## Introduction

For many time, people are annoyed by some process that takes too much system resources. For example, you are doing scientific calculation on your computer and the system tells you “I am going into update software, firewall and most importantly myself. This will rise the occupation of CPU to 99%, networks to 99%. Consequently, your calculation will be delay, your video will be stuck, and you can do nothing to help. When I am done, I will restart the computer regardless of what are you doing.” That is driving people crazy and we would like to make something to help. Linux Process Management is an tools created basing on this purpose.

Github: <https://github.com/Deliangus/Linux_Process_Management.git>

## LPM Design

The Linux\_Process\_Management[LPM] executes a user program and a kernel module in Ubuntu 18.04.1 LTS. The purpose of the project is to monitor processes running in Linux and apply different managing policies to processes. The program requires cooperation between user program and Kernel module. Currently, the kernel module is used to support a system call which collect names and pids of each process running in system. The information collected will be sent to user program in which processes will be classified and apply different policies.

The kernel module is based on sample given by Yuan Xiao. Basic design of kernel module doesn’t differ from sample greatly.

Content:

System\_call.c

The user program is consisted of 4 to 6 parts. Each part performs different functionality.

Content:

Loader.c : Load management policies and working log.

Log.c : Output working log to record the activities of program.

Monitor.c : matches running processes with management policies and applies operation to processes. main() function is designed to be within this part.

Policy.c : provides functions to manage policies and processes manually.

## LPM Implementation:

Kernel module

Kernel Module is consisted of one source file named *system\_call.c*. the main function of kernel module is to provide name and pid of each running process to user program. So the kernel module use *task\_sturct* structure to collect processes’ name and pid. Name and pid of each process will be stored in to an array whose size is 512. Once all processes’ information has been stored, the kernel module use *copy\_to\_user* function to implement the data transferring to user program.

User program

User program accepts the list of processes request from kernel module and manages the processes. It is consisted of 4 parts.

Loader.c provides functions used to modify list of processes of each policy for future management.The loader will maintain four tables, *process\_White\_list, process\_Unknown, process\_Kill and process\_Eliminate.* Each table contains the processes that will be applied with same policy. For example, processes in *process\_Kill* will be killed by LPM whenever LPM finds that it is executing.

Functionality provided by loader:

Load(): initialize tables with information from disk.

Get\_Process(): given a name, search process who has that name. If such process exists, return its pointer. Otherwise add the name to unknown list.

Table\_Update\_Pid(): use information of executing processes given by system\_call to update information of processes including pid and its policy.

Print\_List(): print content of certain table to designated file

Insert\_To\_Table and remove\_From\_Table(): add or remove a process from designated table.

Notice, user could decide the content of each list by modifying four files when the program is not running. The files in classification folder is the set of processes of for policies. For example, file *process­\_Kill* is the content of kill table, the modify to this file could make direct effect to table of Kill.

Files:

Process\_Kill: all processes to be killed.

Process\_Eliminate: all processes to be eliminated.

Process\_Unknown: the processes that LPM doesn’t know what to do with.

Processs\_White\_List: processes that LPM will do nothing with.

Process\_Defined: the content of this file will be identical with process\_Unknown when it has not been modified by user. User can add a note at the end of each line and notes could be ‘W’, ‘K’, ‘E’. For example, let sys2018 be the only process in process\_Unknown. If user makes no difference to *process­\_Eliminate*, the file will contain only one line which is *sys2018.* User can add a note ‘W’ behind the process separating by a blank to indicates that user has decided to put the process sys2018 in white list. Similarly, ‘E’ represents Eliminate and ‘K” means Kill.

Before anytime the program starts, the file process\_Defined must contains only process that has been decided by user, of if none, be cleared.

Log.c provides functions that build a working log for user.

Policy.c defines policies to be applied to each process. Policies includes white list, Kill, Eliminate, Unknown. For process in White list, LPM would do nothing to it. For processes in Kill, LPM kills them when it finds any of them. For processes in Eliminate, LPM is supposed to uninstall the software that creates those proesses, but this function has not been well implemented, so it is a functionality for future using. For processes in Unknown, LPM writes the list to file, and allow user to decide policies for unknown process in file, which will be applied in LPM’s next execution.

Monitor.c helps in configuring kernel module, getting necessary information for other parts, and monitor the activities of executing processes. Once new processes start executing, monitor.c informs policy part and let loader take action to new processes.

Main function is also in monitor.c.

## LPM Data Representation

Tables process\_Unknown, process\_White\_List, process\_Kill and process\_Eliminate share identical structure as indicated as following:



Each table is a hash table, and each item in the table is a head node of a double link list. The operation applied to an item in table has to find the item’s hash value. Then the hash value used to visit the head pointer of link list. By going through the link list, the item is found and eventually operation could be done to it.

## LPM Setup

System requirement:

Ubuntu 18.04.1 LTS

Libraries:

pthread

Run instruction;

1. Uncompress LPM.zip
2. Makes sure that all processes in classification/process\_Defined are marked or cleared.
3. Open terminal in root folder of project: Linux\_Process\_Management
4. Execute commend with administration priority:

“sudo make”

1. Execute commend: “sudo ./bin/sys2018”
2. Input command to control the program or let program run in background.
3. Quit the program and check the log and classification list.
4. Update files in classification list.

The kernel module is automatically loaded by user program at the beginning of execution and unloaded at the end of execution.

## Commands:

Quit: exist the program properly.

Print x: print list designated by x.

Example:

Print Unknown: print table unknown to both terminal and file.

Kill x:

Kill a process designated by x:

Ex:

Kill sys2018: kill a process named sys2018

## Future development:

1. Implement the elimination functionality by using app-get.
2. Use semaphore to monitor the new activities instead of scanning all processes again and again.
3. Based on [2], prevent processes in Kill list from executing.
4. Based on [1], keep database online and updated to support elimination functionality. For example, program is able to search the way to uninstall a program completely with user’s permission, if the program generates processes in Eliminate list.
5. In prove that way to manage unknow process, instead of clearing process\_Defined each time before execution.
6. Get further control on process, including control the system resources assigned to process, priorities of processes, accessibility to certain resources.