Memlog Guide

Author: Kim Miikki
Date: 26.1.2020

Introduction

Memlog monitors Linux memory usage as function of time. The system consist of a memory logging script, a log analyzer and some memory cleanup methods. Memlog is especially designed for Raspberry Pi OS systems.

System requirements

Operating system: Raspberry Pi OS or Debian based Linux

Python 3 with Matplotlib and NumPy

Optional Process and Memory Cleanup Methods

1. Disable screensaver

2. Disable Bluetooth

3. Add a 'clean cache' crontab job

```
$ sudo crontab -e
```

Add following line:

```
* * * * * /bin/sh -c "/bin/sync; echo 3 >
/proc/sys/vm/drop_caches"
```

4. Disable swappiness

```
$ cat /proc/sys/vm/swappiness
60
```

Change value to 0:

```
$ sudo sysctl vm.swappiness=0
```

5. Disable swaps:

```
$ sudo swapoff -a
```

Memory Monitor

Memory monitor logs memory usage which can be later analyzed with memlog.py. Here is a listing of the script:

memlog.sh

```
#!/bin/bash

echo "      date     time $(free -m | grep total | sed -E 's/^
(.*)/\1/g')" |& tee meminfo.log
while true; do
    echo "$(date '+%Y-%m-%d %H:%M:%S') $(free -m | grep Mem: | sed
's/Mem://g')" |& tee -a meminfo.log
    sleep 60
done
```

Update interval is given in seconds after the sleep command. Change script to executable with: `chmod +x memlog.sh`.

Memory usage will be appended to a logfile named meminfo.log.

Memory Log File Analyzer

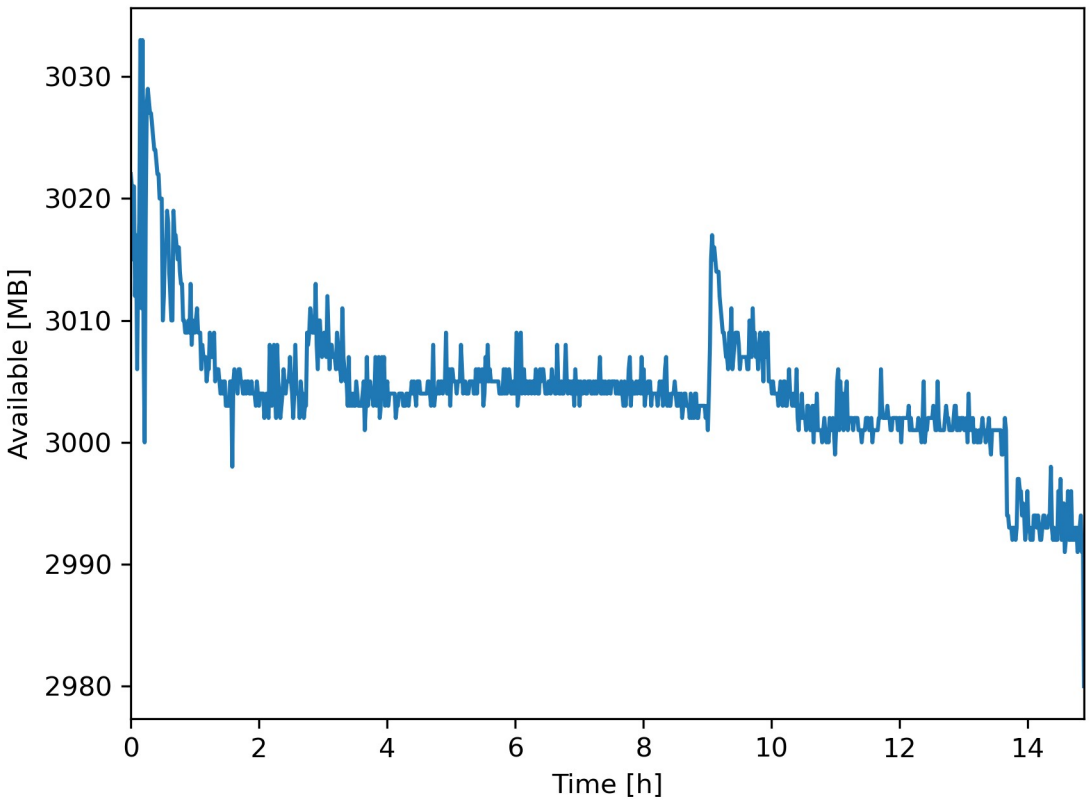
Existing meminfo.log file is analyzed with memanalyze.py python script. The log is read and memory usage of each data columns are saved in graphs as a function of log time. The script should run in same directory as meminfo.log:

```
$ ./memanalyze.py
Memory Log File Analyzer

Analyzing meminfo.log

Saving files:
 17 % done
 33 % done
 50 % done
 67 % done
 83 % done
100 % done
$$ ls -1 *.png
meminfo-available.png
meminfo-buff-cache.png
meminfo-free.png
meminfo-shared.png
meminfo-total.png
meminfo-used.png
```

X-axis auto unit is based on the legth of the monitoring time. Possible units are: s, min, h and d. The unit will change to next when 5 times next unit is reached, i.e. 5 x 60 s = 300 s → 5 min. Available memory is probably the most useful figure as shown in following example:



Monitoring Files in a Directory

The files count can be monitored in a terminal window with following Bash one-liner:

```
$ watch "ls -1 | wc -l"
```

Files will be counted each 2 seconds.

Simple Stress Test

Following one-lner is a simple stress test. It cwill continue until the loop is exited with Ctrl + C.

```
$ while true; do COMMAND; done
```

Replace *COMMAND* with stress or monitoring command, i.e. while true; do **df . | tail -1 && sleep 1**; done.

Use Case: Memory Usage of tlcronmin.py

Very long tame-lapse shootings can cause memory issues. Hence, a memory usge monitoring was conducted for tlcronmin.py. A stress setup was set up for this purpose. A Raspberry Pi camera system is available from github:

https://github.com/kmiikki/rpi-camera

Time-lapse preparation

```
$ tlcronmin.py
Crontab based time-lapse camera ver. 2.0

Current directory:
/media/pi/ssd/memtest

Select image quality (1...100, Default=90: <Enter>):
Default value selected: 90
Select ISO (100, 200, 320, 400, 500, 640, 800; Default=100):
Default value selected: 100

Auto exposure on (Y/N, Default y: <Enter>):
Default selected: EXP auto enabled

AWB mode on (Y/N, Default y: <Enter>):
Default selected: AWB enabled

Select analog gain (1.0...12.0, Default=1.0: <Enter>):
Default value selected: 1.0

Select digits (1...8, Default=4: <Enter>): 5
Select interval (1...525960 min, Default=1: <Enter>): 2
Time-lapse maximum duration: 138.9 d

Preview image (Y/N, Default n: <Enter>): y
Preview enabled
Start time-lapse now (Y/N, Default y: <Enter>):
Default selected: Time-lapse enabled
Old crontab:
User pi crontab cleared

Time-lapse job started.
```

Monitor 1

```
$ ./memlog.sh
```

Monitor 2

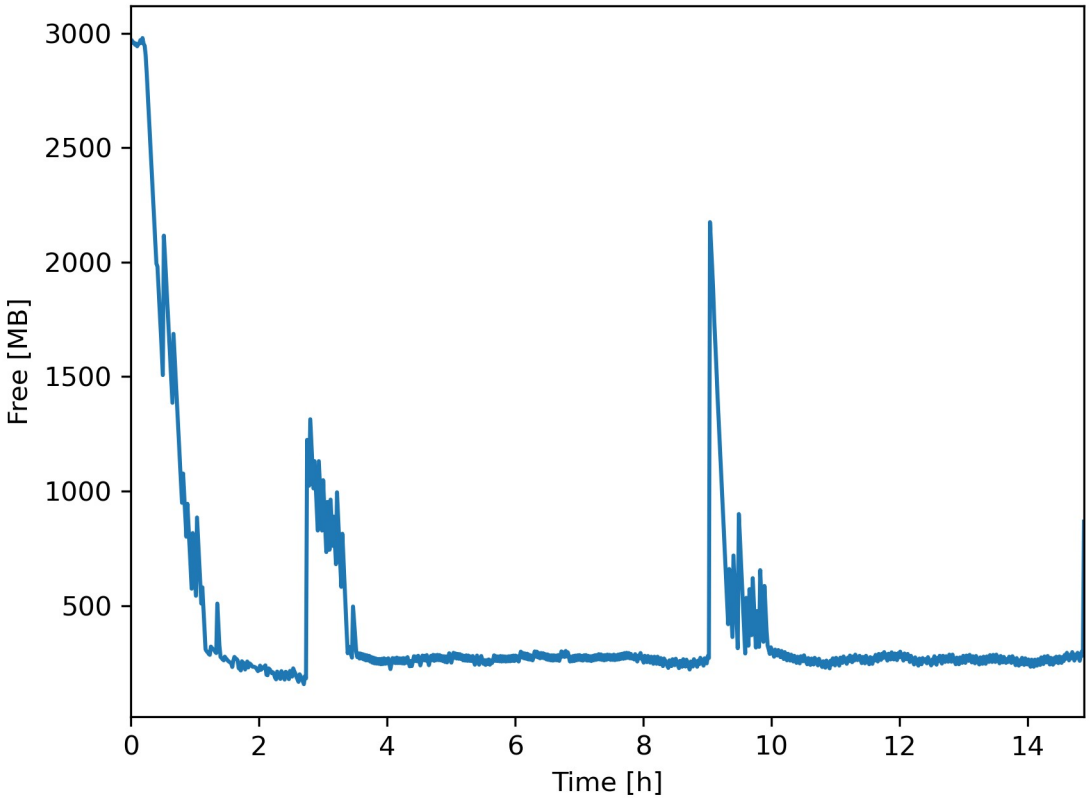
```
$ watch "ls -1 | wc -l"
Every 2.0s: ls -1 | wc -l            ce: Sun Jan 24 21:21:24 2021
```

Stress test: capture images

```
$ while true; do tlcronmin.py -c; done
```

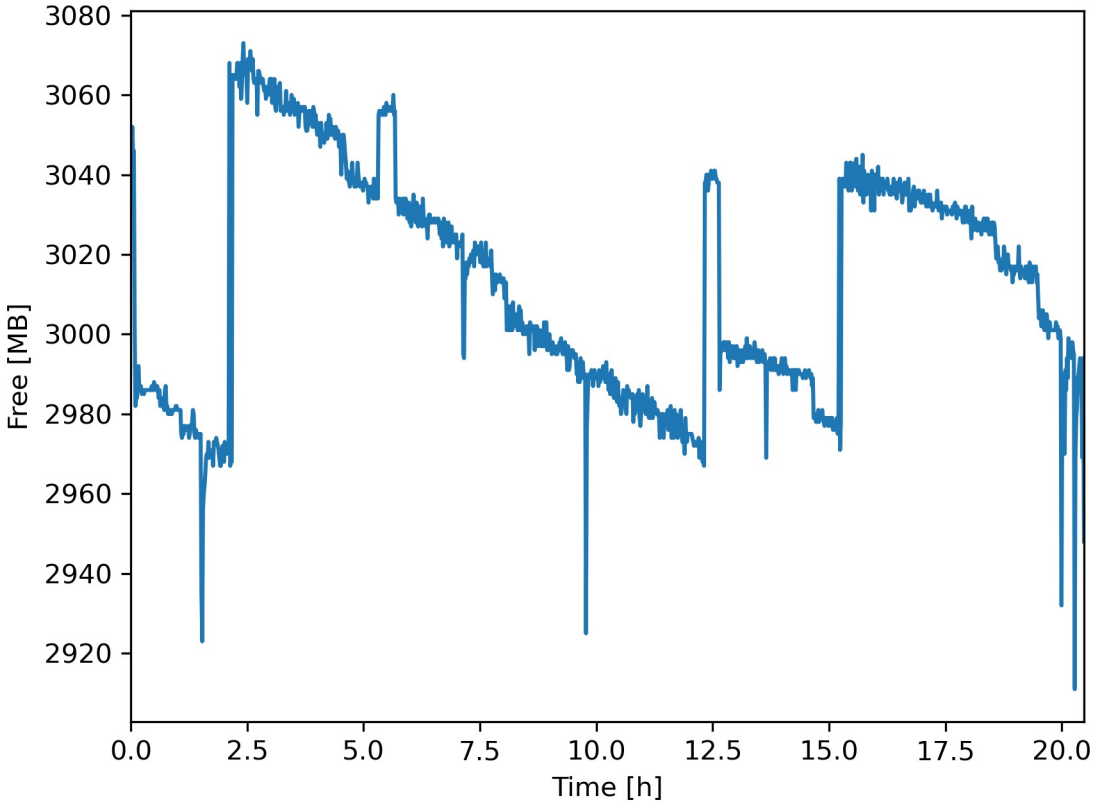
Results

The initial test was done with default memory settings. It resulted in high usage of buff/cache memory and low level of free memory:



However, the available memory did not decrease significantly. The result can be seen in first figure in this document.

A second experiment was performed to keep free memory as high as possible. This was achieved by applying all optional process and memory cleanup methods. The difference is remarkable compared with the initial test.


3080
3060
3040
3020
3000
2980
2960
2940
2920
Free [MB]
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Time [h]