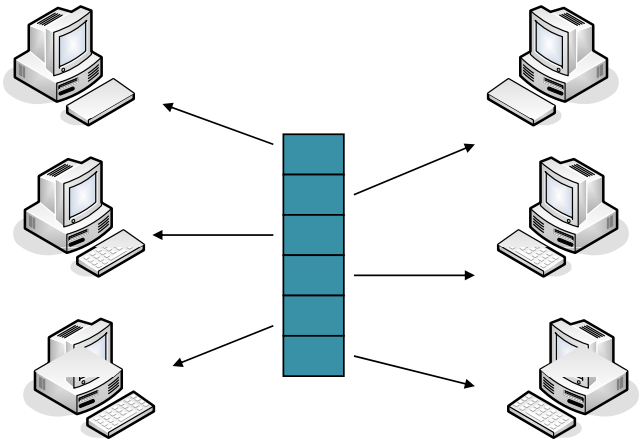
- Distributed Systems
  - No Shared Memory, therefore
    - No Pipe
    - No Semaphore
  - TCP/UDP
    - Developer designed protocol!
  - RPC
  - Web Services, Remoting, …

Message Passing(1)

- As you see
  - Communication is mainly message-based
    - TCP/UDP: send request, receive, response
    - RPC: send request (call method), …
    - Web Service:...
- Distributed Shared Memory
  - Implementation is using message passing!

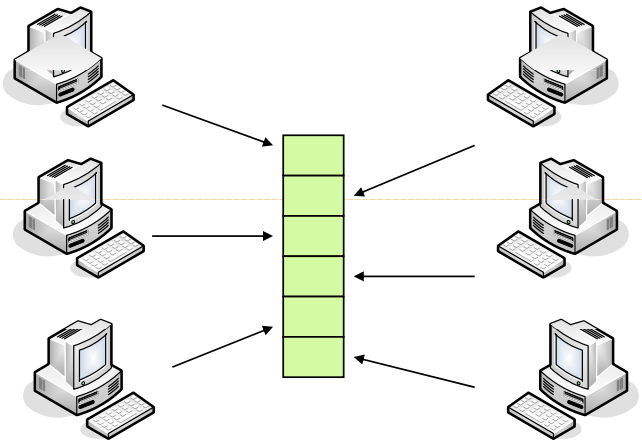
Message Passing(2)

- SPMD (Single Process Multiple Data)
  - The same program running on different machines
  - Requirements for synchronization, communication, data distribution



Message Passing(3)

- …And gathering data!



Message Passing(4)

- Different message passing solutions
- Problems:
  - Porting programs
  - Reusing software/code
- What should be done?
  - Defining Standards

MPI is a Standard (1)

- Specification for a set of functions for
  - Setting up a computing network
  - Locating computing parties
  - Sending message
  - Receiving message
  - Distributing data
  - Gathering it
- About 100 or more functions!
- All 100 functions can be coded using send/receive

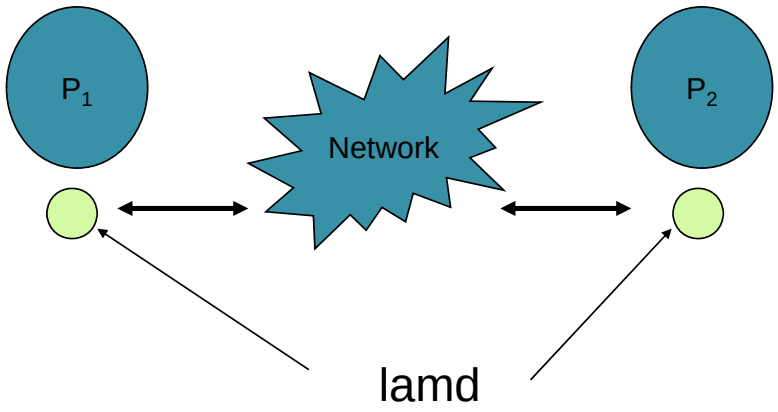
MPI is a Standard (2)

- Different implementations
  - LAM/MPI
  - MPI Pro.
  - Open MPI
  - MPICH
  - MPICH2
  - You can find more!

Details(1)

- MPI contains (LAM/MPI)
  - A wrapper for compiler/preprocessor
    - mpicc, mpiCC,…
  - A set of runtime systems
    - lamboot, lamnodes, lamhalt, mpirun
  - A bunch of libraries for c/Java
    - mpi.h

Details(2)



Examples

- Merge Sort
  - You have an array of numbers
  - One system splits array into sub-arrays
  - And sends sub-arrays to other nodes
  - Each node sorts its sub-array
  - Main node gathers result
  - And merges sub-arrays
  - All nodes run the same code!

MPI_Send

- Sending Message/Data

```
#include "mpi.h"
int MPI_Send(
   void *buf,
   int count,
   MPI_Datatype datatype,
   int dest,
   int tag,
   MPI_Comm comm)
```

MPI_Recv

- Receive message/data

```
#include "mpi.h"
int MPI_Recv(
  void *buf,
  int count,
  MPI_Datatype datatype,
  int source,
  int tag, MPI_Comm comm,
  MPI_Status *status )
```

MPI_Scatter

- Distributing an array between nodes

```
#include "mpi.h"
int MPI_Scatter (
        void *sendbuf,
        int sendcnt,
        MPI_Datatype sendtype,
        void *recvbuf,
        int recvcnt,
        MPI_Datatype recvtype,
        int root,
        MPI_Comm comm )
```

MPI_Gather

- Gathering data from nodes

```
#include "mpi.h"
int MPI_Gather (
   void *sendbuf,
   int sendcnt,
   MPI_Datatype sendtype,
   void *recvbuf,
   int recvcount,
   MPI_Datatype recvtype,
   int root,
   MPI_Comm comm )
```

Merge Sort

```
#include "mpi.h“

void main(int argc, char** args)
{
        MPI_Init(&argc, &args);
        int size = -1;
        MPI_Comm_size(0, &size);
        int myrank = -1;
        MPI_Comm_rank(0, &myrank);
        // you are node #rank of a cluster of #size nodes!
        // let's consider node #0 as root

        int* array;
        if(myrank == 0)
                //fill array
```

Merge Sort (cont.)

```
    int nodesShareSize = ARRAY_LEN/size;
    int* myshare = (int*)malloc(nodesShareSize *
  sizeof(int));
  MPI_Scatter(array, nodesShareSize, MPI_INTEGER,
      myshare, nodesShareSize, MPI_INTEGER, 0, 0);
  MergeSort(myshare);
  MPI_Gather(myshare, nodesShareSize, MPI_INTEGER,
      array, nodesShareSize, MPI_INTEGER, 0, 0);
  // now root must merge the results
  if(myrank == 0)
      // do merge!
  MPI_Finalize();
}
```

Conclusion

- Message Passing in Distributed Systems
- MPI as a Standard
- SPMD
- Think parallel

Need more info?

- Just Google it!
- http://www.mcs.anl.gov/research/projects/mpi/
- http://www.open-mpi.org
- http://www.lam-mpi.org
- And many more in the internet!