Processes Checker

Easy and complex TUI tool to monitor and diagnose Linux processes

Index

1 Purpose and functions of the script

1.1 Script purposes

1. Instant diagnose of all defined processes that should work on current machine
2. Instant info about status of working process based on last update on log files
3. Print processes that need long output from cmd to be recognized
4. Print info about processes each several seconds (works like Top)
5. Search through logs of chosen processes for errors by defined phrases.
6. Print logfiles paths to all existing paths of logfiles for processes
7. Panic kill – will print kill command to immediately kill all application processes
8. Create shell script for start / stop all processes (whole application)
9. Process monitor - sent mail with alert when any process broken

1.2 Script functions

1. Check immediately status of all defined processes for current server. If any of necessary process does not work, then will print command for starting particular crashed process
2. Check during checking processes if logs of KIS and TradeKast clients are updating, to diagnose if process are hung
3. Print processes that need long output from cmd to be recognized – created mostly for Solaris, which is cutting output from ps command for non privileged users. Then script may be lunched as privileged user that see not cut output from cmd, and can compare output to process pattern and recognize if process is working or not.
4. Use script to work as Top, checking processes and refreshing every x seconds
5. Search through KIS and TradeKast clients logs for errors described within config files by error phrases
6. Print logfiles paths and last updates on files (current logfile and last day logfile) for all processes which have defined logpath. Logfiles will be printed with full path, only for currently existing files.
7. Panic kill – will gather PIDs for all processes application and print kill command with -9 SIG (command may be changed in config.py file) and with all gathered PIDs.
8. Create shell script for starting or stopping whole application processes. For stopping script processes will be added in reverse order. Start and stop commands are took from start / stop command assigned to each process.
9. Use script in background to check and sent mail when process broken. Function will not print any output in shell, will sent alert mail if any broken process was found.

2 Manual

2.1 Example output from script lunch

```
1. Checking defined processes – processes working


2. Checking defined processes – processes not working


3. Checking defined processes – several processes not working

4. Print paths to logfiles
```

2.2 Example commands for running the script

```
python /path/to/script/check_processes.py –specyficators
```

Examples:

```
#1
python /path/to/script/check_processes.py
#2
python /path/to/script/check_processes.py 0
#3
python /path/to/script/check_processes.py b
#4
python /path/to/script/check_processes.py 0a
#5
python /path/to/script/check_processes.py m
```

2.3 Adding shelve library environment variable

Before script can be use, it need one environment variable to be exported within the prompt. To define it permanently please edit the file in your home directory:

```
vi $HOME/.bashrc
```

Add new line:

```
export LD_LIBRARY_PATH="/path/to/BerkleyDB/lib"
```

And save file. Now You can reload bash (prompt `exit` then `bash`) and variable will be set each time You login to prompt / bash.

2.4 Adding / changing / removing phrase for function ‘check clients for error phrases

In moment of writing this documentation, there are 4 phrases on list:

```
logs_error_patterns_list = [
                              'Cannot connect to kis server',
                              'Timeout callback error',
                              'ERROR',
                              'Lost connection'
                              ]
```

Add, change or remove phrase, by simply editing this list. If You want to add new element to list, put `,` comma sing after last element, put enter and write another phrase in apostrophe `'` sign.

2.5 Adding new server and his own processes to check

Copy `hostname_empty_pattern.csv` file in `/files` folder and rename file to hostname of machine when you want to use script.
Fulfil script with processes, their properties, and groups of processes you want group them by.
Run the script on hostname you want to add with option **`c`** to add hostname and groups and processes to shelve database file. You can check if processes were added using option **`p`**

2.6 Script arguments

Script have 12 arguments that can be used when lunching:

- 0 - no color
- 1 - use color
- a - check long output processes (only for privileged user)
- b - print paths to logfiles
- c - check basic logs for errors phrases of choosen processes
- d - works like top, refresh each 10 seconds
- e – panic close
- f - create start script
- g - create stop script
- h - help
- i - create shelve file(need for any other functionality)
- j - print inside of shelve file
- k - work in background as monitor and mail if broken process found
- m - menu
- q - quit (only for menu)Command for running the script with examples

3 System Description

Script may be executed by any of the users with privileges to execute by simply typing command that starts the script. Script after start loads the data, including patterns, log paths, name of processes, start command for process, end command for process, how many instances of process should running, and other data to be compared with currently working processes on machine. Script consist of 3 python based files: `config.py`, `text.py` and `check_processes.py`. Each of files is dedicated to special functionality. On user side only one of them is important - `check_processes.py` – which is responsible for loading data information about processes that should working on current machine and successively check each process printing information about them till the end of list.

3.1 Script logic

1. Script may be lunched on any server at anytime.
2. Script main functionalities are consist within file `check_processes.py`
3. `check_processes.py` is script that utilize data file to compare them with currently working processes on machine on which is lunched and present output in human friendly form.

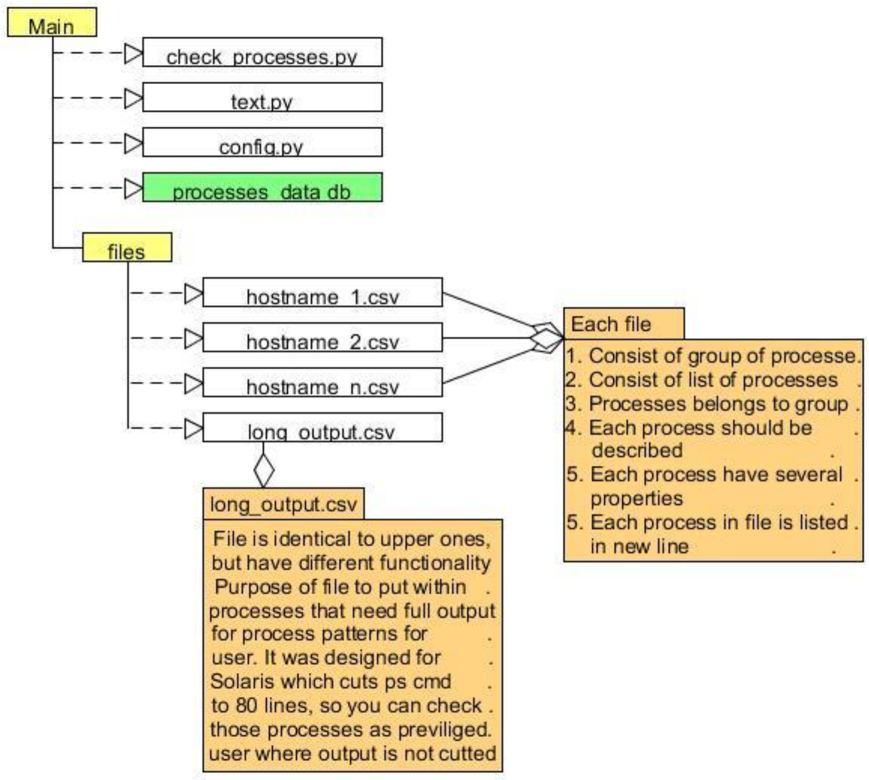
4 System Architecture Model

4.1 System dependency

Script is fully written in Python 2.x, based on OOP. Script requires python v. 2.x to be installed, python shelve library and menu library. Menu library is provided with Check Processes. Additionally scripts require BerkleyDB library, which is utilized by shelve library. BerkleyDB library have to be linked to system by operating system environment variable `LD_LIBRARY_PATH.` Utilizing script as monitor tool to working in background and sanding mail alerts demands to have installed and configured `mailx` Linux application or external mail account having.

4.2 Folder logic

Script consist of several folder and files


Main
check_processes.py
text.py
config.py
processes_data db
files
hostname_1.csv
hostname_2.csv
hostname_n.csv
long_output.csv
Each file
1. Consist of group of processe.
2. Consist of list of processes
3. Processes belongs to group
4. Each process should be described
5. Each process have several properties
5. Each process in file is listed in new line
long_output.csv
File is identical to upper ones, but have different functionality Purpose of file to put within processes that need full output for process patterns for user. It was designed for Solaris which cuts ps cmd to 80 lines, so you can check those processes as previliged. user where output is not cutted

Explanation of above folders structure:

`Main` folder

`check_processes.py` – it is main script that is creating, fulfilling and loading data from / to database `proceses_data`, gathering output from `ps` command, compare data and print detailed info about the processes. File contains all functionality of the script.

`config.py` – consist of configurations within the script. Most of them should not be changed.

`text.py` – file consist all text for communication with user script, so any text values can be easily modified or translate into any language.

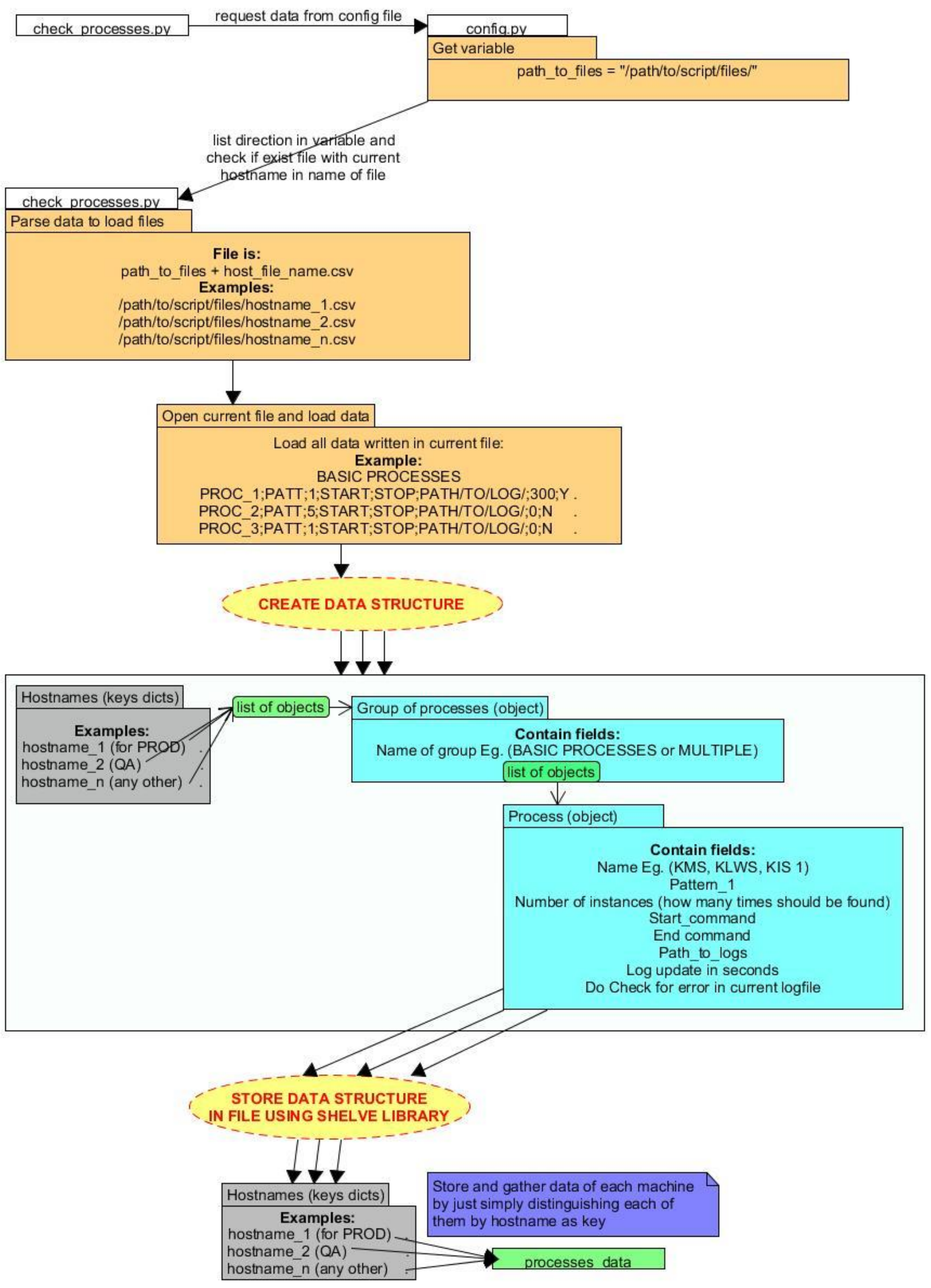
`processes_data` – database file created by `check_processes.py`, consist of object and variables provided to structurized data of processes, for fast load data to compare for `check_processes.py` script.

files folder – here are defined all csv files, each for one server hostname, and within each of csv files are listed detailed info about each process. Processes may be grouped.

`hostname_1.csv` – consist all processes that should be checked on hostname_1
`hostname_2.csv` – consist all processes that should be checked on hostname_2
`hostname_n.csv` – consist all processes that should be checked on hostname_n
[…]

4.3 Files logic

Below is described logic how files containing information necessary to check processes are copied into one shelve library data file, from which Check Processes checker script is loading data.



check_processes.py
request data from config file
config.py
Get variable
path_to_files = "/path/to/script/files/"
list direction in variable and check if exist file with current hostname in name of file
check_processes.py
Parse data to load files
File is:
path_to_files + host_file_name.csv
Examples:
/path/to/script/files/hostname_1.csv
/path/to/script/files/hostname_2.csv
/path/to/script/files/hostname_n.csv
Open current file and load data
Load all data written in current file:
Example:
BASIC PROCESSES
PROC_1;PATT;1;START;STOP;PATH/TO/LOG/;300;Y
PROC_2;PATT;5;START;STOP;PATH/TO/LOG/;0;N
PROC_3;PATT;1;START;STOP;PATH/TO/LOG/;0;N
CREATE DATA STRUCTURE
Hostnames (keys dicts)
Examples:
hostname_1 (for PROD)
hostname_2 (QA)
hostname_n (any other)
list of objects
Group of processes (object)
Contain fields:
Name of group Eg. (BASIC PROCESSES or MULTIPLE)
list of objects
Process (object)
Contain fields:
Name Eg. (KMS, KLWS, KIS 1)
Pattern_1
Number of instances (how many times should be found)
Start_command
End command
Path_to_logs
Log update in seconds
Do Check for error in current logfile
STORE DATA STRUCTURE IN FILE USING SHELVE LIBRARY
Hostnames (keys dicts)
Examples:
hostname_1 (for PROD)
hostname_2 (QA)
hostname_n (any other)
Store and gather data of each machine by just simply distinguishing each of them by hostname as key
processes_data

Explanation of above logic structure:

4.3.1 `config.py` in `Main folder`

Contains several useful fields (variables) that are editable:

4.3.1.1 ### COLORS SECTION

Each variable listed here contains code number for color in bash. Variable were not listed here as they are self-explained.

4.3.1.2 ### BASIC CONFIG VARIABLES SECTION

`path_menu = path.join( path_to_script, 'menu/menu.py')#` Variable stores full path to menu script. Do not change value

`path_menu_text = path.join( path_to_script, 'menu/menu_text.py') #` Variable stores full path to menu text file. Do not change value

`folder_to_processes_files = '/files/' #` name of folder with hostnames processes files. Do not change value

`path_to_script = path.dirname(path.realpath(__file__)) #` value is gathered from function. Do not change value.

`shelve_database = 'processes_data' #` name of shelve database file. Do not change value.

`ps_command = 'ps -fu [user name]#` variable stores command executed in bash which return list of processes currently working on machine and will write the list into python list

`column_with_pid_for_ps_command = 1 #` variable stores number of column which is PID of output from upper ps command

`none = None #` variable stores None value. Do not change value

4.3.1.3 ### HEADER AND FORMAT OUTPUT SECTION

```
header = '|      PROCESS NAME      |INSTANCES| PID 1 | PID 2 |  STATUS  |LOG UPDATE|'
```

# upper variable stores

`one_line_format = '  %20.20s  | %7s | %5s | %5s | %8.8s | %8s '` # variable stores format of printed one line (one line info about each process). Variable should not be changed by someone who do not know Python String formatting.

4.3.1.4 ### STATUSES SECTION

Section contains name of possible statuses of JOBs and related to them logfiles. Statuses are kept in config file, so you can easily edit name of each status due to your preferences. Status value should be possibly short and intuitive. Variables were not putted to documentation, because names of variables as well as written to them statues are self-explained. All statuses were listed in point **3.6** .

4.3.1.5 ### LONG OUTPUT CHECK SECTION

`full_output_ps_command = '/usr/ucb/ps axww'#` variable that stores special command that is prompt when script is started to check status of long output processes, where cmd returned for Solaris as non-privileged user is limited to 80 lines.

`column_with_pid_for_full_output_ps_command = 0 #` variable stores number of column which is PID of output from upper ps command

`long_output_path_to_file = 'long_output.csv' #` variable stores path to files which consist all process that need long output from CMD

`privileged_user = 'user' #` Stores information what should be user when running for checking processes that need long output from process cmd.

4.3.1.6 ### CHECKING FOR ERROR IN LOGFILES SECTION

`how_many_path_to_logs_show_in_show_log_paths_function = 2 #` variable stores digit which represent the logical number of how many lofgiles for each process

```
logs_error_patterns_list = [
                              'Cannot connect to kis server',
                              'Timeout callback error',
                              'ERROR',
                              'Lost connection'
                              ]
```

`#` Upper list stores list of error phrases for which script will be searching within logs of processes which was declared in file of processes for current hostname.

4.3.1.7 ### PANIC KILL SECTION

`command_kill = 'kill -9 ' #` variable stores command to kill immediately all defined processes of by PIDs, using SIG -9

4.3.1.8 ### TOP SECTION

`refresh_interwal_in_seconds = 10 #` variable store as digit numer, which stands in seconds for information: how many seconds is break between next check and refresh info about working processes

`command_clear = 'clear' #` variable stores clear screen command for bash

4.3.1.9 ### BACKGROUND MONITORING

`check_processes_each_how_many_seconds = 5 #` variable stores digit in seconds that informs how often all processes of application will be checked. Digit stands for interval before next check.

`when_found_broken_processes_next_check_in_seconds = 60 #` variable stores digit in seconds, that stands for time interval in seconds between checks, but only in situation when during last check were found broken processes. Reason for existing such variable is not to be spammed with hundreds of mails that process do not works.

`mail_header = '| INSTANCES  | LAST UPDATE ON LOGS | PROCESS NAME \n' #` This is header of mail body.

`mail_one_line_format = '|  %3i / %3i | %8s / %8s |  %s \n' #` This is format of one line in body. Each line is one process not working / hung

`receivers = [ 'receiver_1@mail.com', 'receiver_2@mail.com', 'receiver_3@mail.com' ] #` list with alert mail receivers

`subject = 'WARNING! On Hostname %s Some of processes are not working' #` Subject of alert mail. Should contain %s where is put hostname

`use_python_smtplib = False #` True if you want to use smtplib library and sent mail after login directly to server of mail account

`sender = 'email_address_of_sender@gmail.com' #` variable need for smtplib library, sender e-mail address

`sender_password = '' #` variable need for smtplib library, sender password to e-mail

`smtp_server = 'smtp.gmail.com' #` variable need for smtplib library, sender smtp server address

`smtp_port = 587 #` variable need for smtplib library, sender smtp port number

`use_mailx = False` # True if you want to use mailx application on system (server) host
`sent_body_file_name = 'mail_body'` # any value can be put here, it is name of file, that is storing body of mail and will be used for mailx.

4.3.1.10 ### BACKGROUND MONITORING

`start_script_name = 'start_processes.ksh'` # Name of start script script that will be created when function for creating script was lunched.
`stop_script_name = 'stop_processes.ksh'` # Name of stop script script that will be created when function for creating script was lunched.
`header_script_declaration = '#!/bin/bash'` # Here is the file declaration (file header) that will be put created start / stop script
`access_rights_to_scipt = 0777` # After creating file, here are access rights that will be set to start / stop script
`owner_of_the_script = ''` # if you put something between the apostrophe signs, then user owner of created start / stop script will be set to value assigned to this variable.
`break_between_each_start_stop_command_in_seconds = 0` # variable stores digit, that stands for information if there will be break in seconds between each start / stop command. If you put `2`, then between each command in script there will be `2` seconds time interval. If you put to `0`, then it will not be any break between each command, just will be done one after another.
`command_echo = 'echo'` # shell command echo, better do not change
`command_sleep = 'sleep'` # shell command sleep, better do not change

4.3.2 **`hostname_n.csv`** in **`files/ folder`**

Any line in file stands for one process. Each process have 5 or 6 fields, separated by semicolon **;** . Structure is presented below:

```
GROUP OF PROCESSES
NAME;Pattern_1;Pattern_2;Number_of_instances;Start_command;END_comma
nd;Log_path;Log_update_in_seconds
```

Example file exist in `/files` folder.

Explanation of file fields:

- `GROUP OF PROCESSES` – Printed name of group of processes
- `NAME OF PROCESS` – Printed name of process
- `PATTERN` – Pattern 1 through which process will be recognized
- `NUMBER OF INSTANCES` – Number (digit) of processes that should be found for process. If 2 then process recognized ad double (with x86sol), if 1 then single process, if more than 2 then multiple process. Multiple process is taking only PATTERN 1 and count how many times it found pattern in working processes, then compare content number with the one in NUMBER OF INSTANCES
- `START COMMAND` – Command will be print if process was not found and is recognized as not working
- `END COMMAND` – Command for stop the process
- `LOG FILE PATH` – Path to logfile of process, should contain file name or part of file name through which file can be found in path

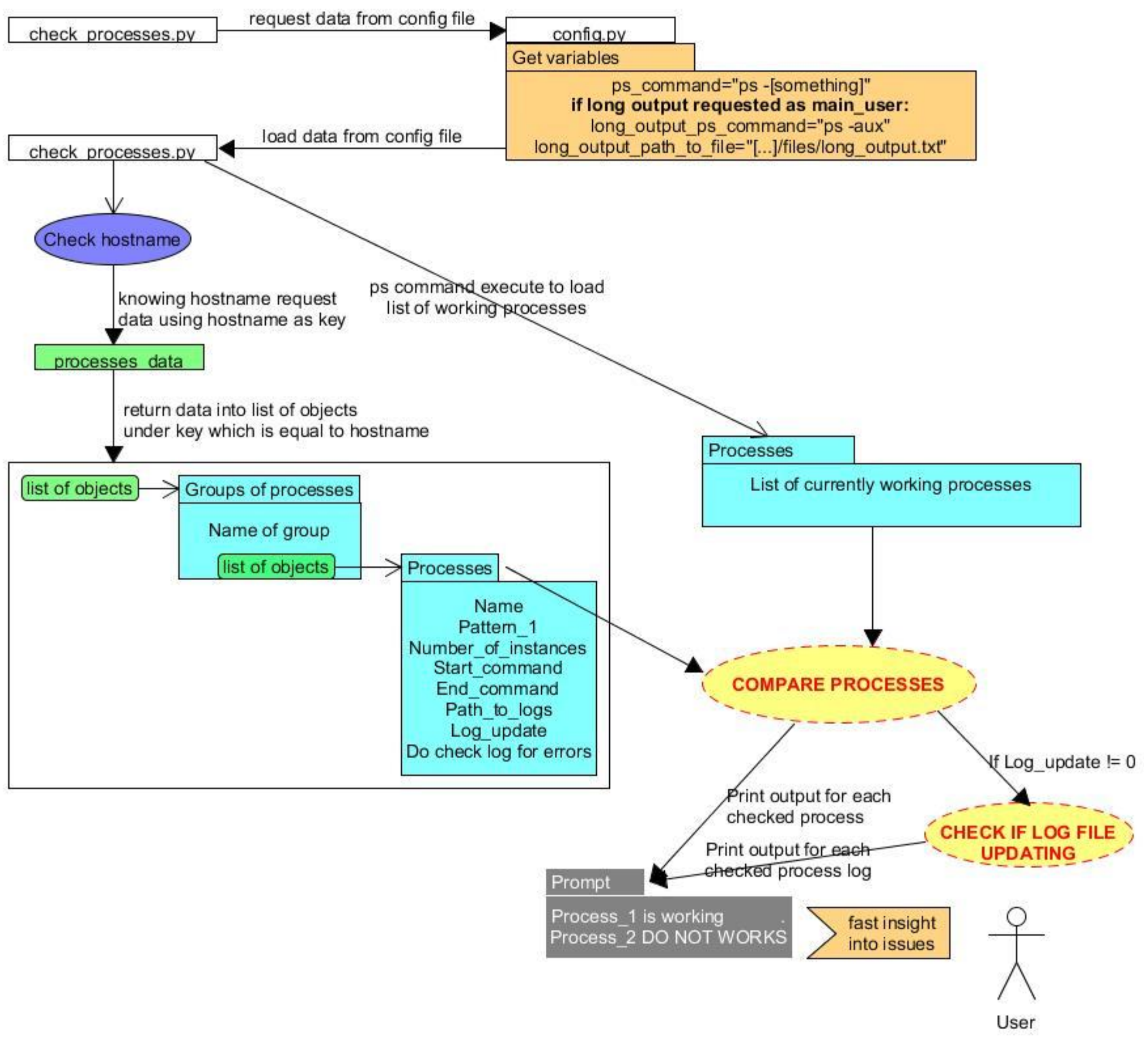
- `LOG UPDATE (IN SECONDS)` – Default value is `0`, which stands for “Do not check”. If you put digital value, for example `300`, it will work as condition. Condition which stand for: “If log file of process was not updating more than value (`300` seconds), then will print additional info for current process that log file is not updating more than value, so process may be hung”. Putting here different value than `0` only if you have `LOG FILE PATH` field fulfilled for that process.
  Option have 3 states: red: logs not updating more than `x`, yellow: logs not updating less than x but more than half of `x` (`x/2`), green: logs are updating less than half of `x` .

- `Do Check for error in current logfile` – May store only 2 char values:

  `Y` – stands for True, if value is set, then logfile will be checked for errors
  `N` – stands for False, if value is set, then logfile will not be checked for errors

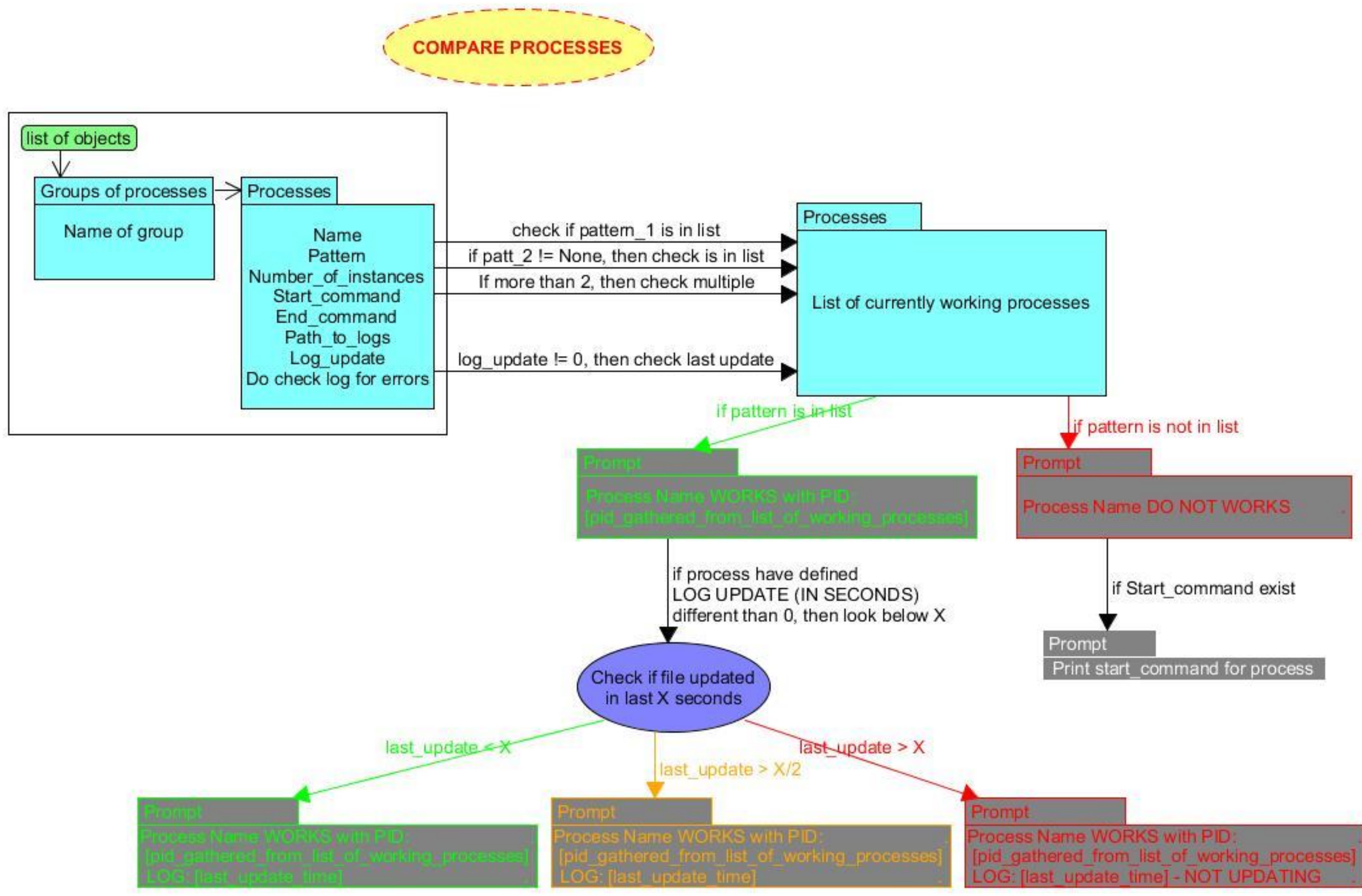
4.4 Algorithm logic

On below picture is presented main functionality algorithm logic:


check_processes.py
request data from config file
config.py
Get variables
ps_command="ps -[something]"
if long output requested as main_user:
long_output_ps_command="ps -aux"
long_output_path_to_file="[...]/files/long_output.txt"
load data from config file
check_processes.py
Check hostname
knowing hostname request data using hostname as key
ps command execute to load list of working processes
processes_data
return data into list of objects under key which is equal to hostname
list of objects
Groups of processes
Name of group
list of objects
Processes
Name
Pattern_1
Number_of_instances
Start_command
End_command
Path_to_logs
Log_update
Do check log for errors
Processes
List of currently working processes
COMPARE PROCESSES
If Log_update != 0
Print output for each checked process
CHECK IF LOG FILE UPDATING
Print output for each checked process log
Prompt
Process_1 is working
Process_2 DO NOT WORKS
fast insight into issues
User

4.5 Insight into comparing processes

Each of loaded list consists of some information. List of Reports to check consist information about File name, average size, abbreviation from file size in percent. List of reports generated consists of information: File name, file size, date of creation. That information are used to check if file is correct, or may be suspicious or wrong.



COMPARE PROCESSES
list of objects
Groups of processes
Name of group
Processes
Name
Pattern
Number_of_instances
Start_command
End_command
Path_to_logs
Log_update
Do check log for errors
check if pattern_1 is in list
if patt_2 != None, then check is in list
If more than 2, then check multiple
log_update != 0, then check last update
Processes
List of currently working processes
if pattern is in list
if pattern is not in list
Prompt
Process Name WORKS with PID:
[pid_gathered_from_list_of_working_processes]
Prompt
Process Name DO NOT WORKS
if process have defined
LOG UPDATE (IN SECONDS)
different than 0, then look below X
if Start_command exist
Prompt
Print start_command for process
Check if file updated
in last X seconds
last_update < X
last_update > X/2
last_update > X
Prompt
Process Name WORKS with PID:
[pid_gathered_from_list_of_working_processes]
LOG: [last_update_time]
Prompt
Process Name WORKS with PID:
[pid_gathered_from_list_of_working_processes]
LOG: [last_update_time]
Prompt
Process Name WORKS with PID:
[pid_gathered_from_list_of_working_processes]
LOG: [last_update_time] - NOT UPDATING

4.6 Possible processes statuses (existing in config.py file)

4.6.1 Log statuses

```
status_log_file_not_exist = ' - LOGS NOT EXISTS'
status_log_file_short_not_updating = ' - NOT UPDATING SHORT'
status_log_file_not_updating = ' - NOT UPDATING LONG'
status_log_file_ok = 'LOGS UPDATING'
```

4.6.2 Processes statuses

```
# SINGLE #
status_process_ok = 'OK'
# MULTIPLE #
status_processes_all_ok = 'All OK'
status_process_all_few_bad = 'FEW BAD'
status_process_all_bad = 'ALL BAD'
status_processes_too_many = 'TOO MANY'
```

4.7 Additional script functionality

Except of processes checking for PROD server, script have additional 3 functions:

1. ***Print processes that need long output from cmd to be recognized*** – created mostly for Solaris, which is cutting output from ps command for non privileged users. Then script may be lunched as privileged user that see not cut output from cmd, and can compare output to process pattern and recognize if process is working or not.
2. ***check tradekast / kis clients logs in search for phrase of error*** – checks throught logs of all kis / tradekast clients. Search in logs for present day, if found matched phrase or word, then print in prompt. List with phrases is easy editable in config.py file
3. ***print logfiles for all processes that have defined path to logfiles*** – function will print current paths to lofiles for current day and previous day (may be change for more or less days in config file). Will print only full path and simple information about last log update
4. ***panic close*** **–** function will gather all working processes and print kill command for processes.
5. ***print output in grey (no color)*** – function useful for anyone using different console than bash.If your prompt does not show colors, but signs of colors, then it is useful option
6. **use script to work as Top** – function is checking processes and refreshing every x seconds, where x is declared as variable `refresh_interwal_in_seconds` in `config.py` file.
7. **Use script in background to check and sent mail when process broken** – script start to works in background and check each x seconds if all processes are working, and logs of defined processes are updating in given thresholds. If there will be found broken process, or with logs not updating, then will be sent mail to receivers, and special break (how long – defined `when_found_broken_processes_next_check_in_seconds` variable in `config.py` file) taken, to avoid spam alert message each several seconds, when bad process found.

5 Contact

If you have any questions, concerns or maybe you found bug within the code, then please do not hesitate to contact with the author of Check Processes.

E-mail: tomasz.wojcik.88@gmail.com