TEMA 8. SISTEMAS OPERATIVOS PARA TIEMPO REAL.

8.1. Introducción.

- Un sistema operativo para tiempo real es un sistema operativo capaz de garantizar los requisitos temporales de los procesos que controla.
- Los sistemas operativos convencionales no son apropiados para la realización de sistemas de tiempo real, debido a que
  - no tienen un comportamiento determinista y
  - no permiten garantizar los tiempos de respuesta.
- Un sistema operativo para tiempo real debe ofrecer las siguientes facilidades:
  - concurrencia: procesos ligeros (*threads*) con memoria compartida.
  - temporización: medida de tiempos y ejecución periódica.
  - planificación: prioridades fijas con desalojo, acceso a recursos con protocolos de herencia de prioridad.
  - dispositivos de E/S: acceso a recursos hardware e interrupciones.
- Existen varios sistemas operativos para tiempo real: QNX, Lynx, VxWorks, RT-Linux, KURT,... e incluso soluciones para usar Windows NT como RTOS.
- Nos centraremos en el estudio de RT-Linux y mostraremos las extensiones de tiempo real ofrecidas por el estándar POSIX (estándar IEEE 1003.1b).

8.2. Problemas de los sistemas operativos convencionales.

- Existen algunas características de los sistemas operativos convencionales que impiden su uso como RTOS (*Real-Time Operating System*).
- Lo veremos en el caso particular de un sistema operativo UNIX.
- Problemas:
  1. Planificación para tiempo compartido:
     - Uso de planificadores que aseguran un uso equitativo del tiempo de CPU entre todos los procesos.
     - Es conveniente para un usuario que usa el sistema desde una terminal.
     - No para procesamiento de tiempo real, ya que la ejecución de cualquier proceso depende de forma compleja e impredecible de la carga del sistema y el comportamiento del resto de procesos.
  2. Baja resolución del temporizador:
     - Históricamente a los procesos de usuario se les proporcionaban señales de alarma y la llamada al sistema *sleep()* con sólo 1 segundo de resolución $\Rightarrow$ No es suficiente para procesamiento de tiempo real.
     - Las versiones más modernas proporcionan medios de especificar intervalos con mayor precisión.

8.2. Problemas de los sistemas operativos convencionales.

- Problemas (cont.):
  3. Núcleo no desalojable:
     - Los procesos que se ejecutan en modo núcleo no pueden ser desalojados.
     - Una llamada al sistema podría tardar demasiado tiempo para poder admitirlo en procesamiento de tiempo real.
  4. Deshabilitación de interrupciones:
     - Muy cercano al anterior está el problema de la sincronización.
     - Para proteger los datos que podrían ser accedidos asíncronamente, algunos diseñadores optan por inhibir las interrupciones durante las secciones críticas $\Rightarrow$ más eficiente que los semáforos.
     - Sin embargo, pone en peligro la posibilidad de responder a eventos externos de forma adecuada.
  5. Memoria Virtual:
     - En STR introduce un nivel de impredecibilidad intolerable.

8.2. Problemas de los sistemas operativos convencionales.

- Algunas soluciones:
  1. MINIX OS:
     - Cambiar el planificador *round-robin* de tiempo compartido por uno basado en prioridades.
     - No hay problemas con la paginación ni el *swapping* (no se usan).
     - Es una solución aceptable para sistemas sencillos.
  2. POSIX.1b-1993:
     - Estándar para introducir características de tiempo real en UNIX.
     - Define planificación con prioridades, bloqueo de páginas de memoria del usuario en memoria, señales para tiempo real, IPC, timers . . .
  3. QNX:
     - Cumple el estándar POSIX.1b.
     - Arquitectura de microkernel $\Rightarrow$ el núcleo implementa sólo cuatro servicios: planificación de procesos, comunicación entre procesos, comunicación de red de bajo nivel y manejo de interrupciones.
     - El resto de servicios se implementan como procesos de usuario.

8.2. Problemas de los sistemas operativos convencionales.

- Algunas soluciones (cont.):
  4. VxWorks:
     - RTOS patentado orientado hacia la aproximación *host*/*target*.
     - Un *host* UNIX es usado para el desarrollo del software y para la ejecución de las partes de la aplicación que no son de tiempo real.
     - El núcleo de VxWorks (llamado *wind*) ejecuta las tareas de tiempo real en la máquina *target*.
     - Las máquinas se comunican utilizando TCP/IP.
     - Proporciona algunas funciones POSIX.1b.
  5. REAL/IX:
     - Es un sistema UNIX (a partir del UNIX System V) al que se le han añadido capacidades para procesamiento de tiempo real.
     - Cumple el estándar POSIX.
  6. Windows NT:
     - Tendencia por usar un SO para todo (incluido tiempo real).
     - Núcleo de Windows NT no sirve para procesamiento de tiempo real.

8.2. Problemas de los sistemas operativos convencionales.

- Algunas soluciones (cont.):

  6. Windows NT (cont.):

     - Varias soluciones proporcionadas por distintas compañías:
       - Intime de RadiSys: modifica la capa Hardware Abstraction Level para atrapar los intentos de Windows para desactivar las interrupciones o resetear el reloj.
       - Los diseñadores de QNX implementan la API de Win32 sobre su SO POSIX-*compliant*.

- Direcciones de interés:
  - QNX: *http://www.qnx.com*
  - VxWorks: *http://www.wrs.com/products/html/vxwks52.html*
  - REAL/IX: *http://209.249.130.161/real_time/c_and_c/c_and_c.html*
  - Real-Time & Windows NT:
    - *http://www.radisys.com/products/rtos/nt_prod.html*

8.3. Sistemas operativos para tiempo real basados en Linux.

- Existen varias posibilidades para la realización de STR utilizando el sistema operativo Linux:
  - Real-Time Linux: Proporciona un núcleo de tiempo real basado en un planificador con desalojo y prioridades fijas. Está preparado para la gestión de tareas críticas.
  - KU Real-Time Linux: Se trata de la realización de un sistema para planificación cíclica de procesos acríticos. Es útil para procesos acríticos como los relacionados con la grabación y reproducción de imágenes en movimiento.
  - POSIX: La versión del núcleo estándar de Linux 2.0.36 implanta algunas llamadas requeridas por la especificación POSIX 1003.1b.

8.3.1. Real-Time Linux (RTL).

- Es una modificación de código de Linux para gestionar tareas críticas.
- Está disponible la versión 1.1 para la versión 2.0.36 del kernel de Linux.
- Por ahora sólo está disponible para la arquitectura del PC.
- Dirección Web: *http://rtlinux.cs.nmt.edu/~rtlinux/*

8.3. Sistemas operativos para tiempo real basados en Linux.

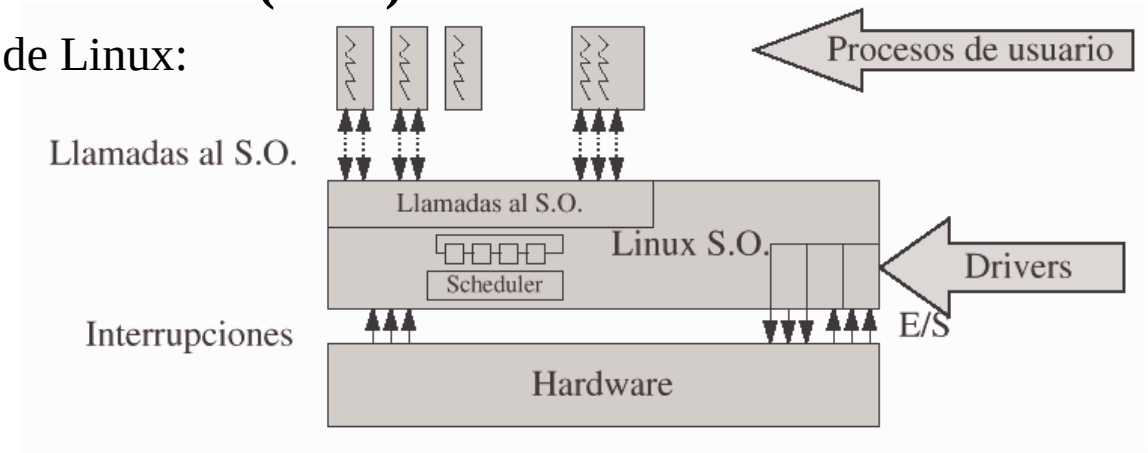
8.3.1. Real-Time Linux (RTL).

- Características:
  - Incluye un planificador con desalojo y prioridades fijas, para la ejecución de tareas críticas de tiempo real.
  - Las tareas pueden ser periódicas o bien activadas mediante una interrupción (esporádicas o aperiódicas).
  - Incorpora mecanismos para la comunicación con los procesos no críticos, que son los de Linux *normal*. Estos mecanismos son colas FIFO.
  - Las tareas de tiempo real se ejecutan con la CPU en modo *supervisor* (pueden acceder a puertos E/S, reprogramar interrupciones, etc...).
  - Convierte al núcleo de Linux en una tarea más, pero de segundo plano (de prioridad mínima).
- RT-Linux es como un microkernel que realiza operaciones muy básicas: gestionar todas las interrupciones y planificar las tareas de tiempo real.
- Linux deja de disponer de las instrucciones *cli*, *sti* e *iret* para tener en su lugar llamadas al propio microkernel de RT-Linux, que las simula.
- Linux pierde el control del sistema y no se ejecutará si las tareas críticas ocupan toda la CPU $\Rightarrow$ El sistema puede bloquearse aparentemente.

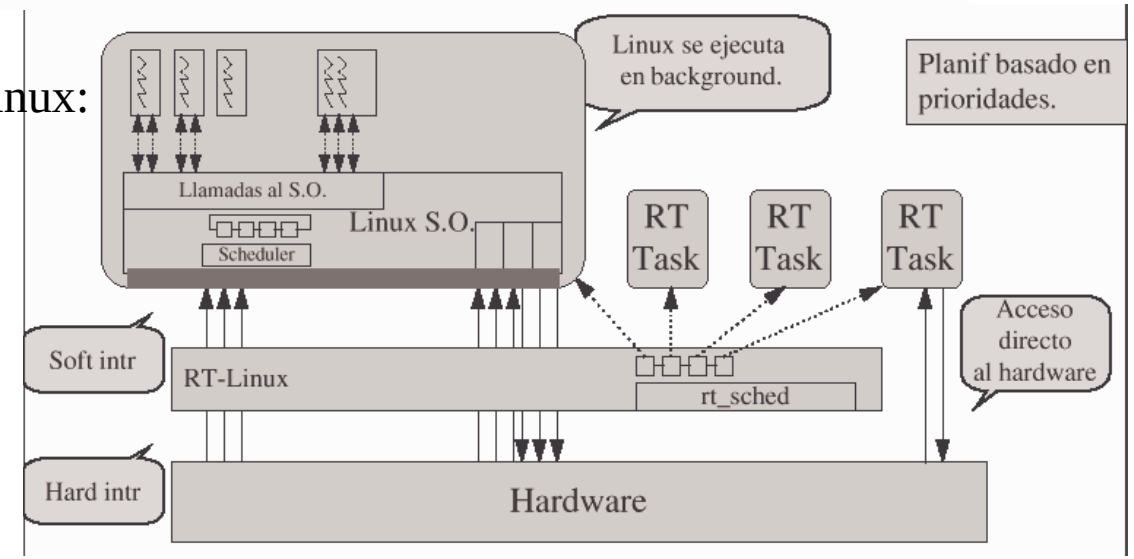
8.3. Sistemas operativos para tiempo real basados en Linux.

8.3.1. Real-Time Linux (RTL).

- Estructura de Linux:



- Estructura de RT-Linux:



8.3. Sistemas operativos para tiempo real basados en Linux.

8.3.1. Real-Time Linux (RTL).

- Las tareas de tiempo real en RT-Linux son código que se ejecuta en modo supervisor de la CPU, para tener acceso directo a los dispositivos de E/S.
- Para ejecutar procesos en modo supervisor no basta con hacerlo desde la cuenta *root*, sino que además es necesario que sea parte del código del núcleo.
- El código del núcleo no se pagina ⇒ un tarea no puede ser expulsada a disco.
- Es necesario programar las taras como módulos de carga (el propio RT-Linux se ha hecho como un módulo).
- Un módulo de carga es un programa que podemos realizar en C en el que:
  - Carece de función *main*.
  - Tiene una función para iniciar el módulo *init_module()* ⇒ se ejecuta al cargarlo y desde ella llamaremos a otras.
  - Tiene una función para finalizar el módulo *cleanup_module()* ⇒ se ejecuta al descargar el módulo.
- El núcleo no dispone de salida estándar, por lo que no podremos usar las funciones de E/S por pantalla habituales, en su lugar usaremos *printk*.

8.3. Sistemas operativos para tiempo real basados en Linux.

8.3.1. Real-Time Linux (RTL).

- Ejemplo:

```
#define MODULE
#include <linux/module.h>
static int x,y;
int init_module (void) { printk("Iniciando módulo..."); return(0); }
int cleanup_module (void) { printk("Finalizando módulo..."); return(0); }
```

- Todo lo que imprimimos con la función *printk* va al anillo de mensajes de Linux, que es explorado por *syslog* para su registro y presentación por pantalla.
- Para compilar el módulo: *gcc -O2 -Wall -D__KERNEL__ -c mimodulo.c*
- Para cargar el módulo: *insmod mimodulo.o* ó *modprobe mimodulo.o*
- Para descargar el módulo: *rmmod mimodulo.o*
- Para ver los módulos cargados: *lsmod*
- Es posible pasar parámetros al módulo, para ello se han definido las variables globales estáticas *x* e *y*, para darles valor: *insmod mimodulo.o x=5 y=6*

8.3. Sistemas operativos para tiempo real basados en Linux.

8.3.1. Real-Time Linux (RTL).

La API de RT-Linux:

- Gestión de tareas según un esquema de prioridades fijas:

rt_task_init	rt_task_suspend
rt_task_delete	rt_task_wait
rt_task_make_periodic	rt_task_wakeup

- Asignación de manejadores de interrupción:

request_RTint	free_RTint

- Comunicación con aplicaciones Linux de usuario:

rtf_create	rtf_get
rtf_destroy	rtf_put
rtf_create_handler	rtf_resize

- Alta resolución de medida de tiempos:

rt_get_time

8.3. Sistemas operativos para tiempo real basados en Linux.

8.3.1. Real-Time Linux (RTL).

- Ejemplo: Tarea periódica.

gcc -O2 -Wall -D__RT__ -D__KERNEL__ -c ejemplo.c

```
#define MODULE
#include <linux/module.h>
#include <linux/rt_sched.h>
RT_TASK mi_tarea;
void accion(int no_usado) {
  static int activacion;
  while (1) {  printk("Activación número %d\n", cont++);
               rt_task_wait();  } }
int init_module (void) {
  rt_task_init(&mi_tarea, accion, 1, 1000, 1);
  rt_task_make_periodic(&mi_tarea, (RTIME) rt_get_time() + (RTIME) RT_TICKS_PER_SEC,
                                   (RTIME) RT_TICKS_PER_SEC);
  return(0); }
int cleanup_module (void) {
  rt_task_delete(&mi_tarea);
  return(0); }
```

Constituye la tarea periódica

Registra la tarea en el planificador

Instante de la primera activación

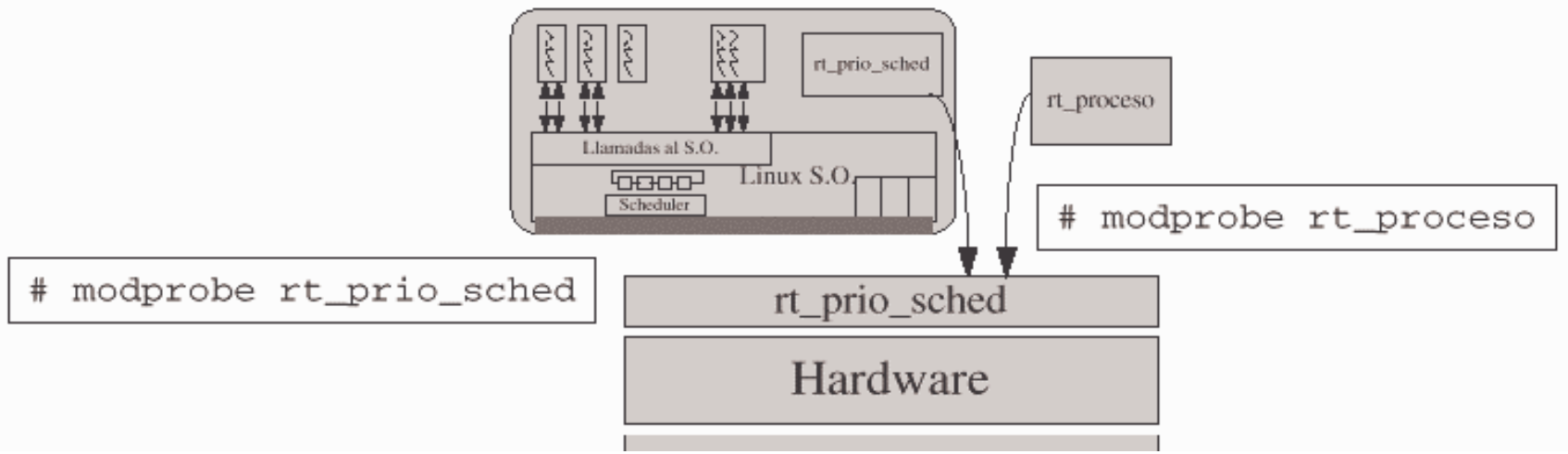
Periodo

Borra la tarea del planificador

8.3. Sistemas operativos para tiempo real basados en Linux.

8.3.1. Real-Time Linux (RTL).

- Para poder ejecutar tareas de tiempo real tenemos que cargar el módulo del planificador. Al hacerlo, Linux pasa a ser una tarea de tiempo real (RT_TASK) con la prioridad más baja.
- Si durante la ejecución de otras tareas de tiempo real se produce una interrupción que debe atender Linux, se guarda temporalmente hasta el momento en que Linux sea la tarea activa.
- Una tarea RT_TASK se carga igual que cualquier módulo.



8.3. Sistemas operativos para tiempo real basados en Linux.

8.3.1. Real-Time Linux (RTL).

- La idea de RT-Linux es la de dividir una aplicación de tiempo real en dos partes:
  - Parte de tiempo real: Incluye el código que es crítico en tiempo y debe mantenerse lo más simple posible.
  - Parte de no-tiempo real: Realiza el procesamiento de los datos, incluyendo interfaces de usuario así como el archivo y distribución de los datos.

¿Cómo se comunican?

- RT-Linux proporciona colas de tiempo real (RT-FIFO) para la comunicación de ambas partes así como entre tareas de tiempo real. Son similares a las tuberías UNIX.
- Las FIFO son globales a todo el sistema y se identifican por un número.
- Las tareas RT_TASK hacen uso de las FIFO usando las funciones de la API para comunicación con tareas de usuario.
- Las tareas de usuario acceden empleando mecanismos estándar de UNIX sobre ficheros (funciones *open*, *read*, *write* y *close*). Concretamente a las FIFO se accede mediante los ficheros especiales */dev/rtf*?, donde ? = 0, 1, 2 ...

8.3. Sistemas operativos para tiempo real basados en Linux.

8.3.1. Real-Time Linux (RTL).

- Otra forma de comunicación es memoria compartida

- RT-Linux se instala como un parche (*patch*) sobre el código fuente de Linux:
  - Este parche modifica el código fuente de Linux para evitar que éste acceda directamente a las interrupciones y a algunas instrucciones máquina importantes (*cli*, *sti*, *iret*).
  - También se instalan los ficheros fuente del planificador de tiempo real y de otras facilidades de tiempo real (*rt-fifo*, *rt-time*).

    patch -p1 < donde_esta/parche
- Hay que recompilar el núcleo de Linux de la forma habitual:

make config; make dep; make clean; make zImage; make modules; make modules_install;

- Se reinicia la máquina.
- En este momento Linux se comporta normalmente, hasta que no carguemos el planificador no tendremos el sistema de tiempo real.

8.3. Sistemas operativos para tiempo real basados en Linux.

8.3.2. KU Real-Time Linux.

- Es útil para sistemas de tiempo real *firm*: sistemas que tienen unos requisitos de temporización de grano fino (típicos de los críticos) junto con los requerimientos de los servicios de los acríticos.
- Por ejemplo: aplicaciones multimedia de vídeo tienen requisitos temporales típicos de un sistema crítico y necesitan muchos servicios del sistema.
- Al contrario que en RT-Linux, las tareas pueden hacer uso de todas las facilidades de Linux.
- Las modificaciones que se han llevado a cabo sobre el núcleo son:
  - Mejorar la granularidad del reloj del sistema: en Linux-i386 la frecuencia con la que se interrumpe el reloj es de 10 ms (100 veces por segundo), y es con esta resolución temporal con la que se toman las acciones de control y se mide el tiempo. KURT programa el chip de reloj para que genere interrupciones bajo demanda, en vez de periódicamente. Se logran una resolución superior al microsegundo.
  - Se ha modificado el planificador para incluir una nueva política de planificación (SCHED_KURT) además de las que Linux ya implementa.
  - Se han añadido nuevas llamadas al sistema para poder hacer uso de las nuevas funcionalidades de tiempo real.

8.3. Sistemas operativos para tiempo real basados en Linux.

8.3.2. KU Real-Time Linux.

- Las tareas de tiempo real son módulos de carga dinámica.
- Se ha implementado un planificador cíclico que se basa en el uso de una tabla (plan) en la que están anotadas todas las acciones de planificación (instante de activación, tarea a ejecutar, duración ...).
- La tabla se construye durante la fase de diseño del sistema y el trabajo del planificador consiste en leer secuencialmente la tabla y seguir sus indicaciones ⇒
  - El planificador es muy sencillo de implementar y eficiente.
  - Dificultad a la hora de construir el plan.

- Más información: *http://hegel.ittc.ukans.edu/projects/kurt/index.html*