

# Linux en Tiempo Real

Carlos M. Cámara Mora

Abril 2004

<i>Linux en Tiempo Real</i>	1
-----------------------------	---

## Índice

Índice	1
Índice de figuras	2
1. Introducción	3
2. Sistemas Operativos en tiempo real	4
3. Planificación de procesos	5
4. Descripción del kernel de Linux	7
5. Linux en tiempo real	8
6. Descripción de RTLinux	9
7. Tareas de tiempo real	11
8. Planificación en RTLinux	12
9. Precisión temporal	12
10. Comunicación entre procesos	13
11. Experiencia práctica con RTLinux	14
12. Conclusiones	16
Referencias	17

## Índice de figuras

1.	Esquema del sistema . . . . .	3
2.	Planificación de tasa monótona . . . . .	7
3.	Arquitectura de RTLinux . . . . .	10
4.	Salida de rlinux para start y stop . . . . .	15
5.	Salida de kiwi . . . . .	15

Copyright-Copyleft (C) 2004 Carlos M. Cámara. Este documento está bajo la licencia Creative Commons Attribution - NonCommercial - ShareAlike - v.1.0: Se permite la copia, distribución, uso y realización de la obra, siempre y cuando se reconozca la autoría y no se use la obra con fines comerciales –a no ser que se obtenga permiso expreso del autor. El autor permite distribuir obras derivadas de esta sólo si mantienen la misma licencia que esta obra. Esta nota no es la licencia completa de la obra sino una traducción de la nota orientativa de la licencia original completa (jurídicamente válida), que puede encontrarse en <http://creativecommons.org/licenses/by-nc-sa/1.0/legalcode>.

## 1. Introducción

En el desarrollo de los distintos sistemas que nos hacen la vida más fácil existen multitud de diseños y representaciones que nos ayudan a describir los objetivos de nuestros sistemas. Una de las maneras más adecuadas de realizar un sistema se basa en la introducción de controladores que nos modifiquen la salida de nuestros sistemas y la vayan acercando a lo que queremos obtener. Estos controladores por lo general no actuarán de forma inmediata y tendrán un cierto retardo, que en muchos casos nos será inadmisibile. Así con la idea de obtener salidas dentro de un margen de tiempo adecuado surgen los sistemas en tiempo real. Estos sistemas se basan en la idea de que no sólo es preciso obtener una salida correcta sino que además es preciso obtenerla dentro de unos márgenes de tiempo adecuados porque sino nos será completamente inútil. Podemos destacar varios ejemplos de aplicaciones que precisan de sistemas en tiempo real, como por ejemplo la monitorización de enfermos en un hospital, el control de una central nuclear, la reproducción de video, etc.

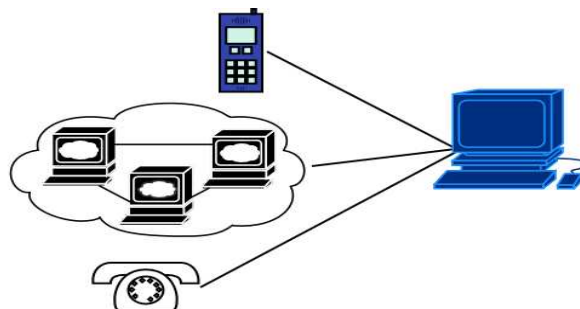


Figura 1: Esquema del sistema

Para automatizar toda esta serie de tareas de control y hacerlas más sencillas en la mayoría

de los casos se opta por implementar las soluciones mediante un ordenador que ejecute un programa que nos sirva de controlador. Esto facilita enormemente el uso posterior del controlador por parte del usuario y facilita la integración del controlador con las diferentes tecnologías actuales como internet, UMTS, etc; con esto conseguimos un sistema como el que aparece en la figura 1, en el que podemos interactuar a través de distintos medios. Esto nos plantea la necesidad de modelar el sistema de forma específica para un ordenador y nos permite distinguir los sistemas en función de las tareas que ejecuten. Así podemos clasificar estos sistemas de tiempo real según las necesidades de proceso que tengan:

**Sistemas de tiempo real monotarea:** Son aquellos sistemas en los que sólo tendremos que ejecutar un proceso que además será el que funcione como controlador. Son sistemas triviales y su dimensionamiento se reduce a determinar las especificaciones de frecuencia de repetición del proceso.

**Sistemas de tiempo real multitarea:** En estos sistemas concurren múltiples procesos que deben de tenerse en cuenta a la hora de cumplir las especificaciones de tiempo. La mayoría de estos procesos nos serán necesarios en el control de las aplicaciones pero otros no y debemos procurar que no afecten al rendimiento de nuestro sistema de control.

Así para manejar los sistemas en tiempo real multitarea podemos optar por aprovechar las ventajas de un sistema operativo que nos facilite la gestión de los recursos. El problema es que normalmente no podremos hacer predicciones sobre tiempos de proceso dentro de los mismos por lo que tendremos que optar por sistemas operativos específicos que sí sean predecibles.

## 2. Sistemas Operativos en tiempo real

Hemos visto como la complejidad de los sistemas nos obligan a la implementación de soluciones que pasen por el uso de un ordenador y un sistema operativo. Existen varios sistemas operativos que intentan dar soluciones para sistemas de tiempo real como INTEGRITI, THREADX, QNX, VXWORKS, LINUX EN TIEMPO REAL, etc; pero todos ellos tienen que tener una serie de características y deberán cumplir unos requisitos mínimos para poder realizar bien su misión. Así para asentar las bases de nuestro sistema podemos considerar la definición de un sistema en tiempo real de Donald Gillies:

*Un sistema en tiempo real es aquel en el que para que las operaciones computacionales sean correctas no depende sólo de que la lógica e implementación de los programas*

*computacionales sea correcto, sino también en el tiempo en el que dicha operación entregó su resultado. Si las restricciones de tiempo no son respetadas el sistema se dice que ha fallado.*

Por tanto no sólo la respuesta del sistema nos determinará su calidad sino que también debemos tener en cuenta que cumpla las especificaciones temporales adecuadas. Para ello podemos establecer una serie de características que deben cumplir los sistemas en tiempo real:

**Determinismo:** Es la característica clave de todo sistema en tiempo real y es que debemos conocer en todo momento cómo nos va a responder el sistema.

**Tiempo de respuesta:** Se trata de la cantidad de tiempo que tardamos en dar una respuesta al proceso una vez que éste ha sido atendido y en el tiempo que tardamos en atender al proceso.

**Control de los procesos sobre el sistema:** Aunque esta característica parezca contraproducente, no es así puesto que los procesos deberán asignarse una prioridad en función de su tiempo de respuesta e importancia de su respuesta. Existirán una serie de tipos de proceso que ayudarán a que todo funcione sin problemas.

**Confiabilidad:** No sólo el sistema debe de estar libre de fallos sino que además su respuesta no puede degradarse y deberá ser capaz de recuperarse después de un fallo grave.

**Operación a prueba de fallos:** Si ocurre algún fallo el sistema además deberá ser capaz de mantener todos sus datos y capacidades anteriores al fallo.

De entre todas estas características nos centraremos en el determinismo y el tiempo de respuesta ya que el hecho de que el sistema sea confiable y a prueba de fallos está más relacionado con la eficacia del sistema que con el tiempo de respuesta.

### 3. Planificación de procesos

En los sistemas multitarea comunes el tiempo de procesador se reparte entre todos los procesos con la ayuda del planificador. Normalmente la política de planificación no respeta en ningún caso las especificaciones de tiempo real para cumplir con el tiempo de respuesta y su asignación del procesador a las tareas se basa en asignaciones cíclicas por orden de llegada. De forma que los procesos que llegan antes a la lista de procesos listos para ejecutarse tienen prioridades altas. Además, en este tipo de políticas, las rutinas de interrupción poseen las prioridades más

altas. Ésto nos deja al sistema en un estado de indeterminismo absoluto puesto que es imposible conocer de antemano cuántas interrupciones va a haber en un sistema o en qué momento van a llegar. Los planificadores de tiempo real se basan en premisas que sí confieren determinismo al sistema aunque en algunos casos de cara al usuario da la impresión de que lo ralentizan. Las políticas de planificación más importantes de tiempo real son dos:

1. Planificación cíclica: En ésta política se sitúan todas las tareas de tiempo real de forma que siempre ejecutamos la misma secuencia de procesos. Es la forma más predecible de planificación pero también es la más complicada de implementar puesto que requiere un diseño específico para cada uno de los casos posibles. Actualmente apenas es usada en algunas industrias y no la trataremos aquí.
2. Planificación por prioridades: Aquí le asignamos prioridades a los procesos y los vamos ejecutando por orden de prioridad. Esta política es la que se usa actualmente en la mayoría de los sistemas operativos modernos, aunque de forma que la asignación de prioridades no se hace orientada a cumplir los requisitos de tiempo real sino a mejorar la velocidad de entrada y salida de cara al usuario.

Los planificadores por prioridades se dividen en dos grandes grupos: *estáticos* y *dinámicos*. En los primeros la asignación de prioridades se realiza en la fase de diseño mientras que en los planificadores dinámicos se van decidiendo las prioridades durante la ejecución. Debemos tener en cuenta también que la planificación puede ser expulsiva, es decir, que podremos expulsar procesos que se estén ejecutando si llega un proceso de mayor prioridad.

Dentro del gran número de políticas de planificación existentes nosotros nos centraremos en una política estática, *Rate Monotonic* y otra dinámica *Earliest Deadline First*:

**Rate Monotonic (Tasa monótona):** En esta realización a cada tarea se le asigna una prioridad inversamente proporcional a su periodo o plazo de finalización. Es una realización expulsiva y además el plazo máximo de ejecución ha de ser igual al periodo de cada tarea. [Figura 2](#)

**Earliest Deadline First (Plazo de finalización más corto en primer lugar):** Esta política de planificación dinámica también es expulsiva y en ella el plazo final de ejecución de una tarea es el punto del tiempo donde debe estar terminada. Este plazo de finalización es calculado al crear la tarea. El planificador elige como tarea a ejecutar aquella que tenga el plazo de finalización más corto. Así, una tarea con un mayor plazo de finalización es expulsada del procesador por otra de menor plazo. Esta política minimiza la tardanza máxima de cualquier conjunto de tareas con respecto a las demás políticas de planificación posibles. Fue desarrollada por Ismael Ripoll.

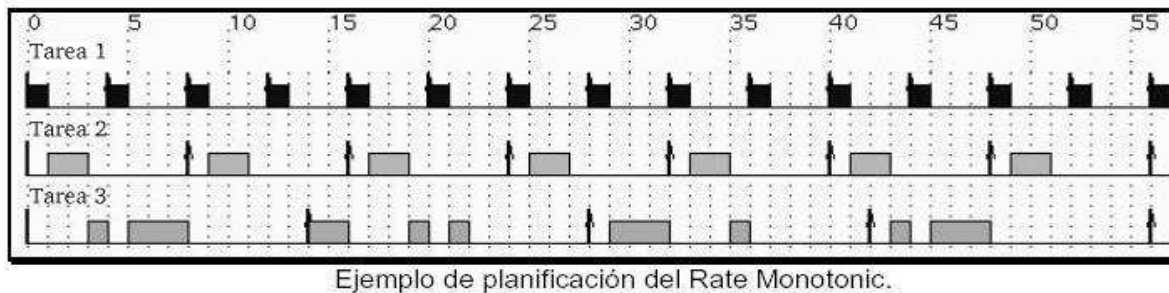


Figura 2: Planificación de tasa monótona

## 4. Descripción del kernel de Linux

Un sistema Linux es un sistema operativo diseñado a partir de un núcleo basado en un sistema Minix y que gracias a la colaboración de cientos de personas alrededor del mundo y al proyecto GNU ha crecido hasta ser capaz de competir con las principales empresas de software del mercado. Esto hace a Linux una opción a tener en cuenta dentro del mundo de los sistemas operativos, y tanto es así que en el intento de aumentar las prestaciones de este sistema operativo muchos desarrolladores han realizado modificaciones en el núcleo para poder tener características de un sistema en tiempo real sin renunciar a las bondades de Linux. Para entender estas modificaciones observemos algunas de las características del núcleo de Linux.

- ⇒ Permite multitarea, multiprocesador y multiusuario.
- ⇒ Kernel modular
- ⇒ Capaz de funcionar en varias plataformas.
- ⇒ Posee protección de memoria entre procesos.
- ⇒ Carga de ejecutables por demanda.
- ⇒ Rapidez y economía del uso de la memoria gracias a su política de copia en escritura para la compartición de páginas de memoria entre procesos.
- ⇒ Memoria virtual con paginación a disco.
- ⇒ La memoria se gestiona como un recurso unificado para los programas de usuario y para el caché de disco.

- ⇒ Se realizan volcados de estado (core dumps) para posibilitar los análisis post-mortem.
- ⇒ Se adapta al estándar POSIX definido por el IEEE.
- ⇒ Buen soporte a los distintos dispositivos.
- ⇒ Sistema de archivos que permite gestionar tanto particiones linux como de otros sistemas.
- ⇒ Soporte a la capa TCP/IP y a otros muchos protocolos de red.
- ⇒ Soporte técnico formado por una gran comunidad de usuarios.
- ⇒ Todo el código fuente está disponible, incluyendo el núcleo completo y todos los drivers, las herramientas de desarrollo y la mayoría de los programas de usuario.

Estas características, y en concreto las que están en rojo nos ayudan a despreocuparnos de la confiabilidad del sistema y tan sólo tener que centrarnos en las características temporales del sistema en tiempo real. Tanto es así que será de lo único que nos tendremos que preocupar para tener un sistema Linux en tiempo real.

Por situarnos un poco numéricamente en el peor de los casos un manejador de interrupciones para un sistema Linux estándar corriendo sobre un procesador con arquitectura x86 puede tardar en ser atendido unos  $600 \mu s$  y llegamos hasta  $20 ms$  si tenemos una tarea periódica, mientras que en un sistema con RTLinux en la misma máquina tendremos un retardo de tan sólo  $15 \mu s$  para el manejador y  $35 \mu s$  para la tarea periódica. Además hay que tener en cuenta que estos resultados están limitados por el hardware y que a medida que éste mejore mejorarán también nuestros resultados.

## 5. Linux en tiempo real

El primer intento de adaptar un sistema linux a las especificaciones de tiempo real se desarrolló en el departamenteo de informática en el Instituto de Minería y Tecnología de Nuevo México y les correspondió a Victor Yodaiken y Michael Barabanov como trabajo de este último para completar su Master of Computer Science. Esta aproximación trajo consigo otras propuestas de diversas empresas y de la comunidad de software libre que brevemente describiremos. Así basándonos en el artículo [The Real-time Linux Software Quick Reference Guide](#) aparecido en [7] el 19 de Junio del 2003 podemos establecer una clasificación de las distintas soluciones para tiempo real en linux:

	Real-time Linux Distribuciones Comerciales	Implementaciones de Código abierto Real-Time Linux
Sistemas basados en Núcleos modificados	TimeSys Reservations uLinux Real-Time Solutions for Linux (Montavista) RedHawk REDICE-Linux	Qlinux RED-Linux KURT
Sistemas basados en microkernels adicionales	RTLinuxPro  BlueCat RT	RTLinux RTAI ADEOS ART Linux Linux/RK

Como podemos observar en la tabla, además de poder clasificar los sistemas como comerciales o de código abierto, los podemos agrupar según la forma que tienen de implementar el tiempo real. Así tendremos sistemas en los que se ha realizado una revisión completa del núcleo optimizándolo para asignar a las tareas más urgentes las prioridades más altas, que son los que hemos denominados "Sistemas basados en Núcleos modificados"; por otro lado tendremos también otro tipo de modificación que fue la que propusieron Yodaiken y Barabanov en la que situaban un núcleo muy pequeño entre el núcleo de Linux y el Hardware para que diera soporte al tiempo real. De todos estos sistemas los más populares y que mejor documentados están son 4: RTLinux, RTAI, KURT y TimeSys©. De hecho la mayoría de los sistemas que sitúan un micronúcleo entre el S.O. y el hardware están basados en RTLinux (Entre ellos RTAI). Nosotros centraremos nuestro estudio en RTLinux ya que al implementarse mediante módulos cargados en el kernel funciona como una tarea más del núcleo de Linux y se integra perfectamente en el sistema.

## 6. Descripción de RTLinux

Como se ha comentado anteriormente RTLinux fue desarrollado por Michael Barabanov y Victor Yodaiken. Su uso se ha extendido mucho en las aplicaciones que requieren tiempo real y ha sido usado por importantes empresas como la NASA o los estudios de Hollywood, tanto en la realización de empresas que requirieran de Tiempo Real, como para las simulaciones de distintos proyectos.

Las características fundamentales de este sistema son:

- ⇒ Tiene un planificador expulsivo con prioridades fijas para la ejecución de las tareas de tiempo real.
- ⇒ Las tareas pueden ser periódicas o activadas por una interrupción.
- ⇒ Incorpora tuberías FIFO para la comunicación con los procesos que no son de tiempo real.
- ⇒ Las tareas de tiempo real se ejecutan en la CPU en modo supervisor. Convierte el núcleo de linux en una tarea más de segundo plano.
- ⇒ Protección absoluta del núcleo de RTLinux sobre cualquier tarea de Linux (en modo usuario o supervisor).
- ⇒ RTLinux se carga en el sistema como un módulo cualquiera del núcleo de Linux.

A la hora de diseñar el sistema, Yodaiken y Barbanov optaron por diseñar un gestor de interrupciones por software que se situara entre el kernel de linux y el sistema operativo, tal y como se muestra en la Figura 3

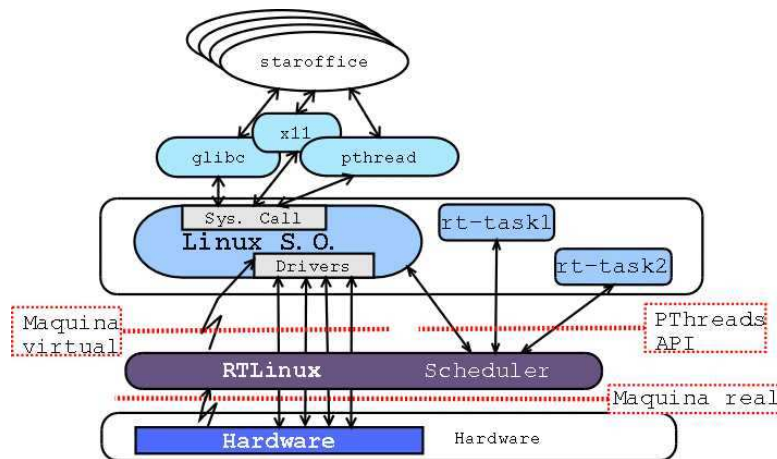


Figura 3: Arquitectura de RTLinux

Así, cuando llega una interrupción Hardware al núcleo, RTLinux la intercepta y si no existe una tarea de tiempo real la pasa al núcleo de Linux. De lo contrario espera a que la tarea de tiempo real termine para pasarla al núcleo. Con esta forma de implementar el tiempo real lo que hicieron Yodaiken y Barabanov fue dotar al sistema de un núcleo de tiempo real que se situaba en el espacio del kernel de Linux y que actuaba de intermediario entre el Hardware y Linux.

Desde este nuevo esquema, el núcleo de Linux pasa a ser una tarea más a ejecutar por el núcleo de RTLinux.

Así no tendremos un núcleo que exclusivamente trabaje en tiempo real o de forma normal, sino que podremos elegir las características de nuestro sistema en tiempo de ejecución. Como ejemplo muy cercano, este trabajo se ha realizado en un AMD 1600XP+ con un núcleo 2.4.4 que estaba adaptado para trabajar en tiempo real pero con el módulo de RTLinux sin cargar y que por tanto no afectaba en absoluto al sistema.

El trabajo en el espacio del núcleo nos aportará la ventaja de que, dado que el código del núcleo no se pagina, no sufriremos retrasos debido a problemas con la memoria (errores muy comunes en los 486).

Las características temporales para el trabajo en tiempo real las obtenemos sacrificando algunas otras cualidades del núcleo. Entre otras cosas nuestro núcleo de tiempo real no pedirá memoria ni realizará ningún tipo de sincronización con el núcleo de Linux, salvo en contadas ocasiones. Estas ocasiones son obligadas ya que no es posible ni beneficioso que el núcleo en tiempo real no se comunique con el núcleo de Linux. El mecanismo de comunicación entre procesos será mediante colas FIFO. En estas comunicaciones la sincronización nunca bloqueará el núcleo de RTLinux, ni lo hará esperar para escribir o leer de una cola.

La clave a la hora de diseñar un sistema en tiempo real con RTLinux es diseñar las aplicaciones de forma que sólo usemos el núcleo en tiempo real cuando nos sea estrictamente necesario y aprovechemos al máximo el núcleo sin tiempo real para las tareas no críticas. Así por una parte usaremos las funciones de un sistema en tiempo real asegurándonos de la realización correcta de las tareas críticas y por otra parte utilizamos un sistema Linux para la ejecución de tareas no críticas como el almacenamiento en disco, inicialización de dispositivos, comunicación con el usuario, etc. Con esto esperamos cumplir todas las especificaciones temporales de un sistema en tiempo real con la confiabilidad que nos aporta un sistema Linux.

En resumidas cuentas podemos considerar a RTLinux como un micronúcleo que se encarga de la gestión de interrupciones y de la planificación de prioridades de tareas y dispositivos críticos.

## 7. Tareas de tiempo real

Para introducir nuestras tareas de tiempo real en el espacio del núcleo aprovecharemos la característica modular del kernel de Linux y cargaremos nuestras tareas de tiempo real en forma de módulos de forma que se ejecutarán todas en el espacio del núcleo. De hecho como ya hemos mencionado el propio RTLinux se carga como un módulo. Como consecuencia de esta arquitectura debemos realizar un gran esfuerzo de programación ya que si cometemos algún error

podemos bloquear todo el sistema. No obstante, desde un punto de vista práctico, un sistema en tiempo real en general tiene que lidiar con periféricos y en ellos es vital no cometer errores en la programación puesto que podemos estropearlos, por tanto este inconveniente no es tan grave.

Otro de los peligros de la arquitectura de RTLinux está en un mal dimensionamiento de las tareas críticas. Uno de los errores más comunes consiste en que siempre halla una tarea crítica ejecutándose ya que el núcleo de Linux no conseguirá nunca el procesador y nos dará la impresión de que el sistema está bloqueado. No hay que olvidar que el núcleo de Linux es como una tarea de mínima prioridad dentro de un pequeño sistema RTLinux.

Además debemos tener siempre en cuenta que una tarea de Linux nunca podrá hacer esperar a una tarea del núcleo de RTLinux ni podrá bloquear las interrupciones de éste. Así el núcleo de RTLinux siempre estará disponible para atender a cualquier tarea que le llegue independientemente del estado del núcleo de Linux.

## 8. Planificación en RTLinux

Normalmente no se puede elegir la planificación en un sistema operativo, no obstante, para aumentar la flexibilidad de RTLinux, Yodaiken y Barabanov optaron por incluir varias políticas de planificación para que los administradores del sistema eligieran la más adecuada a sus requerimientos de tiempo. La primera fue desarrollada por Victor Yodaiken y se trata de una planificación en la que una tarea lista para ejecución puede desalojar del procesador a otra de menor prioridad. En esta política asignamos una prioridad fija a cada tarea y se ejecutarán según estén listas por orden de prioridad. Además podremos también implementar un gestor de interrupciones que se despierte cuando sea necesario. Con esta política podemos usar el algoritmo de planificación de tasa creciente (Rate Monotonic) y gestionar así todas las prioridades. El otro planificador que podemos usar fue implementado por Ismael Ripoll y usa el algoritmo de Earliest Deadline First. Este algoritmo asigna las prioridades de forma dinámica de forma que se le da la máxima prioridad a la tarea cuyo plazo de finalización esté más cercano.

## 9. Precisión temporal

Uno de los principales problemas de un sistema en tiempo real es la falta de precisión en la respuesta temporal del sistema. Esta imprecisión provoca pequeñas desviaciones (jitter) a la hora de ejecutar la planificación que pueden afectar a la eficacia del planificador. Esta falta de precisión o jitter está provocada por el temporizador de baja resolución que utilizan los sistemas

operativos y que está gobernado por las interrupciones periódicas de reloj. Linux tampoco es una excepción a esto y normalmente programa estas interrupciones a una frecuencia de 100 Hz y por tanto sólo podemos obtener unos 10 ms de precisión sin afectar al rendimiento de la máquina. Para saltar esta dificultad RTLinux utiliza el temporizador Intel 8354 presente en la mayoría de los Pc compatibles. Así con este dispositivo funcionando en modo terminal-on-count podemos obtener una precisión de hasta 1  $\mu$ s. Esto hace además que la carga derivada de la interrupción temporal sea mínima.

## 10. Comunicación entre procesos

Tal y como hemos comentado previamente en [Descripción de RTLinux](#), la comunicación entre los procesos del núcleo de Linux y RTLinux debe ser muy cuidada y reducida en la medida de lo posible, no obstante, hay que proporcionar algún tipo de comunicación entre ellos para no imponer demasiadas restricciones al sistema. Ahora bien, tenemos que tener en cuenta que el núcleo de Linux será interrumpido en cualquier momento por el núcleo de tiempo real y por tanto no podremos ejecutar rutinas de Linux desde RTLinux de forma segura.

Para la intercomunicación de procesos usaremos simples colas FIFO a las que llamaremos RT-FIFO para distinguirlas de las colas FIFO del API de Linux. El número de estas colas está limitado a 64, no obstante lo podremos cambiar a la hora de recompilar el kernel y aumentar hasta 248 con una simple modificación del código fuente.

La API de RTLinux incluye funciones para la creación, lectura y escritura de estas colas además de asegurar que la lectura y escritura sean operaciones atómicas, evitando así el problema de inversión de prioridad.

Para el kernel de Linux no obstante estas colas se verán como dispositivos de carácter, esta forma de implementar las colas le permite a los usuarios de Linux una comunicación total con el núcleo de RTLinux.

En la última versión disponible de RTLinux [10] se implementa también la comunicación mediante cola de mensajes que nos da un mayor control en la comunicación permitiéndonos definir el tamaño de los mensajes. No obstante la API de esta nueva característica aun no se encuentra lo suficientemente bien documentada.

## 11. Experiencia práctica con RTLinux

Para probar las características del sistema y poder comprobar la efectividad del mismo se ha optado por instalar RTLinux en un sistema Linux basado en la distribución Debian Potato y en el que se ha modificado el núcleo convenientemente para la instalación. Algunas observaciones que me llamaron la atención sobre su instalación son:

- ⇒ La distribución Debian incorpora en sus repositorios de paquetes un paquete concreto que contiene la última distribución de RTLinux [10].
- ⇒ Para instalar con éxito RTLinux es preciso parchear el núcleo sobre el que lo vayamos a instalar.
- ⇒ La versión RTLinux [10] tan sólo puede trabajar con dos núcleos de Linux, 2.2.19 y 2.4.4.
- ⇒ Las tareas de tiempo real presentan incompatibilidades con las funciones APM y por seguridad deben desactivarse al compilar el nuevo núcleo.

La versión del kernel por la que opté fue la 2.4.4 y la versión de RTLinux fue la 3.1 que bajé desde [10] y tanto el parcheo del núcleo como su compilación no presentaron ningún tipo de inconveniente. De hecho siguiendo la documentación que RTLinux incorpora [4] no hubo ningún tipo de problema.

A la hora de introducir o sacar los módulos de tiempo real y las aplicaciones que realizamos, la distribución trae una pequeña aplicación que se encarga de insertar en el kernel los módulos de RTLinux y de la aplicación. Su sintaxis es la siguiente:

```
$ rtplinux start—stop—status [modulo(s)]
```

Su salida se muestra en la figura 4

A la hora de programar en RTLinux hay que tener en cuenta las consideraciones que hicimos en [Tareas de tiempo real](#) sobre la necesidad de no cometer errores en la programación y además considerar que realmente estamos programando módulos del núcleo y por tanto nos tenemos que atener a las condiciones de programación en el espacio del núcleo. Podemos encontrar una excelente guía de iniciación en [11]. Con esto, tenemos que lidiar con la ausencia de funciones como *printf* que se sustituye por *printk*. Para paliar esta falta, en la API de RTLinux viene implementada la función *rtl\_printf* que posee las mismas características que su homóloga de C con la ausencia de la impresión de tipos float.

Uno de los problemas observados en esta implementación es que presenta una pequeña incompatibilidad con el emulador de terminal y después de haber trabajado con RTLinux no

```

OSK-terminal-emulador
-----
Schese: (-) not loaded, (+) loaded
(+) maquina
(+) mbuf
(+) rtl_fifo
(+) rtl
(+) rtl_posixio
(+) rtl_sched
(+) rtl_time

MORDIB:/home/carcas/Arquitectura de Computadores/Trabajo/programasRT/maquina# rtlunix stop maquina
/usr/bin/rtl-config: line 173: cd: /home/carcas/Arquitectura: No existe el fichero o el directorio
/usr/bin/rtl-config: line 173: cd: /home/carcas/Arquitectura: No existe el fichero o el directorio

Schese: (-) not loaded, (+) loaded
(-) maquina
(-) mbuf
(-) rtl_fifo
(-) rtl
(-) rtl_posixio
(-) rtl_sched
(-) rtl_time

MORDIB:/home/carcas/Arquitectura de Computadores/Trabajo/programasRT/maquina#

```

Figura 4: Salida de rlinux para start y stop

es posible cerrar su ventana excepto matando el proceso. Este problema lo he detectado en los entornos de escritorio KDE 2.2 y 3.1 y en Gnome 2.4.

Para monitorizar la planificación de procesos existen algunas aplicaciones como [kiwi](#) que nos muestran de forma gráfica los procesos que están ejecutándose en el sistema y cómo están usando el procesador (Figura 5)

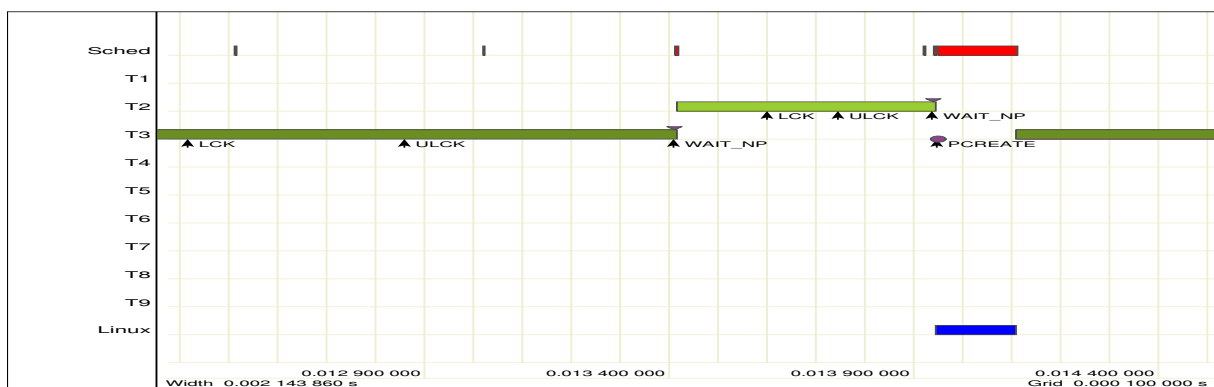


Figura 5: Salida de kiwi

## 12. Conclusiones

Una vez más el intento de la comunidad Linux por aportar soluciones fiables al mundo de la computación ha dado sus frutos en este sistema de tiempo real que basándose en la estabilidad de Linux consigue mejorar sus tiempos de respuesta y darnos una respuesta confiable para nuestras aplicaciones de tiempo real. La aproximación de Yodaiken y Barabanov fue crucial dentro del mundo de Linux en tiempo real aportando una solución en gran medida independiente del núcleo y que no lo modifica para adaptarlo a los requerimientos temporales, sino que colabora con él y se aprovecha de sus características. Tan fiable resulta ser este sistema que Yodaiken liberó el código y a partir de ahí comenzó el desarrollo de un sistema RTLinux comercial con el que da servicios a través de su empresa [FMSLabs](#).

El aporte de RTLinux a la comunidad Linux no se acaba en el desarrollo de este sistema en tiempo real competitivo a nivel comercial, sino que ha dado pie a que se incorpore cada vez más dentro del núcleo soporte para tiempo real, que aunque no está aun lo suficientemente testado, ya en el núcleo 2.6 podemos probar.

## Referencias

- [1] [Charla del Campus Party'98](#)  
Autor: *Juan José Amor Iglesias*
- [2] [RTLinux Manifiesto](#)  
Autor: *Victor Yodaiken*
- [3] [Tutorial de RTLinux](#)  
Autor: *Ismael Ripoll*
- [4] [Instalación de RTLinux](#)  
*Manual de instalación que acompaña a la distribución de RTLinux.*
- [5] [Sistemas de tiempo real](#)  
*Página sobre sistemas RTLinux Webmaster: Ismael Ripoll*
- [6] [Planificación de sistemas en tiempo real](#)  
*Introducción a la planificación de sistemas en tiempo real*
- [7] [Linux Devices](#)  
*Publicación digital dedicada a los sistemas linux y su relación con los distintos dispositivos.*
- [8] [Portal de Linux en tiempo real de la Universidad de Valencia](#)  
Webmaster: *Ismael Ripoll*
- [9] [www.fsmlabs.com](#)  
*Página de la empresa de Victor Yodaiken*
- [10] [RTLinux 3.1](#)  
*Repositorio de RTLinux de la Universidad de Valencia*
- [11] [Linux Kernel Programming Guide](#), Autor: *Peter Jay Salzman*, Versión: *ver 2.4.0*