FIB Facultat d'Informàtica de Barcelona
UNIVERSITAT POLITÈCNICA DE CATALUNYA

CONCEPTES AVANÇATS DE SISTEMES OPERATIUS
Departament d'Arquitectura de Computadors

REALTIME LINUX

(Seminaris de CASO)

Autors
Raúl Mingorance Fernández.
Jesús Tallón Pérez.

Idea original de RTLinux

- La idea original era hacer linux capaz de manejar procesos que requieren procesamiento en tiempo real.
- Linux ya tiene una cierta habilidad "Real Time", pero es un "Soft Real Time" no útil p.e en procesos industriales.
- Desarrollar un SO en tiempo real es muy costoso y difícil de implementar, igualmente modificar un SO ya existente era una tarea complicada y el resultado normalmente era un SO lleno de bugs.
- RTLinux adopta una estrategia intermedia, fue desarrollado para ser realmente en tiempo real, pero es un RTOS sumamente pequeño con las funcionalidades más elementales y este SO ejecuta linux como su proceso de mas baja prioridad y se comunican entre si por memoria compartida y pipe en tiempo real.

Que aporta?

- La ventaja de adoptar esta estrategia es que la parte de tiempo real es muy pequeña y fácil de mantener y la parte de Linux nos permite disponer de un gran número de aplicaciones desarrolladas.
- Las modificaciones de Linux son mínimas , básicamente fue cambiar una macro del kernel, que linux usa para bloquear interrupciones, así todas las interrupciones son realmente controladas por la parte en tiempo real.
- Los programas que requieran tiempo real, se separan en una pequeña parte que es administrada por el RTOS y otra parte que procesa los datos que corre en linux normal y se comunican mediante una pipe en tiempo real que hay entre los dos. Si no hay ningún proceso de tiempo real pendiente, se ejecuta linux y este administra sus procesos como siempre lo hace.

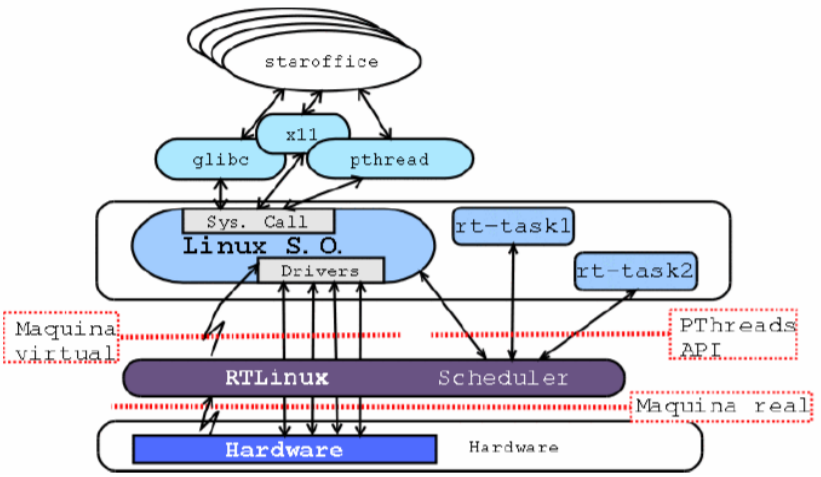
Características

- Sistema Operativo de tiempo real estricto.
- Extensiones para entorno multiprocesador SMP (x86).
- API próximo al de POSIX. Planificador exclusivo por prioridades fijas, señales, sistema de archivos POSIX, semáforos y variables de condición.
- Depuración de código mediante GDB (GNU Debuger).
- Soporte para arquitecturas x86 y PPC.
- Acceso directo al Hardware (Puerto de interrupciones).
- Comunicación con procesos Linux mediante memoria compartida y Pipes.
- Estructura modular para crear sistemas pequeños.
- Eficiente gestión de tiempos. En el peor caso se dispone de una resolución próxima al microsegundo (para un i486).
- Facilidades para incorporar nuevos componentes: Relojes, Dispos. E/S y planificadores.

Arquitectura de RTLinux

- No añade nuevas llamadas al sistema ni modifica ninguna de las existentes.
- Se sitúa entre el hardware y el SO, creando una maquina virtual para que Linux pueda seguir funcionando.
- El scheduler utilizado es Rate Monotonic, aunque han aparecido implementaciones de EDF para ser ejecutadas sobre RTLinux.
- Toma el control de todas las interrupciones implementando un gestor de interrupciones por software.
- Las tareas en tiempo real se ejecutan utilizando el Run Time Suport de RTLinux (RTS)

Arquitectura de RTLinux



Tareas de Tiempo Real (rt-task)

- Comparten el mismo espacio de memoria que el núcleo, por lo que pueden acceder a variables y funciones de este, aunque se podrían producir ínter bloqueos o condiciones de carrera.
- No puede hacer uso de las llamadas al sistema de Linux.
- Se ejecutan en modo privilegiado.
- Las páginas de memoria de datos y programa no pueden realizar intercambio con disco (Swap Out).
- A efectos prácticos el núcleo de Linux se puede considerar como una rt-task pero que es planificada con la mínima prioridad.

Módulos del Núcleo

- Para poner en ejecución una rt-task se tiene que utilizar el sistema de módulos cargables de Linux.
- Los módulos son "trozos de sistema operativo" que se pueden insertar y extraer en tiempo de ejecución.
- Un módulo es un fichero objeto, obtenido a partir de un fuente en "C" compilado pero no enlazado (linkado).
- Existen varias guías que explican la programación de módulos, una de ellas es la de Ori Pomerantz: Linux Kernel Module Programming Guide

Módulos Cargables

- Un módulo es un fichero objeto que se puede "enlazar" y "des-enlazar" en el núcleo de Linux en tiempo de ejecución.
- Con los módulos se agiliza el desarrollo de software crítico ya que no es necesario crear un nuevo núcleo y reiniciar la máquina cada vez que hacemos una prueba.
- Un módulo es un programa "C" sin función *main(),* y que obligatoriamente ha de tener las funciones *init_module* y *cleanup_module*.
- Cada módulo se compila para una versión de núcleo concreta y normalmente sólo se puede cargar sobre esta versión.
- Los módulos se crean (como cualquier fichero objeto) con el compilador de "C" de GNU: gcc.

Utilidades para Trabajar con Módulos

- *insmod:* Instala en el núcleo un módulo.
- *rmmod:* Extrae del núcleo un módulo.
- *modinfo:* Muestra información sobre el módulo.
- *modprobe:* Automatiza/facilita la gestión de módulos.
- *depmod:* Determina las dependencias entre módulos.
- *lsmod:* Lista los módulos cargados.

Ejemplo de Módulo: "Hello World"

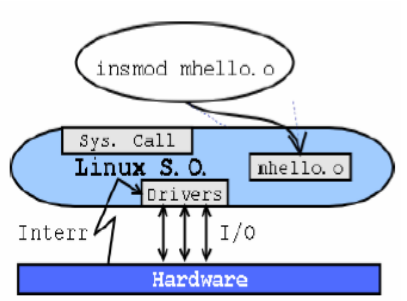
```
#include <linux/module.h>
#include <linux/kernel.h>

static int datos=0;
MODULE_AUTHOR(“Caso");
MODULE_DESCRIPTION("Modulo ejemplo Hello World");
MODULE_PARM(datos,"i");

int init_module(void) {
printk("Hello World, parámentro %d \n", datos);
return 0; }

void cleanup_module(void) { printK(“Bye world. \n”);}
```


Módulos: Compilación.

- Para compilar el mhello.c se tiene que utilizar la siguiente orden:
- gcc -c -O2 -fomit-frame-pointer -DMODULE -D__KERNEL__ mhello.c
- "-c": El compilador compila y ensambla el programa pero no lo enlaza, esto es, únicamente se generará el código objeto.
- "-O2 -fomit-frame-pointer": genera código optimizado.
- "-DMODULE -D__KERNEL__": define las macros MODULE y __KERNEL__, utilizadas por los ficheros de cabecera del núcleo para generar el código apropiado.

Módulos: Ejecución.

- Podemos insertar el módulo con la orden:
  insmod mhello.o
- Para extraer el módulo:
  rmmod mhello
- Para ver todos los mensajes enviados por el núcleo (llamadas a printk) utilizamos la orden: dmesg



Señales e Interrupciones

- En la programación clásica de UNIX, las señales se utilizan como un mecanismo de "comunicación" asíncrono. Si consideramos que un proceso es la abstracción de procesador, entonces las señales hacen el papel de las interrupciones.
- RTLinux hace un uso extensivo del concepto de señal/interrupción. Se utilizarán para comunicar rt-tasks entre sí, modificar el estado de ejecución de rt-tasks , atender interrupciones hardware y disparar eventos en el núcleo de Linux.
- Actualmente toda la gestión de las interrupciones se realiza mediante funciones no estándar. El objetivo de los desarrolladores de RTLinux consisten en gestionar la interrupciones como señales.

Señales

- Al igual que sucede con las interrupciones, un proceso puede establecer a priori la forma en la que se atienden las señales. Cada señal puede estar en uno de los siguientes estados:

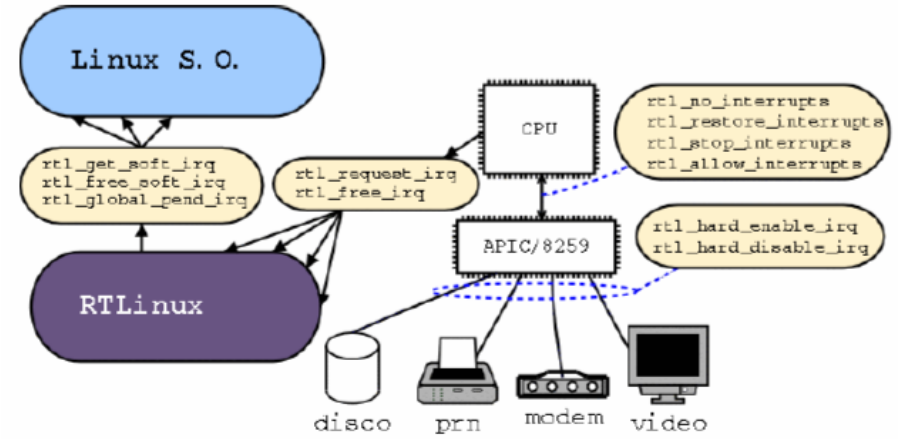
  Ignorada: El proceso no recibe estas señales.

  Capturada: Cada vez que llega una señal se ejecuta una función manejadora asociada.

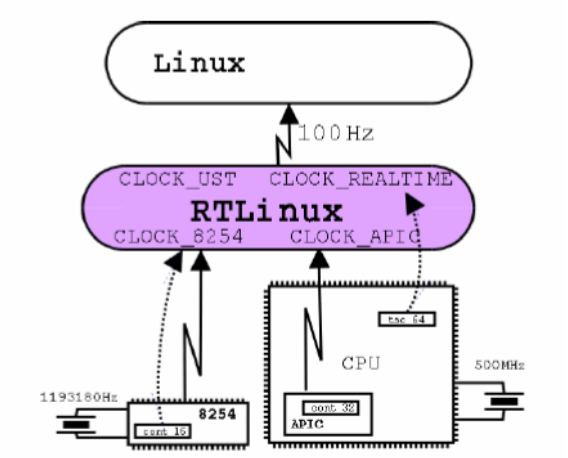
  Opción por defecto: Dependiendo de la señal puede ser suspender o terminar la ejecución del proceso, ignorarla o causar un volcado de memoria (core).

  Bloqueada: Las señales no se "entregan" al proceso hasta que se desbloqueen.

Interrupciones: Esquema global



Gestión del Tiempo: Visión Global



Gestión del Tiempo: Relojes

- *CLOCK_REALTIME:* Reloj del sistema. Puede sufrir ajustes para corregir la fecha.
- *CLOCK_MONOTONIC :* Igual que CLOCK_REALTIME pero no se realizan ajustes, por tanto su cuenta es creciente sin saltos bruscos. Útil para medir duraciones de eventos.
- *CLOCK_8254:* Obtenido a partir del chip 8254. Muy lento, no utilizar.
- *CLOCK_APIC:* Los micros que soportan SMP disponen de un *Advanced Programable Interrupt Controler* (APIC) integrado en el mismo procesador.

POSIX sobre RTLinux

- RTLinux tiene muchas de las funciones POSIX, aunque en algunas de ellas no sigue la semántica Standard.
- Permite la creación y destrucción de Threads.
- Permite utilizar un mecanismo similar a los semáforos binarios clásicos para acceder a regiones criticas. Aunque se cree que es mas eficiente crear secciones criticas atómicas mediante el bloqueo de las interrupciones.
- Fifos adaptadas a un uso en tiempo real (no tienen nada que ver con las fifos clásicas de UNIX). Su funcionamiento es similar al de un buffer circular donde cada operación de lectura elimina los datos leídos.
- Compartición de memoria entre procesos Linux y tareas de RTLinux (funciones mbuff).

Bibliografia

- Enlaces

  http://bernia.disca.upv.es/~iripoll/rt-linux/
  http://people.mech.kuleuven.ac.be/~bruyninc/rthowto/index.html
  http://bernia.disca.upv.es/rtportal/index.shtml
  http://www.linuxhq.com/guides/LKMPG/
  www.opengroup.org.

- Documentos

  Multithreaded Programming with Pthreads de Bil Lewis y Daniel J. Berg. Ed: Sun Microsystems.

  Documento ***"Getting started with RT-Linux"*** de Michael Barabanov, que acompaña los fuentes de RTLinux.

  El libro ***"Programming for the Real World POSIX 1004"***.