

Python Programming III

Offensive and Defensive Tool Construction

Table of Contents

[Objectives 4](#_Toc65107444)

[Background Reading 4](#_Toc65107445)

[Important Information 4](#_Toc65107446)

[Introduction 5](#_Toc65107447)

[Problem 1 30 pts 5](#_Toc65107448)

[Problem 2 30pts 7](#_Toc65107449)

Offensive and Defensive Tool Construction

Python Programming III

Objectives

This lab focuses on the following objectives:

* Define a Python class.
* Create a Python object.
* Use class functions.
* Define class methods.
* Explain inheritance.

Background Reading

Read *How to Think Like a Computer Scientist: Learning with Python*, available at [www.greenteapress.com/thinkpython/thinkCSpy.pdf](http://www.greenteapress.com/thinkpython/thinkCSpy.pdf).

Read the documentation for PID provided on D2L.

# Important Information

**YOU MUST PRESENT IMAGES OF YOUR CODE BEING EXECUTED. DO NOT SUBMIT YOUR ANSWERS IN THIS DOCUMENT. CREATE A BLANK DOCUMENT AND SUBMIT YOUR ANSWERS THERE.**

**YOU WILL LOSE MARKS FOR NOT FOLLOWING THE ABOVE REQUIREMENTS.**

# Introduction

In this lab, we will read and parse contents of the Linux ***/proc*** pseudo-file system. Specifically, you will access the ***/proc/[pid]/stat*** and retrieve pertinent information about each process. There is also another file ***/proc/[pid]/status*** that contains information about the process.

# Problem 1 30 pts

Write a Python program named **m04p01.py** that has a class called **LinuxProcess**. Within the class write methods to extract the fields below. You can have **one or** **multiple getter** method(s) to retrieve each an attribute. You may also have **one or** **multiple setter** method(s) that you provide specific information to set an attribute.

Remember the getter method is used to retrieve information about the object and return it to the requester of that information. The setter method is used to set the information related to that object.

The **LinuxProcess** class **MUST** have a method called **printData** that will print the information provide below.

Test the class by writing code that asks the user for the PID to analyze. You can also recover the PID of your currently executing python program. Apply the class methods and print the results using the format below:

name: m04p01.py

pid: 2345

ppid: 340

rss: 0xfffffffffffffL

rsslim: 0xffffffffffffffffL

start\_code: 0x400000

end\_code: 0x6bb0f4

start\_stack: 0x7fffdd658190

start\_data: 0x8bbdc0

end\_data: 0x9303f4

start\_brk: 0x2872000

arg\_start: 0x7fffdd658661

arg\_end: 0x7fffdd65867d

env\_start: 0x7fffdd65867d

env\_end: 0x7fffdd658fec

**YOU MUST PARSE** **/proc/<PID>/stat** file. In other words you must open the stat file and extract the information requested in the output below.

Some hints:

1. If you decide to have one **setter** or **getter** method, you need some way to let the method know which attribute you are setting or getting.
2. You can pass the information in many different ways for example as a string containing the attribute and the data associated with that data. You can also pass 2 variables to the method.
3. In the scenario you can follow this recipe:
   1. Create the object
   2. Programmatically determine the PID in your main program
   3. Set the PID attribute of your LinuxProcess object
   4. Call the Setter method to set all the attributes
   5. Call the printData method to the data as shown below
   6. Test the scenario where you ask the user for an attribute
   7. Have the getter method return that value
   8. Print that value to the screen from main

If you are looking for a bit more of a challenge also parse **/proc/<PID>/status** to extract some of the values requested below or add other interesting values that could be useful …

You can take some artistic license when presenting the data or adding extra interesting data you found.

You will place a screenshot of the program running at least once to show that it works.

# Bonus 10pts

Write a Python program named **m04p02.py** that creates the **LinuxProcList** class. This class reads all the running processes from ***/proc*** and generates an internal structure that represents the process tree. It also reads the command lines used to invoke each process. You will need to read at least the files ***/proc/[pid]/stat*** and ***/proc/[pid]/cmdline*** for each process. The methods in the class should meet the following criteria:

* LinuxProcList.proclist() returns a list of all process IDs,
* LinuxProcList.cmdline(pid) returns either a string containing the command line for the given process or *None*, and
* LinuxProcList.children(pid) returns a list of the children of the given process.
* Add other methods that may be necessary.

In essence the **LinuxProcList** object is an object that contains **LinuxProcesses**. You created LinuxProcess in the previous example, what is missing from that object is the **cmdline** and **children** attributes for your LinuxProcess.

**YOU MUST PARSE** **/proc/<PID>/stat** and **/proc/<PID>/cmdline** files. As you work through this example, you will see there are some opportunities for code reuse. It is imperative that your code be modular; in other words, you must create functions. You must create classes (with methods and attributes).

**Hints**:

1. Import your **LinuxProcess** class or replicate it in your code
2. **LinuxProcList** is an object that keeps track of a list of **LinuxProcess** objects
3. You can call the **LinuxProcList.proclist()** method that will return a list of processes it knows about
4. A possible approach to test your object:
   1. In main create a **LinuxProcList** object
   2. In your main code get a list of all processes running on your system
   3. Use this list to create LinuxProcess inside the LinuxProcList. You can use a method named **createProc**.
      1. Remember you can modify the LinuxProcess object to contain less information or append to the current structure
   4. Test the program by calling the **LinuxProcList.cmdline(pid)** method and the **LinuxProcList.children(pid)** method.
5. Ask the user to enter a request for processes that don’t have children or processes that have children and print those out.
   1. The print out will contain the processes and their child processes with the cmdline used to execute those processes.

Another possible way to test class is by writing code that presents a list of running processes to the user. Then asks the user to choose a process to get more information on. The program will output the following information to screen:

**LinuxPrompt$** python3 m04p02.py

Here are all the running process on the system …

[ 1, 2, 100, 23143, 2341, 231090, 908, 9]

Please enter a process ID you would like more information about: 2094

Process ID : 2094

PPID : 12  
Process Name : malcontent.elf  
Process CommandLine : /usr/bin/depths\_of\_destruction/  
Process Children :

Worm : 102, /usr/bin/depths\_of\_destruction/

Trojan : 204, /usr/bin/depths\_of\_destruction/

Phishing : 1024, /usr/bin/depths\_of\_destruction/ all\_manner\_of : 1125, /usr/bin/depths\_of\_destruction/

Hajime : 3434, /usr/bin/depths\_of\_destruction/

Mirai : 2032, /usr/bin/depths\_of\_destruction/