CSCI 610 Project 4
Group Assignment
Due: April 13, 2020
Points: 75

Purpose: The objective of this project is to implement several distributed mutual exclusion algorithms and use each to solve a specified synchronization problem.

Requirements:

1. This project is based on Section 6.3 (pages 321 – 329) of the textbook: *Distributed Systems* by van Steen and Tanenbaum, third edition. You are to write two distributed programs. Each is to solve the Dining Philosophers Problem using a different mutual exclusion algorithm. In particular, one program must use the centralized algorithm, and one program must use the distributed algorithm.

2. *Mutual Exclusion Algorithms*: Mutual exclusion is to be used to update shared variables related to the Dining Philosophers Problem. For a description of this problem, see a textbook such as: *Operating System Concepts with Java* by Silberschatz, Galvin and Gagne, 8th edition, pp. 262 – 264.

3. *Dining Philosophers Solution*: A process must be used to represent each dining philosopher; there must be a total of five such processes. The duration of the eating and thinking phases is to be randomly selected at run-time by each process. Your code should display messages that correspond to the start and stop of each of these phases. These phases must not interfere with the communication responsibilities of the process related to mutual exclusion. For example, the reply to a message must not be delayed because the philosopher is eating or thinking. In addition the completion of a phase must not be delayed because I/O is blocked.

   Your solution must be deadlock free.

4. *Shared Variables*: The shared variables are to be accessed in some manner that could be supported in a distributed environment.

5. *Output*: Your output must clearly demonstrate that each mutual exclusion algorithm is effective. Further, it is expected that it will demonstrate that more than one philosopher is able to eat at one time.

6. *Error checking*: Provide appropriate error checking for each system call and take appropriate steps if an error is encountered.

7. *Communications*: You must use Internet type sockets. Though I only expect you to test your program on the virtual machine (199.17.28.75), it should be possible to have the processes located on different machines and still have the programs communicate correctly.

8. *Man page*: create man pages that describe the usage of your programs.

9. *Readability:* Your program must be written using good programming conventions:
   - Variable names and function names should be descriptive of what they represent.
   - Use indentation to show the structure of the program. Typically this means using indentation with *for*, *while*, *do-while*, *if*, and *switch* statements as well as indenting the body of functions. Indentation should show the nesting level of the statements involved.
   - Include some in-line documentation to give a high level view of what the programs and groups of statements is/are doing.

Suggestions:

You might want to consider the use of non-blocking I/O and/or *select*.

Turn-in:

1. A summary report that describes your design and implementation of the two programs. Discuss which program was easiest to create and why.
2. A listing of your programs.
3. Your man page documentation.
4. Output that demonstrates that the code performs correctly. Your output must demonstrate that your programs function correctly. Highlight portions of the output that demonstrate the requirements have been met; for example, show cases where two philosophers are eating at the same time.
5. Leave a copy of your source and executable files under your Coursefile work folder for this class in a folder called Project4. Leave an empty file in the folder that has your name on it.

   Turn in your Project 4 results (summary report, program listing, output, and man page) in a Word or PDF document with the results clearly identified to the D2L Assignment Folder for Project 4. There should be one report per group. Also, each person is to individually turn in their peer evaluation form to the Project 4 Evaluation Assignment folder.

Grading:

Grading of this project will be divided into two parts:

a. A group score (g) consisting of up to 80% of the available points (60 points).
b. An individual (i) score.

These two parts are added together to determine your total score for this project.

The group score is based on the following criteria with potential points allocated approximately as indicated:

a. Report (15%): your report is expected to include any design considerations.
b. Documentation (15%): Your man page descriptions.
c. Readability (10%): Your program must be written using good programming conventions as described above.
d. Correctness (45%): Does your program meet the requirements?
e. Testing (15%): Is there evidence of significant testing.

Each group member is to assign points to the other group members, but not to him/herself. Suppose there are n group members and n*(0.20 * 75) points to be assigned (pool_points). Then each person would assign ((n-1)/n) * pool_points to the other members of the group. This assignment is to be based on the contribution of each group member; in making this decision, consider both the effort expended by each group member and their effectiveness. A signed form with this point distribution is to be given to the instructor by each group member when the project is turned in. The instructor will compute an average potential score, p_k for each group member based on the input received. The actual individual score will then be determined by: $i = p_k * (g / 60)$.

Name: _____

Group : _____

Project: _____

a. Number of people in group: _____ (n)

b. Available Points to Assign: _____ (n – 1) * 0.20 * 75

For each person in the group, except yourself, assign points to that person based on their effort and effectiveness in the project. Note that the sum of the points must not exceed the value computed on line-b above.

Name	Assigned Points	Justification

Signature: _____