EECS268:Lab3

Navigation

Home
Information

Syllabus
Schedule
Lecture Archive

Classwork

Labs
Submitting Work

Contents

- 1 Overview
- 2 File overview
  - 2.1 Example File
- 3 Stack and Queue
- 4 Rubric
- 5 Submission instructions
  - 5.1 Creating a File Archive Using Tar
  - 5.2 Emailing Your Submission

Overview

You will create a mock CPU scheduler. Programs will get in a queue and wait for their turn to have some time in the processor. While a program is using the CPU it call do simple things like add a function to its call stack. You will need to make a node-based implementation of a Queue and a Stack. You'll also design and implement additional classes that will use the Queue and Stack.

- Process a class
  - A Process contains a call stack, which is a stack of function names
  - A process is in charge of managing its call stack by either making additional calls (adding to the call stack) or having a function return (removing from the call stack)
- CPU_Scheduler
  - The CPU_Scheduler is in charge of adding Processes to a Queue, allowing them CPU time, then moving them to the back of the queue if they need

An Input from a file provided by the user will dictate the order the programs are loaded into the queue and what the program is doing with its CPU time.

File overview

The input file you receive will consist of the commands listed below. After each command is executed display appropriate output to terminal.

If a command is impossible (e.g. calling a function when no processes are in the queue) display an appropriate error message.

COMMAND	Details
START <process name>	A new process is created and added to the queue. All processes start with a "main" as their first function call. Print a message to the screen indicating which process was added to the queue.
CALL <function name>	The process at the front of the queue gets some CPU time and calls a function. This put that function on the call stack for that process. After the call is made, that process goes to the back of the line. Print a message to the screen indicating which process called the function and what the name of the function is.
RETURN	The process at the front of the queue has the function at the top of its call-stack return. If the process has any functions left on the call-stack, put it at the back of the queue. Otherwise, if its main returns, simply remove it. Should a process end, display a message indicating this.

Example File

```
START itunes
START firefox
START putty
CALL play
CALL navigate
RETURN
```

Note this file would then display the following process list:

```
itunes added to queue
firefox added to queue
putty added to queue
itunes calls play
firefox call navigate
putty returns from main
putty process has ended
```

Stack and Queue

You will create node-base implementation of the Stack and Queue class. The Node class can be as simple as this:

```
class Node:
    def __init__(self, entry):
        self.entry = entry
        self.next = None
```

Your Stack class will need the following functionality:

Method	Description
push(entry)	Put the entry at the top of the Stack.
pop()	Remove and return the value at the top of the stack. Raise RuntimeError otherwise.
peek()	Return value at the top of the Stack, raise a RuntimeError otherwise.
is_empty()	Return True if Stack is empty, False otherwise.

Your Queue class will need the following functionality:

Method	Description
enqueue(entry)	Put the entry at the front of the Queue.
dequeue()	Remove and return the value at the front of the Queue. Raise RuntimeError otherwise.
peek_front()	Return value at the front of the queue, raise a RuntimeError otherwise.
is_empty()	Return True if Queue is empty, False otherwise.

Rubric

- 30pts Modularity
  - Sensible class design
  - Completely object oriented (e.g. your main should invoke some kind of executive class)
- 45pts Runtime functionality
  - Compatible with the provide format
  - You may assume the file will be formatted correctly but you need to be able to detect if a command is possible and react accordingly.
  - Well formatted output
- 15pts Stability
  - There should be zero unhanded exceptions when provided with a properly formatted file
  - Even if the command would cause an Exceptions to be raised, your program should be able to handle it
- 10pts Comments and documentation:
  - Please provide docstrings in each of your functions and methods

Submission instructions

Send a tarball/zip file with all the necessary files (.py file and input file) to your GTA. Ask them if they use Canvas, Blackboard, etc.

Creating a File Archive Using Tar

The standard Unix utility for created archived files is **`tar`**. Tar files, often called **tarballs**, are like zip files.

1. Create a folder to hold the files you will be sending in. The folder should be named like *LastName-KUID-Assignment-Number*:

   ```
   mkdir Smith-123456-Lab-0#
   ```

2. Now, copy the files you want to submit into the folder:
3. Tar everything in that directory into a single file:

   ```
   tar -cvzf Smith-123456-Lab-0#.tar.gz Smith-123456-Lab-0#
   ```

   That single command line is doing a number of things:
   - `tar` is the program you're using.
   - `-cvzf` are the options you're giving to `tar` to tell it what to do.
     - `c`: **create** a new tar file
     - `v`: operate in **verbose** mode (show the name of all the files)
     - `z`: **zip** up the files to make them smaller
     - `f`: create a **file**
   - `Smith-123456-Lab-0#.tar.gz`: the name of the file to create. It is customary to add the **`.tar.gz`** extension to tarballs created with the `z` option.
   - `Smith-123456-Lab-0#`: the directory to add to the tarball

Please note that it is **your** responsibility to make sure that your tarball has the correct files. You can view the contents of a tarball by using:

```
tar -tvzf filename.tar.gz
```

Emailing Your Submission

Once you have created the tarball with your submission files, email it to your TA. The email subject line **must** look like "[EECS 268] *SubmissionName*":

> [EECS 268] Lab 0#

Note that the subject should be *exactly* like the line above. Do not leave out any of the spaces, or the bracket characters ("[" and "]"). In the body of your email, include your name and student ID.