Indian Journal of Science and Technology, Vol 8(19), DOI:10.17485/ijst/2015/v8i19/77017, August 2015
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Performance Analysis of Real Time Operating System with General Purpose Operating System for Mobile Robotic System

Akhilesh Murikipudi*, V. Prakash and T. Vigneswaran

School of Electronics Engineering, VIT University, Chennai–600127, India; murikipudi.akhilesh2013@vit.ac.in, prakash.v@vit.ac.in, vigneswaran.t@vit.ac.in

Abstract

Background/Objectives: The objective of the paper is to analyze the General Purpose Operation System (GPOS) performance with Real Time Operating System(RTOS) using a mobile robot as a real time application. The mobile robot module is implemented on a single board computer having ARM11 as its core. **Methods/Statistical Analysis:** The method used to calculate response is realfeel method which uses dedicated timer and interrupt to calculate the response. **Findings:** The response of the mobile robot is calculated by interrupting the mobile robot with some obstacles. Findings:In addition to this, other parameters such as speed, rotations time and precision resulted by the sensor are also calculated. RTOS have an average of 20µs where as GPOS have an average of 102µs of response time in real time environment.**Conclusion:**The results show that RTOS has better response time than GPOS. The minimum distance required to stop the mobile robot in RTOS is more accurate than GPOS. Compare to GPOS, the RTOS is able to avoid the obstacle collision even for a shorter distance.

Keywords: GPOS, Latency Calculation, Mobile Robot, Response Time, RTOS, Single Board Computer

1. Introduction

Timing is the most important factor in any real time applications where some applications should react in very small amount of time in those situation applications needs a platform where the timing constrain was meet. In real-time systems, all real-time tasks are differentiated based on their timing, such as sporadic, response time, deadline etc. Real time systems are classified in to two types' hard real-time systems and soft real time systems. Hard real time system means it should complete the task with in the deadline period otherwise its computation is useless. The damages caused by the hard real time systems are irreparable. The system builder's should responsible to choose an operating system that can support and schedule these jobs with respect to their timing criteria so that no deadline will be missed. Soft real time systems require performance assurances from the operating system.

Some applications such as video/audio visual gaming require performance assurance and also acceptable jitter in timing these applications comes under soft real-time applications. The general architecture ofRTOS is shown in Figure 1.

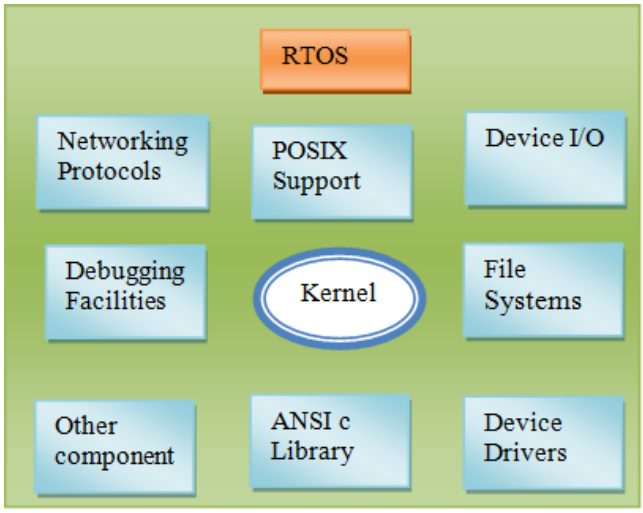

Figure 1. General architecture of RTOS.

* Author for correspondence

Any RTOS basically have the characteristics like Multitasking and Preemptibility, Task Priority, Priority Inheritance, Short Latencies (Task switching latency Interrupt latency, Interrupt dispatch latency), Reliable and Sufficient Inter Task Communication Mechanisms, Control of Memory Management.

The general purpose operating system for the proposed research work is based on Linux flavor (RaspbianOS). Manyresearches from the past few years used Linux as their platform Figure 2 it is inferred that, use of Linux is increasing in developing countries where technology increases[1].

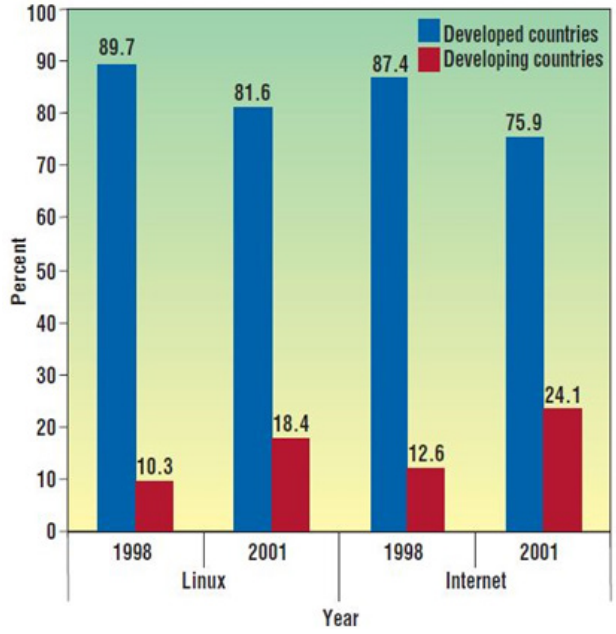

Figure 2. Use of Linux and internet in developed and developing countries.

Increasing complexity of devices and systems, huge competition, shorter time to Market and to reduce the cost of development requirement makes to use high-end software. Acatle began to develop OXO systems. OXO belongs to hard real time systems where it must respond to the event in predefined period whatever the load stress; this makes them to think that existing real time kernel is inadequate to support all the expected services. Development of the OXO systems was divided in to three classes' non real time and soft real time Linux processes in user space and hard real time processes in kernel space and RTLinux shares the kernel space as well[2]. A comparison of windows patched with RT kernel and Linux patched with RTkernel is also carried out in this development on services like voice services, Datapluse internet services, Software development environment, cost and support these results are shown in Table 1.

Table 1. Comparison of Windows RT with Linux RT

SERVICES	WINDOWS + RT EXTENSION	LINUX + RT EXTENSION
Voice Services	Sufficient	Good
Datapluse Internet Services	Very Good	Very Good
Software Development Environment	Good	Sufficient
Support	Good	Good
Cost	High	Low

A new low cost approach for course and laboratory designed to allow students to design robotics, embedded devices that feature IOT, networking, a Real Time Operating System (RTOS) with different languages support like C, C++, python java and others[3]. Power consumption in mobile and handheld devices is the key embedded design issue; by varying the cache parameters in RTOS can reduce the consumption of the power by the embedded systems is designed by S.K. Dash and T. Srikanthan[4]. The xenomai RTOS can perform better than RT patched Linux on overload conditions and RT patched Linux can provide high throughput than Xenomai[5].

Luis Burdalo, Andres Terrasa, Agustin Espinosa, and Ana Garcia-Fornes presents analyzing the effect of gain time on soft task scheduling in real time systems. Results show that, in general, the presence of a significant amount of gain time reduces the performance benefit of the scheduling policies under study when compared to serving the soft tasks in background, which is considered the theoretical worst case. In some cases, this performance benefit is so small that the use of a specific scheduling policy for soft tasks is questionable[6].

Performance comparison of Vxworks, Linux, RTAI, xenomai in hard real time applications presented in[7] and the experimental results shows that open sources software is suitable for real time application. The underlying hardware should be shared by Linux and with other additional component this can be achieved by Xenomai and RTAI by using ADEOS nanokernel which acts as communicator between hardware functionality[8,9]. A hybrid operating system implemented using ARM processor that contains a real time kernel and time sharing OS[10].

This approach also improves real time interrupt latencies of hybrid operating system with two level hardware interrupt. Real time operating system

comparison by Rafael V. Aroca, GlaucoCaurin[11] this paper presents different real time operating system (QNX neutrino, µC/OS II, RTAI, Vxworks) and few General purpose operating system (Windows XP, Linux) comparison on parameters like worst case response time, Interrupt latency, Latency jitter. Worst case response time was calculated by using method proposed by ISA. All these compared parameters are tabulated and shown in Table 2.

Table 2. A: Worst case response time(µs), B: Interrupt Latency(µs), C: Latency jitter(µs)

	Win XP	Linux	Neutrino	µc/OSII	RTAI	Vx Works
A	200	1389	20	192	5	385
B	848	98	352	32	114	134
C	700	776	32	232	7.01	104

The well consolidated systems QNX Neutrino and Vx Works shown determinism and reliability throughout the experiment. QNX Neutrino and Vx works has better performance critical task and noncritical embedded task is very crucial job. It totally depends up on the application and its inter-process communication and platform.

The paper is organized as follows; Section 2 discusses the system design flow of research and parameters calculation steps. Section 3 presents the experimental setup and flow chart for the real time application. The results and its discussion are presented in Section 4. Section 5 concludes the paper.

2. System Design

The proposed system design is implemented on Raspberry pi consists ARM11 as processor running Real time application consists four main components

1. Raspberry pi
2. Ultrasonic sensor module
3. DC motor drivers
4. DC motors.

The proposed system with GPOS and with RTOS showed in Figure 3 and 4 respectively. Figure 3 shows Raspberry pi was booting with GPOS (Raspbian OS) and in Figure 4 shows Raspberry pi was booting with RTOS (RT-Preempt patched Raspbian) and same application was executed on both platforms and response time is calculated.

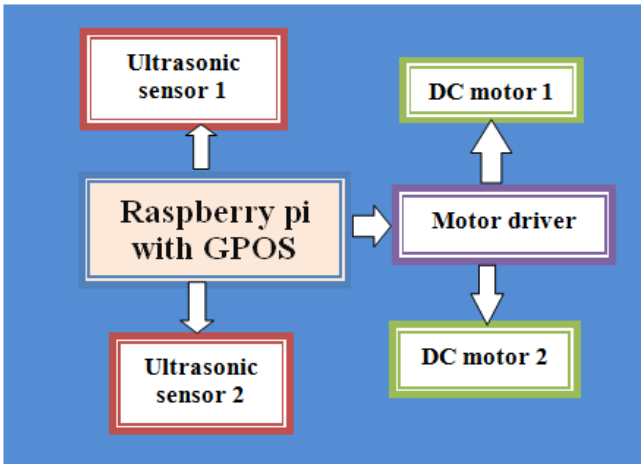

Figure 3. Proposed system with GPOS.

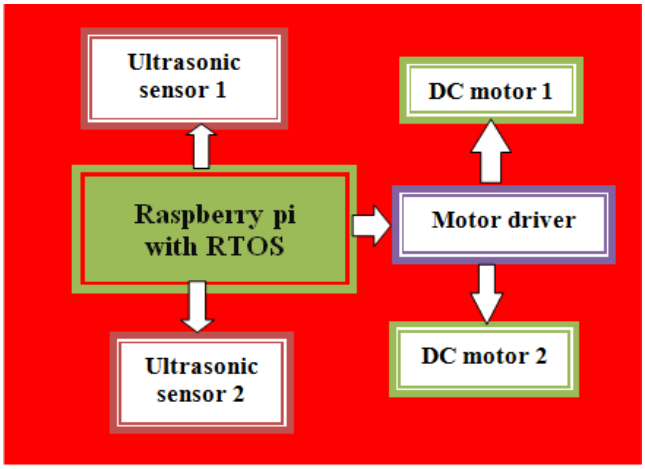

Figure 4. Proposed system with RTOS (RT patched Raspbian).

The main system of mobile robot is control by the raspberry pi in both Methodologies, where the Ultrasonic sensor receive input from the control module and send the acquire data in the form of distance Measurement. The control module will make the decision when obstacle conditions are meeting and move in according to avoid the obstacle.

The motor driver circuit is used to receive the signals from the control module in the form of binary logic and it will be supplied to the DC motors with sufficient voltage and current supply. The motor driver circuit will also protect the control module from DC motors when they get damaged.

The GUI application was design using python where

it will display the distance, latency and create a log file which stores previous latency values to a file in read and write only mode.

The experimental results were carried out on both platforms using mobile robot in real-time environment. Linux was booted first and following steps are carried out, after that same was done on RTOS.

1. Calculate the response time of the mobile robot in real-time environment.
2. Calculate the time taken by mobile robot to move full rotation and half rotation on flat surface.

2.1 Hardware Setup

The Hardware experimental setup was shown in Figure having two ultrasonic sensors used to find the distance between the mobile robot and obstacle with in the threshold set in the application program. When high logic was captured from echo pin of ultrasonic sensor the distance was calculated. Hc-sr04 was the ultrasonic model used in this application. The distance between the obstacle and mobile robot is calculated using Equation 1.

----- (1)

The control module will control the two DC motors of the mobile robot with the help of the Motor driver circuit. Hardware setup was shown in Figure 5.

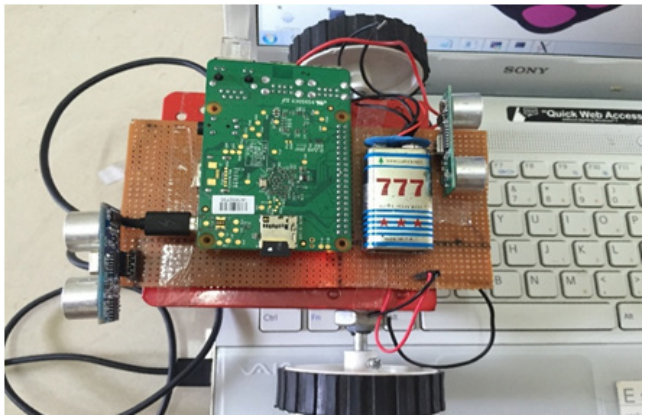

Figure 5. Experimental hardware setup for system design.

2.2 Software Setup

The raspberry was booted with a Raspbian OS having kernel version of 3.12.28+. The Raspberry pi kernel was changed to raspberrypi 3.12.36-rt50+. Where RT-Preempt patch converts the Raspbian OS kernel into a fully pre-emptible kernel. By changing the futures like in-kernel locking-primitives was implemented using spinlocks in Raspbian OS this was reimplemented using rtmutexes. Converting the old Linux timer API into separate infrastructures for high resolution kernel timers and one for timeouts used in user space POSIX timer having high resolution. For in-kernel spinlocks and semaphores, priority inheritance was implemented. Interrupt handlers were converted in to pre-emptible kernel threads. The RT-Preempt patch treats soft interrupt handlers in kernel thread context.

The application was developed on these both operating system and executed and comparison was done. The raspberry pi has general purpose input and output pins these pins can be used by the ultrasonic sensor and DC motor. The ultrasonic sensor is triggered, when echo was generated timer starts counting the time these timers are very high resolution timers. A signal is generated based on the thresh hold value to the DC motor drive then timer should be stopped. The timer value gives the response time of the mobile robot and other parameters were calculated using this approach. Figure 6 shows the flow of the application.

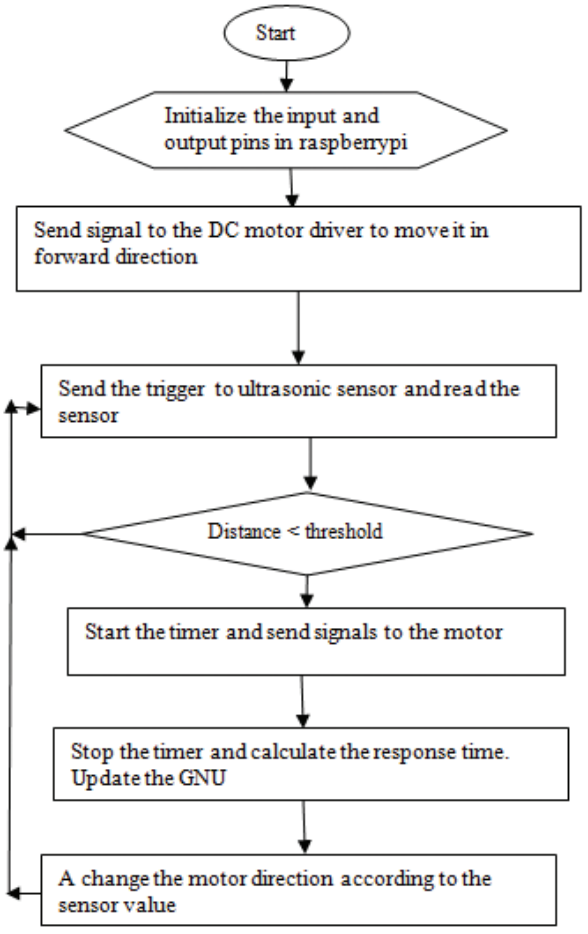

Figure 6. Flow chart of the Robot control.

3. Results

This section presents the results of the research about the calculation of the response time on Raspbian OS and its comparison on RT-Preempt patched with Raspbian OS in real time environment.

When Raspberry pi was booted with Raspbian OS the response time of the mobile robot was shown in Figure 7. The response of the GPOS was measured, obtained results shows that 102μs–105 μs as response time from the ultrasonic sensor to DC motor where the distance change, varying the response time in the GPOS.

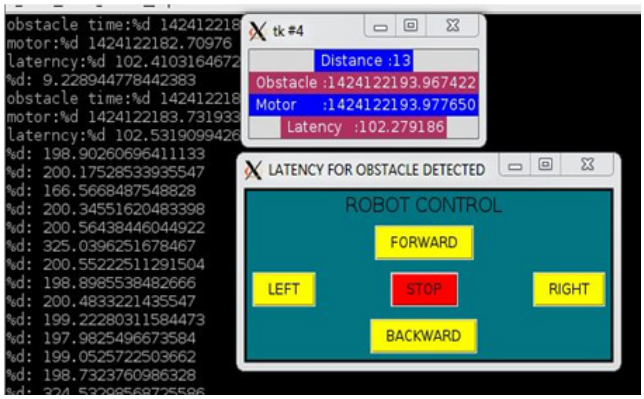

Figure 7. GPOS response time.

The same application was carried out in RTOS when Raspberry pi was booted with Raspbian RT-Preempt patched operating system the measuring system console shown in Figure 8. The response of the RTOS was measured, the obtained results shows that 22–24μs as response time from ultrasonic sensor to DC motor and it is also inferred that the change in distance varies very small change in response time.

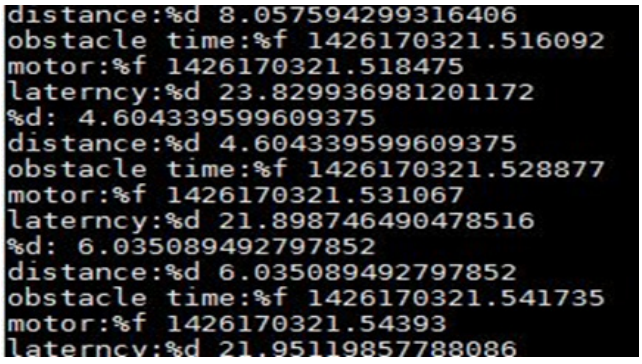

Figure 8. RTOS response time.

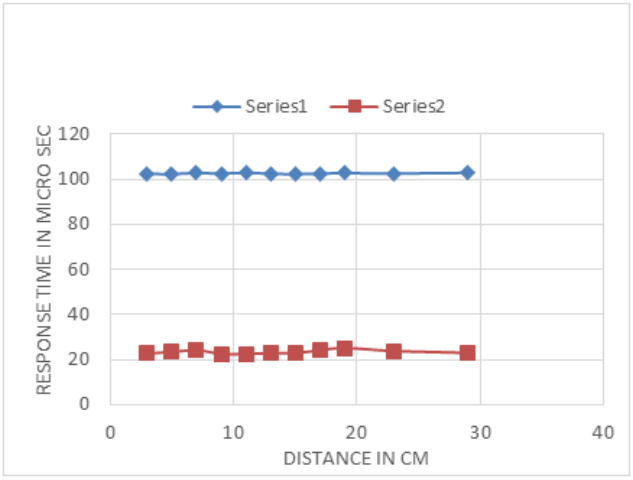

Figure 9. Comparison of response time.

The response time for distance from 2.5cm to 30cm was calculated with a scale of 5cm and tabulated in Figure 9. The graph shows in Table 3 the difference of response between GPOS and RTOS there was an average difference of 82μs was observed. The timer shouldstart accurately to calculate the distance between obstacle and mobile robot and response time.

Table 3. Comparison of response time between GPOS and RTOS

Distance	GPOS Response Time	RTOS Response Time
2.5	Failed to respond	Failed to respond
5	Failed to respond	22.940636
10	Failed to respond	23.004784
15	102.200508	23.549629
20	102.478211	23.400789
25	102.641000	23.829937
30	102.784131	23.89238

The secondary part was to calculate the time taken by the mobile robot for an angle 180° and 360°. It is observed that 1.2sec for 180° and 2.5 for 360°.For one minute the mobile robot did ninety two wheel rotations, this is the speed of the mobile robot.

When GPOS was running on Raspberrypi the minimum distance that mobile robot can detect an obstacle and stop the collision was 13.5cm where it was 4.5cm in RTOS. The reaction time taken by the mobile robot to avoid the obstacle in GPOS was 500ms where it was 250ms in the RTOS. This is the reaction time can by

the hardware in the mobile robot. The major reasons for short response time in RTOS are

1. The scheduling used in the RT-Preempt patched Raspbian was different from the Raspbian OS.
2. The memory used by the application was dedicated by the RTOS and it was shared in the GPOS system.
3. Kernel options are changed for RTOS compared to GPOS

So a difference of on an average of 82µs response time was observed in this real time application.

4. Conclusion

The overall research shows that ultrasonic sensor can detect the obstacle and communicate with control module to avoid the obstacle all this happens in 500ms second in GPOS and it is 250ms in RTOS. The results show that real time operating system have better response time compared to the general purpose operating system. RTOS is reliable even the threshold limit is given as 5cm. The average difference between the response time of GPOS and RTOS is 82µs. This response time is important in many real time applications. They are many direction of future work for this analysis one of it is this can be implemented in a maze and calculate the response of the mobile robot using GPOS and RTOS.This research can be implemented in many industries needs low response reactions, fire rescue operations and advance areas.

5. References

1. Kshetri N.Economics of Linux adoption in developing countries. IEEE Softwares. 2004 Jan/Feb; 20(4):7 4–81.
2. Marchesin A.Using Linux for Real Time Applications. IEEE Softwares.2004; 15:18–21.
3. Hamblen JO, van Bekkum GME.An embedded systems laboratory to support rapid prototyping of robotics and the internet of things. Transactions on Education.2013 Feb; 56(1):121–8.
4. Dash SK,Srikanthan T.Instruction cache tuning for embedded multitasking applications. IET Comput Digit Tech. 2010; 4(6):439–57.
5. Marieska MD, Kistijantoro AI, Subair M. Analysis and benchmarking performance of real time patch Linux and Xenomai in serving a real time application. IEEE Electrical Engineering and Informatics.2011 Jul; 32(8):21–19.
6. Burdalo L, Terrasa A, Espinosa A, Garcia-Fornes A. Analyzing the effect of gain time on soft-task scheduling policies in real-time system. IEEE on Software Engineering.2012 Nov/Dec; 38(6):1305–18.
7. Barbalace A, Luchetta A, Manduchi G, Moro M, Soppelsa A, Taliercio C.Performance comparison of VxWorks, Linux, RTAI and Xenomai in a hard real-time application. IEEE Transactions on Nuclear Science.2008 Feb;55(1):435–9.
8. ADEOS [Online]. 2015 Mar. Available from: http://www.adeos.org
9. Weinberg B.Uniting mobile Linux application platforms. Linux Pundit. 2008.
10. Liu M, Liu D, Wang Y, Wang M, Shao Z. On improving real-time interrupt latencies of hybrid operating systems with two-level hardware interrupts. IEEE Transactions on Computers.2011 Jul; 60(7):978–91.
11. Regehr J,Stankovic JA. HLS: A framework for composing soft real-time schedulers.RTSS'01.The 22nd IEEE Real-Time Systems Symposium; 2001.