Un Caso de Uso de un Sistema Operativo Convencional en Tareas de Tiempo Real Blandas Tolerantes a Fallas

Pablo Novarini[1]
Facultad de Informática – UNLP
(1900) La Plata, Buenos Aires, Argentina
pnovarini@lidi.info.unlp.edu.ar

Rodriguez Christian[1]
Facultad de Informática – UNLP
(1900) La Plata, Buenos Aires, Argentina

Ing. Fernando Romero[2]
LIDI - Facultad de Informática – UNLP
(1900) La Plata, Buenos Aires, Argentina
fromero@ lidi.info.unlp.edu.ar

Lic. Hugo D. Ramón[3]
LIDI - Facultad de Informática – UNLP
(1900) La Plata, Buenos Aires, Argentina
hramon@lidi.info.unlp.edu.ar

Resumen

Se presenta la utilización de un sistema operativo convencional (SO), Linux, en plataforma Intel, para la solución de un problema en tiempo real blanda (STR). Se exponen los inconvenientes de los SO y la forma que se utilizo de atenuar sus efectos en la implementación presentada.

Se justifica la utilización de los SO y no extensiones tipo Posix, caracterizando los problemas capaces de ser resueltos mediante su utilización, con la ventaja comparativa en costos por la reutilización de plataformas ya existentes.

Se desarrollo un driver de una placa analogica-digital (A/D) que amortigua los problemas de los SO para STR y se dos aplicaciones de testeo en modo consola y X-Windows, esta placa A/D posee entradas digitales y analógicas y salidas digitales, de uso conveniente para problemas tolerantes a fallas.

Palabras clave: Sistemas de Tiempo Real, Sistemas Operativos, Aplicaciones distribuidas, Tolerancia a fallas.

[1] Alumno Licenciatura en Informática, Facultad de Informática, UNLP

[2] JTP semidedicacion, LIDI, Facultad de Informática, UNLP

[3] Profesor dedicación exclusiva, LIDI, Facultad de Informática, UNLP

Características de un STR

Dentro de los requerimientos funcionales existe un subconjunto relacionado con la performance o las restricciones temporales para reaccionar ante un evento externo. Esto implica que la aplicación tiene fuerte relación con el tiempo de procesamiento de los módulos concurrentes. Estos módulos concurrentes trabajan bajo restricciones de tiempo que pueden ser duras o blandas [1].

La expiración de procesos periódicos en intervalos adecuados, esta relacionada con las acciones a tomar en caso de no completar, por un determinado proceso, en los tiempos máximos determinados, para evitar inconsistencias del sistema [2].

Debido a la naturaleza distribuida por la diversidad de dispositivos a ser manejados por estos sistemas y a que los mismos operan a diferentes velocidades, se deben implementar mecanismos y políticas para aislar este problema de las restricciones temporales de la aplicación.

Se deben controlar esos dispositivos para que el sistema no sea dependiente de las bajas velocidades, utilizando, entre otras, el uso de interrupciones. El manejo de las mismas conlleva la asignación de prioridades a las distintas tareas que concurren al uso del procesador y a los recursos del sistema en un determinado tiempo. Esto esta relacionado fuertemente con el sistema operativo, el que proveerá mecanismos (primitivas) para la comunicación entre los distintos procesos, tanto sincrónica como asincrónica [3].

Por lo tanto, un sistema operativo para tiempo real (RTOS) debe ofrecer facilidades para implementar concurrencia (tales como threads con memoria compartida), temporización (tanto para la medida de tiempos como para la ejecución periódica), planificación (prioridades fijas con desalojo, acceso a recursos con protocolos de herencia de prioridad) y un manejo adecuado de dispositivos de E/S (acceso a recursos hardware e interrupciones) [4].

Existen varios RTOS:
QNX (http://www.qnx.com/),
Lynx (http://www.lynuxworks.com/),
VxWorks (http://www.gespac.com/html/Software/vxworks.html),
RT-Linux (http://www.rtlinux.org),
KURT (http://www.ittc.ku.edu/kurt/),
y soluciones para usar Windows NT como RTOS (http://www.realtime-info.be/magazine/97q2/winntext.htm).

Inconvenientes de los SO en Aplicaciones de TR

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 por el funcionamiento de los distintos mecanismos que implica la multitarea, lo que lleva como consecuencia no permitir garantizar los tiempos de respuesta. Existen algunas características de los sistemas

operativos convencionales que impiden su uso como RTOS. [5] Lo veremos en el caso particular de un sistema operativo UNIX.

La *planificación para tiempo compartido* asegura un uso de acuerdo a prioridades del tiempo de CPU entre todos los procesos, lo cual lo hace conveniente para un usuario que usa el sistema desde una terminal, pero no lo hace apto para procesamiento de tiempo real, ya que la ejecución de cualquier proceso depende de forma compleja e impredecible de la carga del sistema y del comportamiento del resto de procesos.

La *baja resolución del temporizador,* hace que a los procesos de usuario se les proporcionan señales de alarma y la llamada al sistema *sleep()* con sólo 1 segundo de resolución, lo cual no es suficiente para procesamiento de tiempo real, aunque versiones más modernas proporcionan medios de especificar intervalos con mayor precisión.

El *núcleo no es desalojable*, por lo que los procesos que se ejecutan en modo núcleo no pueden ser desalojados, por lo que una llamada al sistema podría tardar demasiado tiempo para poder ser admitida en procesamiento de tiempo real.

La *deshabilitación de interrupciones* que está relacionada al problema de la sincronización. Para proteger los datos que podrían ser accedidos asincronicamente, se opta por inhibir las interrupciones durante las secciones críticas, lo cual es más eficiente que el uso de semáforos. Sin embargo, esto pone en peligro la posibilidad de responder a eventos externos de forma adecuada.

La *memoria virtual* introduce un nivel de impredecibilidad intolerable, debido a los tiempos excesivos de acceso a disco para los reemplazos en caso de fallo de pagina.

Estas son las características principales cuyos efectos se busca atenuar. La metodología para tratarlo esta desarrollada en el presente trabajo.

Ventajas de la utilización de un SO en Aplicaciones de TR.

Hay ámbitos en los que no es tradicional la utilización de STR como o sistemas distribuidos de tiempo real (SDTR), en los que el creciente uso de la tecnología los hace necesarios (oficinas, consultorios médicos y hogares). En estos ámbitos es común la existencia de plataformas de tipo PC, interconectadas a través de una red de área local (LAN), por lo que su reutilización para la implementación de SDTR trae aparejado un ahorro en el costo del sistema a implementar.

La utilización de estas plataformas hace posible la implementación de sistemas tales como los relacionados a viviendas (sistemas de seguridad y de automatización del hogar), oficinas (controles de acceso, sistemas de seguridad y de incendio). Estas aplicaciones, si bien son STRs, ya que poseen sus características, sus requerimientos de performance son limitados.

Placa ADQ12-B

Se analiza también el funcionamiento de la aplicación en tiempo real en un ambiente multitasking como es Linux, donde factores externos a la aplicación pueden afectar en forma significativa su comportamiento.

Se analiza también el funcionamiento de la aplicación en tiempo real en un ambiente multitasking como es Linux, donde factores externos a la aplicación pueden afectar en forma significativa su comportamiento.

Los datos leídos pueden corresponder a señales unipolares o bipolares. Para la adquisición de datos se utiliza un conversor analógico - digital de 12 bits. La obtención del valor convertido se deberá realizar en dos lecturas sucesivas.

La señal de fin de conversión EOC, indica que se cuenta con un nuevo valor digitalizado.

Posee un puerto de salida llamado OUTBR. Este puerto tiene 8 posibles salidas identificadas como OUTB0-OUTB7. La primera (OUTB0) sirve para enmascarar la entrada de interrupción IN0. Cada OUTBi puede tomar el valor 0 o 1. Para poder setear el valor para cada OUTBi necesitamos referenciar el registro de un byte OUTBR. En él se especifica con los 3 bits menos significativos (b0-b2) a que OUTBi se modificara su valor, mientras que el bit 3 indica el valor. Los bits 7 a 4 no se utilizan. Si se necesitan setear los 7 OUTBi a 1 se necesitan 7 accesos a OUTBR.

Otro registro de gran utilidad es el STINR. En él reside información sobre cuatro señales de entrada IN0-IN4 (b0-b4 respectivamente), el caso de IN0 se utiliza para la generación de interrupciones desde una fuente externa a través de un puerto DB-39. El bit 5, indica el valor antes descripto EOC (end of convertion). El bit 6 muestra el valor de OUTP (flag que indica el estado de salida del Pacer). El bit 7 muestra el valor de OUT2 (flag que indica el estado de salida del contador 2).

ADQ 12-B utiliza la unidad INTEL 8253 [9], la cual dispone de tres temporizadores programables de 16 bits y un registro de control. Cada timer cuenta con:

- Entrada de disparo G0, G1 y G2
- Entrada de reloj CLK0, CLK1 y CLK2
- Salida OUT0, OUT1 y OUT2.

Cada timer está conformado por un contador (CONT0, CONT1 y CONT2) de 16 bits. El contador se carga por software con un valor inicial y tras un disparo inicia una cuenta descendente, al ritmo que impone la señal del reloj, cuando la cuenta arriba a cero la salida OUTn toma el estado alto.

Los timer 1 y 0 conforman un Pacer de 32 bits. La combinación de ambos permite obtener marcaciones temporales entre 16 μseg y 2 horas 23 minutos. La salida se obtendrá del bit 6 del STINR denominado OUTP.

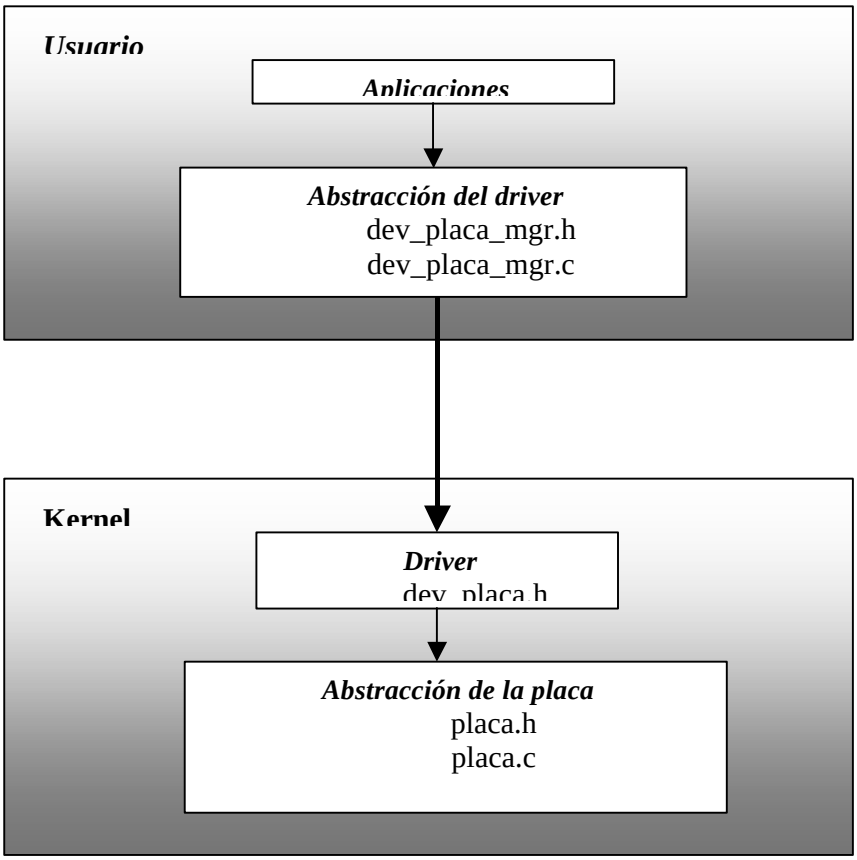
La salida OUTP puede activar uno de los niveles de interrupción de la PC: IRQ5, IRQ4, IRQ3 (configurable por hardware). El Pacer puede emplearse para generar demoras de tiempo programables.

Propuesta de solución a los inconvenientes planteados

Se desarrolló un driver para la placa ADQ12-B [8] en donde las tareas de máxima prioridad, como la adquisición, corren en el espacio del kernel y no en el de usuario. También el uso de un timer externo para obtención de una gama de tiempos con mejor resolución y rango que los provistos por el sistema operativo.

Se analiza también el funcionamiento de la aplicación en tiempo real en un ambiente multitasking como es Linux, donde factores externos a la aplicación pueden afectar en forma significativa su comportamiento.

Para esto se desarollarón dos aplicaciones para verificacón del driver, una en modo consola (CADQ-12B), y la otra utilizando la version KDE de X-Windows (XADQ-12B).



Funcionalidades de las aplicaciones

Entre las funcionalidades provista se destacan el seteo de parámetros propios de la placa: modo de trabajo balanceado o no balanceado, unipolar o bipolar. La selección del canal desde donde se desean adquirir valores y mostrar la gráfica en tiempo real de los valores adquiridos. Se puede setear la velocidad de muestreo. Se manejan de los canales de salida OUTB1 a OUTB7 y se permite visualizarn los canales de entrada IN1 a IN4.

Las aplicaciones utilizan el pacer para lograr la frecuencia de muestreo deseada. Las interrupciones generadas por el pacer son atendidas por un manejador que lee un nuevo valor de la placa y lo gráfica en pantalla.

Debido a que la visualización de los valores adquiridos se realiza en tiempo real, se requiere eficiencia en la visualización a efectos de no perder datos, se han definido opciones de gráfica configurables por el usuario permitiendo variar la calidad.

El valor adquirido se grafica en el eje vertical, y los distintos valores se van desplazando sobre el eje horizontal.

Otro parámetro importante que se puede configurar optimizando altamente la visualización de la señal es el que determina cada cuantos valores adquiridos se actualiza la gráfica de la señal.

Descripción del driver

El driver es un módulo que debe registrarte en Linux [10][11][12][13][14]. Se tiene 4 funciones básicas para operar el dispositivo

- open: Abre el dispositivo.
- read: Lee de a un valor del dispositivo.
- close: cierra el dispositivo.
- ioctl: permite setear los valores sobre el dispositivo según el parámetro CMD

Los valores posibles del CMD a enviar a ioctl están definidos en dev_placa.h y son los que se muestra a la siguiente tabla.

PLACA_SET_MSKP_CTREG *(Establece el valor de la mascara)*	PLACA_SET_CANAL_CTREG *(Establece el canal A/D de entrada)*	PLACA_SET_PGA_CTREG *(Establece la ganancia del amplificador)*
PLACA_SET_BIPOLAR *(Establece si la seàl sera bi o unipolar)*	PLACA_LEER_OUTP_STINR *(Para leer una entrada digital)*	PLACA_GET_MSKP_CTREG *(Obtener el valor de la mascara)*
PLACA_SET_BALANCEADA *(Establece si las entradas seran 8 balanceadas o 16 sin balancear)*	PLACA_GET_CANAL_CTREG *(Obtiene el canal A/D del cual se esta leyendo)*	PLACA_ES_BIPOLAR *(Obtiene en que modo está)*
PLACA_SET_PACER *(Establece los valores del pacer)*	PLACA_GET_PGA_CTREG *(Obtiene los valores de ganancia del amplificador)*	PLACA_ES_BALANCEADA *(Obtiene en que modo está)*
PLACA_LEER_EOC_STINR *(Obtiene el End of Conversion)*	PLACA_GET_GTP_CTREG *(Obtiene los valores de configuracion del pacer)*	PLACA_SET_GTP_CTREG *(Establece los valores de configuracion del pacer)*
PLACA_SET_BIT_OUTBR *(Configura un bit de la salida digital)*		

Pruebas

Se han realizado pruebas de la capacidad de respuesta del driver desarrollado. Para ello seutilizo la aplicación consola la cual adquiere datos en tiempo real a la frecuencia especificada por los parámetros. Al no realizar procesamiento con los datos adquiridos se evalúa en forma fehaciente el driver.

Las siguientes tablas muestran la cantidad de tiempo requerido para adquirir diferentes cantidades de muestras utilizando diferentes frecuencias de muestreo:

Cantidad Muestras	Tiempo Esperado	TiempoUtilizado	Demora
2	20.000	22.977	2.977
10	100.000	114.200	14.200
100	1.000.000	1147.713	147.713
1.000	10.000.000	10507.601	1507.601
10.000	100.000.000	115275.700	15275.700

Frecuencia de muestreo: 10.000 μseg (100 muestras/segundo)

Cantidad Muestras	Tiempo Esperado	Tiempo Utilizado	Demora
2	2.000	2.270	270
10	10.000	11.663	1.663
100	100.000	117.363	17.363
1.000	1000.000	1174.311	174.311
10.000	10000.000	11744.497	1744.497

Frecuencia de muestreo: 1.000 μseg (1.000 muestras/segundo)

Cantidad Muestras	Tiempo Esperado	Tiempo Utilizado	Demora
2	200	230	30
10	1000	1.324	324
100	10.000	13.938	3.938
1.000	100.000	139.993	39.993
10.000	1.000.000	1.400.781	400.781

Frecuencia de muestreo: 100 μseg (10.000 muestras/segundo)

Cantidad Muestras	Tiempo Esperado	Tiempo Utilizado	Demora
2	64	89	25
10	320	632	312
100	3200	6768	3568
1000	32000	67992	35992
10000	320000	680379	360379

Frecuencia de muestreo: 32 μseg (31.250 muestras/segundo)

* Los tiempos se miden en μseg

La columna demora indica la diferencia de tiempo entre el esperado para tomar las muestras y el real (en μseg).

Como se puede observar en las tablas precedentes, el tiempo de respuesta del driver sometiéndolo a una gran cantidad de muestras da una mayor demora.

Conclusiones y trabajos futuros

El presente trabajo se inscribe en el marco de una serie de trabajos que llevan como finalidad explorar las limitaciones de los SO para su empleo en aplicaciones de STR.

Si bien se experimento con una placa de uso general, la experiencia apunta a una metodologia para desarrollar STR usando como plataforma estos SO explotando al máximo sus capacidades y atenuando sus limitaciones. Como trabajos futuros estan proyectados la realización de un Vxd (driver virtual) para Windows 98 y un driver para Windows NT.

El driver del presente trabajo fue probado en un sistema Linux dentro de dos ámbitos: X windows y consola. Para ambos casos, el driver respondió en forma correcta. El hecho de haber realizado dos aplicaciones que utilizan el driver fue para poder apreciar cómo se ve afectada la performance respecto de la cantidad de procesamiento realizada en cada caso.

Sin embargo, el hecho que Linux sea un sistema operativo multitasking hace que el desarrollo de sistemas en tiempo real, es decir que requieren de un tiempo de respuesta crítico, no sea del todo adecuado, ya que la performance se ve afectada por factores propios de este tipo de sistemas operativos. Como fundamento de esta conclusión, un sistema multitasking tratará de asignar tiempo de CPU a todas las tareas que estén ejecutando. Por esta razón si la tarea que se necesita ejecutar en tiempo real es crítica, probablemente un sistema operativo de este tipo no sea el adecuado. En contraposición, si el requerimiento de las aplicaciones de tiempo real esta dentro de los limites de tiempo de respuesta crítico que surgen de las pruebas realizadas, entonces sería posible pensar en este tipo de sistemas operativos.

Bibliografía

[1] Ada in Distributed Real-Time Systems. K. Nielsen, McGraw Hill, 1990.

[2] Software Design Methods for Concurrent and Real-Times Systems. H. Gomaa, Adison Wesley, SEI series in Software Engineering, 1993.

[3]Real-Time Systems and Programming Languages, A: Burns. Addison Wesley,1996

[4] An Introduction to Real-Time Systems. L. Buhr. Prentice Hall, 1999.

[5] Sistemas Operativos. Conceptos Fundamentales, 3rd Ed A. Silberschatz, J. Peterson, P. Galvin, Addison-Wesley Iberoamericana, 1994.

[6]An Embedded Software Primer, D. Simon, 1999. Addison Wesley CEPUB

[7] Real-Time Programming, Grehan. Addison Wesley, 1998.

[8]Manual de la placa ADQ 12-B.

[9] Manual del CI 8253.

[10]Linux device driver (1ra edición) Alessandro Rubini

[11]Device programming. www.kernel.org/LDP

[12]Module programming. www.kernel.org/LDP

[13]Linux programming guide. www.kernel.org/LDP

[14]Kernel hacking. www.kernel.org/LDP

[15]Documentación de Linux. Funciones open, read, close ioctl.