Sistema de Tiempo-Real para el Procesamiento Digital de Audio

Vidblaín Amaro, Arnoldo Díaz-Ramírez, *Miembro IEEE*, Víctor H. Díaz-Ramírez y Heber S. Hernández

***Resumen*—El procesamiento de audio consiste en incrementar la calidad de las señales sonoras en términos de la inteligibilidad y de la reducción de ruido. Existen algoritmos de procesamiento de audio que son computacionalmente eficientes, pero que sacrifican la inteligibilidad al tratar de suprimir el ruido. Este problema puede solucionarse al utilizar técnicas de procesamiento localmente adaptativo. El procesamiento localmente adaptativo es un proceso computacionalmente costoso, ya que depende del calculo de estadísticas locales de la señal de entrada en cada instante. Los algoritmos localmente adaptativos requieren de sistemas de procesamiento confiables y eficientes, que garanticen un correcto funcionamiento en todo momento para aplicaciones críticas. La implementación de estos filtros en procesadores de propósito general representa una alternativa interesante, ya que permite un mejor aprovechamiento de los recursos disponibles en un sistema y a un bajo costo. En este articulo, proponemos un sistema de tiempo-real para el procesamiento confiable de señales de audio basado en Preempt-RT, que es un sistema operativo de tiempo-real para Linux. El sistema propuesto puede suprimir el ruido en señales de audio de longitud infinita, sin destruir su inteligibilidad. Los resultados obtenidos muestran que el sistema de tiempo-real propuesto es capaz de procesar eficientemente señales de audio empleando técnicas de procesamiento localmente adaptativo, con un bajo costo computacional.**

Palabras Claves — Sistemas de Tiempo-Real, Sistemas Operativos de Tiempo-Real, Procesamiento Digital de Audio

I. Introducción

El procesamiento digital de señales (*digital signal processing* o DSP) aborda los aspectos relacionados con la representación digital de las señales y el uso de sistemas digitales para analizar, modificar o extraer información de esas señales. En años recientes se ha desarrollado una intensa investigación relacionada con el desarrollo de algoritmos de DSP. Además, los avances logrados en las áreas de tecnologías digitales y de semiconductores ha permitido la implementación de algoritmos de DSP en sistemas digitales de alta integración electrónica que ofrecen reducción de espacio, bajo consumo de energía y un bajo costo.

Existen muchas ventajas del uso de técnicas digitales para el procesamiento de señales, tales como compresión, eliminación de ruido o extracción de características. En comparación con dispositivos analógicos, los sistemas DSP son mas convenientes debido a su flexibilidad, capacidad de ser programados, su confiabilidad y a que permiten la implementación de aplicaciones sofisticadas [7].

En la Fig. 1 se muestran los componentes básicos de un sistema para el DSP. Puede observarse que el sistema recibe de entrada una señal analógica, que es convertida a una señal digital por medio de un convertidor analógico digital (ADC). La conversión de la señal analógica a digital, conocida también como *digitalización*, se lleva a cabo por medio de los procesos de muestreo (discretización en tiempo) y cuantización (discretización en amplitud). El proceso de muestreo permite representar una señal analógica como una secuencia de valores. Para esto, la señal de entrada es procesada en muestras, en instantes uniformes nT, en donde n es un entero positivo y T es el periodo de muestreo. El proceso de muestreo convierte una señal analógica (véase Fig. 2) en una señal discreta en el tiempo (véase Fig. 3). La amplitud de cada muestra es cuantizada en alguno de los niveles 2^B, en donde B es el número de bits que el ADC utiliza para representar cada muestra.

El periodo de muestreo está definido como

$$T = \frac{1}{f_s}, \tag{1}$$

en donde f_s es la frecuencia de muestreo en *hertz* o ciclos por segundo.

Para representar una señal analógica $x(t)$ con una señal discreta en el tiempo $x(nT)$ de manera correcta, la frecuencia de muestreo f_s debe ser al menos el doble del máximo componente de la frecuencia (f_M) en la señal analógica $x(t)$; esto es,

$$f_s \geq 2f_M, \tag{2}$$

en donde f_M es conocido también como el ancho de banda de la señal $x(t)$.

A la Ec. (2) se le denomina el Teorema de Shannon, y establece que cuando la frecuencia de muestreo es mayor o igual que el doble del mayor componente de la frecuencia contenida en la señal analógica, la señal original $x(t)$ puede ser perfectamente reconstruida utilizando su correspondiente señal discreta en el tiempo $x(nT)$.

Por otra parte, dada una frecuencia de muestreo para una aplicación específica, el periodo de muestreo puede ser determinado con la Ec. (1). Por ejemplo, en los CDs de audio,

Vidblaín Amaro-Ortega es estudiante de Maestría del Instituto Tecnológico de Mexicali, Av. Tecnológico s/n, Col. Elías Calles, Mexicali, B.C., México, 21376 (e-mail: vidblain@itmexicali.edu.mx)

Arnoldo Díaz-Ramírez es profesor-investigador del Instituto Tecnológico de Mexicali, adscrito al Departamento de Sistemas y Computación (e-mail: adiaz@itmexicali.edu.mx)

Víctor Hugo Díaz-Ramírez es profesor-investigador del CITEDI-IPN, Av. del Parque No. 131O, Mesa de Otay, Tijuana, Baja California, México, 22510 (e-mail: vhdiaz@citedi.mx)

Heber Samuel Hernández-Tabares es profesor-investigador del Instituto Tecnológico de Mexicali, adscrito al Departamento de Sistemas y Computación (e-mail: heberhdz@itmexicali.edu.mx)

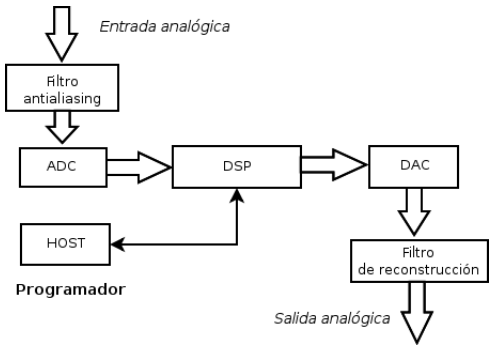

Figura 1. Componentes básicos de un sistema DSP

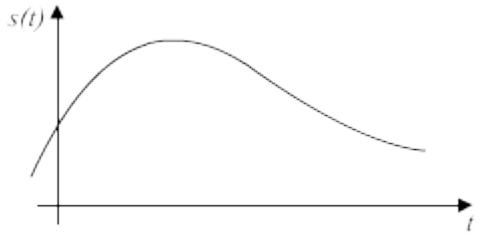

Figura 2. Señal Analógica

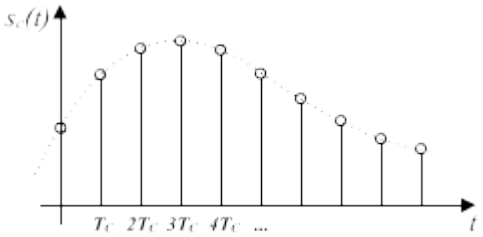

Figura 3. Señal Digital

la frecuencia de muestreo es $f_s = 44{,}1\ kHz$, por lo tanto el periodo de muestreo es $T = {}^{1}/_{44100s} = 22{,}76\mu s$.

Una vez que se tiene la señal digital, ésta puede ser procesada por la unidad DSP. Debido a la flexibilidad de este tipo de sistemas, es posible la implementación de algoritmos para procesar la señal con diversos fines. Por ejemplo, uno de los temas de mayor interés es el del procesamiento de señales digitales de voz que han sido degradadas por procesos de ruido, que consisten en modificaciones indeseables que sufre la señal durante el proceso de grabación, almacenamiento y/o transmisión. Para restaurar o mejorar la calidad de las señales capturadas por un sistema digital se han propuesto una gran cantidad de filtros digitales, entre los que podemos destacar los filtros localmente adaptativos. Los filtros localmente adaptativos pueden procesar una señal variando su acción de transformación en función de los parámetros estadísticos locales de la señal de entrada, calculados en cada instante. Esto permite que el filtrado localmente adaptativo pueda suprimir el ruido presente en las señales sin destruir los detalles finos de las mismas, conservando de mejor manera su inteligibilidad. Sin embargo, es común que estos filtros presenten una complejidad elevada.

La parte final del procesamiento consiste en que la señal digital procesada es convertida de nuevo a una señal analógica utilizando un convertidor digital analógico (DAC).

El procesamiento de una señal digital se lleva a cabo utilizando alguna plataforma de hardware. Entre las opciones de hardware existentes para la implementación de sistemas DSP se encuentran los circuitos integrados específicos de aplicación (*application-specific integrated circuits* o ASIC), *field-programmable gate arrays* (FPGA), procesadores de propósito general, y procesadores generales de procesamiento de señales (procesadores DSP), por mencionar algunos [7].

El uso de procesadores de propósito general ofrece muchas ventajas, tales como flexibilidad, bajo consumo de energía y bajo costo. Uno de los mayores atractivos consiste en que pueden ser utilizados en varias aplicaciones al mismo tiempo. Por ejemplo, con un procesador de propósito general es posible llevar a cabo simultáneamente procesamiento digital de señales, operaciones de control industrial, monitorización de eventos, entre muchos otros más. Sin embargo, estas aplicaciones tienen restricciones temporales críticas. En el caso del procesamiento de una señal digital, el muestreo de la señal debe llevarse a cabo en periodos de tiempo determinados, por lo que la latencia máxima en el procesamiento de la señal debe estar acotada. Por esta razón, además de la utilización de un procesador de propósito general es necesario utilizar un sistema operativo de tiempo-real, de tal manera que las restricciones temporales de las tareas del sistema siempre se cumplan.

En este artículo se presenta un sistema de tiempo-real para el procesamiento de señales digitales de audio. El sistema propuesto utiliza Preempt-RT, que es una sistema operativo de tiempo-real basado en Linux. Los resultados obtenidos muestran que el sistema es capaz de procesar eficientemente señales de audio con un bajo costo computacional.

El resto del documento está organizado de la siguiente manera. En la Sección II se explican brevemente las características de los sistemas operativos de tiempo-real. La Sección III presenta la arquitectura del sistema propuesto. En la Sección IV se muestran detalles de implementación del sistema, así como los resultados obtenidos. Finalmente, en la Sección V se presentan las conclusiones y el trabajo futuro.

II. Sistemas Operativos de Tiempo-Real

Un sistema operativo está compuesto por un conjunto de programas que actúan como intermediario entre los programas aplicación y el hardware de una computadora. El sistema operativo se encarga de administrar eficientemente todos los recursos existentes, a fin de que una aplicación pueda hacer uso de éstos sin interferir con el funcionamiento de otras aplicaciones o del sistema operativo [9].

Los sistemas operativos pueden clasificarse en *sistemas operativos de propósito general* y *sistemas operativos de tiempo-real*. Los primeros tienen como objetivo soportar aplicaciones de tiempo compartido (*time-sharing*) y por lo tanto deben asignar de manera equitativa los recursos disponibles a las tareas del sistema. En contraste, los sistemas operativos de tiempo-real tienen como objetivo soportar aplicaciones de

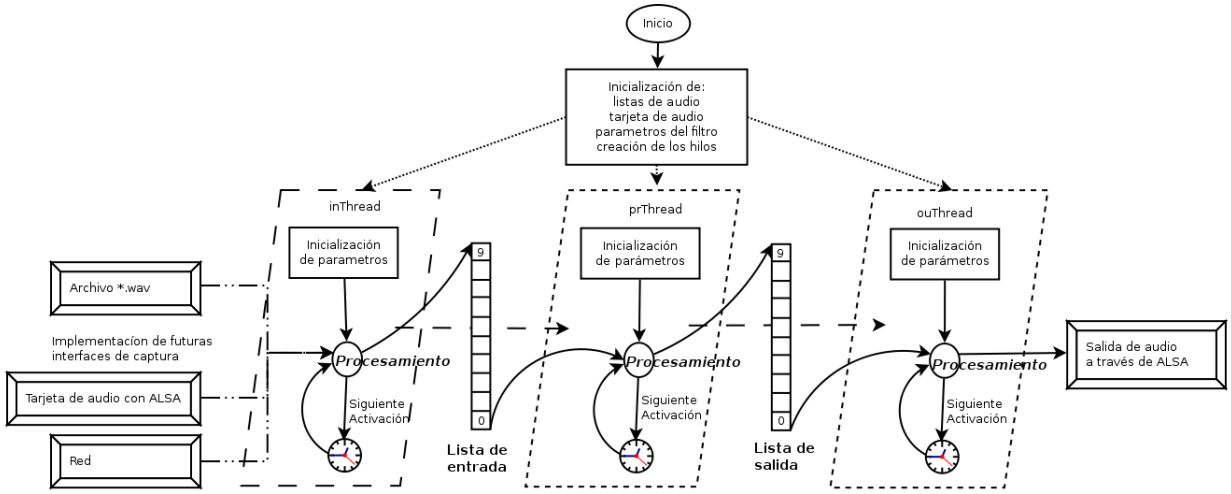

Figura 4. Arquitectura del sistema

tiempo-real, por lo que utilizan algoritmos que permitan el cumplimiento de las restricciones temporales de este tipo de aplicaciones.

Los sistemas de tiempo-real son sistemas informáticos que deben reaccionar en instantes de tiempo específicos a eventos del entorno. Como consecuencia, el correcto funcionamiento de estos sistemas no depende tan solo de los valores generados, sino también del tiempo en el cuál esos valores son generados. Las consecuencias de no entregar resultados a tiempo pueden llegar a ser catastróficas en algunos casos [2].

Un sistema de tiempo-real está compuesto por varias tareas. La mayoría de los requerimientos temporales del sistema pueden expresarse en los plazos, tiempos de arribo y tiempos de ejecución de las tareas. El uso de algoritmos de planificación eficientes, así como el uso de técnicas confiables de análisis de planificabilidad, son necesarios para garantizar el cumplimiento de las restricciones temporales de las tareas.

En un sistema operativo de tiempo-real (SOTR) las tareas comúnmente se implementan como *threads*[1]. Un thread, también conocido como *proceso ligero*, es la mínima unidad de ejecución administrada por el sistema operativo y está contenido en un proceso.

Un sistema operativo de tiempo-real consiste de un núcleo o *kernel* que proporciona las funciones básicas del sistema y toma el control de la ejecución de los threads. El núcleo del SOTR responde a las llamadas al sistema, proporciona servicios de planificación y temporización, y administra las interrupciones externas.

Una llamada al sistema permite a los threads la invocación de funciones proporcionadas por el núcleo. Las llamadas al sistema se definen por medio de funciones API (*Application Program Interface*). Se han propuesto funciones API estándar para permitir la portabilidad de aplicaciones, de entre los que destaca el estándar POSIX para tiempo-real [3].

Uno de los componentes mas importantes del núcleo de un SOTR es el planificador. El planificador es el módulo que asigna tiempo de procesamiento a las tareas de un sistema, por medio de alguna política o algoritmo de planificación. Por ejemplo, existen algoritmos de planificación *expulsivos*, que expulsan del procesador al thread que está en ejecución si algún thread de mayor prioridad ha sido activado. Para llevar a cabo sus funciones correctamente, el planificador utiliza temporizadores e interrupciones del reloj.

La correcta elección de un SOTR es un aspecto fundamental de diseño. Actualmente existen una buena cantidad de sistemas operativos de tiempo-real [1]. Uno de los SOTR que mas atención han recibido recientemente es Preempt-RT, que es un parche de tiempo-real desarrollado originalmente por Ingo Molnar [5]. Este parche permite que el núcleo de Linux sea expulsable, con lo que en cuanto algún thread de alta prioridad es activado, obtiene el procesador con una baja latencia. Preempt-RT es compatible con el estándar POSIX para tiempo-real.

III. Arquitectura del Sistema Propuesto

La arquitectura del sistema propuesta se muestra en la Fig. 4. Como puede observarse, el sistema está compuesto por tres módulos o tareas, que llevan a cabo las siguientes funciones:

1. Captura.- Este módulo recibe de entrada una señal digital de audio (e.g. un archivo wav). Cada una de las muestras son almacenadas en una lista.
2. Procesamiento.- Este módulo es el responsable del procesamiento de la señal para eliminar el ruido, utilizando por ejemplo un filtro adaptativo. Las muestras procesadas son almacenadas en otra lista.
3. Reproducción.- Este módulo se encarga de reproducir la señal que ha sido procesada.

El sistema se basa en el uso de un sistema operativo de tiempo-real, en donde los módulos se implementan como threads periódicos. En su primer activación llevan a cabo un conjunto de actividades de inicialización, como la determinación del tamaño de las listas, el tamaño que requiere cada muestra para ser almacenada, la inicialización de la tarjeta de sonido y la magnitud del periodo de activación de cada thread. Una

[1]hilo

vez que han completado las actividades de inicialización, los threads se activarán periódicamente.

Algunos de los conceptos utilizados en la arquitectura propuesta son los siguientes:

- Resolución (*sample width*): es el número en bits en el cual la muestra es almacenada. Entre mayor sea, mayor será el rango de valores que puede adquirir una muestra y por lo tanto la calidad de audio será mayor. Por ejemplo, si cada muestra de audio es almacenada utilizando 8 bits, el número de combinaciones que puede haber es de hasta 256, o bien si son almacenadas en 16 bits puede ser hasta 65536.

$$resolución = (numero\,de\,bits/(8\,bits))\,bytes = 16\,bits/8\,bits = 2\,bytes \quad (3)$$

- Muestra (*sample*): es la representación numérica de un instante en el tiempo de una señal digital de audio .
- Frecuencia de muestreo (*sample rate*): es el número de muestras tomadas por unidad de tiempo. Este parámetro determina la frecuencia con la que nuestras muestras deben ir reproduciéndose.
- *Frame*: es el conjunto de muestras que componen un instante en el tiempo, considerando todos los canales utilizados por la señal de audio. Por ejemplo, un audio estéreo se compone de 2 canales, por lo que un frame estará compuesto de 2 muestras:

$$frame = numChannels * sample\,in\,bytes = (2\,bytes * 2\,bytes) = 4\,bytes(32\,bits) \quad (4)$$

- Período (*period*): es el número de interrupciones que generará el hardware de la tarjeta de sonido para poder recorrer todo el buffer de la tarjeta de sonido. Para asignar el valor del período utilizamos la equivalencia siguiente:

$$periodo = frame \quad (5)$$

- Tamaño del periodo (*period size*): es el número de *frames* entre cada interrupción del hardware. Esto es, cuántos frames serán reproducidos en cada interrupción.

$$tamaño\,del\,período = 512 \quad (6)$$

- *Buffer*: es el espacio de memoria donde se alojarán las muestras, ya sea para su reproducción o bien para la captura en la tarjeta de sonido (en ALSA es representado como un *buffer* de tipo anillo, viene dado en frames y por lo general siempre viene dado en potencia de 2). El tamaño del buffer está definido como

$$buffer = periodo * tamaño\,del\,período\,(frames)$$

- Tiempo del período: es el tiempo en el cual cada interrupción hará la llamada al sistema operativo. Para esto, es necesario tener como mínimo un período en el *buffer*. Se define como

$$tiempo\,del\,periodo = (tamaño\,del\,período/frecuencia\,de\,muestreo) * 1E9ns \quad (7)$$

A continuación se describen brevemente las principales funciones de las tareas que forman el sistema propuesto.

III-A. Thread de captura (inThread)

Este thread es el encargado de llevar a cabo la captura de cada una de las muestras de audio que se deseen procesar. Como parte de las actividades de inicialización, se lleva a cabo el cálculo del tiempo de su periodo de la siguiente manera:

$$tiempo\,de\,activación = tiempo\,del\,período * 1 \quad (8)$$

De acuerdo a la manera en que se definieron los periodos de las tareas, éste hilo recibirá la menor prioridad ya que tendrá el mayor valor del periodo. Por esta razón, y para evitar que el hilo que procesa la señal no tenga muestras suficientes para procesar, en su primer activación el thread inThread llenará toda la lista *inputList*. Posteriormente, en cada nueva activación almacenará muestras en la lista mientras ésta cuente con espacio disponible. Una vez que se hayan capturado el numero de muestras de acuerdo al tamaño del período asignado a la tarjeta de sonido, el hilo calculará el siguiente instante de su siguiente activación e invocará la la llamada al sistema de `clock_nanosleep()`, que desactivará al thread hasta su siguiente periodo.

III-B. Thread de procesamiento (prThread)

Es el que llevará a cabo las operaciones que implementen el filtro digital que es aplicado a las muestras almacenadas en la lista *inputList*. Cada muestra procesada es almacenada en la lista *outputList*. De igual modo que los demás hilos, al inicio calcula el tamaño de su período utilizando la siguiente expresión:

$$tiempo\,de\,activación = tiempo\,del\,período * 0{,}8 \quad (9)$$

Una vez concluidas sus actividades de inicialización, en cada periodo lleva a cabo las siguientes operaciones:

1. Mientras existan muestras disponibles en *inputList*, así como espacio disponible en *outputList* y aún no se haya excedido el tamaño del periodo de muestras, se procede a llevar a cabo el procesamiento y su almacenamiento en un puntero temporal.
2. Dependiendo del filtro utilizado, el thread leerá *n* muestras de la lista *inputList*. El valor de *n* representa el número de muestra requeridas por el filtro para determinar el valor final de la muestra que se está procesando. En nuestro caso, el valor de *n* es el tamaño de la ventana deslizante utilizado por el filtro adaptativo [6].
3. Cada vez que obtenga una muestra de *inputList*, éste siempre quedara en el del conjunto de muestras que se enviaran al filtro.

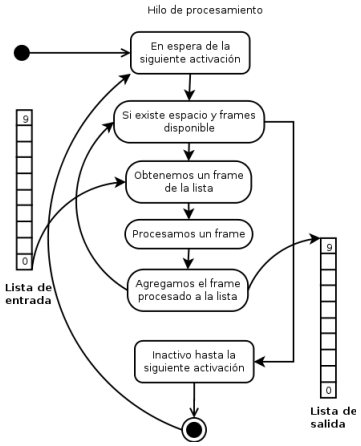

Figura 5. Actividades del thread de procesamiento

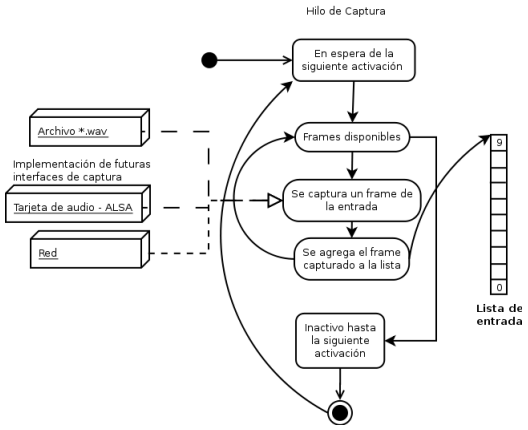

Figura 6. Actividades del thread de entrada

4. El proceso se llevará a cabo hasta que el numero de muestras procesadas sea igual al tamaño del periodo asignado a la tarjeta de audio.
5. Se calculará el siguiente instante de activación de la tarea.

III-C. Thread reproductor (ouThread)

En la primer activación de este thread, se inicializa el puntero en donde se almacenará las muestras del periodo, para su posterior envío al *buffer* de la tarjeta de sonido. De igual modo, se calcula el tiempo del periodo para el hilo reproductor.

$$tiempo\,de\,activación = tiempo\,del\,período * 0{,}6 \qquad (10)$$

Como parte de las actividades de inicialización, el thread deberá esperar hasta que la lista *outputList* se encuentre totalmente llena, para comenzar con la respectiva reproducción de cada conjunto de muestras. Una vez concluidas las operaciones de inicialización, en cada activación el thread reproductor hará lo siguiente:

1. Mientras existan muestras disponibles en *outputList* y no se haya excedido el tamaño de las muestras por periodo, se pasará cada una de las muestras a un *buffer* temporal para su posterior reproducción.

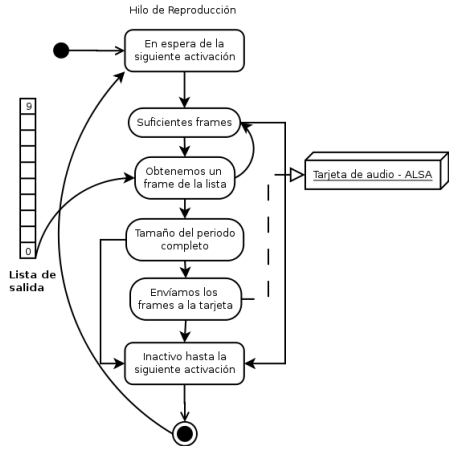

Figura 7. Actividades del thread de salida

Thread	Periodo	WCET
inThread	0.0640	0.000340339
prThread	0.0512	0.005350114
ouThread	0.0384	0.000639310

Cuadro I
PARÁMETROS DE LOS THREADS (SEGUNDOS)

2. Se verifica si el número de *frames* disponibles para reproducir es mayor o igual al tamaño del periodo asignado inicialmente a la tarjeta de sonido, así como también que el tamaño del *buffer* temporal sea igual al numero de muestras en un periodo. En ese caso se podrá enviar al *buffer* de la tarjeta de sonido todo un periodo completo.
3. Si aún no se tiene un periodo completo lleno de muestras, se continúa hasta completar el número de muestras necesarias.

IV. IMPLEMENTACIÓN

El sistema se basa en el uso de Preempt-RT, y en el uso de la política de planificación *rate-monotonic* [4], que asigna mayor prioridad a las tareas mas frecuentes. Debido a que las muestras se almacenan en listas que son compartidas por los threads, se utiliza el protocolo de sincronización de herencia de prioridades (*priority inheritance protocol*) [8].

Para el desarrollo de la aplicación se utilizó el lenguaje de programación C y el API de tiempo-real del estándar POSIX, el cual ofrece además de su eficiencia, compatibilidad con sistemas basados en UNIX. Para la reproducción del sonido utilizó del API de ALSA, que el sistema de audio utilizado por los sistemas Linux.

La aplicación crea dos listas circulares enlazadas estáticas, que son utilizadas para el almacenamiento temporal de las muestras que se reciben de entrada, así como de las muestras que estén listas para reproducirse. Estas listas son llamadas *inputList* y *outputList*. Para el acceso a las regiones críticas se utilizan semáforos (mutex).

La implementación del sistema propuesto se llevó a cabo en una computadora con procesador Intel(R) Pentium(R) 4 a 2.80GHz, con 1Gb de memoria RAM y una tarjeta de sonido

Realtek ALC861. La señal de audio utilizada se tomó de una archivo con formato *wav*, con una señal digital de un canal, con una frecuencia de 8,000 Hz y una resolución de 16 bits por muestra. Se utilizó el filtro adaptativo propuesto por Sandoval *et al*. en [6].

Para el procesamiento de la señal de audio se calcularon los periodos y los tiempos de ejecución en el peor caso (WCET) de los threads, que se muestran en el Cuadro I. La utilización del procesador de cada tarea se define como $u = {}^{WCET}/_{Periodo}$, mientras que la utilización total del sistema formado por *n* tareas está definida como $U = \sum_{i=1}^{n} u_i$. Utilizando los valores del Cuadro I se tiene que la utilización total del sistema es 12.65 %.

V. Conclusiones y trabajo futuro

El procesamiento de audio consiste en incrementar la calidad de las señales sonoras en términos de la inteligibilidad y de la reducción de ruido. Existen algoritmos de procesamiento de audio que son computacionalmente eficientes, pero que sacrifican la inteligibilidad al tratar de suprimir el ruido. Este problema puede solucionarse al utilizar técnicas de procesamiento localmente adaptativo, pero tienen el inconveniente de ser computacionalmente muy costosos.

En este artículo se propuso un sistema de tiempo-real para el procesamiento digital de audio. El sistema puede ser implementado en dispositivos que utilicen un procesador de propósito general, que ofrecen reducción de espacio, bajo consumo de energía y un bajo costo. El sistema está basado en el uso de un sistema operativo de tiempo-real denominado Preempt-RT, que es una variante del popular sistema operativo Linux. Los resultados experimentales mostraron que el sistema propuesto es capaz de procesar eficientemente señales de audio empleando técnicas de procesamiento localmente adaptativo, con un bajo costo computacional.

Como trabajo futuro se planea la implementación del la arquitectura propuesta en un sistema inmerso, así como la integración de capacidades de procesamiento de vídeo.

Referencias

[1] Sanjeev Baskiyar and Natarajan Meghanathan. A survey of contemporary real-time operating systems. *Informatica (Slovenia)*, 29(2):233–240, 2004.

[2] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages*. Addison Wesley Longmain, April 2009.

[3] 2008 Edition IEEE Std 1003.1. *The Open Group Technical Standard Base Specifications, Issue 7. Base Definitions*. Institute of Electrical and Electronic Engineers and The Open Group, Dec, 2008.

[4] Liu, C.L. and Layland, W. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, May 1973.

[5] S. Rostedt and D. V. Hart. Internals of the rt patch. In *Proceedings of the Linux Symposium*, pages 161–172, Ottawa, Canada, June 2007.

[6] Yuma Sandoval-Ibarra, Victor H. Diaz-Ramirez, and Juan J. Tapia-Armenta. Algoritmo de orden localmente-adaptativo para la mejora de senales de voz. *Congreso Internacional en Ciencias Computacionales (CiComp)*, 3:185–190, 2010.

[7] Wenshun Tian Sen M Kuo, Bob H Lee. *Real Time Digital Signal Processing - Implementations and Applications*. Second edition, 2006.

[8] Sha, L., Rajkumar, R., and Lehoczky,J.P. Priority inheritance protocols: An approach to realtime synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, Sep 1990.

[9] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Fundamentos de Sistemas Operativos*. McGraw-HIll, 2006.

VI. Biografías

Vidblaín Amaro Ortega nació en Puebla, Puebla, el22 de Enero de 1987. Actualmente es estudiante de la Maestría en Ingeniería Electrónica del Instituto Tecnológico de Mexicali. Se graduó de la carrera de Ingeniería en Sistemas Computacionales en la misma institución en el 2008. Sus áreas de interés son las los Sistemas de Tiempo-Real y el Procesamiento Digital de Señales.

Arnoldo Diaz-Ramirez nació el 1ro de de marzo de 1964 en Mexicali, Baja California. Obtuvo el grado de Doctor en Ciencias por la Universidad Politécnica de Valencia, España, y el grado de Maestro en Tecnología de Redes e Informática por el Cetys Universidad, Campus Mexicali. Estudio la carrera de Ingeniería en Ciencias Computacionales en esa misma universidad. Actualmente es Profesor-Investigador del Instituto Tecnológico de Mexicali. Sus áreas de interés son los Sistemas de Tiempo Real, las Redes de Sensores Inalámbricas y la Computación Ubicua.

Víctor H. Díaz Ramírez nació el 29 de Agosto de 1976. Obtuvo el grado Maestro en Ingeniería Electrónica por el Instituto Tecnológico de Mexicali en 2003 y el grado de Doctor en Ciencias en Ciencias de la Computación por el Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE) en 2007, México. Actualmente es profesor del Instituto Politécnico Nacional (CITEDI), México. Sus temas de investigación incluyen Procesamiento Digital de Señales e Imágenes, Reconocimiento de Patrones y Procesamiento Opto-Digital de Información.

Heber Samuel Hernández Tabares nació en Orizaba, Veracruz, el 8 de marzo de 1966, estudio la Licenciatura en Informática en el Instituto Tecnológico de Orizaba, posteriormente realizo un posgrado en Cetys Universidad en Tecnología de Redes e Informática. Actualmente se desempeña como Coordinador de Proyectos de Investigación en la División de Posgrado del Instituto Tecnológico de Mexicali. Sus áreas de interesé son las Redes Inalámbricas y la Computación Ubicua.