*International Journal of Engineering & Technology*, 7 (2.7) (2018) 710-713

International Journal of Engineering & Technology

*Website: www.sciencepubco.com/index.php/IJET*

***Research Paper***

# Implementation of asymmetric processing on multi core processors to implement IOT applications on GNU/Linux framework

**Poonam Jain. S[1], Pooja. S[2], Sripath Roy. K[3*], Abhilash. K[4] and Arvind. B.V[5]**

*[1,2,4,5]Student, Fourth year, Department of ECE, KLEF, Guntur, AP*
*[3]Asst Professor, KLEF, Guntur, AP*
**Email: koganti_sripathroy@kluniversity.in*

**Abstract**

Internet of Things brought in a bigger computing challenges where there came a need for running tasks in a multi-sensor and large data processing is involved. In order to implement this requirement multiprocessors are being used for implementation of IoT Gateways. There comes a need for specific tasks having a resource dedicated for its job. To fulfill this we face a hurdle in choosing dedicated processor or shared processor in a Symmetric Processing Architecture. Dedicated processor are the one in which all the tasks are being processed on a single core where as in fair share processors specific processes are assigned to specific cores. Symmetric processing makes use of dedicated processors where as Asymmetric processor makes use of shared processors. Asymmetric Multi Processing can be used in real time applications in order to solve real time problems, one such platform is IOT. In this paper we have evaluated Asymmetric processing on GNU/Linux Platform to test multiple threads running on different multi-core processors architectures to realize the same for running IOT applications having higher computational requirements in the future.

***Keywords***: *Asymmetric Multiprocessing, GNU/Linux, Internet of Things, Scheduling, Symmetric Multiprocessing.*

## 1. Introduction

The primary idea of building a real time kernel with asynchronous multi processing is to solve real time problems. Here the real time processes are executed by this real-time kernel and normal Linux processes are turned down during this lapse. The scheduler of the real-time kernel treats the normal Linux kernel as an idle work, which when given a chance to implement, executes its own scheduler to program normal Linux tasks. But these normal Linux tasks can at any time be pre-empted by a real time task.

Symmetric Multi-Processing is a multiprocessor software and hardware architecture in which two or more identical processors are connected to a single and shared main memory and are controlled by a single operating system which treats all processors equally, reserving none for special purposes. In the case of multi-core processor, the SMP architecture treats them as separate processors.

In asymmetric multiprocessing system (AMP), not all CPUs are treated equally; for example, a system might allow only one CPU to execute operating system code or might allow only one CPU to perform I/O operations. Other AMP systems would do something different so that they were symmetric with respect to processor roles, but attached some or all peripherals to particular CPUs, so that they were asymmetric with respect to the peripheral attachment. AMP is used in applications that are dedicated i.e. when individual processors can be dedicated to specific tasks at design time.

But in real time there is much need in which kernel executes the task based on the priorities of the tasks i.e. it chooses it own task. Here each CPU performs its own task. Such requirement can be implemented by Asymmetric Multi Processing. Hence this drives the basic idea of implementing asymmetric multi processing on different processors used for IOT applications.

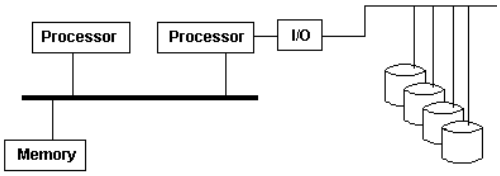



**Fig 1.** Asymmetric Processing

## 2. Analysis

A thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler. Asymmetric Multi Processing involves multiple threads to be executed on multiple cores depending upon the configuration of the system. Here tasks are assigned to specific processors unlike in symmetric processing2.1. The paper should have the following structure

### 2.1 Multi threading

Multi Threading is the ability of the processor or CPU to execute multiple threads or processes. It tries to enhance the utilization of a single processor by processing both threads and instruction sets in a parallel way. Multi Threading uses common memory heap

**Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.**

for the allocation of the threads. Multi threading provides faster switching between the threads. Asymmetric multi-processing delegates system tasks to be executed by some processors and applications on others. This is basically not well organized as symmetric processing due to the reality that below specific characteristic one processor may be completely active while the other is inactive. At the operating system level, multiprocessing is rarely used to point out the implementation of various synchronous processes in a system, with each process executing on a individual CPU or core, as conflicting to one process at any one instant. Multiprocessing reflects real parallel running of multiple processes using many processors. Multiprocessing certainly don't convey one process or task uses many processor [19-24].

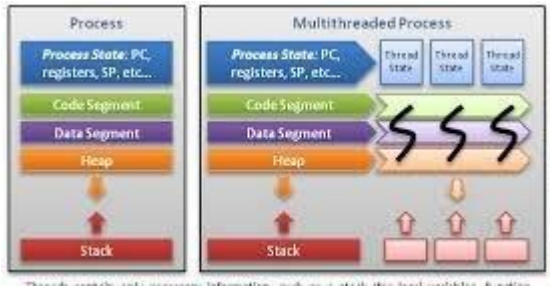



**Fig 2.** Multi threaded Execution

## 3. Scheduling mechanisms

Scheduling is the strategy by which work indicated by a few means is appointed to resources for completion of work. The work might be virtual calculation.
Components, for example, threads, procedures or data flows are in turn scheduled onto processors, network links or expansion cards which are hardware resources.
A scheduler carries out scheduling activity. Schedulers are used table various clients to share framework assets viably, or to accomplish an to keep the system resources busy, it allows multiple users to share resources equally. When used schedulers in implementation, follows a particular algorithm upon which threads are assigned to processors of the systems.
FIFO is one simplest scheduling algorithm which means First In First Out it is also known as First Come First Serve. It simply queues the items in the order they arrive. In this scheduling the task which come first gets served and remaining are in waiting. It is based on queuing and is generally used inside other scheduling mechanisms.

## 4. 4. Implementation methodology

In order to implement Asymmetric processing first we need set few pre-requisites.

- Preferred Platform
- Selection of CPU
- Kinds of tests to be performed

1. Preferred Platform:

We have chosen Linux to be the suitable platform because of its Free and Open Source Nature having a larger community working towards solving problems. Support of the community and availability of API can be used to solve real time problems. Hence we implemented asymmetric processing on Linux kernel.

2. Selection of CPU:

We have implemented asymmetric multi processing on various multi core processors like a. x86 i3 processor with Frequency- 1900 MHz. b. AMD II X3 720 Processor, Frequency- 800 MHz and c. x8 i5 Processor, Frequency- 1400.158 MHz. The system performance and latency was measure for the same. We chose general purpose computing platforms as a base to test the computational power as we are testing the system to work as a IoT Gateway Server.

3. Performing tests :

At first we have created many threads using POSIX APIs and set priority the tasks .Later by using scheduler we assigned the tasks to the specific CPU cores.
For example if there are 10 tasks , and if tasks 2, 7 are assigned to CPU1 and 1,4,8,9,10 are assigned to CPU3 and tasks 3,5,6 are assigned to CPU4,Here CPU2 is left idle. Then the CPU 1,3,4 perform only related and internal tasks where as CPU2 performs only internal tasks, rest of the time it remains idle. Hence in this way we have implemented asymmetric processing on multi core and checked for CPU performance and measured the start time, end time and latency of all the tasks.

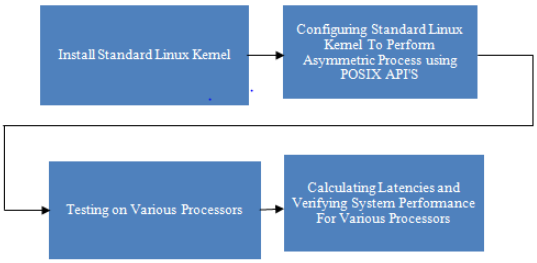



**Fig 3.** Block diagram and explanation

1. Installing Standard Linux Kernel.

The first step is to install standard Linux kernel and configuring the kernel to perform Asymmetric Multi Processing.

2. Configuring standard Linux kernel for Asymmetric multi processing using POSIX API'S.

After installing Standard Linux kernel we implemented multi processing with the help of POSIX (portable operating system interface) threads.

We implement multi processing by creating and processing large number of POSIX threads .As a part of asynchronous multi processing, we allocated specific threads to a particular processor. We defined 4 tasks and tasks were assigned to each processor
3. Testing on various processors

Next we worked on various processors to implement asynchronous process by creating and processing a
number of threads where threads were assigned to process specific tasks. We implemented multi processing successfully and thread parameters for each thread are obtained.
4. Calculating Latencies and verifying system performance for various processors.

After getting the thread related parameters by asynchronous processing, we found latency for each thread. It gives timing for each thread, which is most important for an real time operating system. We also verified multiprocessing on the system level using system monitor performance. Later the graphs were plotted for asynchronous processing for various processors.

## 5. Testing and verification

After the successful building of the kernel, we need to test the behavior and performance of the kernel to justify the real time behavior of the kernel. To verify the functional behavior and per-

formance metrics of the kernel, we need to carry some standard test benches.

Latency Test:

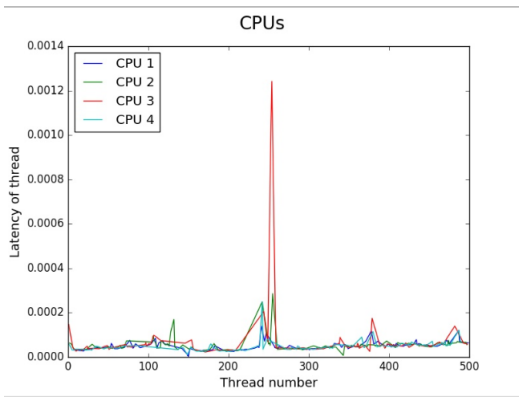
Latency test is the best marker of your ongoing execution, it checks the general execution of your framework. On the off chance that you have effectively introduced an accurately working constant part this test will tell you quickly. It gauges the distinction in time between the time when an assignment is really called by the scheduler and the time between the normal switch time. This test gives least, normal, and greatest latencies for that period and also least and most extreme general latencies that happened over the whole test for each one moment

## 6. Results

1. Asynchronous process on standard Linux kernel (x86 i3 processor, Frequency- 1900 MHz)



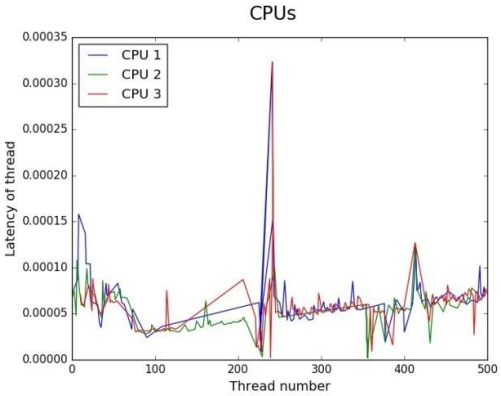
1.1 Asynchronous Processing plot on standard Linux kernel (x86 i3 processor, Frequency- 1900 MHz)




2. Asynchronous Processing on standard Linux kernel (AMD II X3 720 Processor, Frequency-800 MHz)



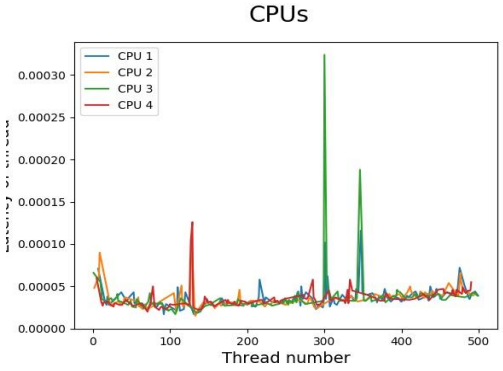
2.1 Asynchronous Processing plot on Standard Linux Kernel (AMD II X3 720 Processor, Frequency-800 MHz)




3. Asynchronous Processing on standard Linux kernel (x86 i5 Processor, Frequency-1400.158 MHz)



3.1 Asynchronous Processing plot on standard Linux kernel (x86 i5 processor, Frequency-1400.158 MHz).




## 7. Conclusion

We implemented the asymmetric processing on various processors like i3, AMD, i5 processors and calculated the latency values and, and we also verified the multi processing on system level using system monitor performance, and graphs were plotted for various asymmetric processing on various processors.

## Acknowledgement

This project has been implemented and worked upon by all the authors mentioned. We are very grateful to Sripath Roy and thankful to Koneru Lakshmaiah Educational Foundation for their guidance and support to this project

## References

[1] Nikola Markovic, Daniel Nemirovsky, Osman Unsal, Mateo Valero and Adrian Cristal “Kernel-to-User-Mode Transition-Aware Hardware Scheduling” IEEE Micro ( Volume: 35, Issue: 4, July-Aug. 2015 )pp. 0272-1732.

[2] M.Aater Suleman,Onur Mutlu, Moinuddin Qureshi, "Accerlating Critical ection Execution with Asymmetric Multicore Architecture" IEEE Micro (Volume:30, " , Issue: 1 , Jan.-Feb. 2010).

[3] Junji Sakai, Inoue Hiroaki, Sunao Torii, Masato Edahiro. "Multitasking Parallel Method for High-End Embedded Appliances." IEEE Micro (Volume: 28, Issue: 5, Sept.-Oct. 2008 ) **pp:** 0272-1732.

[4] H. Roth, A. Chandra "POSIX standards for fault management in a real-time environment". IEEE Xplore**:** 06 August 2002 pp. 0-8186-7515-2.

[5] Emily Blem, Hadi Esmaeilzadeh, Renee St.Amant, Karthikeya Sankaralingam, Doug Burger, " MultiMore Model from Abstract Single Core Inputs". IEEE computer Architecture Letters (Volume: S12,issue 2 dec.2013); pp.1556-6056

[6] Yang Zhou, Qiaodi Zhou. " The embeded real-time Linux operation system based on the Xenomai." Electrical and Control Engineering (ICECE), 2011 International Conference on 16-18 Sept. 2011 Published on 24 October 2011

[7] M.D.Marieska, A.I.Kistijantoro, M.Subair." Analysis and benchmarking performance of Real Time Patch Linux and Xenomai in serving a real time application". Electrical Engineering and Informatics (ICEEI), 2011 International Conference on 17-19 July 2011 Published on 19 September 2011.

[8] http://ask.xmodulo.com/view-threads-process-Linux.html

[9] http://marc.merlins.org/linux/linux.conf.au2001/Day1/threads.pdf

[10] http://www.cs.fsu.edu/~baker/realtime/restricted/notes/pthreads.html

[11] https://www.Linux.com/learn/how-install-and-configure-conky

[12] https://www.ibm.com/developerworks/library/l-async/

[13] http://www.cs.kent.edu/~ruttan/sysprog/lectures/multi-thread/multi-thread.html

[14] https://randu.org/tutorials/threads/#pthreads

[15] http://scitechconnect.elsevier.com/asymmetric-multi-processing-amp-vs-symmetric-multi-processing-smp/

[16] https://randu.org/tutorials/threads/#pthreads

[17] http://github.com/dankex/tools/tree/master/Linux-kernel/wake_latency/

[18] http://list.xmodulo.com/logstash.html

[19] ANNABATTULA, J., KOTESWARA RAO, S., SAMPATH DAKSHINA MURTHY, A., SRIKANTH, K.S. and DAS, R.P., 2015. Underwater passive target tracking in constrained environment. Indian Journal of Science and Technology, 8(35), pp. 1-4.

[20] HUSSAIN, S.N. and KISHORE, K.H., 2016. Computational Optimization of Placement and Routing using Genetic Algorithm. Indian Journal of Science and Technology, 9(47),.

[21] Vudatha, C.P., Nalliboena, S., Jammalamadaka, S.K.R., Duvvuri, B.K.K., Reddy, L.S.S., Automated generation of test cases from output domain of an embedded system using Genetic algorithms, ICECT 2011 - 2011 3rd International Conference on Electronics Computer Technology 5,5941989, pp. 216-220

[22] Sastry, J.K.R., Ganesh, J.V., Bhanu, J.S., I2C based networking for implementing heterogeneous microcontroller based distributed embedded systems, Indian Journal of Science and Technology, Volume 8, Issue 15, 2015

[23] Sastry, J.K.R., Naga Sai Tejasvi, T., Aparna, J., Dynamic scheduling of message flow within a distributed embedded system connected through a RS485 network, ARPN Journal of Engineering and Applied Sciences, Volume 12, Issue 9, 1 May 2017, Pages 2809-2817

[24] Sastry, J.K.R., Suresh, A., Bhanu, S.J., Building heterogeneous distributed embedded systems through rs485 communication protocol, ARPN Journal of Engineering and Applied Sciences, 2015, 10(16), pp. 6793-6803