ANÁLISIS COMPARATIVO DE LOS TIEMPOS DE EJECUCIÓN SOBRE SBC PARA DOS SISTEMAS OPERATIVOS DE TIEMPO REAL

Diana Lizet González Baldovinos

Instituto Politécnico Nacional ESIME Culhuacan

glez_lizet@hotmail.com

Jose Luis Cano Rosas

Instituto Politécnico Nacional ESIME Culhuacan

lucskyr@gmail.com

Pedro Guevara López

Instituto Politécnico Nacional ESIME Culhuacan

pguevara@real-time.com.mx

Resumen

El análisis del desempeño de los Sistemas Operativos de Tiempo Real (SOTR) permite aplicarlos adecuadamente en diferentes ámbitos y desarrollar posibles aplicaciones de tiempo real de manera efectiva. En este sentido, el presente trabajo considera los tiempos de ejecución como parte del desempeño total del sistema operativo; para ello se plantea la comparativa de respuesta de los tiempos de ejecución generados al realizar una tarea de alta complejidad temporal, implementada en dos extensiones de Tiempo Real basadas en Linux, particularmente PREEMPT_RT y Xenomai. La tarea consiste en un algoritmo basado en inversión de matrices, mediante el método de Gauss-Jordan con dimensión de 2^n, en el rango desde 2×2 hasta 128×128. La finalidad de este trabajo, es mostrar que los tiempos de ejecución, son parte del desempeño total de los Sistemas Operativos de Tiempo Real y que un mismo hardware puede tener diferente desempeño con diferentes extensiones de tiempo real, al manejar prioridad y políticas de planificación. Para los experimentos se utilizó una SBC (Single Board Computer) Raspberry Pi 3.

Palabras Claves: Complejidad Temporal, inversión de matrices, SBC, SOTR, tiempos de ejecución.

Abstract

The analysis of Real-time Operating Systems (RTOS) performance allows to be properly apply them in different environments and development real-time applications. In this sense, this paper consider the response of computing times as part of total performance of the operating system. So, we propose the comparison of execution times response generated by a task of high temporal complexity, implemented in two Real-time extensions based on Linux, PREEMPT_RT and Xenomai. The task is created by an algorithm based on Gauss-Jordan of matrix inversion method, with dimension of 2^n, in the range of 2 × 2 to 128 × 128. The goal of this paper is to demonstrate that execution times, are part of the total performance of Real-time operating systems and the same hardware could have different performance with two real-time operating system extensions, modifying priority level and scheduling policies. For the experiments we used an SBC (single board computer) Raspberry Pi 3.

***Keywords*:** *Execution times, temporal complexity, matrix inversion, RTOS, SBC.*

1. Introducción

Las computadoras embebidas o SBC como Raspberry Pi, pueden emplearse para el desarrollo de sistemas en tiempo real; Sin embargo, generalmente cuentan con un sistema operativo de tiempo compartido, donde su núcleo (kernel) puede adecuarse para dar soporte de tiempo real. Los Sistemas Operativos de Tiempo Real dan preferencia a los procesos del mundo físico sobre el usuario, brindan soporte para el manejo de tareas y sincronización, manejo de memoria, comunicación entre procesos, manejo de dispositivos, así como del reloj de la computadora. Existen algunas extensiones de Tiempo Real basadas en Linux, tales como: PREEMPT_RT y Xenomai. PREEMPT_RT es una extensión de tiempo real, que ejecuta al kernel de Linux como un thread (hilo de ejecución) de menos prioridad que las tareas de tiempo real. Con este diseño, las tareas de

tiempo real y los manejadores de interrupciones nunca se ven retrasados por operaciones que no son de tiempo real. Estas tareas y los manejadores, ejecutan cuando se necesitan en detrimento de lo que estuviera ejecutando Linux.

Xenomai es un proyecto de software libre desarrollado en 2001, que implementa un framework de tiempo real en plataformas Linux [Xenomai, 2017]. Por lo anterior, se puede afirmar que Xenomai no es un sistema operativo de tiempo real, en cambio, es una extensión que permite realizar tareas en tiempo real y cuya criticidad dependerá de la integridad del sistema en su totalidad y no únicamente de la extensión, es decir, que podrá trabajar con limitantes. Xenomai utiliza un kernel dual mediante una extensión al kernel nativo de Linux, mismo que se encarga de atender interrupciones y planificar tareas de tiempo real [Xenomai, 2017].

Los trabajos que han servido como punto de referencia para esta investigación, se mencionan a continuación: Del trabajo Doctoral [Valdez, 2015], se toma el estudio sobre los tiempos de ejecución, donde se menciona que c_k es el tiempo en que se procesa la información en el intervalo k hasta completarse el procesamiento, sin considerar los bloqueos por lectura o escritura en los canales de comunicación, desalojos del procesador u otro tipo de suspensiones. Se menciona, que el tiempo de respuesta del sistema en tiempo real, depende en gran parte del comportamiento de los tiempos de ejecución $c_{i,k}$ de cada instancia $j_{i,k}$ de cada tarea en tiempo real J_i, el cual varía debido a diversos factores. Éste trabajo también hace referencia a [Stappert, 2000] y [Bernat, 2003], donde los autores exponen que el comportamiento fluctuante de los tiempos de ejecución $c_{i,k}$, se debe a los factores computacionales como son: el caching, pipeline, la búsqueda de la ruta de ejecución considerando a la exclusión mutua, número de lazos, predicciones y otras interacciones.

De acuerdo al trabajo de [Brown, 2010], se presentó un sistema de prueba para evaluar el rendimiento de tres núcleos; el kernel base de Linux, el mismo kernel de Linux con parche PREEMPT_RT y el mismo kernel de Linux pero con el parche de Xenomai aplicado. Se midieron los tiempos para realizar dos tareas; La primera tarea consistió en alternar una salida de E/S de propósito general (GPIO) en un

período fijo. La segunda tarea consistió en responder a un pin GPIO de entrada cambiante haciendo que el valor de un pin de salida GPIO lo siga. Para esta tarea, en lugar de realizar sondeos, utilizaron una interrupción para notificar cuándo cambia la entrada GPIO. Todas estas pruebas fueron realizadas en el sistema embebido BeagleBoard, los autores exponen que el kernel con parche PREEMPT_RT, es el que tiene un mejor tiempo de respuesta cuando se alterna una salida de GPIO.

En el trabajo de [Parikh, 2013] se analiza un panorama fundamental de los parámetros que determinan el rendimiento de algunos Sistemas Operativos de Tiempo Real. Con esta investigación, se observó que la planificación es el parámetro que más influye en el desempeño de los SOTR. Además, el documento proporciona una descripción intuitiva de tres Sistemas operativos de Fuente abierta, los cuales son: RT-Linux, FreeRTOS y eCos. Los autores afirman que el planificador, es el parámetro más importante que rige el funcionamiento de todo el sistema, incluido el hardware. Los sistemas de tiempo real crítico, requieren interrupciones y latencias extremadamente bajas y para ello se necesita un sistema de prevención con un mecanismo eficiente de manejo de interrupciones. RT-Linux es el SOTR más prometedor de los tres, ya que proporciona amplias funcionalidades sin comprometer lo esencial de un SOTR.

2. Métodos

La parte principal en el desarrollo de este trabajo, es experimentar el efecto de los tiempos de ejecución en una SBC Raspberry Pi 3, con dos extensiones de tiempo real basadas en Linux.Para ello, se ha desarrollado e implementado un algoritmo programado en ANSI C denominado matrices.h, descrito como una biblioteca de funciones para la inversión de matrices basada en los fundamentos teórico-prácticos que caracterizan a la matriz y su inversión, empleando principalmente el método de Gauss-Jordan.

El algoritmo está elaborado con el propósito de generar carga computacional y se emplea para hacer la medición de los tiempos de ejecución. Dicho algoritmo, fue implementado en los dos sistemas operativos de tiempo real, en cada instancia se

realiza la inversión de la matriz con dimensión de 2^n, abarcando desde 2×2 hasta 128×128. La medición de los tiempos de ejecución se realiza dentro del algoritmo, para efectuar las mediciones, existe una serie de funciones dentro de la biblioteca time h.

La función clock(), es una herramienta de los sistemas POSIX, también se encuentra la función time(), ClockTime() y clock_gettime(), siendo esta última la que se usa en esta investigación. La diferencia de esta última con la función clock(), es que la anterior trabaja con múltiplos enteros de "ticks" de reloj y registra valores en el orden de los microsegundos, en cambio con la función clock_gettime(), la resolución puede ser ajustada permitiendo obtener valores más precisos de los tiempos de ejecución, medidos en el orden de los nanosegundos [Cano, 2015]. En la figura 1 se muestra el fragmento de código donde se realiza la medición del tiempo de ejecución.

```
for (k=0; k<ITERACIONES; k++){
clock_gettime(CLOCK_REALTIME, &inicio);
matriz_inversa=inversa(matriz, M);
clock_gettime(CLOCK_REALTIME,&fin);
milisegundos_i=(inicio.tv_sec*1000.0)+((double)inicio.tv_nsec/1000000.
0);
milisegundos_f= (fin.tv_sec*1000.0) + ((double)fin.tv_nsec/1000000.0);
ejecucion= milisegundos_f-milisegundos_i;
printf ("c(%d)= %f milisegundos\n", k, ejecucion);
sprintf(buffer, "\n%f\n", ejecucion);
write(des, buffer, sizeof(buffer));
}
```

Figura 1 Medición de los tiempos de ejecución para la inversión de una matriz.

Los tiempos de ejecución medidos se guardan en un archivo .txt para posteriormente ser graficados fuera de línea en el software Octave. En el análisis se consideran los primeros dos momentos de probabilidad, el primer momento de probabilidad es la media o valor esperado de una variable aleatoria y es denotada por μ. La función con la cual se obtiene, se expresa en la ecuación 1.

$$E\{x_i\} = \sum_{i=1}^{n} \frac{x_i}{n} = \mu \qquad (1)$$

El segundo momento de probabilidad o varianza de una variable aleatoria, es una medida de la dispersión de sus valores alrededor de la media μ y se denota por σ^2 [Bernat, 2003]. La función con la cual se obtiene, se define mediante ecuación 2.

$$E\{x_i\}^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu)^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \left[\frac{1}{n}\sum_{i=1}^{n} x_i\right]^2 = \sigma^2 \quad (2)$$

Las métricas que se emplean para realizar la comparativa de los tiempos de ejecución en cada SOTR, consisten en el manejo de prioridades y manejo de política de planificación. Se hace uso del manejo de prioridad por amabilidad, empleando el comando nice, la cual consiste en establecer la preferencia de ejecución de un proceso, la escala de prioridad está estandarizada con los valores de -20 a 19 donde el negativo representa la más alta prioridad y el positivo indica baja prioridad. Para establecer la máxima preferencia de ejecución al proceso, se ingresa la siguiente línea de comando desde la terminal: sudo nice –n -20 ./programa. Al utilizar el comando nice se observa que los tiempos de ejecución disminuyen un poco, sin embargo pueden disminuir aún más. Por lo tanto se hace uso de la función setpriority()que se encuentra dentro de la librería sys/resource.h, la función setpriority() requiere tres argumentos int setpriority(int *which*, **id_t** *who*, **int** *value*) los procesos de destino se especifican mediante los valores de los argumentos which y who. El primer argumento puede ser uno de los siguientes valores: PRIO_PROCESS, PRIO_PGRP o PRIO_USER, el segundo argumento who debe interpretarse como un ID de proceso, un ID de grupo de proceso o un ID de usuario efectivo, respectivamente, finalmente el tercer argumento indica el valor de prioridad [Linux, 2017], en la figura 2 se muestra el uso de la función.

```
#include <sys/resource.h>
...
int which = PRIO_PROCESS;
id_t pid;
int priority = -20;
int ret;

who = getpid();
ret = setpriority(which, who, priority);
```

Figura 2 Uso de la Función setpriority().

Por otro lado se emplea también la llamada al sistema sched_setscheduler() la cual se encuentra en la librería sched.h, ésta llamada establece tanto la política de planificación como los parámetros para el hilo cuyo ID se específica en el pid, en la figura 3 se muestra el uso de la función.

```
#include <sched.h>

int sched_setscheduler(pid_t pid, int
policy, const struct sched_param *param);
```

Figura 3 Uso de la Función sched_setscheduler().

En este trabajo se utilizaron dos políticas de planificación de tiempo real, éstas políticas son SCHED_FIFO una política de primero en entrar, primero en salir; y SCHED_RR una política round-robin, los valores de prioridad que manejan respectivamente van desde 1 a 99 donde 99 es la más alta prioridad [SCHED, 2017].

3. Resultados

En el siguiente apartado, se presenta un compendio de figuras donde se muestran los resultados experimentales de la medición de los tiempos de ejecución, al realizar inversión de matrices con manejo de prioridades en PREEMP RT y Xenomai. Se expone la diferencia de tiempo de ejecución de la misma tarea, para 1000 instancias efectuadas en la misma SBC.

PREEMPT RT

Figuras 4 y 5 muestran el resultado de los tiempos de ejecución con prioridad *nice* para la inversión de matrices de diferentes dimensiones, en este caso se introdujo desde la terminal la línea de comando: *sudo nice –n -20 ./ejecución*.

Puesto que el propósito de este trabajo es mostrar una comparativa de los tiempos de ejecución cuando se ejerce alta carga temporal, se han elegido las gráficas que muestran los tiempos al invertir matrices de dimensiones 64×64 y 128×128, con el

respectivo manejo de prioridades. En la figura 6 se observan los tiempos de ejecución al invertir matrices de 64×64, el valor de prioridad asignado para setpriority() y *nice* es -20, y el valor para sched_setscheduler() de acuerdo a la política de planificación FIFO y Round Robin se estableció en 95.

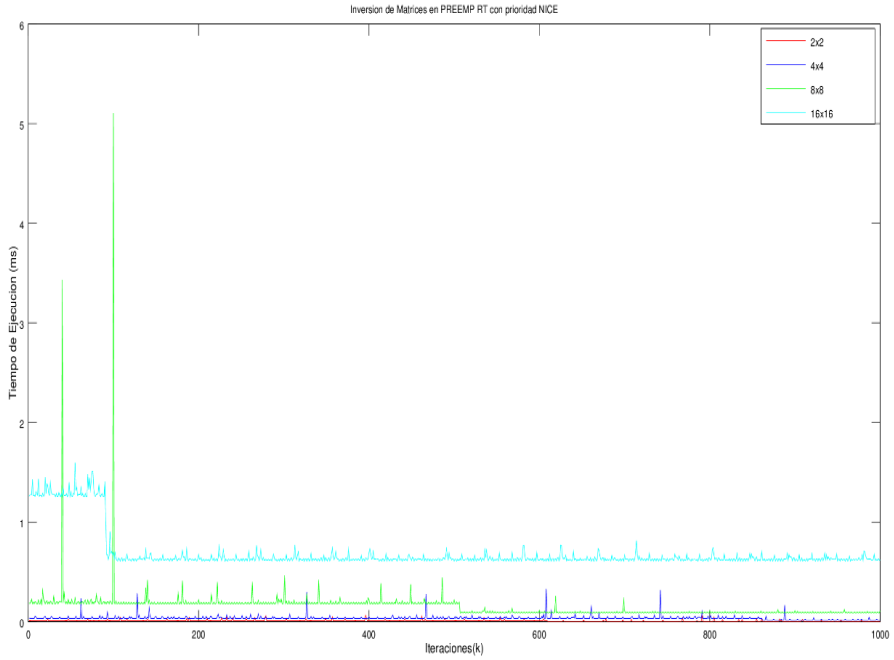

Figura 4 Inversión de matrices desde 2×2 hasta 16×16, PREEMP_RT prioridad *nice*.

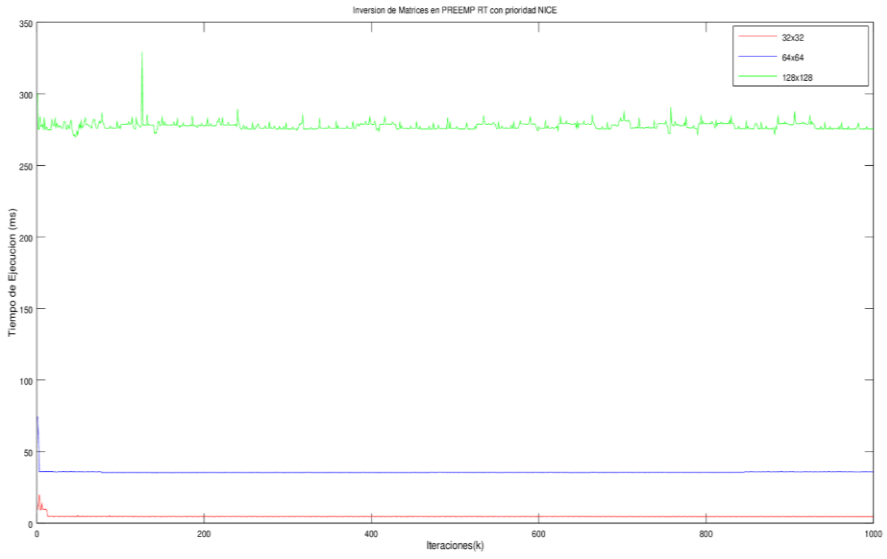

Figura 5 Inversión de matrices desde 32×32 hasta 128×128, PREEMP RT prioridad *nice*.

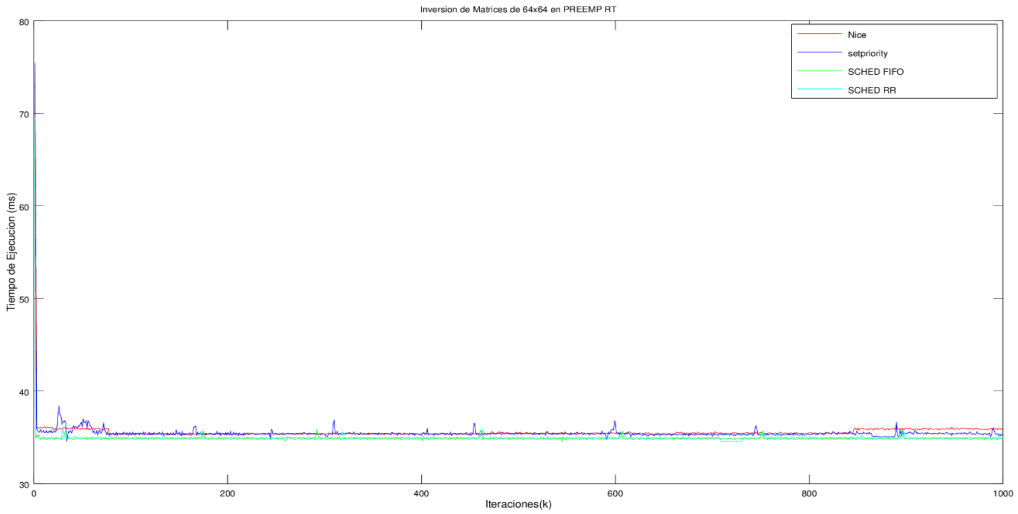

Figura 6 Tiempos de ejecución inversión de matrices de 64×64, manejo de prioridades.

En la figura 7 se muestran los tiempos de ejecución al invertir matrices de 128×128, con los valores de prioridad mencionados anteriormente.

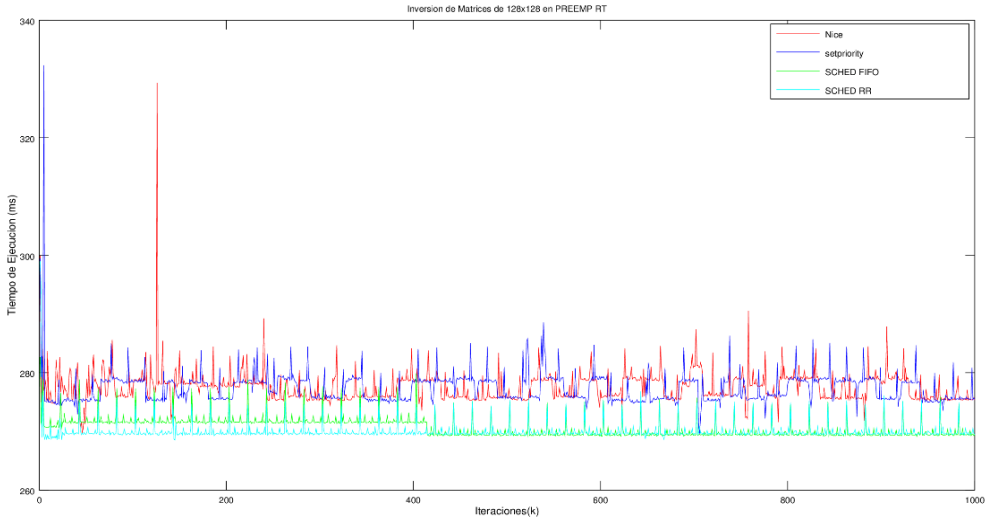

Figura 7 Tiempos de ejecución inversión de matrices de 128×128, manejo de prioridades.

Con esta extensión de tiempo real, se observó que el manejo de prioridad por planificación Round Robin presenta tiempos de ejecución menores a diferencia de los demás. En la figura 8 se presentan la media y varianza de los tiempos de ejecución de la inversión de matrices de 128×128 con prioridad por planificador Round Robin.

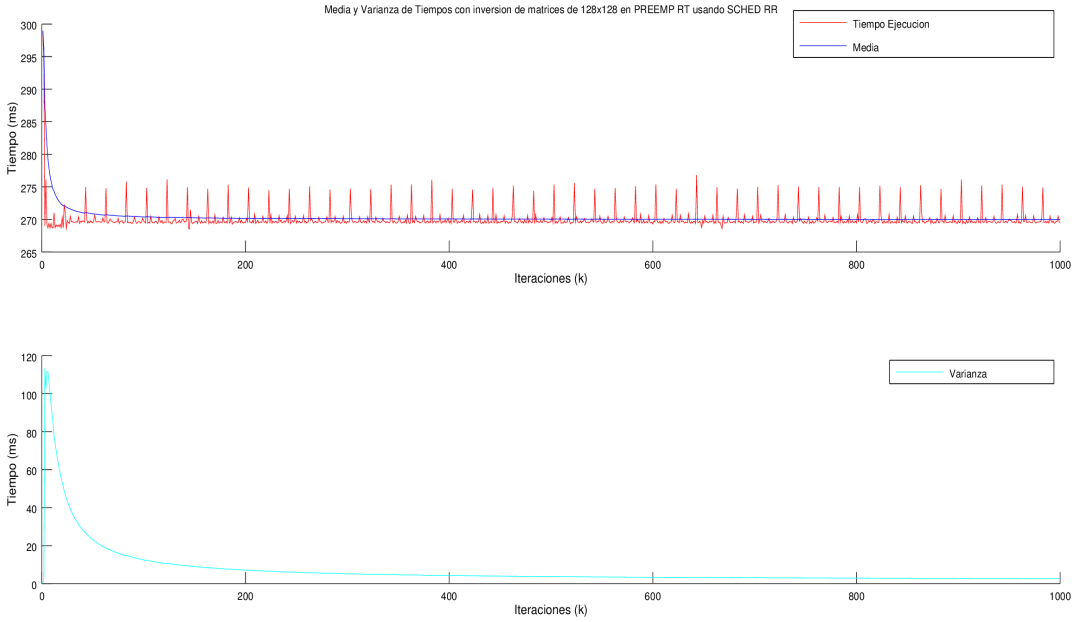

Figura 8 Media y Varianza de Tiempos e ejecución de inversión de matrices de 128×128, con prioridad Round Robin.

Xenomai Los tiempos de ejecución obtenidos, son considerablemente mayores a los que se obtuvieron en PREEMPT_ RT, a continuación se presentan las gráficas resultantes al invertir matrices de dimensión 64×64 y 128×128. En figuras 9 y 10 se muestran los tiempos de ejecución, con los valores de prioridad nombrados inicialmente.

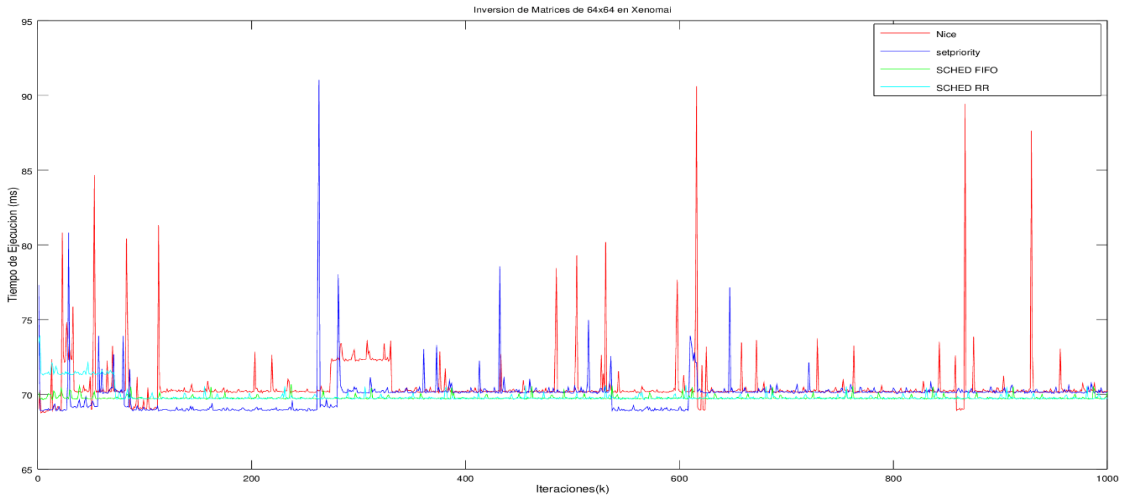

Figura 9 Tiempos de ejecución inversión de matrices de 64×64, manejo de prioridades.

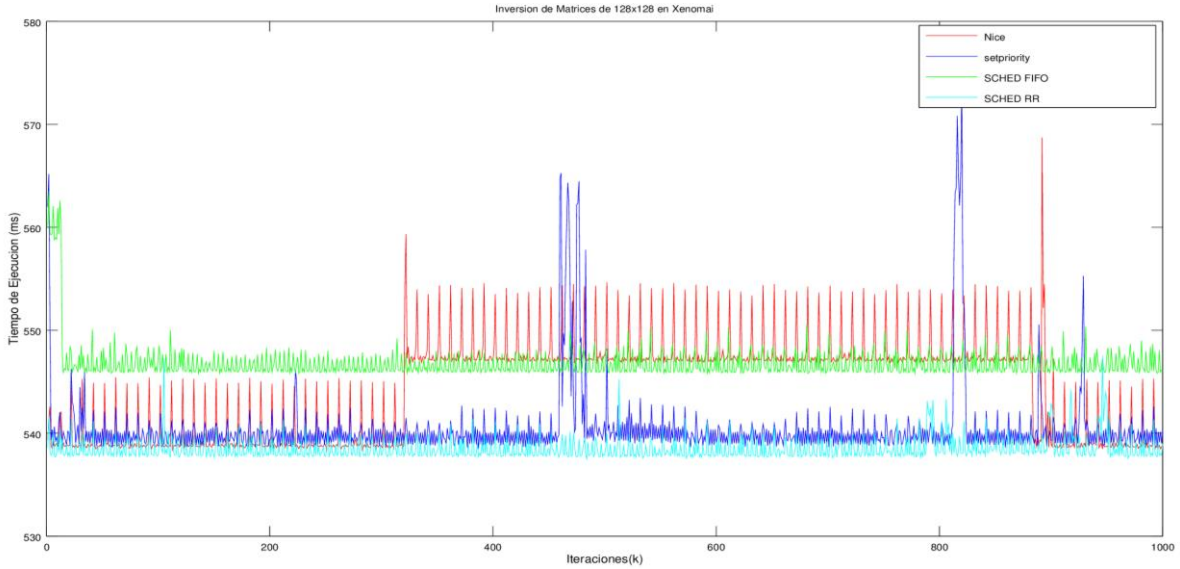

Figura 10 Tiempos de ejecución inversión de matrices de 128×128, manejo de prioridades.

De igual forma que en PREEMPT_ RT el manejo de prioridad por planificación Round Robin presenta tiempos de ejecución menores a diferencia de los demás. En la figura 11 se presentan la media y varianza de los tiempos de ejecución de la inversión de matrices de 128×128 con prioridad por planificador Round Robin.

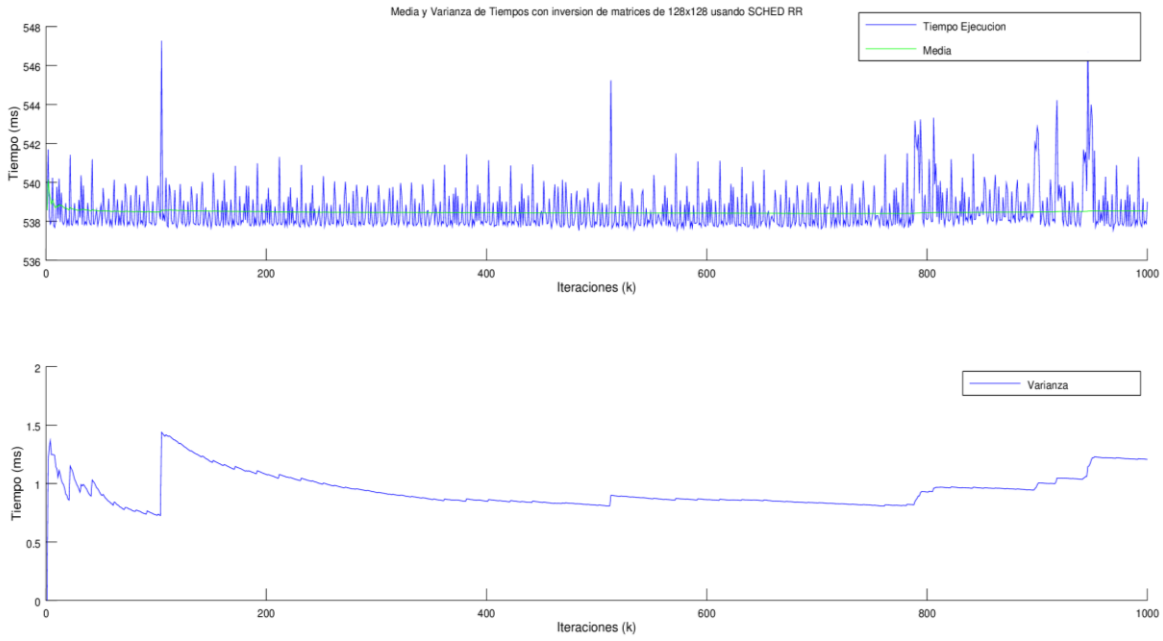

Figura 11 Media y Varianza de Tiempos de ejecución de inversión de matrices de 128×128, con prioridad por planificador Round Robin.

4. Discusión

El comportamiento de los tiempos de ejecución, de acuerdo al algoritmo de inversión de matrices con alta complejidad computacional temporal, realizado en los sistemas operativos de tiempo real basados en el kernel de Linux, tiene una diferencia sustancial en cuanto a tiempo de procesamiento, como se ilustra en la figura 12. Es posible comprobar que en PREEMPT_ RT los tiempos de ejecución son menores, debido a que esta extensión trata al kernel de Linux como una tarea de menor prioridad y en contraste Xenomai trabaja con kernel dual. Por lo tanto, hace que los recursos se dividan para ambos y el tiempo de procesamiento sea mayor. También se observa que los tiempos de ejecución en PREEMPT_ RT no son tan variables, esto otorga mayor estabilidad y fiabilidad al sistema cuando se aplica alta carga computacional temporal, a diferencia de Xenomai que muestra gran variabilidad en los tiempos.

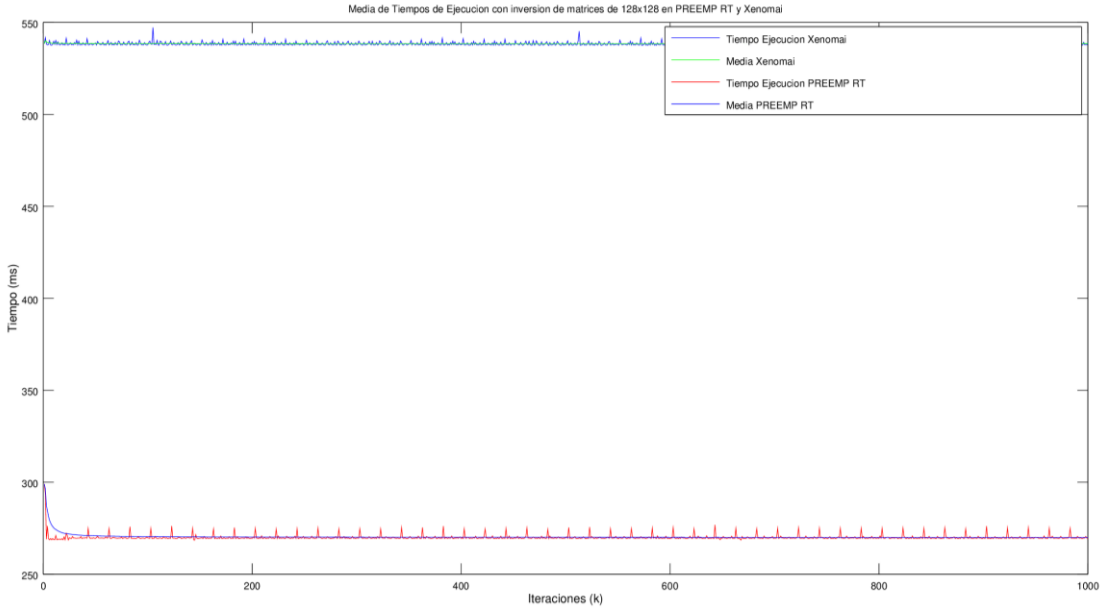

Figura 12 Media y Varianza de Tiempos de ejecución inversión de matrices de 128×128, en PREEMP RT y Xenomai con prioridad por planificador Round Robin.

5. Conclusiones

Las computadoras basadas en placa reducida (SBC) como Raspberry Pi, son una excelente alternativa para el desarrollo de sistemas de tiempo real embebidos;

algunas razones son su pequeño tamaño, bajo consumo energético y sobretodo el soporte para software libre; todo esto hace que el resultado final sea modular y de muy bajo costo. En este proyecto, se instaló en la SBC Raspberry Pi 3 el sistema operativo Raspbian, que es un derivado del sistema operativo GNU/Linux Debian con características de tiempo compartido. Para dar soporte de tiempo real se utilizaron PREEMPT_RT y XENOMAI, ambas extensiones se basan en arquitecturas y funcionamiento diferente; para analizar su comportamiento sobre los tiempos de ejecución se llevó a cabo este estudio experimental donde se manipularon prioridades, se midieron tiempos de ejecución y se realizó un análisis cuantitativo basado en los primeros momentos de probabilidad de las mediciones, utilizando como banco de pruebas un algoritmo de complejidad NP, como la inversión de matrices por el método de Gauss-Jordan. Los resultados obtenidos reflejaron que en PREEMPT_ RT los tiempos de ejecución fueron menores debido a que esta extensión trata al kernel de Linux estándar como una tarea de menor prioridad, en contraste con Xenomai cuya ejecución fue de mayor demanda temporal. Pudo observarse que en PREEMPT_RT los tiempos tuvieron mayor estabilidad, es decir, su varianza no fue demasiado alta, lo cual determina mayor confiabilidad en ésta extensión, cuando se genera alta carga temporal.

6. Bibliografía y Referencias

[1] Bernat G, Colin A, Petters SM, PWCET: A tool for probabilistic Worst-Case Execution Time Analysis of Real-Time Systems. Technical Report YCS-2003-353. Department of Computer Science, University of York, Reino Unido, York, pp. 1-18, 2003.

[2] Brown, J. H., & Martin, B., How fast is fast enough Choosing between Xenomai and Linux for real-time applications. In proc. Of the 12th Real-Time Linux, 2010.

[3] Cano Rosas J. L., Efecto Del Overclocking Sobre Los Tiempos De Ejecución Generados Por Inversión De Matrices En Una Computadora Embebida. Sección de Estudios de Posgrado e Investigación, ESIME Culhuacan IPN, 2015.

[4] Linux, Linux Setpryority Man Page, 2017: https://linux.die.net/man/3/setpriority, May 2017.

[5] Parikh, H., Shah, R., Shah, U., & Deshmukh, S., Performance parameters of RTOSs; comparison of open source RTOSs and benchmarking techniques. In Advances in Technology and Engineering (ICATE), 2013 International Conference on IEEE, pp. 1-6, 2013.

[6] Stappert F and Altenbernd P., Complete Worst-Case Execution Time Analysis of Straight-line Hard Real-Time Programs. Journal of Systems Architecture V. 46, I. 4, pp. 339–355, 2000.

[7] SCHED SETSCHEDULER, Linux Programmer's Manual: http://man7.org/linux/manpages/man2/sched_setscheduler.2.html, May 2017.

[8] Valdez, J. S. (2015). Medición, Caracterización y Reconstrucción de los Tiempos de Ejecución y Tiempos de Transporte para Sistemas de Telecontrol en Tiempo Real. Tesis Doctoral. Sección de Estudios de Posgrado e Investigación. ESIME Culhuacan. IPN.

[9] Xenomai, About Xenomai, Xenomai.org, 2017: https://xenomai.org/about-xenomai/, May 2017.

[10] Xenomai, Start Here, Xenomai.org, 2017: https://xenomai.org//start-here/#How_does_Xenomai_deliver_real-time, May 2017.