CS 420
HOMEWORK ASSIGNMENT H4

DUE DATE: Wednesday, October 19, 2011

1. The OS Chicken (this exercise worth 70% of the total assignment grade)

The OS chicken is modeled by 4 threads, representing the function of different body parts. The *eyes* look for food and inform the *brain* when food is detected. Once food is located, the brain informs the *feet* to move a certain distance to reach the food. Once the feet have moved the indicated distance they inform the brain, which tells the *mouth* to start eating. These chickens do not have memory; at any point the eyes may identify other food, which will cause the brain to tell the feet to stop any walking, the *mouth* to stop eating, and then move toward the new food and eat it.

Develop a program that simulates this behavior. Your threads should behave as above, with parameters as follows:

- *eyes:* look for food every 1000–2000ms, finding food 50% of the time. Each time they look for food print out a string "looking!".
- *brain:* receives input from the *eyes* indicating food has been found. Once so informed emit a string "saw food!" and then tells the *mouth* to stop eating, the *feet* to stop any current movement and to move toward the newly identified food. Once the *feet* indicate the destination is reached a string is emitted "arrived food!" and the *mouth* is told to start eating.
- *feet:* receives input from the *brain* indicating new food. Once so told, it stops the current movement if any, and moves random 5–10 steps to the new food, 400ms per step. Before each step is taken print out a string "step x" where x is the number of this step toward the current food location. Once the steps have all been taken it tells the *brain* it has walked to the destination.
- *mouth:* waits for a signal from the *brain* to start eating. Eating lasts until it is informed to stop; eating each mouthful takes 400ms and should emit a string "munching!".

The simulation terminates only when the user kills the program.

Develop <u>two</u> solutions to this based on the above specification:

- Use only semaphores (from Java library) for synchronization. Try and minimize the use of shared variables other than the semaphores.
- Use only monitors based on `synchronized` for the solution. There should be no shared variable access outside of your monitor(s).

Create an MS-DOS batch file named chicken.bat that will execute the program. Store all files necessary for the execution of your application in folders named Prob1A and Prob1B that you submit electronically for the assignment, as instructed below. Be sure the output of the program clearly indicates what the program is doing.

2. Safe States

Assume one computer system has several concurrent processes and multiple instances of one resource used by these processes. For each of the following snapshots of a state of resource usage, identify if the system is in a "safe" state or an "unsafe" state. If it is a safe state, show one safe sequence, if it is in an unsafe state, show a sequence of execution that could result in deadlock.

Snapshot 1 – 12 resource instances total

Process	Max Need	Allocated	Claimed
P_1	4	1	3
P_2	6	4	2
P_3	8	5	3
P_4	2	0	2

Snapshot 2 – 12 resource instances total

Process	Max Need	Allocated	Claimed
P_1	4	1	3
P_2	6	4	2
P_3	8	6	2
P_4	2	0	2

Snapshot 3 – 12 resource instances total

Process	Max Need	Allocated	Claimed
P_1	10	8	2
P_2	5	2	3
P_3	3	1	2

3. One method for denying the "wait-for" condition needed for deadlock requires that processes must request all of the resources they will need before the system may let them proceed. The system grants resources on an "all-or-none" basis. Discuss the pros and cons of this method.

4. A system has three processes and four identical resources. Each process requires at most two of the resources at any given time.

- **a.** Can deadlock occur in this system? Explain.
- **b.** If there are *m* processes, and each could request up to *n* resources, how many resources must be available in the system to ensure that deadlock will never occur?
- **c.** If there are *m* processes and *r* resources in the system, what maximum number of resources, *n*, could each process request, if all processes must have the same maximum?

Either type your solutions or print legibly. Solutions that cannot be easily deciphered are incorrect!

General Instructions:

- Printed parts of homework solutions are to be submitted at the start of class on the due date.
- Homework submissions will be machine printed on standard letter size white paper with multiple pages stapled together securely to form one "package".
- Homework submissions must be prepared using computer document preparation applications such a word processor or similar editor. Handwritten solutions are not acceptable – neatness, readability and grammar count!
- Homework submissions will be clearly marked with the student's name, date and assignment identification at the top of the first page.
- All homework is to be completed by each student individually and represent that student's original, unassisted work. Any material copied in any way from other sources must be clearly identified and attributed.

Programming Instructions:

- Put a block of comments at the beginning of every physical file containing program source code that includes your name, the course name and number, information identifying what functions the program is designed to perform, and instructions how to execute the program. (required)
- *Turn in a printed copy of the source code* for your programs with your solution to any non-programming exercises at the start of class on the due date – see general instructions above.
- Place the Java source files, class files, batch files and all other files necessary to execute each program you write into a Windows folder that is named with your email ID. Create a separate subfolder within that folder for each problem in the assignment and name these folders Problem1, Problem2, etc. Make this folder your working directory before creating the zip file as described below.
- Place that folder (containing all the `ProblemX` subfolders) into a zip file. Create the zip file so that the folder structure (path) is also recorded by selecting the "save full path info" option. Use your email ID as the zip file name. Example of file structure for submission:

  Zip file named `brixiusn.zip` contains a folder named `brixiusn` that contains subfolders named `Problem1`, `Problem2 and Problem3`. Each subfolder has source code, class files and any required data files. The zip file records the path information for each file.

- Submit the zip file in the course Blackboard site using the assignment submission capability in the same location you accessed this assignment information. You can also add comments when you submit a file for the assignment.
- Each assignment must be submitted on Blackboard by 0030 on the day the assignment is due.
- Submit only one zip file with your entire assignment.
- If you have already submitted a homework assignment and then decide you must resubmit the assignment before it is due, you can submit another zip file to replace previous submissions. You can also use comments with the submission to further explain your submission to the grader. You may resubmit as many times as you find necessary before the assignment due date.
- In summary, your complete homework submission consists of:
  - The electronic submission of the zip file to the Blackboard by 0030 on due date
  - Any printed documents required will be submitted at the start of class