Introduction to MPI - Worksheet 4 – Chapter 4
Point to Point Communication

1. Point to point communication are two-sided and require active participation from the processes on both sides. One process (the source) _____ (sends/or receives?) and the other process (the destination) _____(sends/or receives?).

2. These source and destination processes operate _____ (synchronous or asynchronous?) meaning that they are _____ (synchronized or not synchronized?)

3. The sent messages that have not yet been received are called _____ messages.

4. Pending messages are stored in a FIFO queue data structure. _____ (True/False)

5. Messages have 2 main sections: _____ and _____

6. List the 4 parts of an MPI message envelope.

   _____, _____, _____, _____

7. What is the communicator that we've been using? _____

8. List the 3 parts of the MPI message body.

   _____, _____, _____

9. Which of the parts in #8 contains the actual data that is being sent?

10. MPI_Send and MPI_Recv are what type of send and receive? (Blocking or Nonblocking?)

11. Explain briefly what “blocking” means.

12. In the example from section 4.2.3,
A. describe what "a" is, the data that is being sent.
B. where is it being sent?
C. is there a tag? if yes, what is the tag?
D. what is the "communicator"?

13. Describe briefly what are the two things that may happen at runtime to the message being sent with MPI_Send.
1.

2.

14. MPI_Send and MPI_Recv block the calling process. Does either return before the communication operation it invoked is completed? (yes/no)

15. Describe briefly what "completion" in #14 means, include a separate description for MPI_Recv and MPI_Send

MPI_Recv:

MPI_Send:

16. Blocking creates the possibility of deadlock. What does deadlock mean?

17. A. Describe the situation in the example from 4.2.6 that causes a deadlock.

B. What is changed in the example in 4.2.6.1 in order to avoid the deadlock?

18. In the example in section 4.2.6.2 both processes issue a Send first, then a Recv second. Does this necessarily cause a deadlock? Why/why not?

19. What is the change in program example 4.2.6.3 to make a probable deadlock situation? (note that the program is basically the same as 4.2.6.2, what's different?)

NON-BLOCKING SENDS and RECVs

20. Nonblocking sending and receiving requires two calls per communication operation.
The first call does what?

The second call does what?

21. MPI_Isend, the nonblocking send, includes an additional output argument (parameter) – a request handle. What is it's purpose?

22. If a send or receive is posted by a nonblocking routine, its completion status can be checked by calling one of a family of completion routines. These completion routines can be either blocking or nonblocking. What is an MPI completion routine that is blocking.

23. What MPI routine checks for the posted operation's completion?

24. What's an advantage of using nonblocking routines?

25. Why does the program in 4.3.6 not cause a deadlock? Both processes begin by posting a receive. (compare with program in 4.2.6

26. MPI_Send uses "Standard Mode Send". What are the other send modes and their corresponding send routines? (there are 3 more send modes)

27. MPI_SEND is used to send an array of 10 4-byte integers. At the time MPI_SEND is called, MPI has over 50 Kbytes of internal message buffer free on the sending process. Choose the best answer.

A. This is a blocking send. Most MPI implementations will copy the message into MPI internal message buffer and return.

B. This is a blocking send. Most MPI implementations will block the sending process until the destination process has received the message.

C. This is a non-blocking send. Most MPI implementations will copy the message into MPI internal message buffer and return.

28. MPI_SEND is used to send an array of 100,000 8-byte reals. At the time MPI_SEND is called, MPI has less than 50 Kbytes of internal message buffer free on the sending process. Choose the best answer.

A. This is a blocking send. Most MPI implementations will block the calling process until enough message buffer becomes available.

B. This is a blocking send. Most MPI implementations will block the sending process until the destination process has received the message.

29. MPI_SEND is used to send a large array. When MPI_SEND returns, the programmer may safely assume

A. The destination process has received the message.

B. The array has been copied into MPI internal message buffer.

C. Either the destination process has received the message or the array has been copied into MPI internal message buffer.

30. MPI_ISEND is used to send an array of 10 4-byte integers. At the time MPI_ISEND is called, MPI has over 50 Kbytes of internal message buffer free on the sending process. Choose the best answer.
A. This is a non-blocking send. MPI will generate a request id and then return.
B. This a non-blocking send. Most MPI implementations will copy the message into MPI internal message buffer and return.

31. MPI_ISEND is used to send an array of 10 4-byte integers. At the time MPI_ISEND is called, MPI has over 50 Kbytes of internal message buffer free on the sending process. After calling MPI_ISEND, the sending process calls MPI_WAIT to wait for completion of the send operation. Choose the best answer.
A. MPI_Wait will not return until the destination process has received the message.
B. MPI_WAIT may return before the destination process has received the message.

Why?

32. MPI_ISEND is used to send an array of 100,000 8-byte reals. At the time MPI_ISEND is called, MPI has less than 50 Kbytes of internal message buffer free on the sending process. Choose the best answer.
A. This is a non-blocking send. MPI will generate a request id and return.
B. This is a blocking send. Most MPI implementations will block the sending process until the destination process has received the message.

33. MPI_ISEND is used to send an array of 100,000 8-byte reals. At the time MPI_ISEND is called, MPI has less than 50 Kbytes of internal message buffer free on the sending process. After calling MPI_ISEND, the sending process calls MPI_WAIT to wait for completion of the send operation. Choose the best answer.
A. This is a blocking send. In most implementations, MPI_WAIT will not return until the destination process has received the message.
B. This is a non-blocking send. In most implementations, MPI_Wait will not return until the destination process has received the message.
C. This is a non-blocking send. In most implementations, MPI_WAIT will return before the destination process has received the message.