COMP 4040

Exam 3

Fall 2013

Name (Last, First): _____

Rules:

- No potty breaks.
- Turn off cell phones/devices.
- Closed book, closed note, closed neighbor.

- WEIRD! Do not write on the backs of pages. If you need more pages, ask me for some.

Reminders:

- Verify that you have all pages.
- Don't forget to write your name.
- Read each question carefully.
- Don't forget to answer every question.

Additional Items:

- For questions that involve writing code:
  - You may omit `import`/`#include` statements.
  - You may omit exception-handling code.

1. [2pts] Parallel programming mainly enhances which one of the following software quality attributes?

    a. Reusability

    b. Comprehensibility

    c. Reliability

    d. Performance

    e. Maintainability

Imagine that the Open MPI "Hello World" program in Figure 1 was run with the option "`-np 8`".

2. [2pts] The variables `rank` and `size` would be set to what values?

    a. `rank` = 8, `size` = number in range 0–7

    b. `size` = 8, `rank` = number in range 0–7

    c. `size` = 8, `rank` = number in range 1–8

    d. `rank` = 8, `size` = 8

    e. Undefined (can't know what values)

3. [2pts] How many times would the "Hello world…" message be printed?

    a. 0

    b. 1

    c. 7

    d. 8

    e. Undefined (can't know how many)

4. [2pts] In what order would the messages be printed?

    a. There would be only 0 or 1 messages

    b. In numeric order from 0 to 7

    c. In numeric order from 1 to 7

    d. In numeric order from 1 to 8

    e. Undefined (can't know the order)

5. [10pts] Write code to complete the Open MPI program in Figure 3 such that the program works as follows: One process, let's call it *teacher*, is going to take roll (i.e., see that all other processes are in attendance). Let's call all the other processes *students*. To do the roll call, the teacher sends a message to each student. This message should contain only the integer 0. After each student receives this message, the student should send a message back to the teacher, this time containing only the integer 1. Once the teacher has confirmed that all students are in attendance, it should print the message "Everyone's here!"). See Figure 2 for relevant API documentation. If you need more room, there's space on the next page.

6. [10pts] Now, imagine an Open MPI program that has one process (#0) that plays the role of *accumulator*, and the rest of the processes play the role of *worker*. The accumulator has a bunch of jobs, each with a different integer ID. To process a job, the accumulator sends the job's ID to a worker. When the worker receives the ID, the worker executes a function with this signature:

```
int do_job(int jobID)
```

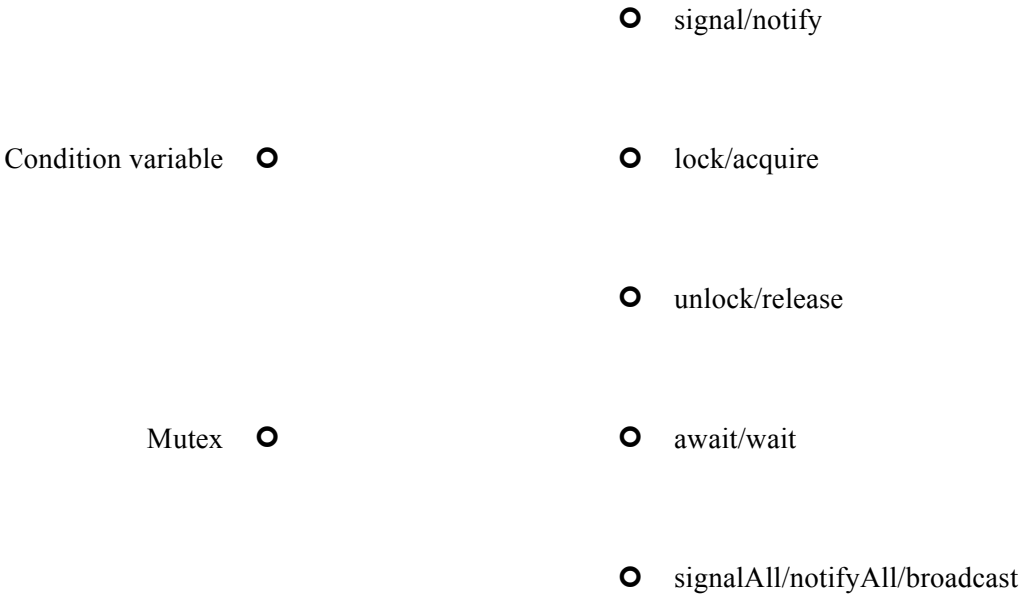
Then, the worker sends the result returned from the function back to the accumulator. Workers do jobs in this manner until there are none left. When the accumulator has no more jobs, it sends each worker a job ID with value -1 to tell the worker to exit.

Write code that implements the worker's behavior (not the accumulator). You can call `do_job()` in your code, but do not define it–assume it has already been defined. There is space on this and the next page to write your answer. As before, see Figure 2 for relevant API documentation.

7. [6pts] Consider the multithreaded Java program in Figure 4. Fill in this table such that it lists all the different values for `a` and `b` that this program might print. Also, give the thread schedule (using comments A–D in the figure) that produced the values. Some rows in the table may be left empty.

Schedule	`a`	`b`

8. [5pts] Draw lines that connect each operation to its associated synchronization mechanism.



9. [5pts] Turning again to Figure 4, imagine that the program contains the following error. If a thread should execute line A, then it must also execute line B before any other thread reads or writes variable `a` or variable `b`. Similarly, if a thread should execute line C, then it must also execute line D before any other thread reads or writes variable `a` or variable `b`. However, the current program does not have these properties.

   Describe in plain English how you would use synchronization mechanisms to correct this error. Tell which mechanism(s) you would use and how many. Also, tell exactly what operations you would invoke and when. Hint: See the previous question for all the possible mechanisms and operations.

10. [2pts] According to question #9, what type of error does the program in Figure 4 contain?

    a. Deadlock

    b. Starvation

    c. Memory Leak

    d. Race Condition

    e. Time of Check to Time of Use

11. [14pts] Give the result of evaluating each of the following Racket expressions.

`5`	
`(+ 1 5)`	
`'(+ 1 5)`	
`(reverse '(+ 1 5))`	
`(reverse (+ 1 5))`	
`(reverse (list (+ 1 5) 7)))`	

`(if (= (- 6 1) 5)`
`  (first '(x y z))`
`  (rest '(x y z)))`	

12. [10pts] Consider this Racket function definition.

```
(define (nth-to-last n list) ( "YOUR ANSWER HERE" ))
```

    Rewrite this function such that it (recursively) finds the item in `list` that is `n`th from the end of the list. For example,

    - the expression `(nth-to-last 1 '(a b c d))` should evaluate to `'d`,
    - the expression `(nth-to-last 2 '(a b c d))` should evaluate to `'c`,
    - the expression `(nth-to-last 3 '(a b c d))` should evaluate to `'b`,
    - etc.

    Do not worry about checking whether there are at least `n` elements in the list, or whether `n` is greater than 0. Hint: Essentially all the functions you need to solve this are in question #11.

    Write your answer on the next page.

Extra Credit Questions

13. [1pts] What was your favorite topic that we covered this semester?

14. [1pts] What programming languages topic was not covered in this course that you wish had been?

15. [1pts] What did you learn in this course that you think will be most useful to you going forward?

```
int main(int argc, char* argv[])
{
  int rank;
  int size;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank); /*** Set rank value ***/
  MPI_Comm_size(MPI_COMM_WORLD, &size); /*** Set size value ***/

  printf("Hello world: %d, %d\n", rank, size);

  MPI_Finalize();
  return 0;
}
```

Figure 1. Open MPI "Hello World" program.

MPI_Recv

Syntax:

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag,
             MPI_Comm comm, MPI_Status *status)
```

- `buf` - Initial address of receive buffer (choice).
- `count` - Maximum number of elements to receive (integer).
- `datatype` - Datatype of each receive buffer entry (handle).
- `source` - Rank of source (integer).
- `tag` - Message tag (integer).
- `comm` - Communicator (handle).
- `status` - Status object (status).

Examples:

```
// Receive a number from process 5.
int number;
MPI_Recv(&number, 1, MPI_INT, 5, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

// Receive a number from any process.
MPI_Status status; // A place to store the rank of the sender.
MPI_Recv(&number, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status);
// Now, status.MPI_SOURCE is the rank of the sender.
```

MPI_Send

Syntax:

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag,
             MPI_Comm comm)
```

- `buf` - Initial address of send buffer (choice).
- `count` - Maximum number of elements to send (nonnegative integer).
- `datatype` - Datatype of each send buffer element (handle).
- `dest` - Rank of destination (integer).
- `tag` - Message tag (integer).
- `comm` - Communicator (handle).

Example:

```
// Send the number 69 to process 9.
int number = 69;
MPI_Send(&number, 1, MPI_INT, 9, 0, MPI_COMM_WORLD);
```

Figure 2. Open MPI API documentation.

```
int main (int argc, char* argv[])
{
  int rank;
  int size;

  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);

  /************************
   * YOUR CODE GOES HERE! *
   ************************/

  MPI_Finalize();
  return 0;
}
```

Figure 3. Open MPI code skeleton.

```
public class MyApp {
       private static int a = 0;
       private static int b = 0;

       public class T1 implements Runnable {
              public void run() {
/*A*/                  a += 3;
/*B*/                  b = a - 1;
              }
       }

       public class T2 implements Runnable {
              public void run() {
/*C*/                  b *= 2;
/*D*/                  a = b;
              }
       }

       public static void main(String[] args) throws InterruptedException {
              MyApp app = new MyApp();
              Thread t1 = new Thread(app.new T1());
              Thread t2 = new Thread(app.new T2());
              t1.start();
              t2.start();
              t1.join();
              t2.join();
              System.out.println("a = " + a + ", b = " + b);
       }
}
```

Figure 4. Example of "simple" multithreaded Java program.