University College of Northern Denmark

Computer science

Technology project: Raspberry Pi OS

Kristian Hjelm

Branislav Macko

Ondřej Špok

Pavol Halás

Adam Petříček

15. December 2020

Introduction

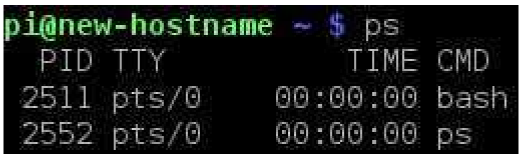
We have chosen the Raspberry Pi OS, because we had a hands-on experience with the new Raspberry Pi 4 model B 2GB while working on this report. We found that it is quite interesting little machine, that can fit into a palm. It's variety of uses are almost endless.

System processes

A running program is represented by a process. When you execute a command in the terminal, it launches a program and creates a process for it. A process id (PID) is assigned to each process and is linked to a specific user or group account.

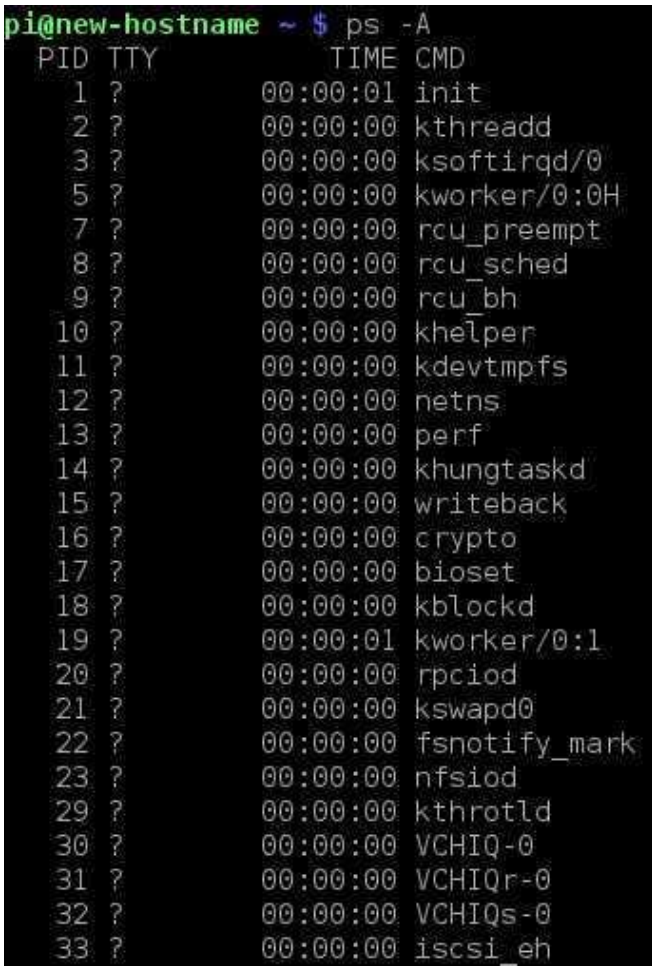
The PS (short for process status) command displays a list of currently running processes on your Raspbian system. It accepts a variety of solutions that can help you troubleshoot your device.

PS shows only processes initiated from the current terminal when run without any options:



```
pi@new-hostname ~ $ ps
  PID TTY          TIME CMD
 2511 pts/0    00:00:00 bash
 2552 pts/0    00:00:00 ps
```

The output shown above does not provide a lot of valuable detail. In order to learn more about our method, we must run the PS command with different choices. For instance, we can use the ps -A command to get information about all processes running on our system:

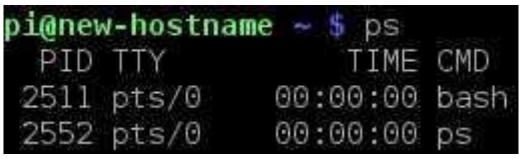


```
pi@new-hostname ~ $ ps -A
  PID TTY          TIME CMD
    1 ?        00:00:01 init
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    7 ?        00:00:00 rcu_preempt
    8 ?        00:00:00 rcu_sched
    9 ?        00:00:00 rcu_bh
   10 ?        00:00:00 khelper
   11 ?        00:00:00 kdevtmpfs
   12 ?        00:00:00 netns
   13 ?        00:00:00 perf
   14 ?        00:00:00 khungtaskd
   15 ?        00:00:00 writeback
   16 ?        00:00:00 crypto
   17 ?        00:00:00 bioset
   18 ?        00:00:00 kblockd
   19 ?        00:00:01 kworker/0:1
   20 ?        00:00:00 rpciod
   21 ?        00:00:00 kswapd0
   22 ?        00:00:00 fsnotify_mark
   23 ?        00:00:00 nfsiod
   29 ?        00:00:00 kthrotld
   30 ?        00:00:00 VCHIQ-0
   31 ?        00:00:00 VCHIQr-0
   32 ?        00:00:00 VCHIQs-0
   33 ?        00:00:00 iscsi_eh
```

A **process** is an instance of a running program. When you run a command in the terminal, a program is run and a process is created for it. Each process has a process id (**PID**) and it's associated with a particular user and group account.

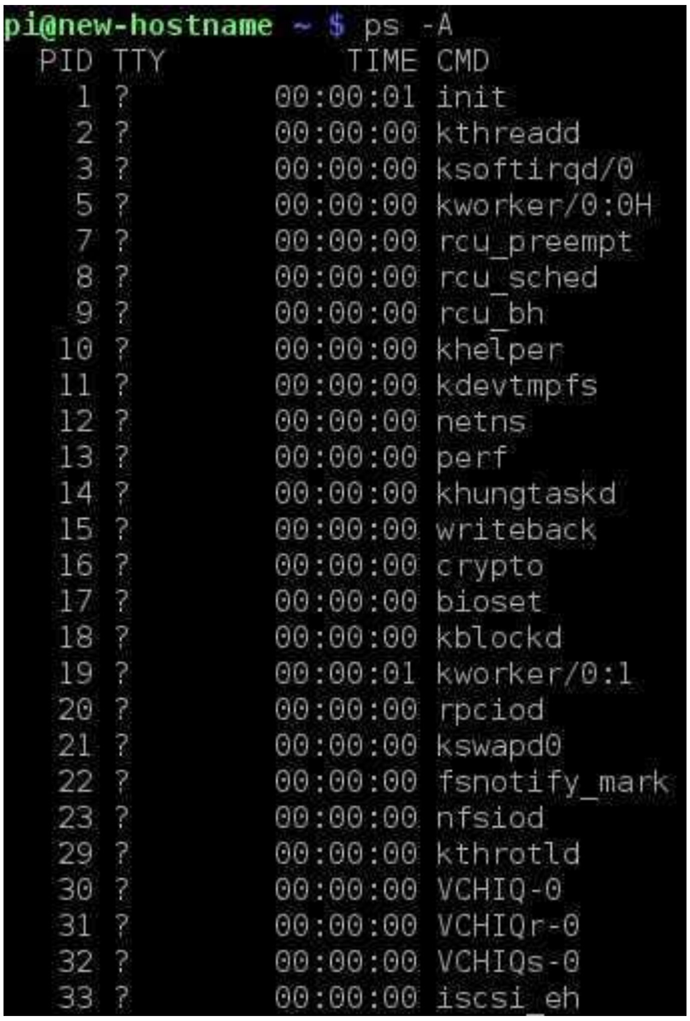
The **ps** (short for **process status**) command is used to list processes currently running on your Raspbian system. It can accept a lot of options that can come in handy when troubleshooting your system.

Used without any options, **ps** displays only processes started from the current terminal:



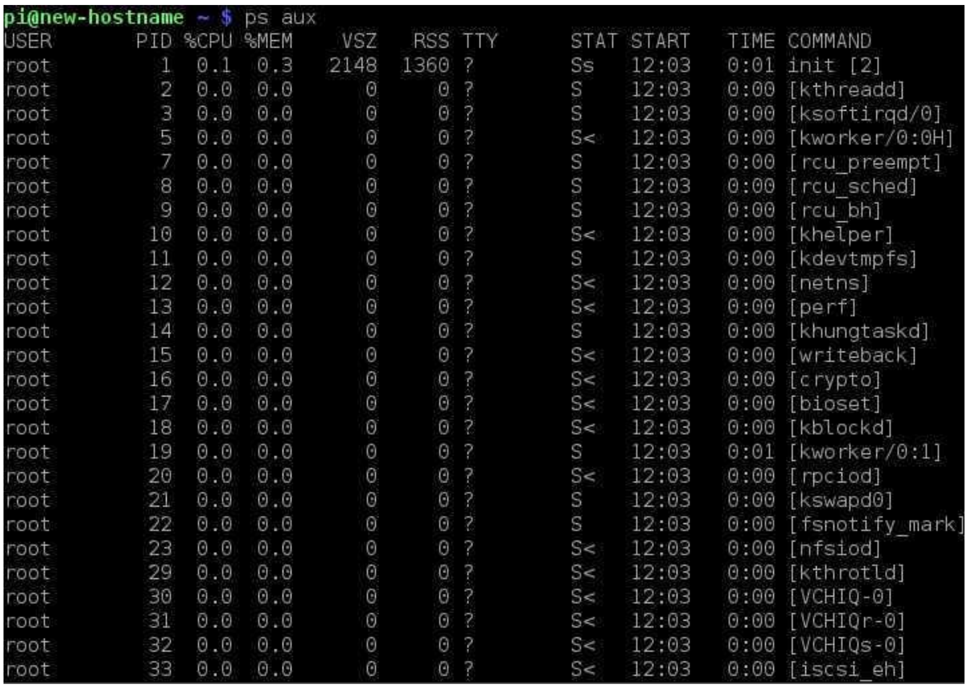
```
pi@new-hostname ~ $ ps
  PID TTY          TIME CMD
 2511 pts/0    00:00:00 bash
 2552 pts/0    00:00:00 ps
```

The output above doesn't provide many useful information. We need to run the **ps** command with various options in order to get more info about our system. For example, to get information about all processes running on our system, we can use the **ps -A** command:



```
pi@new-hostname ~ $ ps -A
  PID TTY          TIME CMD
    1 ?        00:00:01 init
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    7 ?        00:00:00 rcu_preempt
    8 ?        00:00:00 rcu_sched
    9 ?        00:00:00 rcu_bh
   10 ?        00:00:00 khelper
   11 ?        00:00:00 kdevtmpfs
   12 ?        00:00:00 netns
   13 ?        00:00:00 perf
   14 ?        00:00:00 khungtaskd
   15 ?        00:00:00 writeback
   16 ?        00:00:00 crypto
   17 ?        00:00:00 bioset
   18 ?        00:00:00 kblockd
   19 ?        00:00:01 kworker/0:1
   20 ?        00:00:00 rpciod
   21 ?        00:00:00 kswapd0
   22 ?        00:00:00 fsnotify_mark
   23 ?        00:00:00 nfsiod
   29 ?        00:00:00 kthrotld
   30 ?        00:00:00 VCHIQ-0
   31 ?        00:00:00 VCHIQr-0
   32 ?        00:00:00 VCHIQs-0
   33 ?        00:00:00 iscsi_eh
```

The information in the output above is useful, such as the PIDs of the running processes and the commands used to start them. However, the most widely used **ps** command options are a, u, and x. (p**s aux**). When used with these options, the **ps** command will show all processes operating on our device, as well as details such as the process's owner's username, CPU loads, the process's start time, the command that initiated the process, and so on:



```
pi@new-hostname ~ $ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.1  0.3   2148  1360 ?        Ss   12:03   0:01 init [2]
root         2  0.0  0.0      0     0 ?        S    12:03   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    12:03   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   12:03   0:00 [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    12:03   0:00 [rcu_preempt]
root         8  0.0  0.0      0     0 ?        S    12:03   0:00 [rcu_sched]
root         9  0.0  0.0      0     0 ?        S    12:03   0:00 [rcu_bh]
root        10  0.0  0.0      0     0 ?        S<   12:03   0:00 [khelper]
root        11  0.0  0.0      0     0 ?        S    12:03   0:00 [kdevtmpfs]
root        12  0.0  0.0      0     0 ?        S<   12:03   0:00 [netns]
root        13  0.0  0.0      0     0 ?        S<   12:03   0:00 [perf]
root        14  0.0  0.0      0     0 ?        S    12:03   0:00 [khungtaskd]
root        15  0.0  0.0      0     0 ?        S<   12:03   0:00 [writeback]
root        16  0.0  0.0      0     0 ?        S<   12:03   0:00 [crypto]
root        17  0.0  0.0      0     0 ?        S<   12:03   0:00 [bioset]
root        18  0.0  0.0      0     0 ?        S<   12:03   0:00 [kblockd]
root        19  0.0  0.0      0     0 ?        S    12:03   0:01 [kworker/0:1]
root        20  0.0  0.0      0     0 ?        S<   12:03   0:00 [rpciod]
root        21  0.0  0.0      0     0 ?        S    12:03   0:00 [kswapd0]
root        22  0.0  0.0      0     0 ?        S    12:03   0:00 [fsnotify_mark]
root        23  0.0  0.0      0     0 ?        S<   12:03   0:00 [nfsiod]
root        29  0.0  0.0      0     0 ?        S<   12:03   0:00 [kthrotld]
root        30  0.0  0.0      0     0 ?        S<   12:03   0:00 [VCHIQ-0]
root        31  0.0  0.0      0     0 ?        S<   12:03   0:00 [VCHIQr-0]
root        32  0.0  0.0      0     0 ?        S<   12:03   0:00 [VCHIQs-0]
root        33  0.0  0.0      0     0 ?        S<   12:03   0:00 [iscsi_eh]
```

Each column is described below:

- **USER** – the user who owns the process (the user **pi** in this case).
- **PID** – process ID of the process (**2570**).
- **%CPU** – the CPU time used divided by the time the process has been running.
- **%MEM** – the ratio of the process's resident set size to the physical memory on the machine.
- **VSZ** – virtual memory usage of entire process (in KiB).
- **RSS** – resident set size, the non-swapped physical memory that a task has used (in KiB).
- **TTY** – controlling tty (terminal).
- **STAT** – multi-character process state.
- **START** – starting time or date of the process.
- **TIME** – cumulative CPU time.
- **COMMAND** – the command used to start the process (**tail -f /var/log/messages**).

Thread model

Raspberry Pi 4 model B comes with both single and multi-threaded configurations. According to website LeMaRiva “Raspberry Pi 4 model B was able to solve 100 iterations 2.30x times faster then the previous model.”

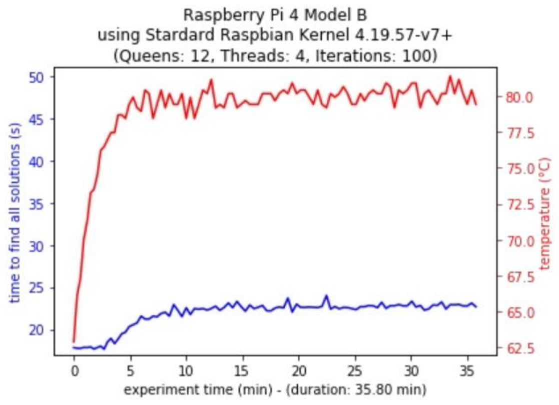


Fig. 2a: Raspberry Pi 4B - Multi-thread Configuration

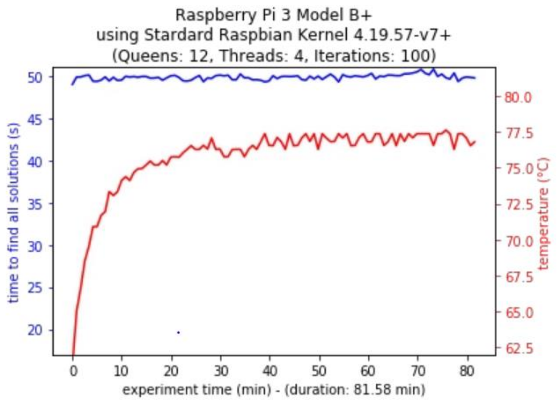


Fig. 2b: Raspberry Pi 3B+ Multi-thread Configuration

The Raspberry Pi 4 Model B specs the SoC used is BCM2711. The BCM2711 uses an ARM Cortex-A72 core.
It has four cores, each of which can be configured to run either individual or multiple threads at once. However, in the single threaded config., each core switches between (software) threads hundreds or thousands of times per second, so it can still handle several threads "at once" from a human perspective, much like any Intel- or AMD-based machine.

Memory management

The amount of memory to set aside for the GPU's exclusive use in megabytes; the remaining memory is allocated to the ARM CPU for use by the OS. The default value for Pi’s with less than 1GB of memory is 64; for Pis with 1GB or more of memory, the default value is 76.

The GPU's memory is used for display, 3D, codec, and camera functions, as well as some basic firmware maintenance. The maximums mentioned below presume that you're using both of these options. Smaller values of gpu mem can be used if you aren't.

Set gpu mem to the smallest value possible to get the best output out of Linux. If a specific graphics function isn't working, try increasing the value of gpu mem, keeping in mind the suggested maximums mentioned below.

Unlike GPUs found on x86 machines, where increasing memory can boost 3D performance, the VideoCore's architecture means that defining values larger than required has little performance benefit and can even damage performance.

The 3D part of the GPU on the Raspberry Pi 4 has its own memory management unit (MMU) and does not use the gpu mem allocation. Instead, memory in Linux is distributed dynamically. In comparison to previous versions, this allows a smaller value for gpu mem to be defined on the Pi 4.

The recommended maximum values are as follows:

total RAM	gpu_mem recommended maximum
256MB	128
512MB	384
1GB or greater	512 , 256 on the Pi4

Setting gpu mem to larger values is possible, but it should be avoided because it can trigger issues, such as stopping Linux from booting. Although the minimum value is 16, this disables certain GPU features.

You can also use gpu mem 256, gpu mem 512, and gpu mem 1024 to switch SD cards between Pis with different amounts of RAM without having to edit config.txt every time:

For Raspberry Pis with 256MB of memory, the gpu mem 256 command sets the GPU memory in megabytes. (If the memory size is less than 256MB, it is ignored.) This takes precedence over gpu mem.

For Raspberry Pis with 512MB of RAM, the gpu mem 512 command configures GPU memory in megabytes. (If the memory size is less than 512MB, this parameter is ignored.) gpu mem is overridden.

For Raspberry Pis with 1GB or more memory, the gpu mem 1024 command sets the GPU memory in megabytes. (If the memory size is less than 1GB, it is ignored.) This takes precedence over gpu mem.

This parameter can be used to force a Raspberry Pi's memory space to be limited: enter the total amount of RAM, in megabytes, that you want the Pi to use. For instance, to make a 4GB Raspberry Pi 4B behave like a 1GB model, use the following commands:

```
total_mem=1024
```

Setting this to 1 prevents the CPU from accessing the GPU's L2 cache and necessitates the use of an L2 disabled kernel. On the BCM2835, the default value is 0. Since the ARMs on the BCM2836, BCM2837, and BCM2711 have their own L2 cache, the default is 1. This disparity in cache setting is reflected in the regular Pi kernel.img and kernel7.img builds.

Scheduling details

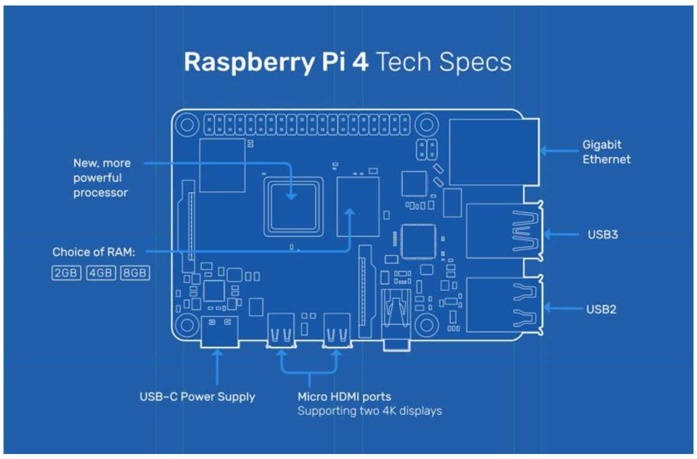
A multiprogramming operating system's process scheduling is critical. Multiple processes can be loaded into executable memory at the same time in such operating systems, and the loaded processes share the CPU by time multiplexing.

The process manager's task is process scheduling, which involves removing a running process from the CPU and selecting a new process based on a set of rules.

For a novice, scheduling commands or scripts on a Raspberry Pi, or on Linux in general, is difficult. On the Raspberry Pi, there is a tool called "crontab" that can be used to schedule tasks. This utility can be used to run a script at a given time or when the computer boots up. The Cron is a service that starts automatically when the Raspberry Pi boots up, allowing the user to run scheduled commands.

I/O model

The Raspberry Pi's row of GPIO (general-purpose input/output) pins along the top edge of the board is a powerful feature. Both new Raspberry Pi boards have a 40-pin GPIO header (unpopulated on Pi Zero and Pi Zero W). Boards had a shorter 26-pin header prior to the Pi 1 Model B+ (2014).



A variety of programming languages and resources can be used to manipulate GPIO pins, such as Python, C/C++, Processing 3 (Java)

A GPIO pin labelled as an input pin can be read as high (3V3) or low (3V4), (0V). The use of internal pull-up or pull-down resistors makes this simpler. Pull-up resistors are fixed on GPIO2 and GPIO3, although they can be programmed in software for other pins.

Funny facts about Raspberry PI Os

Resources

https://geek-university.com/raspberry-pi/list-running-processes/#:~:text=When%20you%20run%20a%20command,running%20on%20your%20Raspbian%20system.

https://www.raspberrypi.org/documentation/configuration/config-txt/memory.md

https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/

https://www.raspberrypi.org/documentation/usage/gpio/

Raspberry Pi 4B: How much faster is the new CPU? - LeMaRiva|tech