URJC Escuela Técnica Superior de Ingeniería de Telecomunicación

Sistemas Empotrados y de Tiempo Real

RTOS: Real Time Operative System

Grado en Ingeniería de Robótica Software

Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

Roberto Calvo Palomino
roberto.calvo@urjc.es

RTOS

- Existen multitud de RTOS
  - https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems
- Dependiendo del uso, características y plataforma soportada habrá RTOS que se ajusten más a nuestros requisitos
- Existen RTOS Open Source y propietarios.
- Compatibles con POSIX (Linux)
- FreeRTOS, VxWorks, QNX, RTLinux

Características

- Un RTOS es un sistema operativo multi-tarea diseñado para ejecutar aplicaciones de tiempo real.
- Comportamiento predecible y determinista.
- Garantizar la ejecución completa de la tarea en un tiempo determinado.
- Debe implementar herencia de prioridades para evitar bloqueos y mal funcionamiento del sistema
- 2 tipos:
  - Soft RTOS
  - Hard RTOS

Funciones básicas

- Manejo de tareas
  - Tiempo de ejecución, periodo y tiempo limite.
- Manejo de interrupciones
  - Síncronas y asíncronas
- Administración de memoria
  - Normalmente no hay memoria virtual, ni reserva dinámica.
- Excepciones
  - Timeouts, bloqueos, tiempo limite no cumplido, …
- Sincronización de tareas
  - Semáforos, mutex, spinlock, cerrojos para lectura/escritura
- Planificador de tareas
  - Basados en prioridad, RMS, EDFS, RR, …
- Manejo del tiempo
  - Reloj hardware programado para interrumpir al procesador a intervalos fijos.

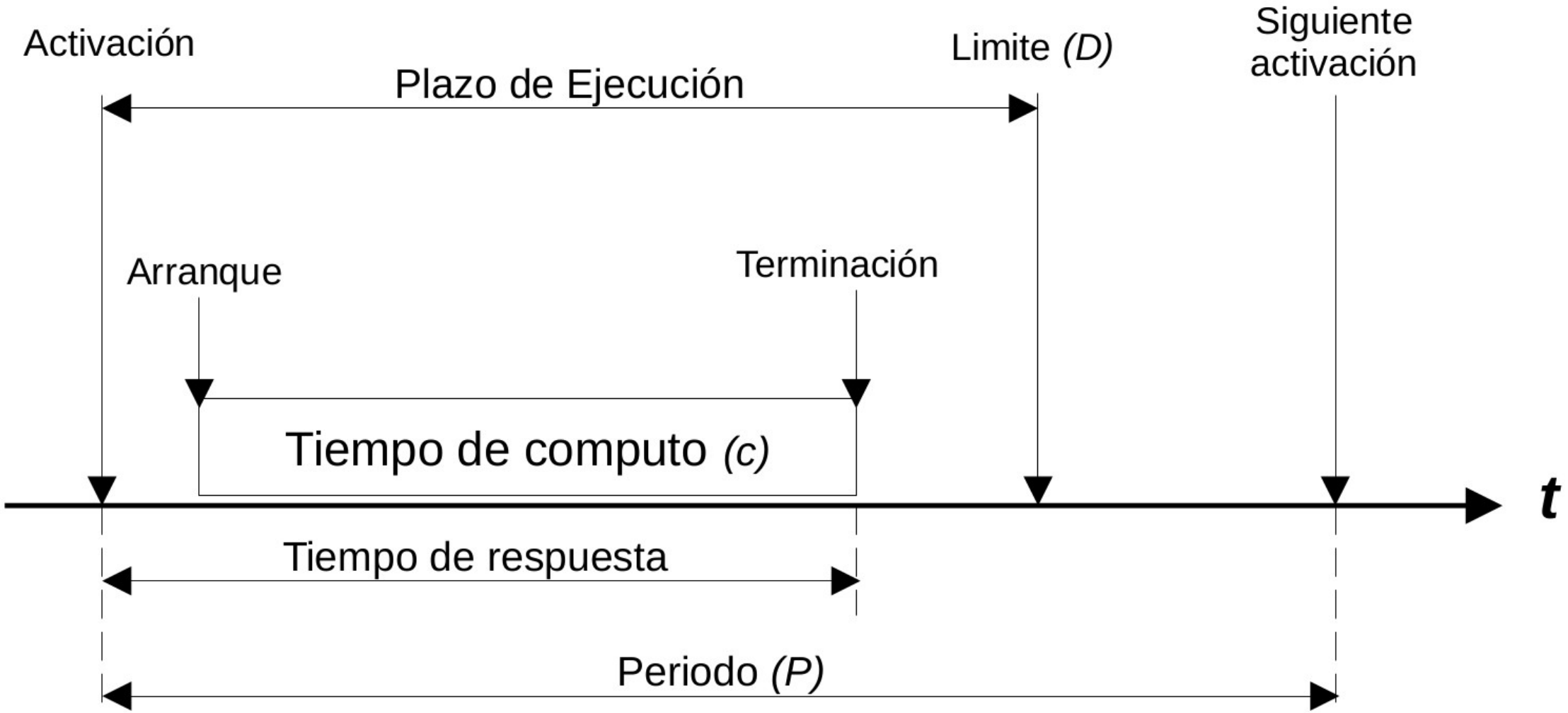
¿Qué RTOS elegir?

- Dependiendo de las características del sistema:
  - Menor **latencia** en cambio de contexto
  - Menor **latencia** en tratar interrupciones
  - Menor tamaño de kernel
  - Mayor flexibilidad en los algoritmos de planificación
  - Mayor rango de arquitectura suportadas
  - Soporte para Mono o multi núcleo
  - APIs starndars, POSIX, etc.

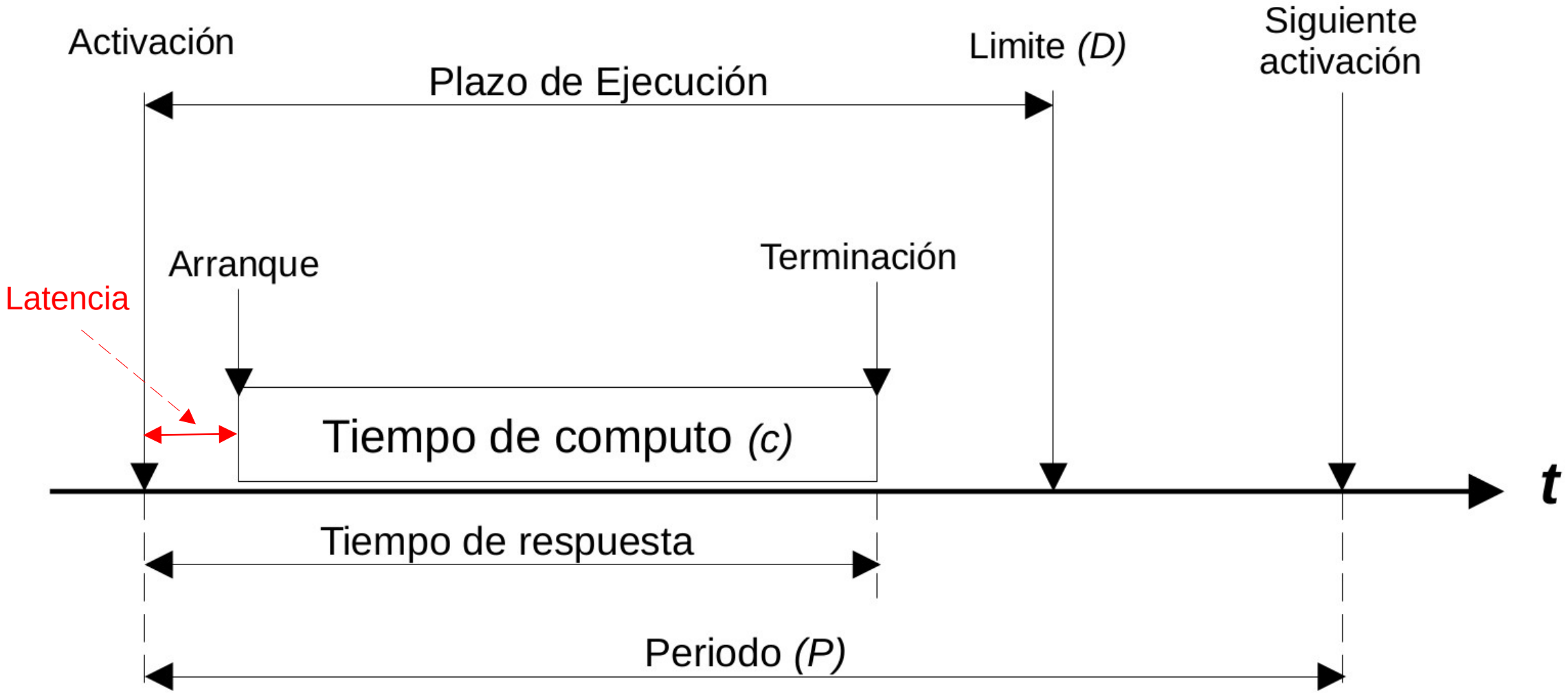
Latencia

- **Latencia**: intervalo de tiempo entre estímulo y respuesta.
- Normalmente es el indicador más importante a la hora de elegir un RTOS
  - Latencia de interrupción: tiempo entre que una interrupción se genera y el instante que la interrupción es atendida (ISR: Interrupt Service Routine). Depende del microprocesador, controladores, sistema operativo, etc.
  - Latencia de planificación: tiempo entre que una tarea debe ejecutarse y el instante en el que verdaderamente se ejecuta.
- Las capacidades y rendimiento de un RTOS puede medirse mediante la latencia de planificación.

Latencia de planificación

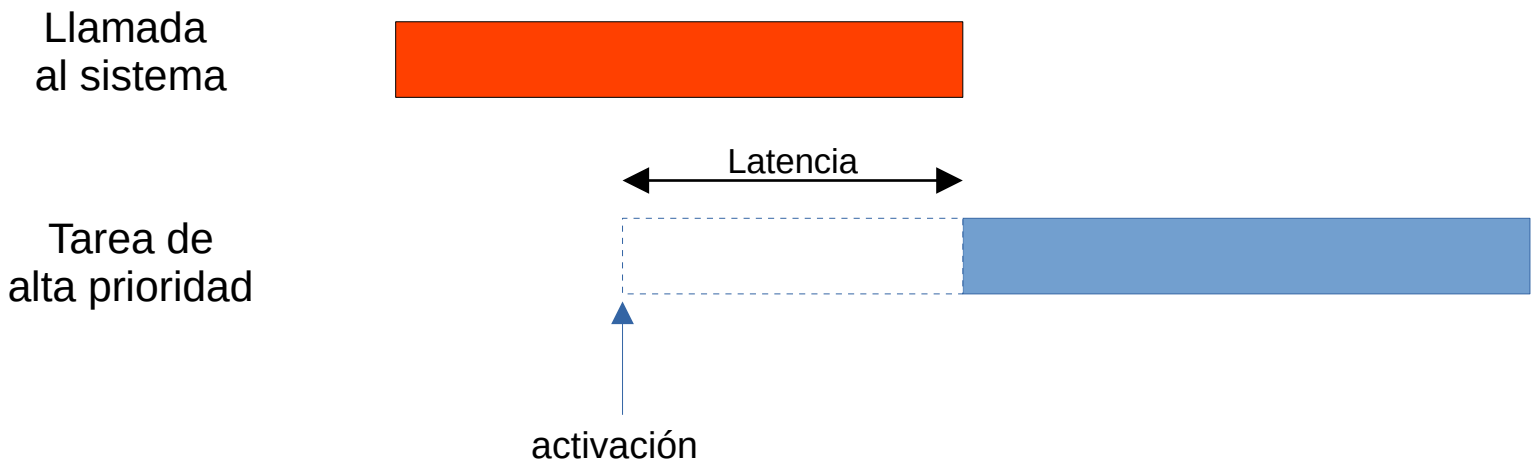


Latencia de planificación



Causas de alta latencia

- Prioridades ilimitadas.
- Latencia en el algoritmo de planificación.
- Latencia en las interrupciones y en acceso a memoria.
- En SO generalistas, procesos que usan llamadas al sistema no son “expulsables” de la CPU en cualquier momento.



Kernel de Linux

Kernel de Linux

- **No** es un sistema operativo de tiempo real.
- El planificador no es determinista ni predictivo.
- Las **llamadas al sistema** tienen preferencia y no pueden expulsarse de la CPU en cualquier momento.
- Los procesos que ejecutan en **espacio de usuario** son siempre expulsables (preemptible).
  - Un bucle infinito en espacio de usuario no bloquea el sistema.

- Aún así, el kernel de linux es usado en dispositivos empotrados como routers y smartphones.

Scheduler en el kernel de Linux

- El planificador/scheduler de Linux en versiones mayores a 2.6.23 es **CFS: "Completely Fair Scheduler"**
- Este algoritmo tiene como objetivo el maximizar el uso de la CPU pero permitiendo el uso interactivo de la máquina.
  - Minimizar latencia en procesos interactivos
- CFS utiliza una granularidad de nanosegundos y no hace uso de los **"timeslices"** de antiguos planificadores.
- Está basado en un árbol binario de búsqueda equilibrado (árbol rojo-negro, RBTREE).
- Complejidad de O(log(N)).

CFS

- CFS implementa 3 colas de ejecución:
  - SCHED_NORMAL: Usado para tareas regulares
  - SCHED_BATCH: No expulsa tan a menudo como en tareas regulares. Favorece tareas de ejecución largas que hace mejor uso de caches a costa de la interactividad.
  - SCHED_IDLE: Usado para tareas de muy baja prioridad.
- CFS ofrece políticas de “real-time”
  - SCHED_FIFO: se basa en política First-In, First-Out
  - SCHED_RR: se basa en política round robin (quantum fijo).
  - SCHED_DEADLINE: se basa en GEDF (Global Earliest Deadline First)

```
$ man sched
```

CFS

- Configurar nuestro thread para ejecutar en la cola SCHED_FIFO con prioridad 99

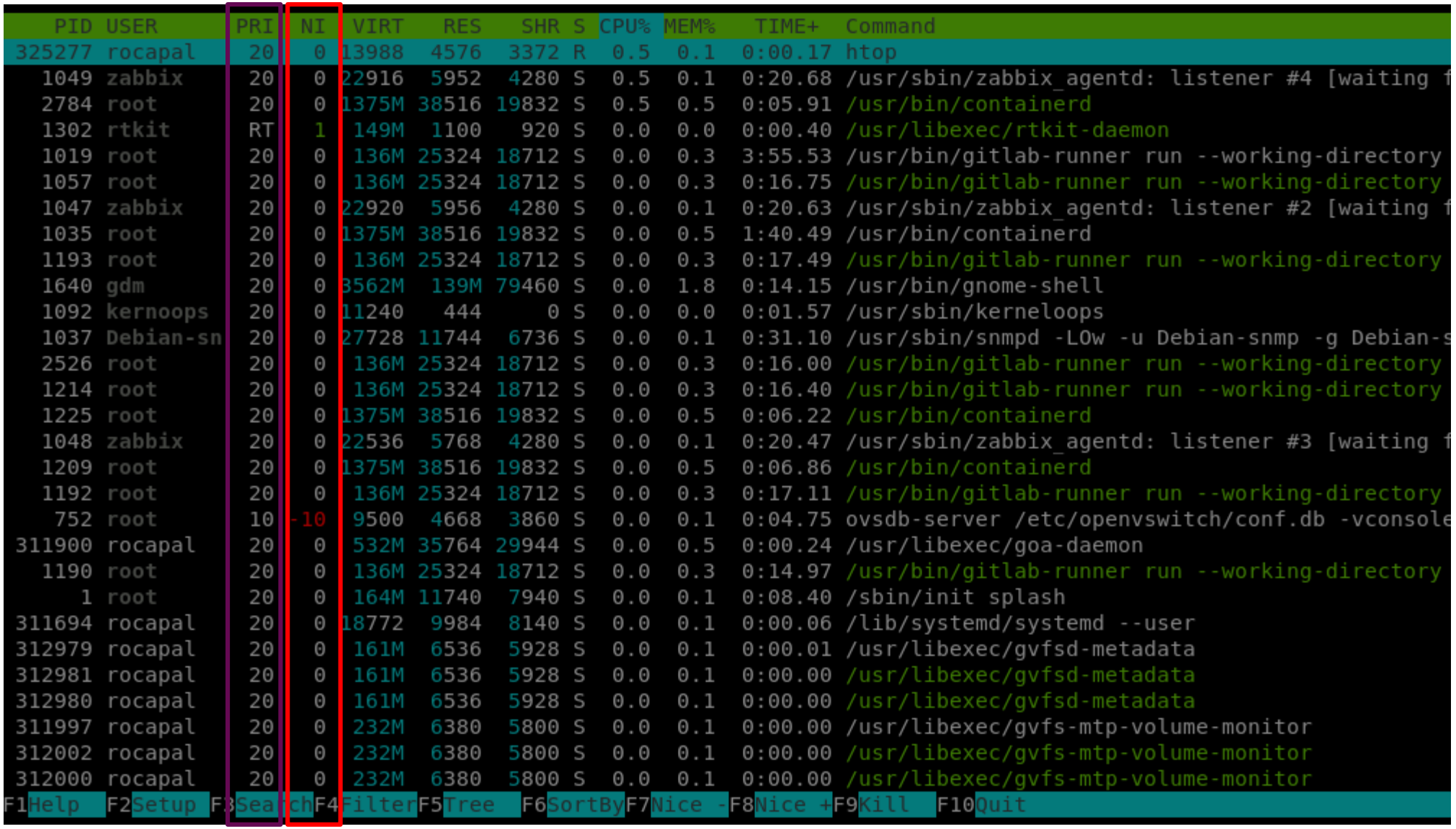
```
struct sched_param param;
param.sched_priority=99;

pthread_setschedparam (pthread_self(), SCHED_FIFO, &param);
```

Prioridades y NICE

- El valor de **prioridad** de un proceso se visualiza como PR o PRI (usando *ps* o *htop*). Es el valor de prioridad que el kernel/planificador utiliza.
  - Prioridad a nivel de espacio de usuario 100-139
  - Prioridad real-time: 1 (baja) a 99(alta) (SCHED_FIFO, SCHED_RR)
- **Nice**: Valor utilizado en espacio de usuario para cambiar la prioridad de un proceso
  - Rango de -20 (alta) a 19(baja) , 0 por defecto.
- Prioridad = 20 + Nice , nice=[-20, 19]
  - Los procesos linux tiene una prioridad de **20** por defecto (nice=0)

Prioridades y NICE

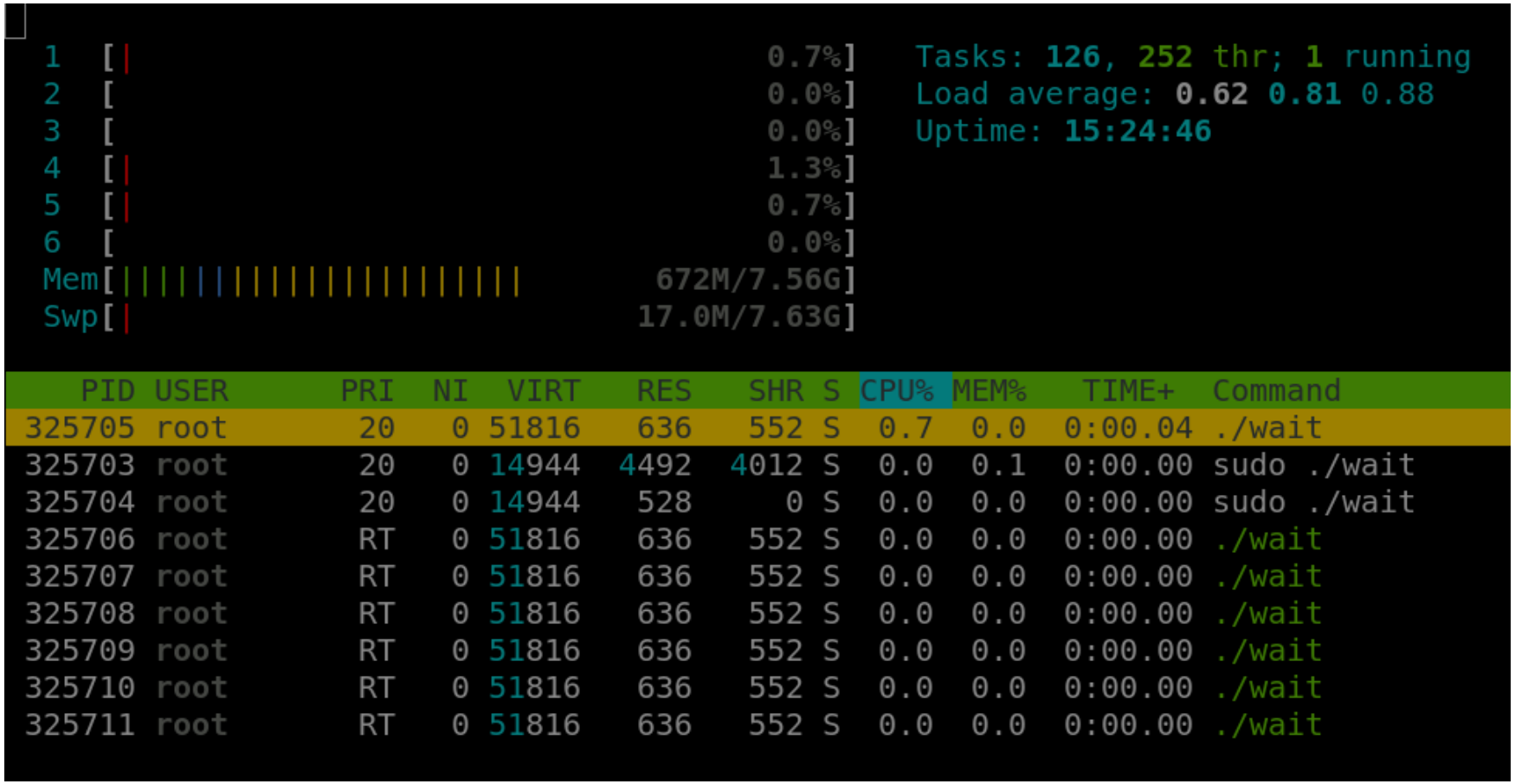


PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
325277	rocapal	20	0	13988	4576	3372	R	0.5	0.1	0:00.17	htop
1049	zabbix	20	0	22916	5952	4280	S	0.5	0.1	0:20.68	/usr/sbin/zabbix_agentd: listener #4 [waiting f
2784	root	20	0	1375M	38516	19832	S	0.5	0.5	0:05.91	/usr/bin/containerd
1302	rtkit	RT	1	149M	1100	920	S	0.0	0.0	0:00.40	/usr/libexec/rtkit-daemon
1019	root	20	0	136M	25324	18712	S	0.0	0.3	3:55.53	/usr/bin/gitlab-runner run --working-directory
1057	root	20	0	136M	25324	18712	S	0.0	0.3	0:16.75	/usr/bin/gitlab-runner run --working-directory
1047	zabbix	20	0	22920	5956	4280	S	0.0	0.1	0:20.63	/usr/sbin/zabbix_agentd: listener #2 [waiting f
1035	root	20	0	1375M	38516	19832	S	0.0	0.5	1:40.49	/usr/bin/containerd
1193	root	20	0	136M	25324	18712	S	0.0	0.3	0:17.49	/usr/bin/gitlab-runner run --working-directory
1640	gdm	20	0	3562M	139M	79460	S	0.0	1.8	0:14.15	/usr/bin/gnome-shell
1092	kernoops	20	0	11240	444	0	S	0.0	0.0	0:01.57	/usr/sbin/kerneloops
1037	Debian-sn	20	0	27728	11744	6736	S	0.0	0.1	0:31.10	/usr/sbin/snmpd -LOw -u Debian-snmp -g Debian-s
2526	root	20	0	136M	25324	18712	S	0.0	0.3	0:16.00	/usr/bin/gitlab-runner run --working-directory
1214	root	20	0	136M	25324	18712	S	0.0	0.3	0:16.40	/usr/bin/gitlab-runner run --working-directory
1225	root	20	0	1375M	38516	19832	S	0.0	0.5	0:06.22	/usr/bin/containerd
1048	zabbix	20	0	22536	5768	4280	S	0.0	0.1	0:20.47	/usr/sbin/zabbix_agentd: listener #3 [waiting f
1209	root	20	0	1375M	38516	19832	S	0.0	0.5	0:06.86	/usr/bin/containerd
1192	root	20	0	136M	25324	18712	S	0.0	0.3	0:17.11	/usr/bin/gitlab-runner run --working-directory
752	root	10	-10	9500	4668	3860	S	0.0	0.1	0:04.75	ovsdb-server /etc/openvswitch/conf.db -vconsole
311900	rocapal	20	0	532M	35764	29944	S	0.0	0.5	0:00.24	/usr/libexec/goa-daemon
1190	root	20	0	136M	25324	18712	S	0.0	0.3	0:14.97	/usr/bin/gitlab-runner run --working-directory
1	root	20	0	164M	11740	7940	S	0.0	0.1	0:08.40	/sbin/init splash
311694	rocapal	20	0	18772	9984	8140	S	0.0	0.1	0:00.06	/lib/systemd/systemd --user
312979	rocapal	20	0	161M	6536	5928	S	0.0	0.1	0:00.01	/usr/libexec/gvfsd-metadata
312981	rocapal	20	0	161M	6536	5928	S	0.0	0.1	0:00.00	/usr/libexec/gvfsd-metadata
312980	rocapal	20	0	161M	6536	5928	S	0.0	0.1	0:00.00	/usr/libexec/gvfsd-metadata
311997	rocapal	20	0	232M	6380	5800	S	0.0	0.1	0:00.00	/usr/libexec/gvfs-mtp-volume-monitor
312002	rocapal	20	0	232M	6380	5800	S	0.0	0.1	0:00.00	/usr/libexec/gvfs-mtp-volume-monitor
312000	rocapal	20	0	232M	6380	5800	S	0.0	0.1	0:00.00	/usr/libexec/gvfs-mtp-volume-monitor

F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice - F8Nice + F9Kill F10Quit

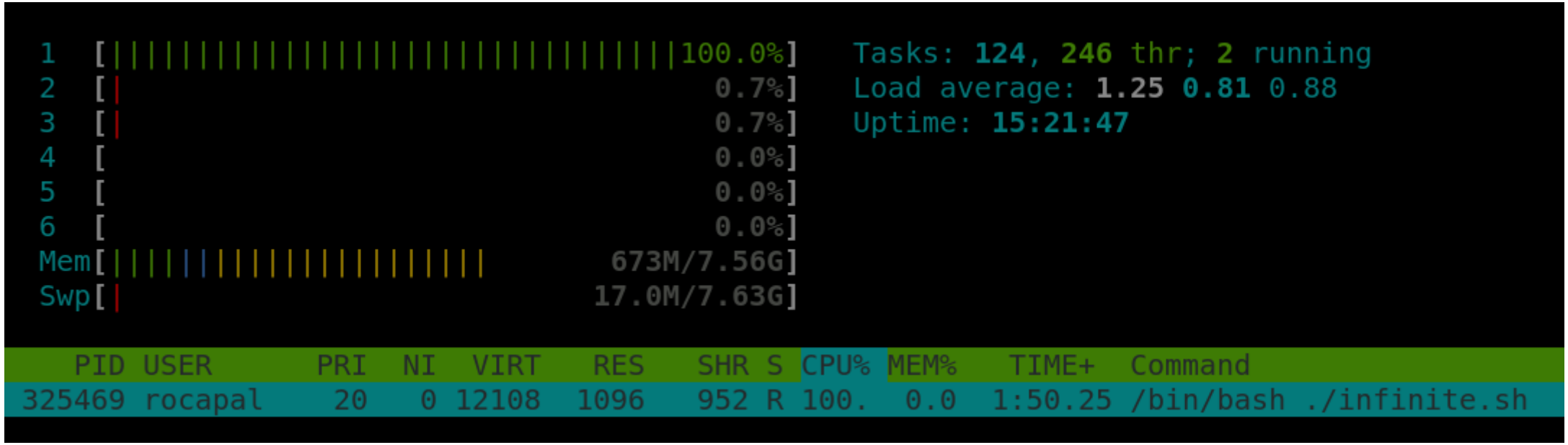
Prioridades y NICE

- Cuando un proceso están en la cola de prioridades de real time (SCHED_RR ó SCHED_FIFO), aparece **RT** en la columna *PRI* de *htop*.



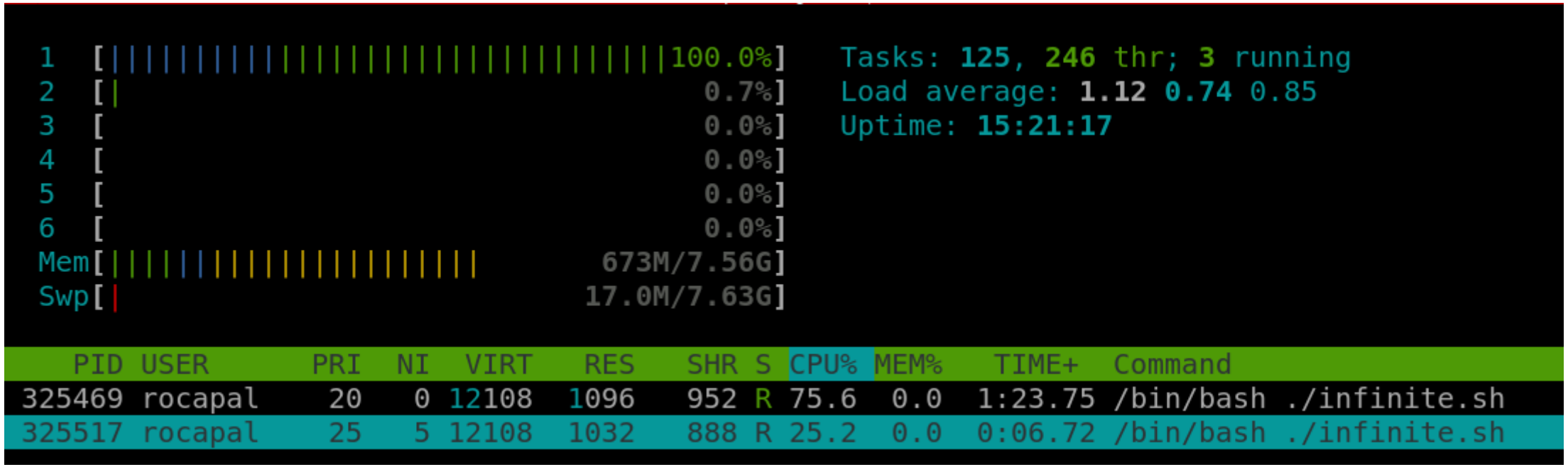
Prioridades y NICE

- Lanzamos un proceso en la CPU-0
  - taskset -c 0 ./infinite.sh &



Prioridades y NICE

- Lanzamos otra instancia del mismo proceso con un valor de **nice** especifico.
  - taskset -c 0 nice -n 5 ./infinite.sh &



Gestion de tiempos: Relojes

- Relojes **Hardware**
  - RTC (Real Time Clock), mantiene la gestión del tiempo cuando el equipo está apagado.

- Relojes **Software** (system clock o kernel clock)
  - CLOCK_REALTIME: Reloj del sistema. Puede sufrir ajustes para corregir la fecha (NTP).
  - CLOCK_MONOTONIC : Igual que CLOCK_REALTIME pero no se realizan ajustes, por tanto su cuenta es creciente sin saltos bruscos. Avanza constantemente por cada tick.
    Útil para medir duraciones entre eventos.

Gestion de tiempos: Relojes

- CLOCK_MONOTONIC vs CLOCK_REALTIME

```
clock_gettime(CLOCK_MONOTONIC, &begin);
[...]
// Transcurren 1000 ms
clock_gettime(CLOCK_MONOTONIC, &end);
end - begin ~= 1000 ms



clock_gettime(CLOCK_REALTIME, &begin);
[...]
// Transcurren 1000 ms
clock_gettime(CLOCK_REALTIME, &end);
end - begin pueden no ser 1000 ms
```

Kernel de Linux de Tiempo Real RTLinux

Historia

- **RTLinux** fue inicialmente desarrollado por Michael Barabanov y Victor Yodaiken, y fue adquirido por Wind River en 2007.
- Primeras versiones ofrecía un API muy reducido y no POSIX.
- En Octubre de 2015 el proyecto pasa a estar bajo la **Linux Foundation**[1] junto con la colaboración de Google, Intel, IBM, Qualcomm o ScanDisk
- RTLinux no es un código independiente, es un parche que se aplica sobre una versión del kernel de linux (versión estable 5.6.19)
  - https://git.kernel.org/pub/scm/linux/kernel/git/rt/linux-stable-rt.git

[1]https://www.linuxfoundation.org/press-release/2015/10/the-linux-foundation-announces-project-to-advance-real-time-linux/

RT Linux

- Permite un sistema híbrido de:
  - Tareas de tiempo real
  - Tareas regulares
- La mayoría del código realizado en Linux puede ejecutarse con muy pocas modificaciones en RTLinux (POSIX)
- Minimiza la latencia lo máximo posible
- Minimiza el código de kernel no PREEMPT.
- Sistema operativo de tiempo real estricto.
- Extensiones para entorno multiprocesador SMP.

RT Linux

- ¿Dónde está la magia del parche?
- Fuerza la interrupción de threads
  - Permite priorizar los controladores de interrupciones.
- Permitir que los locks se duerman
  - rt_spinlocks, rt_mutexes, semaphores, …
- Eliminar secciones críticas
  - Evitar zonas no expulsables
  - Interrupciones
  - Spinlocks
- Herencia de prioridad

RT Linux

- Sistema de tiempo real duro / 95%
- Latencia estable pero con algunos picos
  - CPU idle states
    - /sys/devices/system/cpu/cpu0/cpuidle/
  - Voltaje y frecuencia dinámica
  - Multi-Threading (cambio de contexto)
  - Hardware

Políticas de Tiempo Real

- SCHED_FIFO, SCHED_RR
- SCHED_DEADLINE: se basa en GEDF (Global Earliest Deadline First)
  - Modelo para tareas esporádicas
    - Requisitos temporales críticos, activaciones a instantes aleatorios.

```
struct sched_attr task;
pid_ttid = syscall(SYS_gettid);
task.size = sizeof(struct sched_attr);
task.sched_policy = SCHED_DEADLINE;
task.sched_flags = 0;
task.sched_nice = 0;
task.sched_priority = 0;
task.sched_runtime = 100 * 1000;
task.sched_deadline = 200 * 1000;
task.sched_period = 200 * 1000;

if (sched_setattr(tid, &task, 0) < 0)
   handle_err("Could not set SCHED_DEADLINE attributes");
```

Políticas de Tiempo Real

- SCHED_DEADLINE:
  - Muy utilizado en sistemas de tiempo real
  - Puede contar las veces que se no se ha cumplido el deadline.
  - NO es capaz de controlar tareas que no han cumplido el deadline.

Non PREEMPT y PREEMPT RT

```
$ uname -a

Linux f-l3202-pc10 5.4.0-47-generic #51-Ubuntu
SMP Fri Sep 4 19:50:52 UTC 2020 x86_64 x86_64
x86_64 GNU/Linux
```

```
$ uname -a

Linux f-l3202-pc11 5.4.66-rt38 #1 SMP PREEMPT_RT
Sun Oct 4 19:59:00 CEST 2020 x86_64 x86_64 x86_64
GNU/Linux
```

¿Cómo podemos medir la latencia de planificación de un sistema Linux ?

Cyclictest

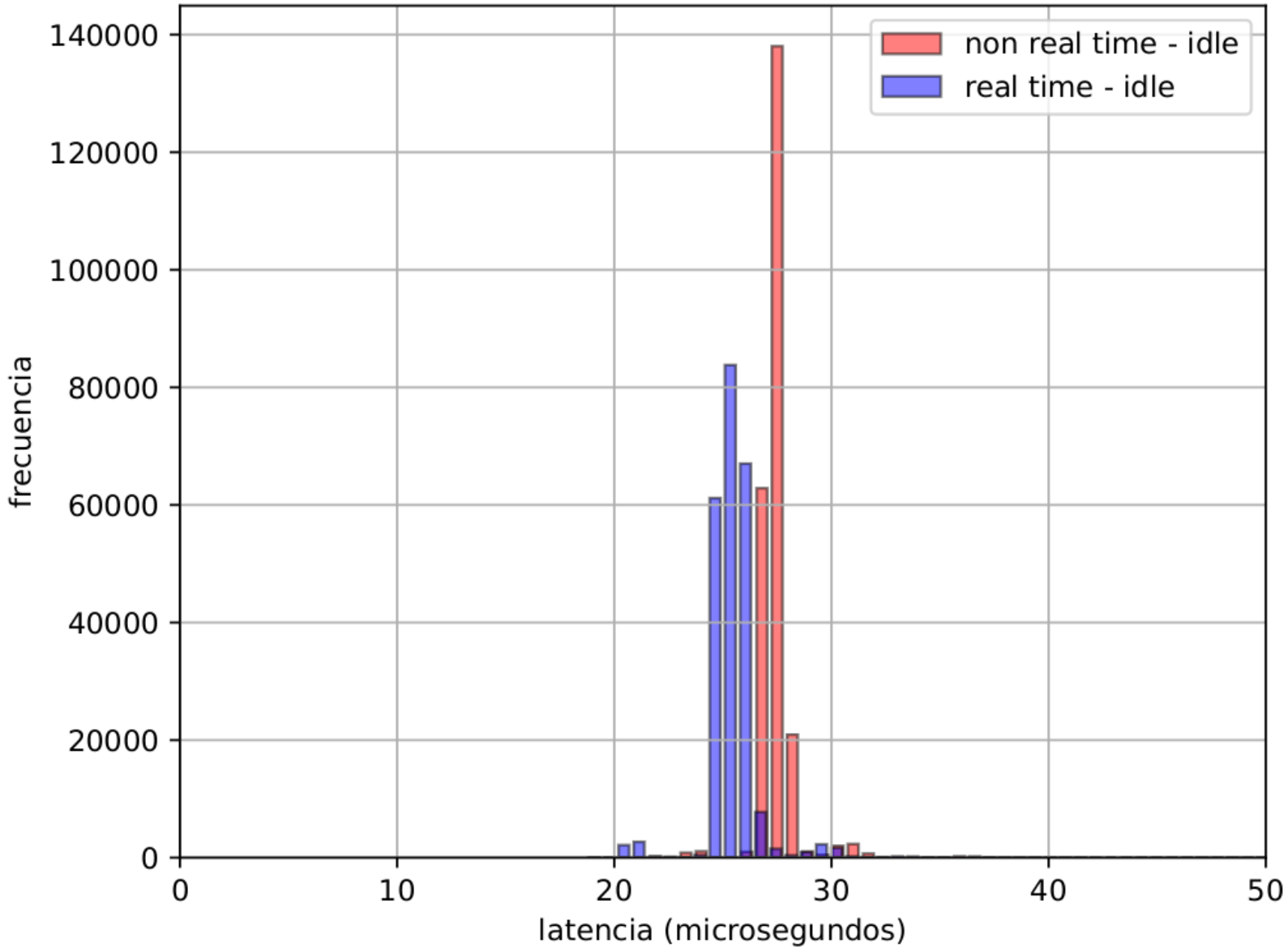
- Utilidad para medir la latencia en sistemas operativos
- Mide la latencia a nivel de planificador
  - Tiempo entre que una tarea se despierta (wake up) y el verdadero instante en el que empieza a ejecutar
- Adoptado como banco de pruebas para entornos RT basados en GNU/Linux.
- La latencia se debe medir en diferentes escenarios
  - IDLE
  - Numerosas operaciones I/O (interrupciones)
  - Multitud de procesos ejecutando en los cores (stress o hackbench)

Otras herramientas

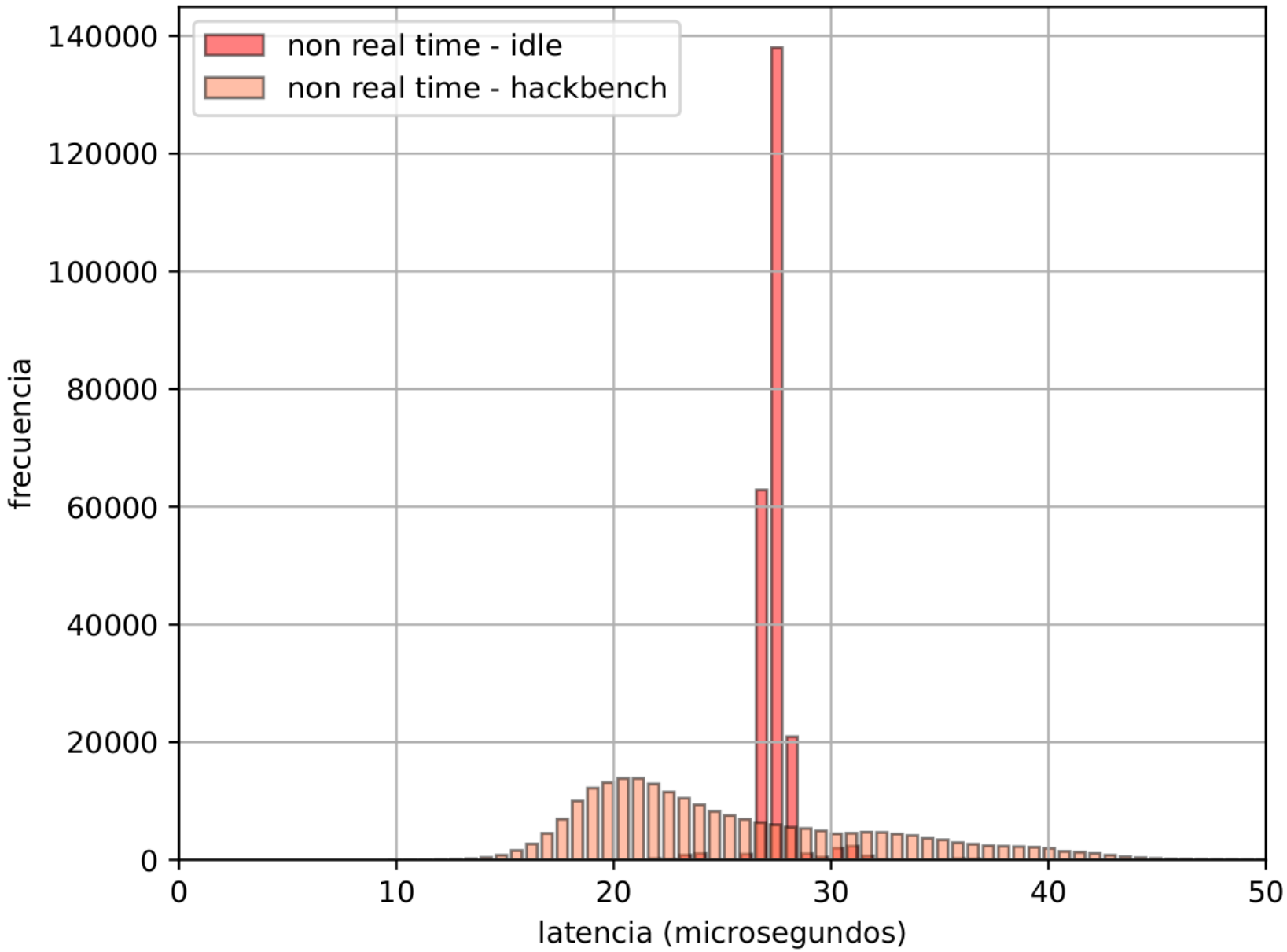
- **stress**: Utilidad para generar sobrecarga computacional en el sistema
- **hackbench**: Utilidad para testear y estresar el planificador.
  - Define número de threads o procesos hijo.
  - Define el número de mensajes intercambiados entre threads.
  - Mensajes enviados a través de pipes en vez de sockets.

- **boonie++**: Utilidad para testear I/O y sistema de ficheros.
  - Define modos de escritura (buffering, no buffering).
  - Genera muchas operaciones I/O con sincronización protegida por semáforos.

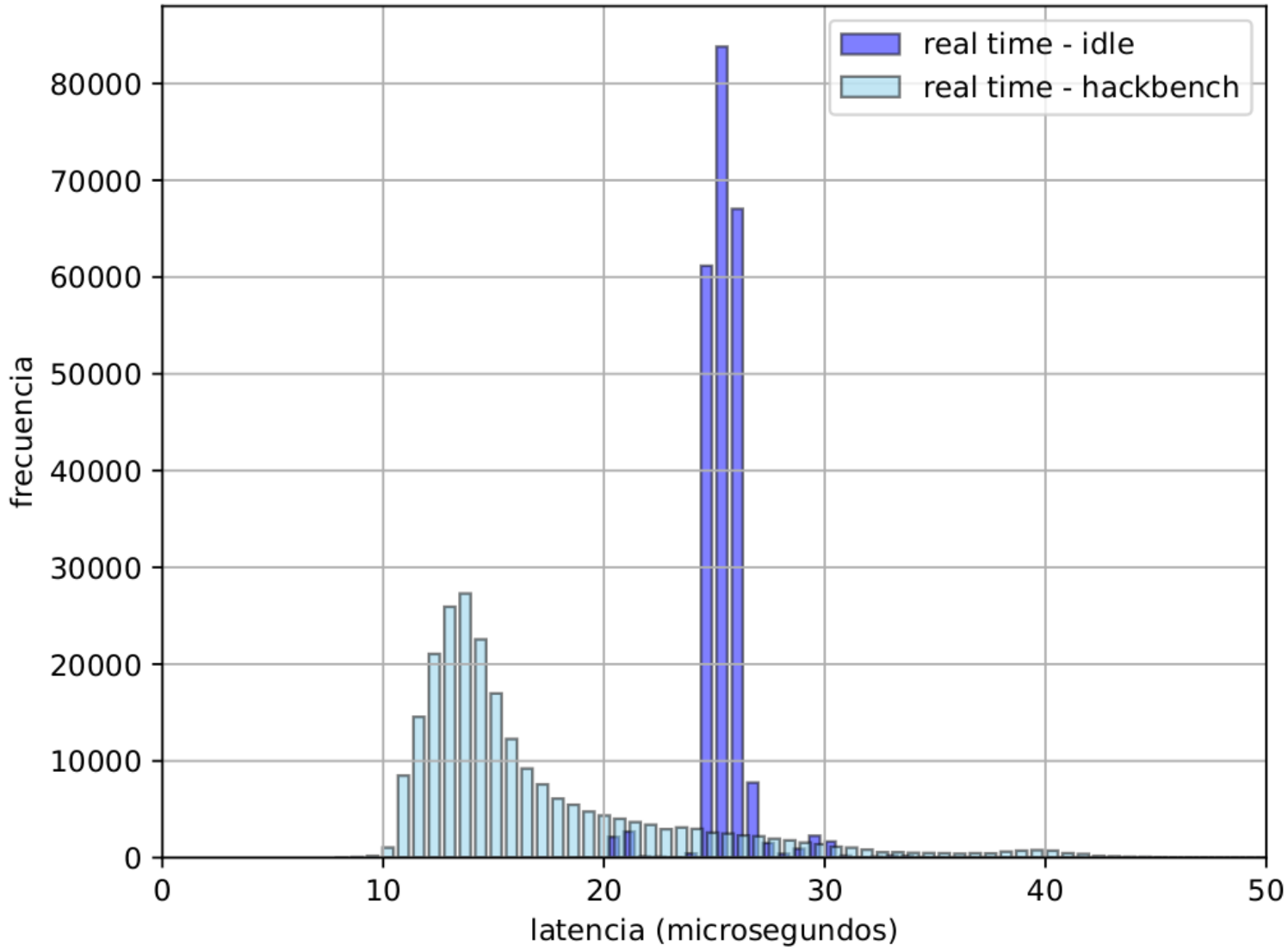
Histograma


non real time - idle
real time - idle
frecuencia
latencia (microsegundos)
0
20000
40000
60000
80000
100000
120000
140000
0
10
20
30
40
50

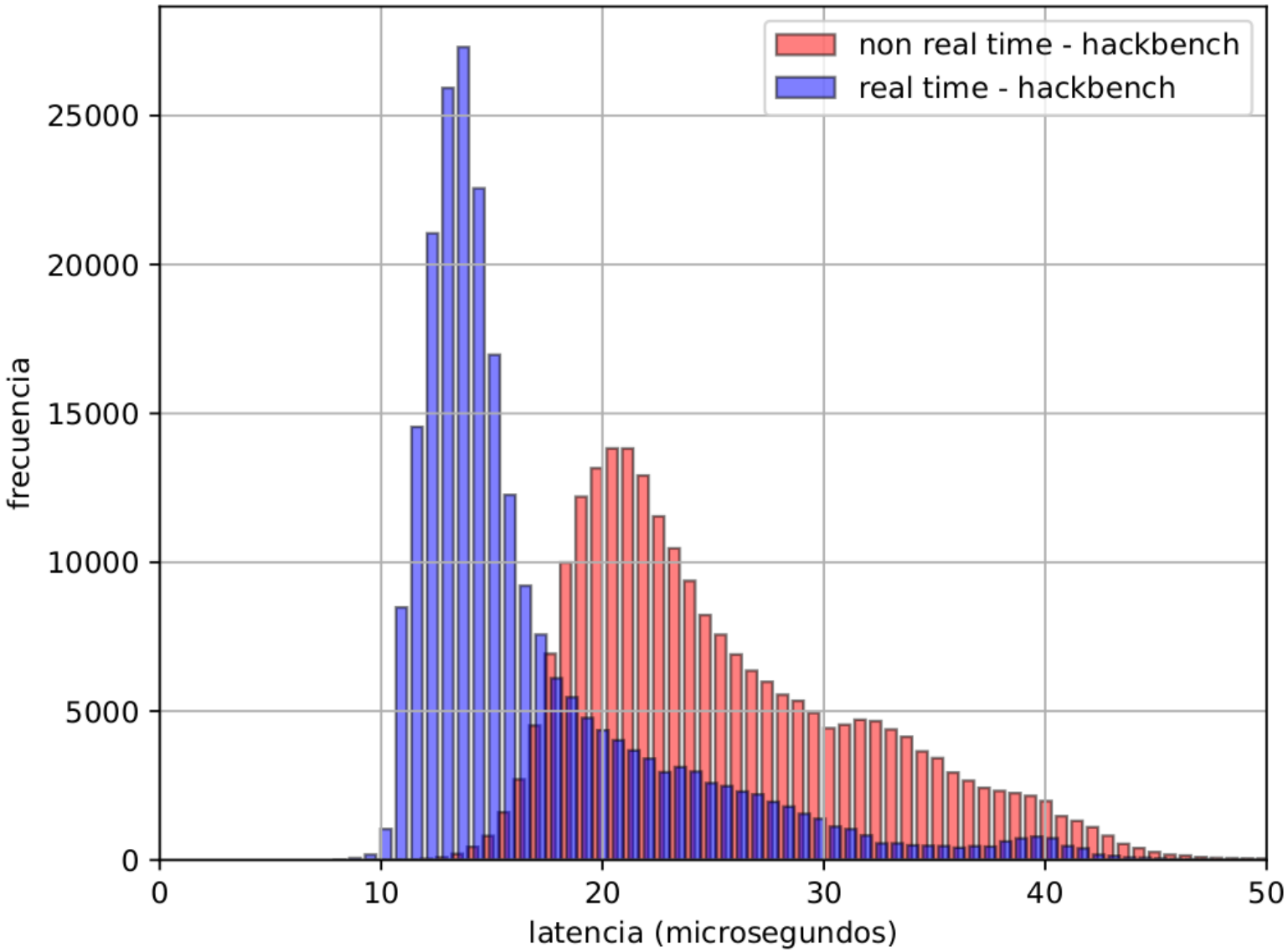
Histograma


non real time - idle
non real time - hackbench
frecuencia
latencia (microsegundos)
0
20000
40000
60000
80000
100000
120000
140000
0
10
20
30
40
50

Histograma


real time - idle
real time - hackbench
frecuencia
latencia (microsegundos)
0
10000
20000
30000
40000
50000
60000
70000
80000
0
10
20
30
40
50

Histograma


non real time - hackbench
real time - hackbench
frecuencia
0
5000
10000
15000
20000
25000
0
10
20
30
40
50
latencia (microsegundos)

RTLinux en Ubuntu y RaspBerryPi

- Ubuntu
  - https://chenna.me/blog/2020/02/23/how-to-setup-preempt-rt-on-ubuntu-18-04/

- RaspBerryPi
  - https://www.get-edi.io/Real-Time-Linux-on-the-Raspberry-Pi/
  - https://metebalci.com/blog/latency-of-raspberry-pi-4-on-standard-and-real-time-linux-4.19-kernel/
  - https://lemariva.com/blog/2019/09/raspberry-pi-4b-preempt-rt-kernel-419y-performance-test

Bibliografía

- CFS Scheduler
  - https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt
- A Comparison of Scheduling Latency in Linux, PREEMPTRT, and LITMUS
  - https://people.mpi-sws.org/~bbb/papers/pdf/ospert13.pdf
- The Real Time Linux Project
  - https://wiki.linuxfoundation.org/realtime/start
- A realtime preemption overview
  - https://lwn.net/Articles/146861/

URJC Escuela Técnica Superior de Ingeniería de Telecomunicación