Process ID Assignment

Overview

An operating system's pid manager is responsible for managing process identifiers. At process creation, the pid manager assigns the process a unique pid. When the process completes execution, it returns the pid to the pid manager which in turn may later reassign this pid to another process. Chapter 3 provides a full discussion of process identifiers. What is most important here is recognizing that process identifiers must be unique; no two active processes can have the same pid.

Design

1. The first task in this assignment creates the pid manager whose implementation can simply be a single class. Of course, you can create any other classes you might need to implement the pid manager. You may use any data structure of your choice to represent the availability of process identifiers. One strategy adopts Linux's approach of a bitmap in which a value of 0 at position i indicates that a process id of value i is available and a value of 1 indicates that the process id is currently in use. Use the following constants to identify the range of possible pid values: MIN_PID is 300 and MAX_PID is 5000.

   Have your pid manager implement the following API for obtaining and releasing a pid:
   - **int allocate map( )** – Creates and initializes a data structure for representing pids; returning 0 if unsuccessful or 1 if successful.
   - **int allocate pid( )** – Allocates and returns a pid; returns 1 if unable to allocate a pid, all pids are in use.
   - **void release pid(int pid)** – Releases a pid.

2. The second task in this assignment consists of writing a multithreaded program that tests your pid manager. Implementing the threads uses either **extends Thread** or **implements Runnable**. Create a number of threads where each thread requests a pid, sleeps for a random period of time, releases the pid, and then terminates. Sleeping for a random period of time approximates the typical pid usage in which a new process acquires a pid, the process executes and then terminates, releasing the pid upon its termination. Download the SleepUtilities.java file needed for the assignment. A thread sleeps by calling the SleepUtilities.nap(duration) function, passing an integer value representing the number of seconds to sleep, where duration is a randomly generated integer between 60 and 300. Before sleeping, each thread should print out (on a new line) the message, "My PID is: x.", where x is the actual pid for the thread.

3. Create a driver class and make the name of the driver class **Assignment1** containing only one method: `public static void main(String args[]).`
   The main method itself is fairly short containing code to do the following:
   a. Create the pid manager object.
   b. Create 100 threads.
   c. For each thread, pass the pid manager into the thread and begin the execution of the thread.
   d. The main method needs to keep track of the threads and take care of each thread before it can end. The methods of the Thread class you'll need for this are join() and isAlive(). If the thread is dead then execute a join on it in order to properly dispose of that thread. If the thread is still alive then move on to the next thread. The main method keeps doing this check until the last remaining thread has died and been properly disposed.

4. You must declare public each class you create which means you define each class in its own file.

5. You must declare private the data members in every class you create.

6. You can only use extends in this assignment (extends Thread) when defining your thread class. Though, you don't have to use "**extends Thread**" to define your thread class. Remember, "**implements Runnable**" is the other way in Java to define your thread class.

7. **Tip:** Make your program as modular as possible, not placing all your code in one .java file. You can create as many classes as you need in addition to the classes described above. Methods being reasonably small follow the guidance that "A function does one thing, and does it well." You will lose a lot of points for code readability if you don't make your program as modular as possible. But, do not go overboard on creating classes and methods. Your common sense guides your creation of classes and methods.

8. Do **NOT** use your own packages in your program. If you see the keyword **package** on the top line of any of your .java files then you created a package. Create every .java file in the **src** folder of your Eclipse project, if you're using Eclipse.

9. Do **NOT** use any graphical user interface code in your program!

10. Do **NOT** type any comments in your program. If you do a good job of programming by following the advice in number 7 above then it will be easy for me to determine the task of your code.

Grading Criteria

The total assignment is worth 20 points, broken down as follows:

1. If your code does not implement the task described in this assignment then the grade for the assignment is zero.
2. If your program does not compile successfully then the grade for the assignment is zero.
3. If your program produces runtime errors which prevents the grader from determining if your code works properly then the grade for the assignment is zero.

If the program compiles successfully and executes without significant runtime errors then the grade computes as follows:
Followed proper submission instructions, 4 points:

1. Was the file submitted a zip file.
2. The zip file has the correct filename.
3. The contents of the zip file are in the correct format.
4. The keyword **package** does not appear at the top of any of the .java files.

Code implementation and Program execution, 12 points:

- The driver file has the correct filename, **Assignment1.java** and contains only the method **main** performing the exact tasks as described in the assignment description.
- The code performs all the tasks as described in the assignment description.
- The code is free from logical errors.
- Program output, the program produces the correct results for the input.

Code readability, 4 points:

- Good variable, method, and class names.
- Variables, classes, and methods that have a single small purpose.
- Consistent indentation and formatting style.
- Reduction of the nesting level in code.

Late submission penalty: assignments submitted after the due date are subjected to a 2 point deduction for each day late.

Late submission policy: you **CAN** submit your assignment early, before the due date. You are given plenty of time to complete the assignment well before the due date. Therefore, I do **NOT** accept any reason for not counting late points if you decide to wait until the due date (and the last possible moment) to submit your assignment and something happens to cause you to submit your assignment late.

Submission Instructions

Go to the folder containing the .java files of your assignment and select all (and **ONLY**) the .java files which you created for the assignment in order to place them in a Zip file. The file can **NOT** be a **7z** or **rar** file! Then, follow the directions below for creating a zip file depending on the operating system running on the computer containing your assignment's .java files.

Creating a Zip file in Microsoft Windows (any version):

1. Right-click any of the selected .java files to display a pop-up menu.
2. Click on **Send to**.
3. Click on **Compressed (zipped) Folder**.
4. Rename your Zip file as described below.
5. Follow the directions below to submit your assignment.

Creating a Zip file in Mac OS X:

1. Click **File** on the menu bar.
2. Click on **Compress ? Items** where ? is the number of .java files you selected.
3. Mac OS X creates the file **Archive.zip**.
4. Rename **Archive** as described below.
5. Follow the directions below to submit your assignment.

Save the Zip file with the filename having the following format:

your last name,
followed by an underscore _,
followed by your first name,
followed by an underscore _,
followed by the word **Assignment1**.

For example, if your name is John Doe then the filename would be: **Doe_John_Assignment1**

Once you submit your assignment you will not be able to resubmit it!
Make absolutely sure the assignment you want to submit is the assignment you want graded.
There will be **NO** exceptions to this rule!

You will submit your Zip file via your CUNY Blackboard account.
The only accepted submission method!

Follow these instructions:

Log onto your CUNY BlackBoard account.
Click on the CSCI 340 course link in the list of courses you're taking this semester.
Click on **Content** in the green area on the left side of the webpage.
You will see the **Assignment 1 – Process ID Assignment**.
Click on the assignment.
Upload your Zip file and then click the submit button to submit your assignment.

Due Date: Submit this assignment by Thursday, November 14, 2019.