Operating Systems
Programming Assignment General Guidelines

2015-05-27

1 Plagiarism

Texas A&M University-Commerce does not tolerate plagiarism and other forms of academic dishonesty. Conduct that violates generally accepted standards of academic honesty is defined as academic dishonesty. "Academic dishonesty" includes, but is not limited to, plagiarism (the appropriation or stealing of the ideas or words of another and passing them off as one's own), cheating on exams or other course assignments, collusion (the unauthorized collaboration with others in preparing course assignments), and abuse (destruction, defacing, or removal) of resource material. http://www.plagiarism.org/

As mentioned in the syllabus, I expect students to perform the course assignments on your own, they are not intended to be group projects. I want to see your level of understanding and ability at implementing the concepts we are covering for this course. All works submitted for credit must be original works created by the scholar uniquely for the class. It is considered inappropriate and unethical to make duplicate submissions of a single work for credit in multiple classes, unless specifically requested by the instructor. Plagiarism is passing off another's ideas or work as ones own, and that applies equally as much to software source code as it does to written assignments and tests. That being said, it is fine to discuss the assignments and approaches to them with other students. But there is a line that you should be aware of when helping each other. Definitely over or crossing this line is any copying, sharing, e-mailing, etc. of your code to someone else or receiving same from someone else. This includes from past students or from anyone for that matter. Work submitted is to be your own, developed to satisfy the assignments for the course this semester. If you are caught copying or passing off someone elses work as your own, you may receive a 0 for the assignment or even be assigned a failing grade for the course.

2 C/C++ Tutorials and Hints

I do expect that you are already familiar with programming in C and/or C++, as these are prerequisites to our program. All of the programming assignments for CSci 430/530: Operating Systems are required to be completed in C or C++. You may use any of the ANSI standard C/C++ libraries in your assignments (stdio, iostream, etc.). You may also use the C++ Standard Template Library (STL) for your assignments.

That being said, here are some resources that students have used or mentioned in the past, that they have found helpful for brushing up on C/C++ programming:

- This site shows up at the top of the Google search for C++ tutorials. And I do agree it has both a pretty good tutorial, plus it makes a good online reference. You can also find references at this site here for the standard C and STL library usage.

- A popular plain C tutorial, if you wish to avoid the object oriented aspects and program with straight up functions.
- The Student Resources page for the Stallings textbook website has several links to tutorials for C and C++.

3 Coding Guidelines

All programming assignment source code submitted to CSci 430/530: Operating systems must meet the following requirements. I consider the following as **minimal** requirements for a program to be acceptable enough to begin to try and read and understand it. Failure to at least attempt to conform to these minimal standards may result in me returning your assignment unlooked at, or even being assigned a grade of 0 if you repeatedly fail to understand and follow these guidelines.

Good coding standards and practices are, in my opinion, not tedious busy work. Good standards actually reflect good practices that help make you a better programmer. Also, you should not think of developing code as primarily a task of communicating with the machine to get it to perform a particular task. Good software is written primarily with human readability in mind, and is targeted at making complex algorithms both executable by the machine but also understandable by developers, both current and future, who may read and need to understand what the program is doing. This is the philosophy behind Don Knuth's Literate Programming methodology: that code should be written primarily to communicate its purpose to humans.

Here are the specifics of the minimal guidelines that you should follow:

4 File header documentation

All assignment files you create for this course must have a file document header included at the beginning of the source code file. We will use a pseudo doxygen format, and ask you to provide the following information at the beginning of each of your source files submitted for assignments for this course:

```
/**
 * @author Joe Student
 * @cwid   123 45 678
 * @class  CSci 530, Summer 2015
 * @ide    Visual Studio Express 2013
 * @date   June 8, 2015
 * @assg   prog-01
 *
 * @description Provide a short description of the problem
 *      and the approach you took to solving the problem.
 */
```

5 Function header documentation

In a similar manner, all functions you write for this course should include a function header declaration and documentation using (pseudo) doxygen formatted comments. Almost all programming shops require that all classes and functions (including member functions) be documented in this manner where they are declared. The purpose is mainly to document the input parameters to the function, and any output results the function returns. Use the following format to document all functions you write for assignment for this course:

```
/** Check for process arrivals
 * Check the process table information to find processes that are
 * arriving.  Place all arriving processes at the end of the round
 * robin scheduling queue.
 *
 * @param currentTime An int value, the current time step of the
 *              simulation, check for processes in table arriving at
 *              this time.
 * @param processTable A pointer to a ProcessTable struct, a list of
 *              all the process information for processes we are
 *              simulating, including their arrival times.
 * @param rrQueue A STL list holding pointers to Process struct items.
 *              This is our simulated round robin queue. This parameter
 *              is passed as a reference parameter.  We will add
 *              any arriving processes to the end of this queue as a
 *              side effect of calling this function, since it is a
 *              reference parameter.
 * @returns bool True if a process arrived, false otherwise
 */
bool checkProcessArrivals(int currentTime, ProcessTable* processTable,
                          list<Process*>& rrQueue)
{
  for (int pid=0; pid < processTable->numProcesses; pid++)
  {
    Process* p = processTable->process[pid];
    if (p->arrivalTime == currentTime)
    {
      DEBUG(cout << "   Process arrives: "
                 << p->processName << endl;)
      rrQueue.push_back(p);
      return True
    }
  }

  return False
}
```

6 Indentation

Make sure you pay special attention to the Deital style guidelines regarding proper indentation of course code. For this course we require you to use spaces (no embedded tabs) for indentation. You are required to use 2 spaces as the unit of indentation for all code/block levels for code submitted for this class. DO NOT use hard coded (embdeded) tabs in your submitted programs.

7 Function and Variable Names

All functions and variables should follow camelCaseNameing convention for your programs for this course. When creating a user defined type, like a class or a struct, you should use CamelCaseNameing

with the initial letter capitalized (this differentiates functions and variables from classes and user defined types). Constants, in enumerated types or otherwise, should use `ALL_CAPS_UNDERSCORE` convention for nameing. Make sure you choose meaningful variable, function, class and constant names for your programs, as meaningful well chosen names make your programs more readable and reduce the need for extensive comments. See Deitel guidelines for more hints on how to choose meaningful variable names for your programs.

8 Brace Placement for Control Blocks

For this course you are required to place all braces defining a control block (like a for loop or if statement) on a separate line by themselves, indented appropriately. For example, this function has 2 levels of indentation, and all levels are consistently indented and all opening/closing braces appear on their own line for readability:

```
/** Display a matrix
 * A helper function for debugging.  Display a state matrix to
 * standard output
 *
 * @param rows The number of rows in the matrix
 * @param cols The number of cols in the matrix
 * @param m A 2 dimensional array of rows x cols integers
 */
void displayMatrix(int rows, int cols, int v[MAX_PROCESSES][MAX_RESOURCES])
{
  int r, c;

  // display column headers
  cout << "   "; // extra space over for row labels
  for (c = 0; c < cols; c++)
  {
    cout << "R" << c << " ";
  }
  cout << endl;

  // now display data in matrix
  for (r = 0; r < rows; r++)
  {
    cout << "P" << r << " ";
    for (c = 0; c < cols; c++)
    {
      cout << setw(2) << v[r][c] << " ";
    }
    cout << endl;
  }
  cout << endl;
}
```

9 Whitespace

Pay special attention to the programming guidelines regarding space within statements and between blocks of code and functions. For example, always put a single space before and after all binary operators (like +, -, «, etc.). Put a single space after commas (,) and semicolons (;) separating lists of parameters or declarations in functions/control blocks. But a single blank line before and after a control block inside of a function. For this course, you should place 2 blank lines between the end of each function and the beginning of the next function documentation. In general, pay attention to the formatting of whitespace in our Deitel textbook and follow the conventions shown there.