

[FAVORITAR](#)

 6 dias atrás

 Pedro Bertoleti

 Nenhum comentário

Análise de desempenho do Linux embarcado



[ÍNDICE DE CONTEÚDO](#)



- ▶ Análise de desempenho do Linux embarcado: o que isso engloba?
- ▶ Uso de CPU – geral e por processos específicos
- ▶ Uso de memória RAM – geral e por processos específicos
- ▶ Memory swap
- ▶ Presença de memory thrashing
- ▶ Monitoramento de memory leaks
- ▶ Conclusão
- ▶ Referências
- ▶ Saiba mais

Introdução

O Linux é cada vez mais utilizado nos mais diferentes dispositivos eletrônicos que podemos imaginar, de eletrodomésticos e dispositivos portáteis até grandes servidores. Na vertente de Linux embarcado, isso não é diferente, havendo muitos produtos que já contam com este sistema embarcado e muita coisa está sendo desenvolvida neste momento utilizando o Linux embarcado como sistema operacional também.

Sendo o Linux embarcado um sistema operacional de grande versatilidade, é extremamente importante garantir que ele tenha um bom desempenho, tanto para ser fluido para o usuário final quanto para executar com eficácia e eficiência as tarefas nas quais foi designado.

Este artigo mostrará algumas formas e ferramentas para se analisar o desempenho do Linux embarcado, de forma a ser possível quantificar as métricas importantes de desempenho e se ter um norte quais processos e aplicações podem estar causando problemas de desempenho no Linux embarcado como um todo.

Hardware utilizado

Para demonstrar o uso das ferramentas usadas na análise do desempenho, fizemos uso de um sistema de borda de uma das placas-mãe da nossa linha de placas de desenvolvimento.



(figura 1), devido a sua popularidade e facilidade de uso.

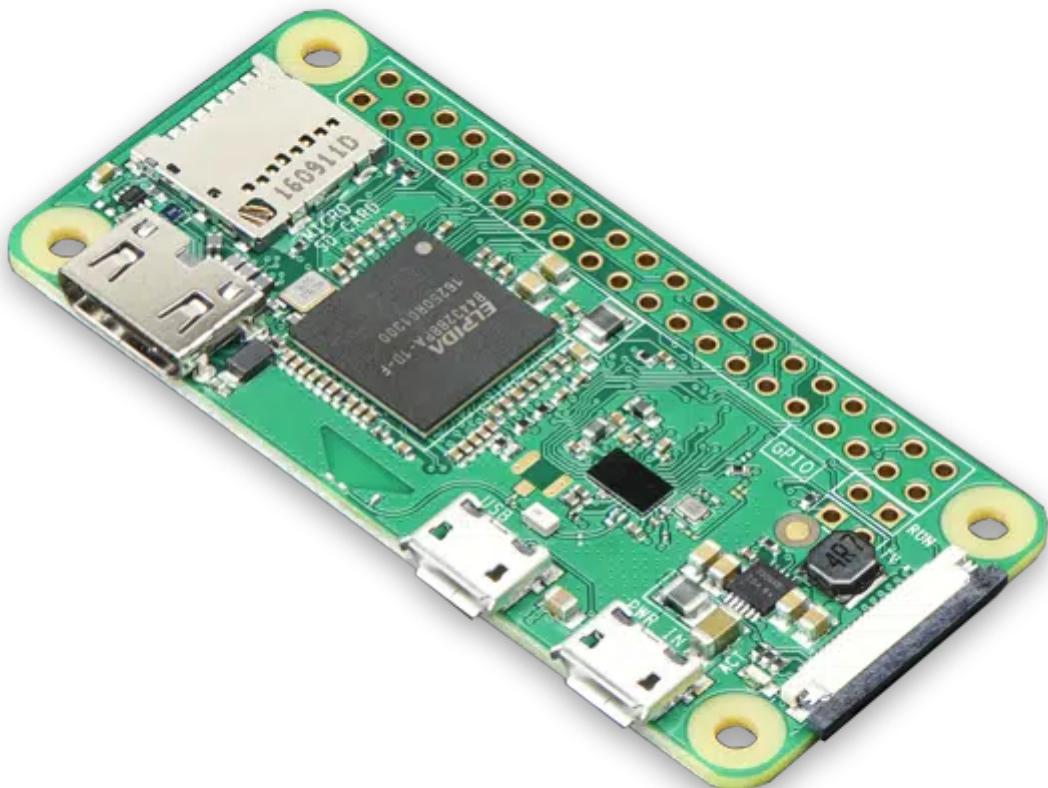


Figura 1 – placa Raspberry Pi Zero W

Além da Raspberry Pi Zero W, é necessário ter um micro-SD card (tamanho recomendado: 8GB ou maior) com a imagem mais recente do Raspberry Pi OS gravada. Recomenda-se o uso do software Raspberry Pi Imager para o processo de download da imagem mais recente do Raspberry Pi OS e gravação da imagem dele no micro-SD card. Para obter o software Raspberry Pi Imager, acesse este link: <https://www.raspberrypi.com/software/>

Análise de desempenho do Linux embarcado: o que isso engloba?

Quando se fala em análise de desempenho, deve-se ter em mente que esta análise abrange uma série de fatores e métricas, ou seja, não é somente uma coisa para se monitorar e ficar de olho.

Em linhas gerais, no Linux embarcado, é comum se monitorar métricas relacionadas ao uso de CPU e memória RAM. De forma mais detalhada, as métricas mais comuns a serem analisadas no Linux embarcado são:

- Uso de CPU, tanto uso geral quanto por processos específicos;
- Uso de memória RAM, tanto uso geral quanto por processos específicos;
- Memory swap
- Monitoramento de memory leaks
- Memory thrashing

Cada um destes itens será abordado neste artigo, de forma que você, leitor, possua meios para fazer uma análise geral de desempenho de uma solução com Linux embarcado.

Uso de CPU – geral e por processos específicos

O uso adequado de CPU em sistemas com Linux embarcado – e em qualquer solução em sistemas embarcados – é de suma importância para o bom desempenho geral do sistema, assim como para otimizar o consumo de energia elétrica da solução. O uso excessivo / alto de CPU por muito tempo pode levar, além do aumento do consumo de energia elétrica e aquecimento do SoC / SiP, a uma lentidão geral do Linux embarcado, prejudicando, portanto, seu desempenho.

verificar se o uso de CPU apresentado (por processo e também geral) é alto e, ainda, se fica nessa situação por muito tempo. Em caso afirmativo, uma das duas hipóteses abaixo torna-se verdadeira:

1. O SoC ou SiP utilizados possui CPUs sub-dimensionadas para a aplicação desejada;
2. Ou, como acontece na maioria das vezes: o hardware utilizado possui processamento suficiente para a solução que está sendo desenvolvida, mas por alguns problemas e erros no desenvolvimento, há processos ou aplicações da solução que estão demandando muito mais processamento das CPUs do que o esperado.

Na hipótese 1, normalmente não há muito a se fazer além de se trocar o hardware da solução para um que suporte-a, uma vez que neste caso o hardware possui poder computacional aquém do necessário. Já na hipótese 2, trata-se de um cenário em que melhorias no design de software das aplicações ou processos da solução e também correção de eventuais bugs podem otimizar muito o uso de CPU, provocando uma melhora geral de desempenho do Linux embarcado. Porém, como mensurar o uso de CPU geral e por processo, de forma a saber se o uso de CPU pode ser otimizado em algum processo ou aplicação da solução?

Para se mensurar isso, há diversas ferramentas disponíveis no Linux. Uma delas é uma ferramenta chamada **htop**. Infelizmente, não é muito comum esta ferramenta vir instalada por padrão numa distro Linux, mas por outro lado, é muito comum ela estar disponível para instalação nas ferramentas de gerenciamento de pacotes (apt-get, por exemplo). No caso da Raspberry Pi Zero W, o **htop** pode ser instalado com o comando abaixo:

```
1 sudo apt-get updatesudo apt-get install htop
```

Esta ferramenta mostra, em tempo real, uma visão dinâmica de várias métricas de funcionamento do Linux, nos quesitos de: uso de CPU, uso de memória RAM, uso de swap, uptime de processos, memória virtual dos processos, etc.

- **Uso percentual individual instantâneo de cada CPU disponível**, destacado em verde na figura 2;
- **Uso de CPU por processo**, destacado em laranja na figura 2;
- **Uso médio de CPU geral por tempo**, chamado de *Load Average*, em destaque na cor azul-claro na figura 2: uso de CPU médio no último minuto (primeiro valor do Load Average), nos últimos 5 minutos (segundo valor do Load Average) e nos últimos 15 minutos (terceiro valor do Load Average).

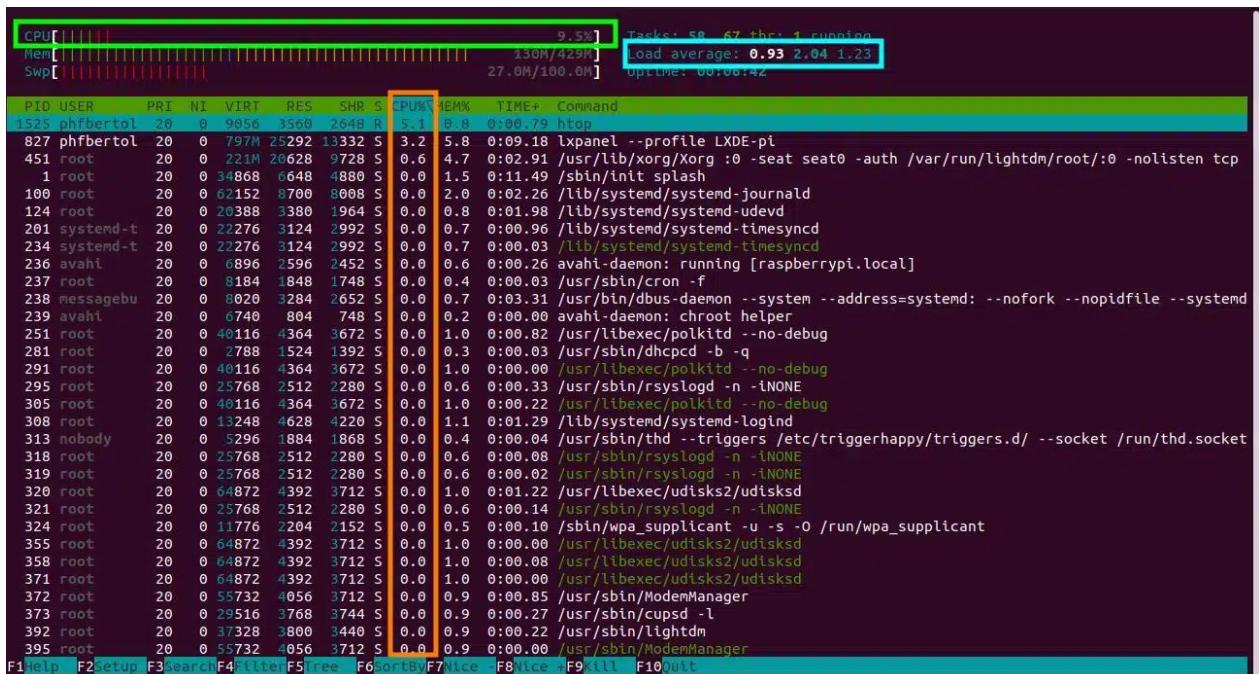


Figura 2 – HTOP funcionando na Raspberry Pi

Ainda, utilizando as teclas de F3 a F9 no **htop**, é possível pesquisar por processos específicos (algo extremamente útil para monitorar só um processo), filtrar a visualização de processos e matar processos também.

Logo, o **htop** é uma ferramenta muito útil para monitorar o uso de CPU no Linux embarcado, permitindo se ter ideia tanto do uso instantâneo de CPU (geral e por processos) quanto o uso médio de CPU nos últimos minutos.

Outra ferramenta que pode ser usada para analisar o uso de CPU (por processo) é o **ps**. Sua principal funcionalidade é o monitoramento de

processos estão efetivamente rodando ou não, sendo possível ter um panorama geral de execução de processos no Linux embarcado. Veja um output do ps obtido da Raspberry Pi Zero W na figura 3.

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	1.4	1.5	34868	6648	?	Ss	17:50	0:11	/sbin/init splash
root	2	0.0	0.0	0	0	?	S	17:50	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	I<	17:50	0:00	[slub_flushwq]
root	4	0.0	0.0	0	0	?	I<	17:50	0:00	[netns]
root	8	0.0	0.0	0	0	?	I<	17:50	0:00	[mm_percpu_wq]
root	9	0.0	0.0	0	0	?	S	17:50	0:00	[rcu_tasks_rude_]
root	10	0.0	0.0	0	0	?	S	17:50	0:00	[rcu_tasks_trace]
root	11	0.2	0.0	0	0	?	S	17:50	0:02	[ksoftirqd/0]
root	12	0.0	0.0	0	0	?	S	17:50	0:00	[kdevtmpfs]
root	13	0.0	0.0	0	0	?	I<	17:50	0:00	[inet_frag_wq]
root	15	0.0	0.0	0	0	?	S	17:50	0:00	[kauditd]
root	16	0.0	0.0	0	0	?	S	17:50	0:00	[khungtaskd]
root	17	0.0	0.0	0	0	?	S	17:50	0:00	[oom_reaper]
root	18	0.0	0.0	0	0	?	I<	17:50	0:00	[writeback]
root	19	0.2	0.0	0	0	?	S	17:50	0:02	[kcompactd0]
root	37	0.0	0.0	0	0	?	I<	17:50	0:00	[kblockd]
root	38	0.0	0.0	0	0	?	I<	17:50	0:00	[blkcg_punt_blo]
root	39	0.0	0.0	0	0	?	S	17:50	0:00	[watchdog]
root	41	0.1	0.0	0	0	?	I	17:50	0:01	[kworker/u2:2-events_unbound]
root	42	0.0	0.0	0	0	?	I<	17:50	0:00	[rpciod]
root	43	0.2	0.0	0	0	?	I<	17:50	0:01	[kworker/u3:0-brcmf_wq/mmc1:0001:1]
root	44	0.0	0.0	0	0	?	I<	17:50	0:00	[xprtiod]
root	45	0.2	0.0	0	0	?	S	17:50	0:01	[kswapd0]
root	46	0.0	0.0	0	0	?	I<	17:50	0:00	[nfsviod]
root	47	0.0	0.0	0	0	?	I<	17:50	0:00	[iscsi_eh]
root	48	0.0	0.0	0	0	?	I<	17:50	0:00	[iscsi_conn_clea]
root	49	0.0	0.0	0	0	?	I<	17:50	0:00	[dwc_otg]
root	50	0.0	0.0	0	0	?	I<	17:50	0:00	[DWC_Notifyatio]
root	52	0.0	0.0	0	0	?	S<	17:50	0:00	[vchiq-slot/0]
root	53	0.0	0.0	0	0	?	S<	17:50	0:00	[vchiq-recy/0]
root	54	0.0	0.0	0	0	?	S<	17:50	0:00	[vchiq-sync/0]
root	55	0.0	0.0	0	0	?	I<	17:50	0:00	[zswap-shrink]
root	57	0.0	0.0	0	0	?	I<	17:50	0:00	[mmc_complete]
root	58	0.2	0.0	0	0	?	I<	17:50	0:02	[kworker/0:1H-mmc_complete]
root	60	0.0	0.0	0	0	?	S	17:50	0:00	[jbd2/nmcblk0p2-]
root	61	0.0	0.0	0	0	?	I<	17:50	0:00	[ext4-rsv-conver]
root	62	0.0	0.0	0	0	?	I<	17:50	0:00	[mid]
root	63	0.0	0.0	0	0	?	I<	17:50	0:00	[tpv6_addrconf]
root	100	0.2	1.9	62152	8700	?	Ss	17:50	0:02	/lib/systemd/systemd-journald
root	124	0.2	0.7	20388	3386	?	Ss	17:50	0:01	/lib/systemd/systemd-udevd
root	141	0.0	0.0	0	0	?	S	17:50	0:00	[vchiq-keep/0]
root	144	0.0	0.0	0	0	?	S<	17:50	0:00	[SMIO]
root	170	0.0	0.0	0	0	?	I<	17:50	0:00	[mmal-vchiq]
root	171	0.0	0.0	0	0	?	I<	17:50	0:00	[mmal-vchiq]
root	173	0.0	0.0	0	0	?	I<	17:50	0:00	[mmal-vchiq]
root	175	0.0	0.0	0	0	?	I<	17:50	0:00	[mmal-vchiq]
root	176	0.0	0.0	0	0	?	I<	17:50	0:00	[mmal-vchiq]
root	177	0.0	0.0	0	0	?	I<	17:50	0:00	[mmal-vchiq]
root	178	0.0	0.0	0	0	?	I<	17:50	0:00	[mmal-vchiq]
systemd+	201	0.1	0.7	22276	3124	?	Ssl	17:50	0:00	/lib/systemd/systemd-timesyncd
avahi	236	0.0	0.5	6896	2596	?	Ss	17:50	0:00	avahi-daemon: running [raspberrypi.local]
root	237	0.0	0.4	8184	1848	?	Ss	17:50	0:00	/usr/sbin/cron -f
message+/-	238	0.4	0.7	8020	3284	?	Ss	17:50	0:03	/usr/bin/dbus-daemon --system --address=systemd: --nofork --nrepidfile --systemd-activation --syslog-only
avahi	239	0.0	0.1	6740	804	?	S	17:50	0:00	avahi-daemon: chroot helper

Figura 3: Saída do ps na Raspberry Pi

A ferramenta **ps** possui muitas opções / parametrizações possíveis, sendo uma ferramenta bem completa. Para conhecer tudo o que ele pode oferecer, recomendo a leitura de sua documentação: <https://man7.org/linux/man-pages/man1/ps.1.html>

Uma terceira ferramenta muito útil na análise de uso de CPU no Linux embarcado é o **sar**. O **sar** (acrônimo para System Activity Reporter) é uma ferramenta para monitorar a atividade geral de cada CPU do Linux embarcado, com métricas relativas a uso percentual, tempo de ociosidade, iowait, etc. É uma ótima opção para se verificar a carga das CPUs da solução e se ter, portanto, um panorama completo do uso de CPUs pelo Linux embarcado. No caso da distro Raspberry Pi OS, o **sar** não vem instalado por padrão. Para instalar essa ferramenta, execute os comandos a seguir:



É possível parametrizar o **sar** no tempo de monitoramento (tempo entre monitoramentos) e quais CPUs ele deve monitorar. Por exemplo, para monitorar a única CPU disponível na Raspberry Pi Zero W com intervalo entre monitoramentos de 500ms, utilize o comando abaixo:

```
1 sar -p 1 500
```

Veja um output do **sar** obtido da Raspberry Pi Zero W na figura 4.

phfbertoleti@raspberrypi:~ \$ sar -p 1 500							
Linux 5.15.84+ (raspberrypi)		11/04/23		_armv6l_		(1 CPU)	
18:07:11	CPU	%user	%nice	%system	%iowait	%steal	%idle
18:07:13	all	0.00	0.00	0.00	0.00	0.00	100.00
18:07:14	all	0.00	0.00	2.00	0.00	0.00	98.00
18:07:15	all	0.00	0.00	1.98	0.00	0.00	98.02
18:07:16	all	0.00	0.00	2.00	0.00	0.00	98.00
18:07:17	all	0.00	0.00	1.01	0.00	0.00	98.99
18:07:18	all	1.00	0.00	0.00	0.00	0.00	99.00
18:07:19	all	1.01	0.00	0.00	0.00	0.00	98.99
^C	Average:	all	0.29	0.00	1.00	0.00	98.71

Figura 4: Saída do **sar** na Raspberry Pi

Assim como o **ps**, o **sar** possui muitas opções / parametrizações possíveis, sendo uma ferramenta bem completa. Para conhecer tudo o que ele pode oferecer, recomendo a leitura de sua documentação:

<https://man7.org/linux/man-pages/man1/sar.1.html> ↗

Uso de memória RAM – geral e por processos específicos

O bom uso de memória RAM é vital para um bom desempenho de um sistema embarcado, uma vez que, em linhas gerais, ela armazena e, em tempo de uso, todos os aplicativos, aplicações e tudo que for essencial para o sistema operacional (no caso, Linux embarcado) rodar, em tempos de acesso muito superiores aos de uma memória de massa.

O uso de uma quantidade excessiva de memória RAM por parte de certos

dados temporários (buffers) e tudo mais que for necessário para o Linux embarcado executar. Isso força o sistema operacional a gerenciar o uso de memória RAM com memory swap, ou até mesmo a forçar o fim de processos para liberar memória. Tal situação leva a redução significativa de desempenho, chegando em casos extremos ao memory thrashing, onde o tempo gasto no gerenciamento de paginação de memória é tão ou mais significativo que o tempo gasto efetivamente rodando os processos e aplicações necessários, causando extrema lentidão do Linux embarcado de forma geral. Os tópicos de memory swap e memory thrashing serão vistos em detalhes mais à frente neste artigo.

Portanto, ao se observar que uma solução com Linux embarcado apresenta lentidão e desempenho insatisfatório, além do já citado uso de CPU, uma das providências a ser tomada é verificar se o uso de memória RAM nos processos e aplicações da solução é muito significativo e, ainda, se essa situação perdura por muito tempo. Em caso afirmativo, tem-se um problema de uso de memória RAM. Problemas deste tipo podem ter como causa desde um uso excessivo de memória RAM por alguns processos devido a falhas no design do software ou até mesmo um memory leak (condição que também será vista em mais detalhes a seguir neste artigo).

Todas as ferramentas abordadas até agora (**htop**, **ps** e **sar**) podem também ser usadas para monitorar uso de memória RAM. No caso do **htop**, cada processo reportado contém a informação percentual de memória utilizada (referida como MEM%), assim como a quantidade total de memória em uso e o uso do Swap. Portanto, o **htop** é uma ferramenta muito útil para investigar o uso percentual instantâneo de um ou mais processos no Linux Embarcado. Veja na figura 5, em verde, o uso percentual de memória RAM por processo e, em azul, o uso geral de memória RAM e o uso do swap no **htop**.

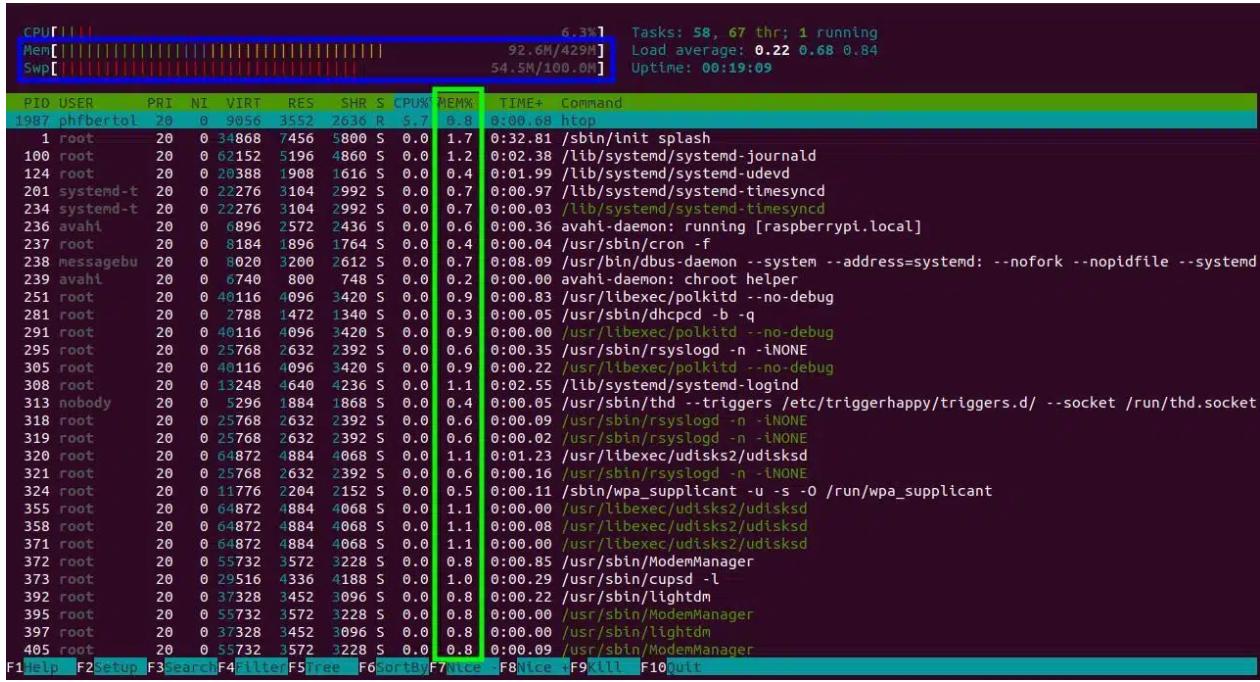


Figura 5: Uso de memória RAM por processo e geral no htop

Além do **htop**, outra ferramenta importante para se analisar o uso de memória RAM no Linux Embocado é o **free**. O **free** tem como diferencial fornecer uma visão geral de uso de memória do Linux Embocado, na qual comprehende: memória RAM total, livre, usada, compartilhada entre processos, etc. É possível até observar, através de suas métricas, memory leaks e memory thrashing.

```
phfbertoleti@raspberrypi:~ $ free
              total        used      free      shared  buff/cache   available
Mem:       439752      90576     192728       4196     156448      292432
Swap:      102396      55808      46588
phfbertoleti@raspberrypi:~ $
```

Figura 6: Output do **free**

A ferramenta **free** possui muitas opções / parametrizações possíveis, sendo uma ferramenta bem completa. Para conhecer tudo o que ele pode oferecer, recomendo a leitura de sua documentação: <https://man7.org/linux/man-pages/man1/free.1.html> ↗

Primeiramente, é preciso definir o que é o memory swap. De forma simples, o memory swap é uma operação realizada pelo sistema operacional (incluindo o Linux embarcado) para salvar em memória de massa (SSD, micro-SD card, HD, etc.) dados contidos na memória RAM que estão sendo pouco utilizados.

Dessa forma, libera espaço em RAM para uso geral e, ainda, é possível resgatar novamente para a memória RAM os dados salvos em memória de massa, caso estes venham a ser necessários novamente. Portanto o memory swap ajuda a manter uma quantidade saudável de memória RAM.

O memory swap é um recurso essencial em todo sistema operacional, sobretudo aqueles com pouca disponibilidade de memória RAM, o que costuma ser o caso de muitos sistemas embarcados com Linux embarcado (lembre que: mais memória RAM = mais custo de hardware, então não se deve esperar quantidades gigantescas de memória RAM em um sistema embarcado).

Entretanto, o memory swap tem um preço caro a ser pago: lentidão. A velocidade de leitura, escrita e comunicação com uma memória RAM é muito superior se comparado com uma memória de massa, principalmente se comparar com os HDs. Cada operação de swap tem como gargalo a velocidade geral (leitura, escrita e comunicação) da memória de massa, sendo portanto uma operação extremamente lenta de ser feita do ponto de vista do sistema operacional.

Para visualizar a quantidade de memória swap ocupada no Linux embarcado, as maneiras mais rápidas são utilizar as ferramentas **htop** ou **free**, conforme já mostrado neste artigo. É uma estratégia interessante monitorar, de forma manual ou automatizada, a quantidade usada de swap ao longo do tempo, para se ter uma ideia do quanto agressivo está o consumo de memória RAM no Linux embarcado.

O memory swap é algo a ser evitado, em prol de um melhor desempenho geral do Linux embarcado. Para evitá-lo, é preciso conhecer as principais possíveis causas dele. As três principais causas de maior frequência de ocorrência do memory swap são:

1. Memória RAM subdimensionada: em um cenário onde a quantidade de memória RAM disponível é subdimensionada para a solução, a tendência é que o memory swap ocorra com grande frequência, contribuindo para uma maior lentidão do sistema embarcado.

2. Má parametrização do swap: o memory swap começa a agir (ou seja, começa a mover dados da memória RAM para a memória de massa) quando a quantidade de memória RAM livre do sistema atinge certo valor. No Linux embarcado, isso é parametrizado a partir do **swappiness**, parâmetro visualizável pelo arquivo **/proc/sys/vm/swappiness**. Esse arquivo tem como conteúdo um número, de 0 a 100, correspondendo à porcentagem de memória RAM livre a partir da qual faz o memory swap entrar em ação (mas, se for configurado em zero, o memory swap é desativado). Logo, valores grandes (mais próximos de 100) de swappiness significam que o memory swap vai entrar em operação ainda com muita memória RAM livre, o que em muitos cenários pode ser desnecessário e só servirá para gerar lentidão geral no Linux embarcado. Como exemplo, se o swappiness estiver configurado como 80, quando a memória RAM ocupada atingir 20%, o memory swap começa a agir. Logo, maiores valores de swappiness (em torno de 60) costumam ser utilizados para levar a um uso moderado de memory swap, sem provocar muita lentidão do Linux embarcado. Para se alterar o valor do swappiness no Linux embarcado (no caso da Raspberry Pi Zero W, usando Raspberry Pi OS), basta escrever alterar o valor de **vm.swappiness** dentro do arquivo **/etc/sysctl.conf**. É importante mencionar que isso só pode ser feito como superusuário.

3. Uso exagerado de memória RAM por um processo ou aplicação: algumas vezes, por más decisões dos desenvolvedores, muita memória RAM é alocada em um determinado processo ou aplicação, levando a um maior uso da memória RAM geral do sistema e aumento do uso de memory swap. Essa situação é particularmente comum quando se porta uma solução que rodava em um computador pessoal (ou servidor) para um Linux embarcado, pois no ambiente anterior possivelmente a quantidade de memória RAM disponível era muito maior que em um sistema embarcado.

Presença de memory thrashing

O memory thrashing é, na verdade, uma consequência do uso exagerado de memory swap. Conforme foi abordado no tópico anterior, o memory swap tem como gargalo a velocidade geral (escrita, leitura e comunicação) da memória de massa.

Quando o Linux embarcado entra em uma situação onde a quantidade de memória RAM livre é tão baixa que muito memory swap precisa ser feito, ocorre, por consequência, muita interação com a memória de massa. Como esta interação é lenta, pode-se chegar ao ponto de o sistema operacional gastar mais tempo fazendo memory swap do que executando de fato as aplicações e processos que deve, levando a uma lentidão extrema do Linux embarcado. É a esse fenômeno que chamamos de memory thrashing.

Em uma analogia livre, é como cavar um poço profundo usando somente uma colher. A colher (memória RAM) comporta pouco material (dados, buffers, aplicações, etc.), logo quem está cavando o buraco (sistema operacional) deve fazer várias viagens até um reservatório (memória de massa) para esvaziar a colher nele (memory swap) e continuar cavando. Nesse caso, dada a baixa capacidade da colher, o processo de esvaziá-la leva um tempo significativo em relação à operação principal, que é cavar. Isso faz com que o sistema operacional gaste muito tempo esvaziando a colher (memory thrashing).

Para diagnosticar o memory thrashing, é preciso observar alguns comportamentos de algumas das métricas relativas a uso de memória RAM no Linux embarcado, métricas estas disponíveis nas ferramentas abordadas até agora (**htop**, **sar** e **free**):

- Uso elevado de memória de swap, sobretudo em situações cuja ocupação da memória RAM não seja tão alta;
- Uso de memória swap crescendo muito conforme o tempo passa;

- Aumento da porcentagem do I/O Wait (chamado de iowait no Linux)

tempo uma (ou mais) CPUs gastaram esperando a resposta de um dispositivo de I/O, como uma memória de massa, por exemplo. Portanto, quanto mais ocorrer memory swap, maior será o valor de iowait observado;

Como o memory thrashing é algo a ser evitado no Linux embarcado, é importante evitar portanto o memory swap, o que pode ser feito observando (e mitigando) as causas de memory swap apresentadas no tópico anterior.

Monitoramento de memory leaks

Um memory leak é uma falha grave em programas, na qual consiste em situações onde um programa aloca dinamicamente uma certa porção de memória RAM e não a desaloca após o uso, deixando esta referida porção de memória RAM indisponível para qualquer outro programa utilizá-la. Se esse problema ocorre em uma rotina que é chamada continuamente, em pouco tempo será esgotada a memória RAM livre do sistema, causando sérios problemas de desempenho (forçando o memory swap, levando a memory thrashing, etc.) e funcionamento do sistema operacional. Em alguns casos, até mesmo pode causar o travamento de todo o sistema, forçando seu reinício.

Um caso clássico de memory leak é quando um programa em C faz alocação dinâmica de memória RAM (com malloc(), por exemplo) porém não é feita, após o uso da memória alocada, a sua liberação com o free().

Uma maneira de detectar memory leaks no Linux embarcado é monitorar, de forma manual ou automatizada, de tempos em tempos, a ocupação de memória RAM (geral e/ou por processo). Isso pode ser feito usando algumas das ferramentas usadas até agora neste artigo: **htop**, **ps** e **free**. Se for observado que um determinado processo ou aplicação tem a sua quantidade de memória RAM somente aumentando com o tempo, há ali um indício de memory leak.

destinada à análise dinâmica de programas, e pode ser usada também para detecção de memory leaks. Esta ferramenta pode ser instalada no Raspberry Pi com os seguintes comando:

```
1 sudo apt-get update
2 sudo apt-get install valgrind
```

Por se tratar de uma ferramenta de análise dinâmica, ela funciona como um “wrapper” para o programa desejado. Para usar o **valgrind** na detecção de memory leaks, basta chamá-lo com os parâmetros `--leak-check=full`, passando também como parâmetro o programa no qual se deseja verificar se há vazamentos de memória. Supondo que o programa chama-se `programa_suspeito`, o uso do **valgrind** com ele seria:

```
1 valgrind --leak-check=full ./programa_suspeito
```

Caso houver algum memory leak detectado pela ferramenta, este será informado assim que acontecer no referido programa.

Conclusão

Neste artigo, foram mostradas técnicas e ferramentas para a análise de desempenho do Linux embarcado, sob o ponto de vista de uso dos seguintes recursos computacionais: CPU e memória RAM.

Com o conteúdo aqui apresentado, é possível fazer uma análise sob estes aspectos de forma a determinar se processos ou aplicações estão utilizando mais recursos computacionais que o esperado. Dessa forma, o desenvolvedor será capaz de compreender as possíveis razões de um desempenho aquém do esperado do Linux embarcado de uma determinada solução e, ainda, saber quais processos são os responsáveis (sendo um norte para a depuração do problema).

Referências

- [https://man7.org/linux/man-pages/man1/top.1.html ↗](https://man7.org/linux/man-pages/man1/top.1.html)

- <https://www.embedded.com/performance-analysis-of-linux-based-embedded-systems-part-1/> ↗
- <https://man7.org/linux/man-pages/man1/ps.1.html> ↗
- <https://man7.org/linux/man-pages/man1/sar.1.html> ↗
- <https://www.ic.unicamp.br/~rafael/materiais/valgrind.html> ↗

Saiba mais

Como se tornar um especialista em Linux Embarcado

Utilizando watchdog no Linux embarcado

Otimização de Tempo de Boot no Linux Embarcado

--- publicidade ---

LEIA TAMBÉM



Ensemble Graphics Toolkit: GUI Linux gratuita para
microprocessadores da Microchip

TREINAMENTO ONLINE EMBARCADOS 

Desenvolvimento de aplicações para



A dark rectangular banner with white text. At the top left is the text "TREINAMENTO ONLINE EMBARCADOS" next to a graduation cap icon. Below it is a large, bold title "Desenvolvimento de aplicações para". To the right of the title is a portrait of a man with a yellow dashed circle drawn around his head.

Novo Treinamento sobre Desenvolvimento de aplicações para
Linux Embarcado



BeagleBone - Primeiros passos para trabalhar com Linux
Embarcado

POR QUE DOMINAR
**SISTEMA
OPERACIONAL
LINUX** deveria ser a
sua prioridade?



Webinar gravado: Por que dominar sistema operacional Linux
deveria ser a sua prioridade?



Como se tornar um especialista em Linux Embarcado

JUNTE-SE HOJE À COMUNIDADE EMBARCADOS

FAZER PARTE



Pedro Bertoleti

61 posts

<https://pedrobertoleti.com.br/> ↗

Sou engenheiro eletricista formado pela Faculdade de Engenharia de Guaratinguetá (FEG - UNESP) e trabalho com Android embarcado em Campinas-SP, no Instituto de Pesquisas Eldorado (<https://www.eldorado.org.br/>).

Curioso e viciado em tecnologia, sempre busco me aprimorar na área de sistemas embarcados (modalidades bare-metal, RTOS, Linux embarcado e Android embarcado) e em Internet das Coisas.

Esta obra está licenciada com uma Licença Creative Commons Atribuição-Compartilhagual 4.0 Internacional ↗.

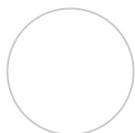


✉ Raspberry Pi, Linux Embarcado

[Home](#) » [Linux Embarcado](#) » Análise de desempenho do Linux embarcado

COMENTÁRIOS:

✉ Notificações ▾



Seja o Primeiro a Comentar!

B I U S ≡ ≡ “ </> 🔗 { } [+]



0 COMENTÁRIOS



TALVEZ VOCÊ GOSTE:

Nenhum resultado encontrado.

SÉRIES

[ULWOS – Multitarefa no RL78](#)

[Controlador VGA](#)

[Monitoramento de água com IoT](#)

[Trazendo o mundo real para dentro do processador](#)

[Sistemas Operacionais de Tempo Real](#)

[A arte de especificar e encontrar componentes](#)

[Projetos de desenvolvimento: Antes de começar](#)

[Boas práticas para o desenvolvimento de software](#)

[GNU ARM Cross-toolchain](#)

[Shape The World](#)

[Ver todas as séries →](#)

NEWSLETTER

Receba os melhores conteúdos sobre sistemas eletrônicos embarcados, dicas, tutoriais e promoções.

Seu e-mail

Concordo com o **Termo de Uso e Política de Privacidade** do Embarcados *

CADASTRAR E-MAIL

INSTITUCIONAL

O Embarcados

Seja Colaborador

Contato

NAS REDES



COMUNIDADE

Oportunidades

Sites e Blogs



LEGAL

Legal

Política de Privacidade

Política de Governança

Política de Cookies

Termos de Uso

© Embarcados – Todos os direitos reservados. Termos de Uso.

Desenvolvido por

