Gist of the project

Asymmetric and symmetric Multi-Processing on multi-core processors

Kernel:

In short, it's a vital part of your operating system, if not the most important. An operating system is responsible for letting your programs function, by allowing them access to your hardware. However, it's which actually carries all these jobs out.

Symmetric Multi-Processing

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 pro-cessors.

Asymmetric multiprocessing

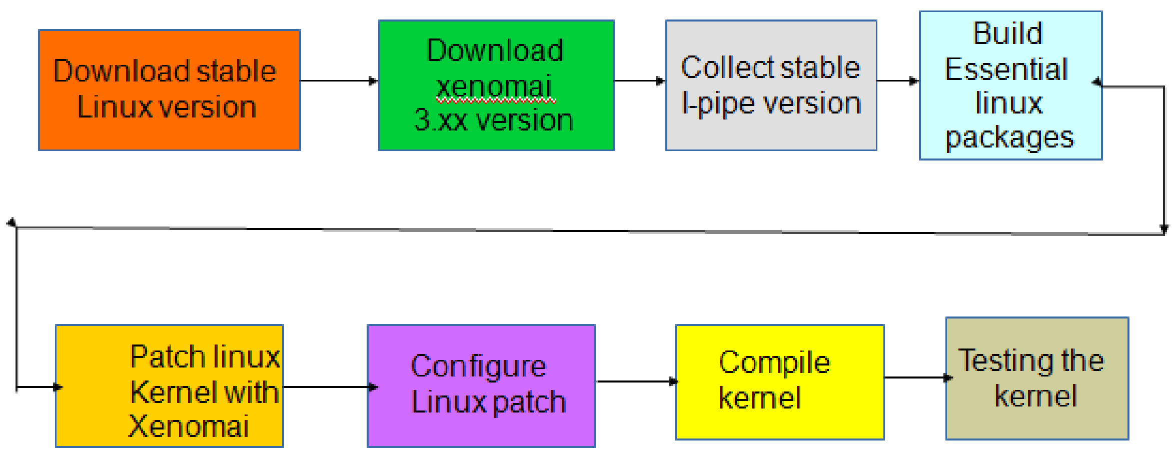
In asymmetric multiprocessing system (AMP), not all CPUs are treated equally; for ample, a system might allow only one CPU to ecute operating system code or might allow only one CPU to perform I/O operations. Other AMP systems would do something different so that they were symmetric with respect to processor 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.

Kernel Installation Steps:

The installation process for building the kernel is as follows

- Download stable Linux version

- Download Xenomai-3.0.5 version
- Collect stable I-pipe version
- Build essential Linux packages
- Patch Linux Kernel with Xenomai
- Configure Linux Patch
- Compile Kernel
- Testing the Kernel



1 Downloading stable Linux source:

Stable Linux version has to be downloaded from the official website.

https://mirrors.edge.kernel.org/pub/linux/kernel/.

This file will we downloaded with an extension of Linux-xx-tar.xz, extract the file and place in some directory

2 Downloading Xenomai stable version:

Stable version has to be downloaded.

https://xenomai.org/downloads/xenomai/stable/latest/

This file will we downloaded with an extension of xenomai-xx-tar.bz2, extract the file and place in same directory

3 Collecting Stable I-pipe Core:

Download the stable I-pipe core from

http://xenomai.org/downloads/ipipe/

Select the patch according to your Linux version,

In this project the Linux version is 4.9.51 so the ipipe selected is -4.9.51-x86-4 patch.

Note: It is better to download ipipe first and then download Linux.Xenomai can be of any version

4 Build essential packages in Linux:

Essential packages were built in downloaded Linux kernel using

Before doing this store all the downloaded files in single folder and place it in some directory like downloads,documents etc.

Open terminal and navigate to your directory where you have the folder.

cd Documents/

To list the files type *ls*

Now navigate to the folder where you have stored the files

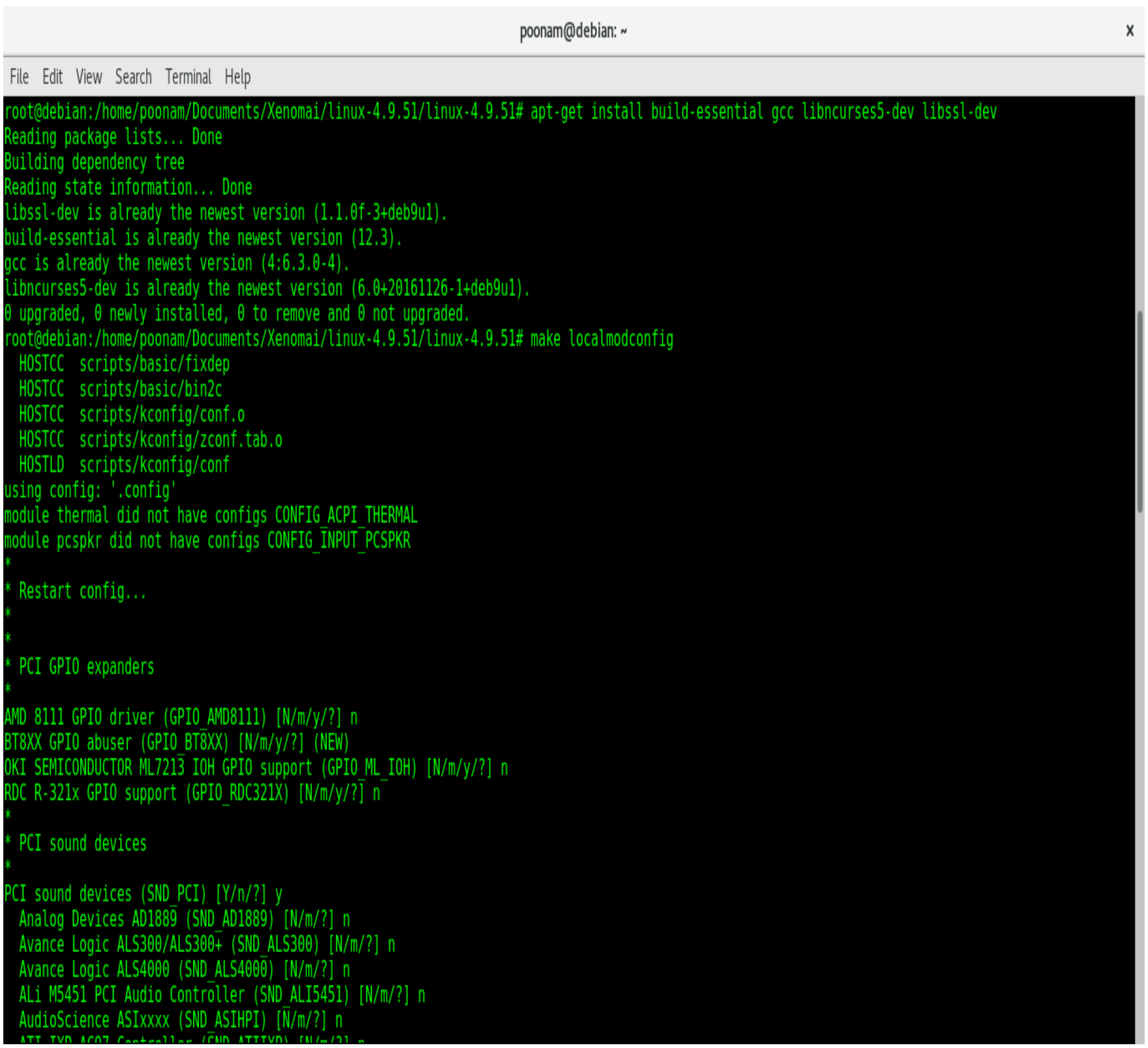
cd Xenomai/ (Here the folder name is Xenomai)

Now navigate to the Linux kernel file present in the folder

cd Linux-4.9.51/

In this Linux kernel now build the essential packages

apt-get install build-essential gcc libncurses5-dev libssl-dev libncurses5-dev package will make the kernel easier to configure.



5 Patch Linux Kernel with Xenomai

Now we need to patch our downloaded stable linux kernel with Xenomai .

For this navigate back to the folder and then navigate to the xenomai file

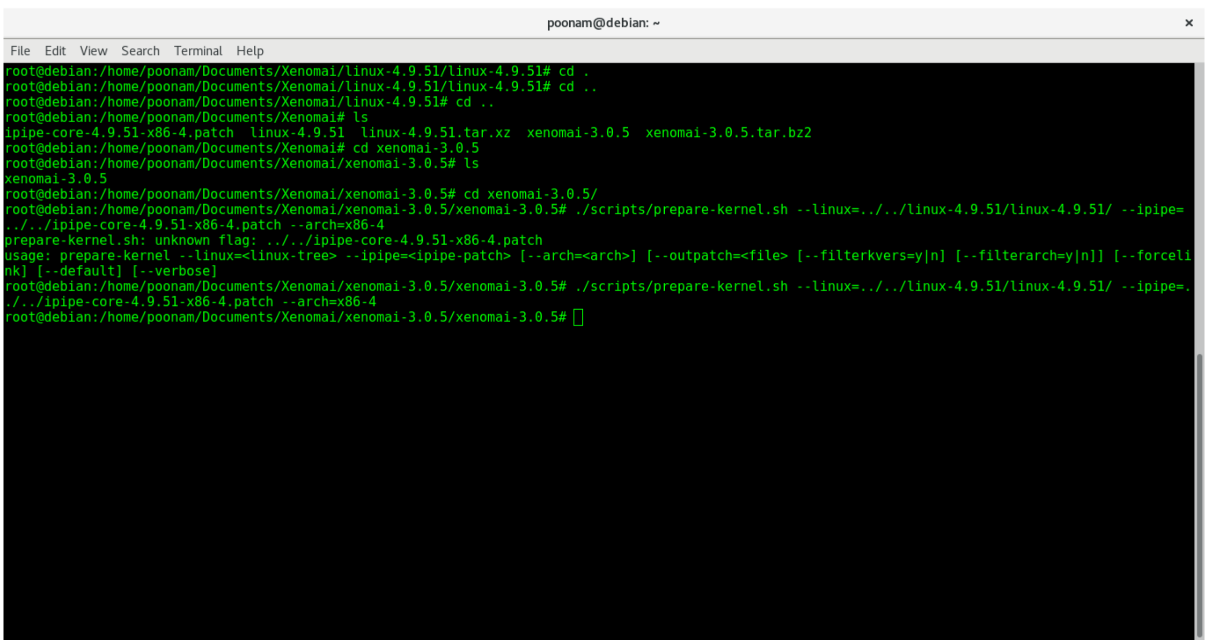
Steps:

1 *cd ..* (This makes us navigate back to the folder so that we can navigate to Linux file

2 *cd xenomai-.xx.yy* (x represents the version)

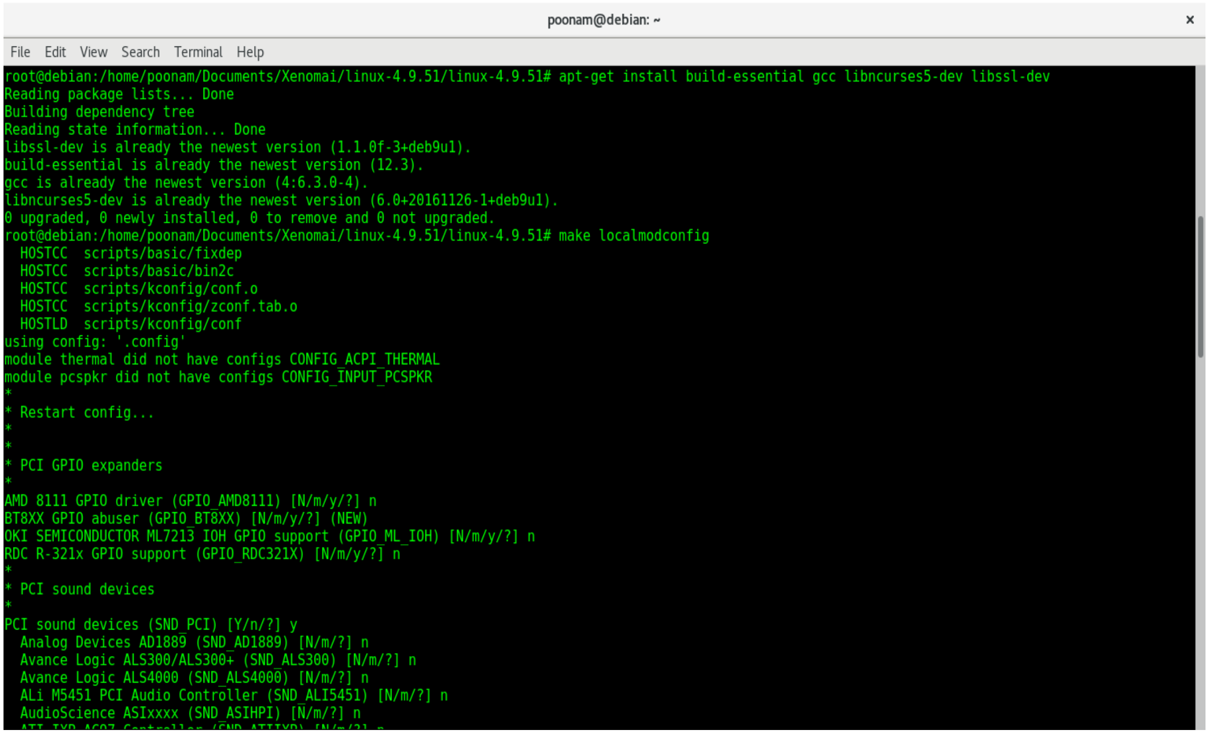
Now after navigating to the Linux file through terminal type the following command for patching.Make sure ipipe,xenomai and Linux files are in the same folder

/scripts/prepare-kernel.sh-linux=../../linux-4.9.51/linux-4.9.51/--ipipe=../../ipipe-core-4.9.51-x86-4.patch-arch=x86-4



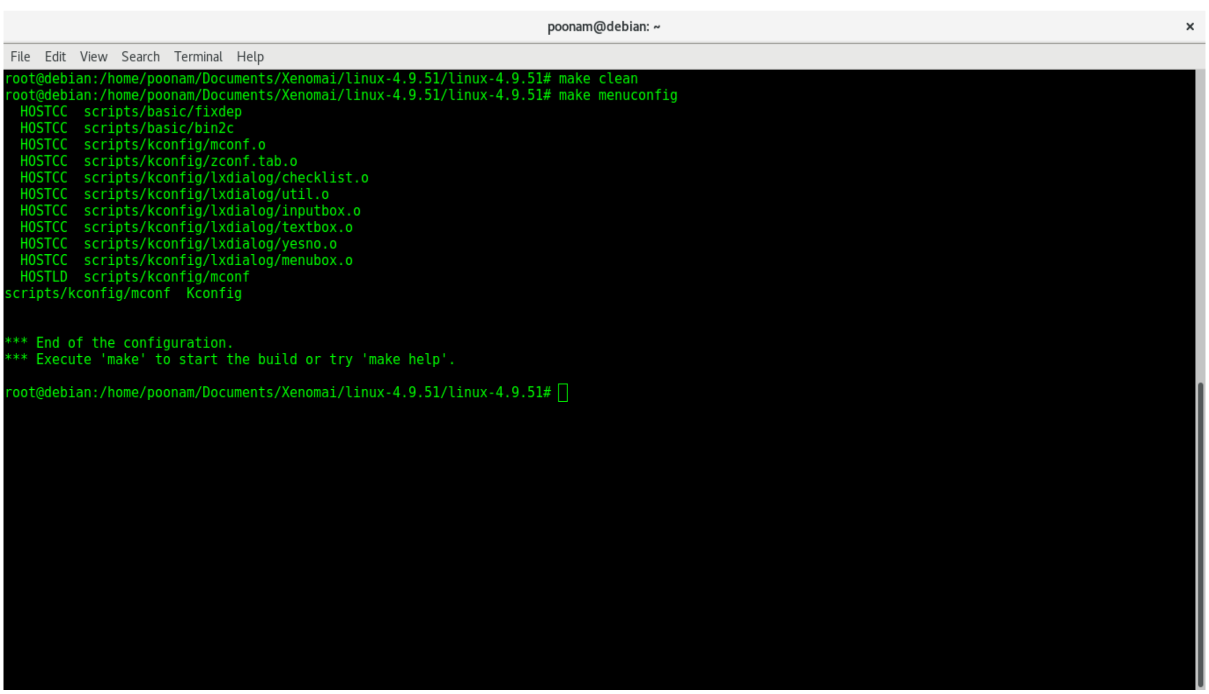
6 Configuring Patched Kernel:

Using ***make local modconfig*** detects currently running kernel components and marks them for compilation.



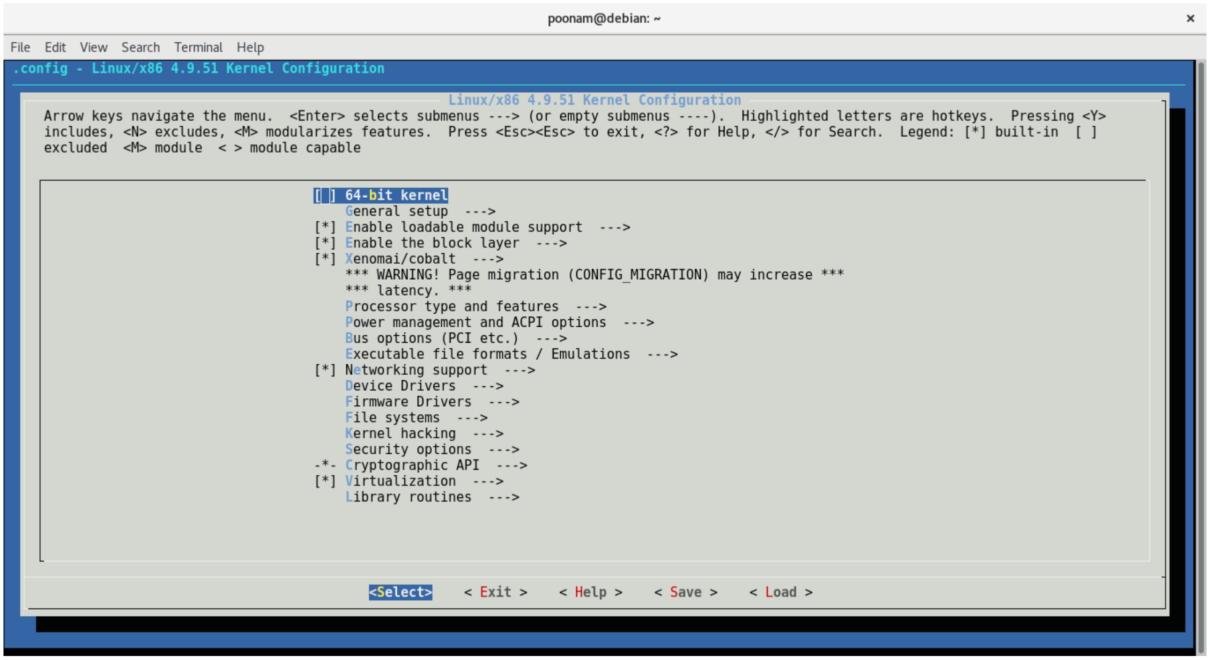
```
poonam@debian: ~
File Edit View Search Terminal Help
root@debian:/home/poonam/Documents/Xenomai/linux-4.9.51/linux-4.9.51# apt-get install build-essential gcc libncurses5-dev libssl-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
libssl-dev is already the newest version (1.1.0f-3+deb9u1).
build-essential is already the newest version (12.3).
gcc is already the newest version (4:6.3.0-4).
libncurses5-dev is already the newest version (6.0+20161126-1+deb9u1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@debian:/home/poonam/Documents/Xenomai/linux-4.9.51/linux-4.9.51# make localmodconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/basic/bin2c
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
using config: '.config'
module thermal did not have configs CONFIG_ACPI_THERMAL
module pcspkr did not have configs CONFIG_INPUT_PCSPKR
*
* Restart config...
*
*
* PCI GPIO expanders
*
AMD 8111 GPIO driver (GPIO_AMD8111) [N/m/y/?] n
BT8XX GPIO abuser (GPIO_BT8XX) [N/m/y/?] (NEW)
OKI SEMICONDUCTOR ML7213 IOH GPIO support (GPIO_ML_IOH) [N/m/y/?] n
RDC R-321x GPIO support (GPIO_RDC321X) [N/m/y/?] n
*
* PCI sound devices
*
PCI sound devices (SND_PCI) [Y/n/?] y
  Analog Devices AD1889 (SND_AD1889) [N/m/?] n
  Avance Logic ALS300/ALS300+ (SND_ALS300) [N/m/?] n
  Avance Logic ALS4000 (SND_ALS4000) [N/m/?] n
  ALi M5451 PCI Audio Controller (SND_ALI5451) [N/m/?] n
  AudioScience ASIxxxx (SND_ASIHPI) [N/m/?] n
```

make menuconfig is used to choose what to compile. Here we make changes and generate a .config file.



```
poonam@debian: ~
File Edit View Search Terminal Help
root@debian:/home/poonam/Documents/Xenomai/linux-4.9.51/linux-4.9.51# make clean
root@debian:/home/poonam/Documents/Xenomai/linux-4.9.51/linux-4.9.51# make menuconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/basic/bin2c
  HOSTCC  scripts/kconfig/mconf.o
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTCC  scripts/kconfig/lxdialog/checklist.o
  HOSTCC  scripts/kconfig/lxdialog/util.o
  HOSTCC  scripts/kconfig/lxdialog/inputbox.o
  HOSTCC  scripts/kconfig/lxdialog/textbox.o
  HOSTCC  scripts/kconfig/lxdialog/yesno.o
  HOSTCC  scripts/kconfig/lxdialog/menubox.o
  HOSTLD  scripts/kconfig/mconf
scripts/kconfig/mconf  Kconfig


*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

root@debian:/home/poonam/Documents/Xenomai/linux-4.9.51/linux-4.9.51#
```

Kernel configuration file:



During configuring we need to enable and disable some settings.

To set configurations refer the link

https://xenomai.org//2014/06/configuring-for-x86-based-dual-kernels/

7 Power Management:

Power management should not be disabled globally, the only which are to be disabled in this area are

- **CONFIG_APM**
- **CONFIG_ACPI_PROCESSOR**

CPU Frequency scaling:

- **CONFIG_CPU_FREQ – Disable**

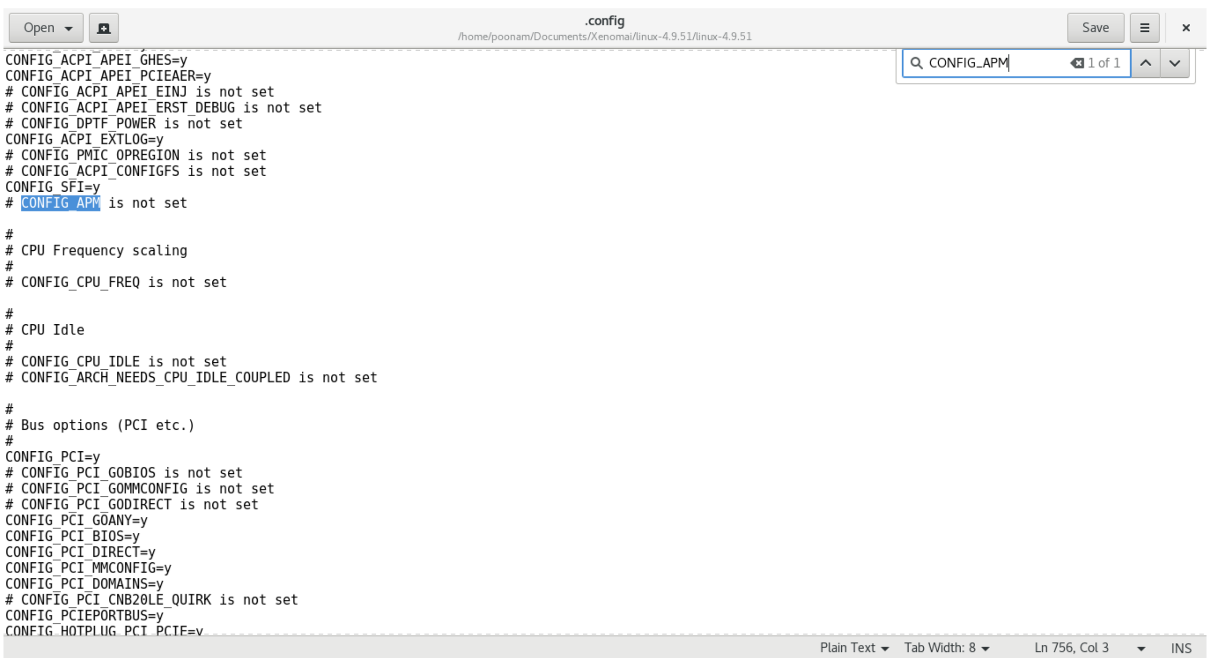
  Numerous CPUs change the TSC tallying recurrence which makes it pointless for act planning when the CPU clock can change.Also a few CPUs take a few milliseconds to increase to full speed. Along these lines, by impairing this enables the CPU to be regulated with workload.

- **CONFIG_CPU_IDLE – Disable**

  This enables the CPU to enter profound rest states , panding the time it removes to get from these rest states. Timers used by Xenomai stop functioning when entered into these sleep states.

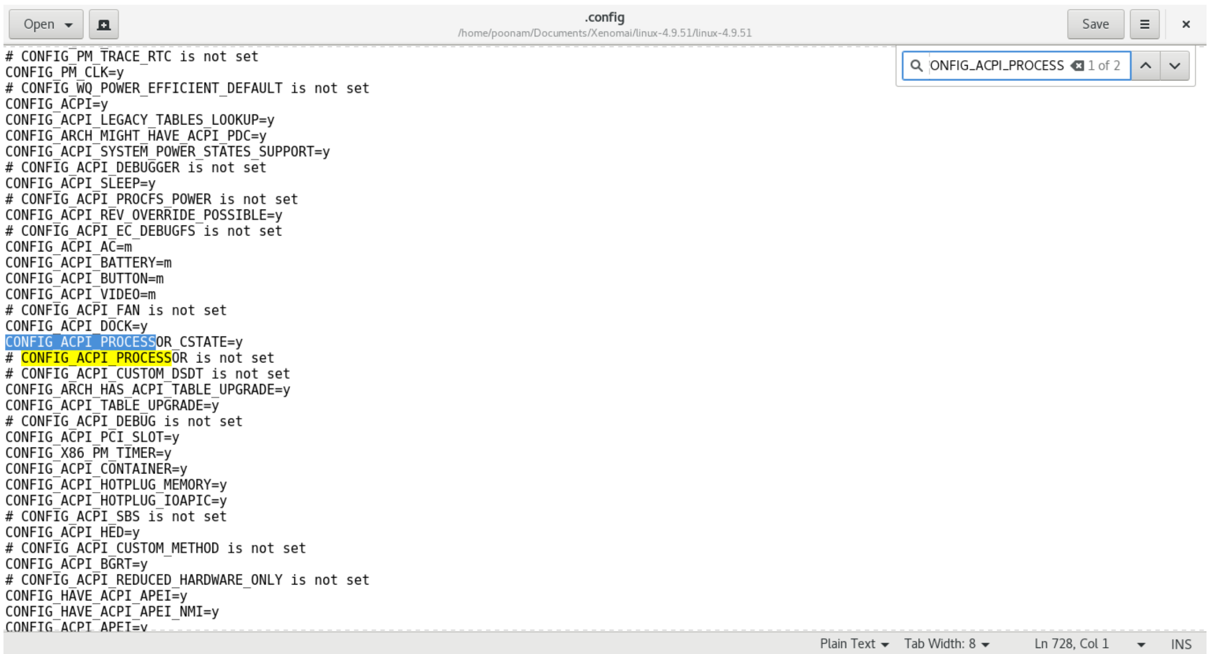
- **CONFIG_APM – Disable**

  Power management control to the BIOS is assigned by the APM model. Profiles code is never composed because of best inactivity. APM schedules are summoned with SMI need if designed, this sidesteps the I-pipe altogether.



- **CONFIG_ACPI_PROCESSOR – Disable**

The ACPI processor module disables the local ACPI which will cause the Xenomai timer initialization to fail due to this reason we disable this option.



.config
/home/poonam/Documents/Xenomai/linux-4.9.51/linux-4.9.51

```
# CONFIG_PM_TRACE_RTC is not set
CONFIG_PM_CLK=y
# CONFIG_WQ_POWER_EFFICIENT_DEFAULT is not set
CONFIG_ACPI=y
CONFIG_ACPI_LEGACY_TABLES_LOOKUP=y
CONFIG_ARCH_MIGHT_HAVE_ACPI_PDC=y
CONFIG_ACPI_SYSTEM_POWER_STATES_SUPPORT=y
# CONFIG_ACPI_DEBUGGER is not set
CONFIG_ACPI_SLEEP=y
# CONFIG_ACPI_PROCFS_POWER is not set
CONFIG_ACPI_REV_OVERRIDE_POSSIBLE=y
# CONFIG_ACPI_EC_DEBUGFS is not set
CONFIG_ACPI_AC=m
CONFIG_ACPI_BATTERY=m
CONFIG_ACPI_BUTTON=m
CONFIG_ACPI_VIDEO=m
# CONFIG_ACPI_FAN is not set
CONFIG_ACPI_DOCK=y
CONFIG_ACPI_PROCESSOR_CSTATE=y
# CONFIG_ACPI_PROCESSOR is not set
# CONFIG_ACPI_CUSTOM_DSDT is not set
CONFIG_ARCH_HAS_ACPI_TABLE_UPGRADE=y
CONFIG_ACPI_TABLE_UPGRADE=y
# CONFIG_ACPI_DEBUG is not set
CONFIG_ACPI_PCI_SLOT=y
CONFIG_X86_PM_TIMER=y
CONFIG_ACPI_CONTAINER=y
CONFIG_ACPI_HOTPLUG_MEMORY=y
CONFIG_ACPI_HOTPLUG_IOAPIC=y
# CONFIG_ACPI_SBS is not set
CONFIG_ACPI_HED=y
# CONFIG_ACPI_CUSTOM_METHOD is not set
CONFIG_ACPI_BGRT=y
# CONFIG_ACPI_REDUCED_HARDWARE_ONLY is not set
CONFIG_HAVE_ACPI_APEI=y
CONFIG_HAVE_ACPI_APEI_NMI=y
CONFIG_ACPI_APEI=y
```

- **CONFIG_INTEL_IDLE – Disable**

  This causes gigantic latencies in light of the fact that the ACPI clock that Xenomai uses may not fire any longer, additionally simply like CONFIG_ACPI_PROCESSOR, this sit driver sends the CPU into profound C states

- **Disabling CONFIG_CC_STACKPROTECTOR:**

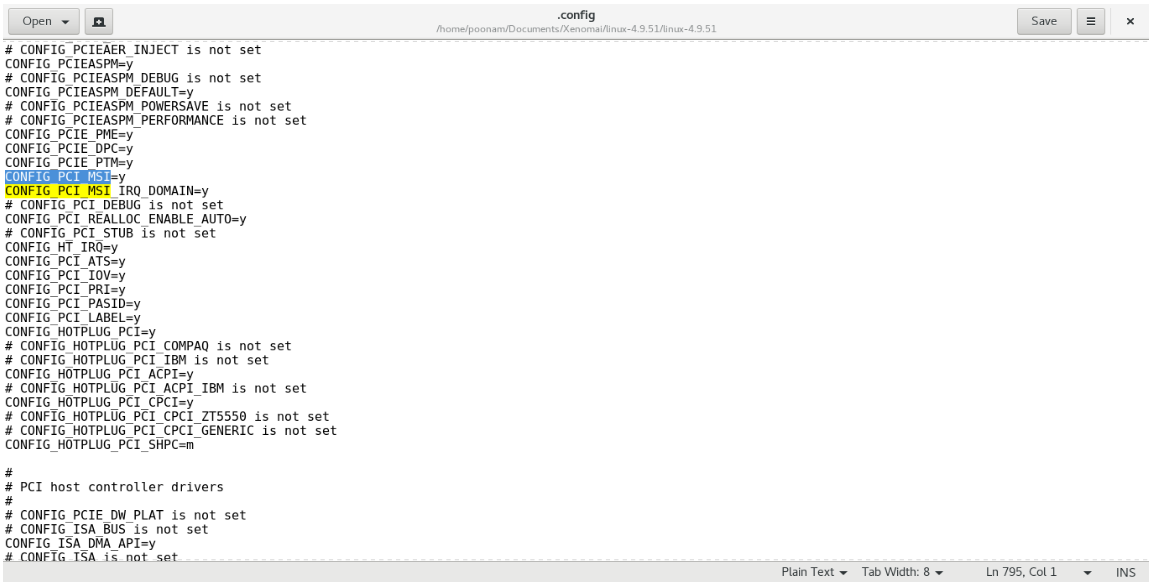
.config

/home/poonam/Documents/Xenomai/linux-4.9.51/linux-4.9.51

```
CONFIG_ARCH_HAVE_NMI_SAFE_CMPXCHG=y
CONFIG_HAVE_CMPXCHG_LOCAL=y
CONFIG_HAVE_CMPXCHG_DOUBLE=y
CONFIG_ARCH_WANT_IPC_PARSE_VERSION=y
CONFIG_HAVE_ARCH_SECCOMP_FILTER=y
CONFIG_SECCOMP_FILTER=y
CONFIG_HAVE_GCC_PLUGINS=y
# CONFIG_GCC_PLUGINS is not set
CONFIG_HAVE_CC_STACKPROTECTOR=y
#CONFIG_CC_STACKPROTECTOR is not set
# CONFIG_CC_STACKPROTECTOR_NONE is not set
# CONFIG_CC_STACKPROTECTOR_REGULAR is not set
CONFIG_CC_STACKPROTECTOR_STRONG=y
CONFIG_HAVE_ARCH_WITHIN_STACK_FRAMES=y
CONFIG_HAVE_IRQ_TIME_ACCOUNTING=y
CONFIG_HAVE_ARCH_TRANSPARENT_HUGEPAGE=y
CONFIG_MODULES_USE_ELF_REL=y
CONFIG_ARCH_HAS_ELF_RANDOMIZE=y
CONFIG_HAVE_ARCH_MMAP_RND_BITS=y
CONFIG_HAVE_EXIT_THREAD=y
CONFIG_ARCH_MMAP_RND_BITS=16
CONFIG_HAVE_COPY_THREAD_TLS=y
# CONFIG_HAVE_ARCH_HASH is not set
# CONFIG_ISA_BUS_API is not set
CONFIG_CLONE_BACKWARDS=y
CONFIG_OLD_SIGSUSPEND3=y
CONFIG_OLD_SIGACTION=y
# CONFIG_CPU_NO_EFFICIENT_FFS is not set
# CONFIG_HAVE_ARCH_VMAP_STACK is not set

#
# GCOV-based kernel profiling
#
# CONFIG_GCOV_KERNEL is not set
CONFIG_ARCH_HAS_GCOV_PROFILE_ALL=y
CONFIG_HAVE_GENERIC_DMA_COHERENT=y
CONFIG_SLABINFO=y
```

Plain Text Tab Width: 8 Ln 281, Col 27 INS

- **Enable CONFIG_PCI_MSI:**



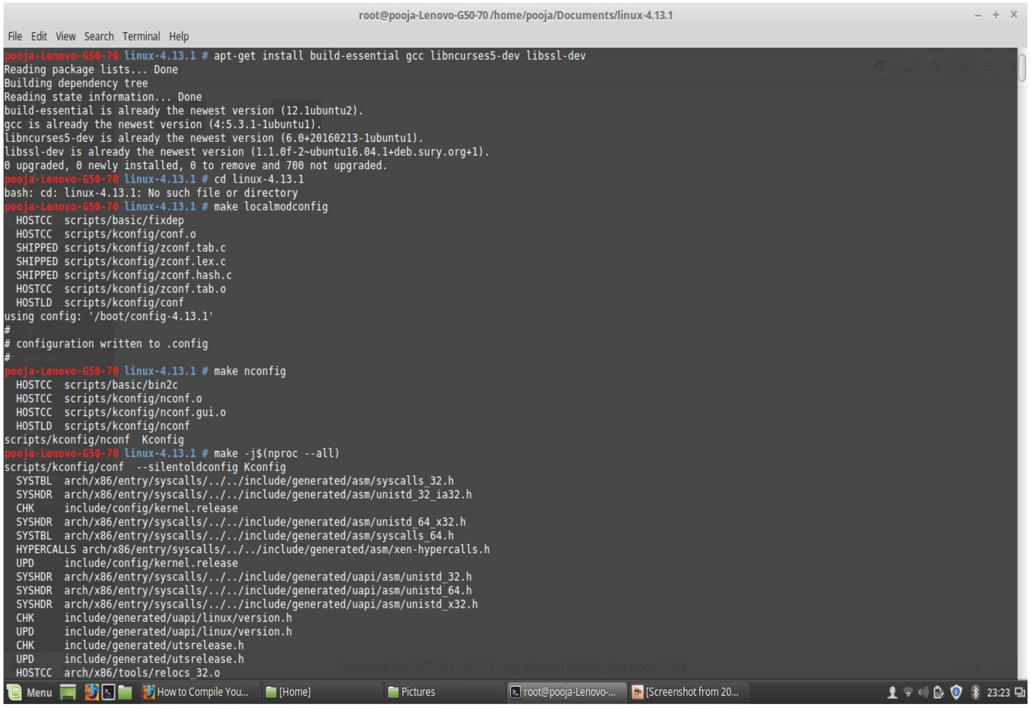

```
# CONFIG_PCIEAER_INJECT is not set
CONFIG_PCIEASPM=y
# CONFIG_PCIEASPM_DEBUG is not set
CONFIG_PCIEASPM_DEFAULT=y
# CONFIG_PCIEASPM_POWERSAVE is not set
# CONFIG_PCIEASPM_PERFORMANCE is not set
CONFIG_PCIE_PME=y
CONFIG_PCIE_DPC=y
CONFIG_PCIE_PTM=y
CONFIG_PCI_MSI=y
CONFIG_PCI_MSI_IRQ_DOMAIN=y
# CONFIG_PCI_DEBUG is not set
CONFIG_PCI_REALLOC_ENABLE_AUTO=y
# CONFIG_PCI_STUB is not set
CONFIG_HT_IRQ=y
CONFIG_PCI_ATS=y
CONFIG_PCI_IOV=y
CONFIG_PCI_PRI=y
CONFIG_PCI_PASID=y
CONFIG_PCI_LABEL=y
CONFIG_HOTPLUG_PCI=y
# CONFIG_HOTPLUG_PCI_COMPAQ is not set
# CONFIG_HOTPLUG_PCI_IBM is not set
CONFIG_HOTPLUG_PCI_ACPI=y
# CONFIG_HOTPLUG_PCI_ACPI_IBM is not set
CONFIG_HOTPLUG_PCI_CPCI=y
# CONFIG_HOTPLUG_PCI_CPCI_ZT5550 is not set
# CONFIG_HOTPLUG_PCI_CPCI_GENERIC is not set
CONFIG_HOTPLUG_PCI_SHPC=m

#
# PCI host controller drivers
#
# CONFIG_PCIE_DW_PLAT is not set
# CONFIG_ISA_BUS is not set
CONFIG_ISA_DMA_API=y
# CONFIG_ISA is not set
```


8. Compile Kernel:

Now we need to compile our linux kernel using

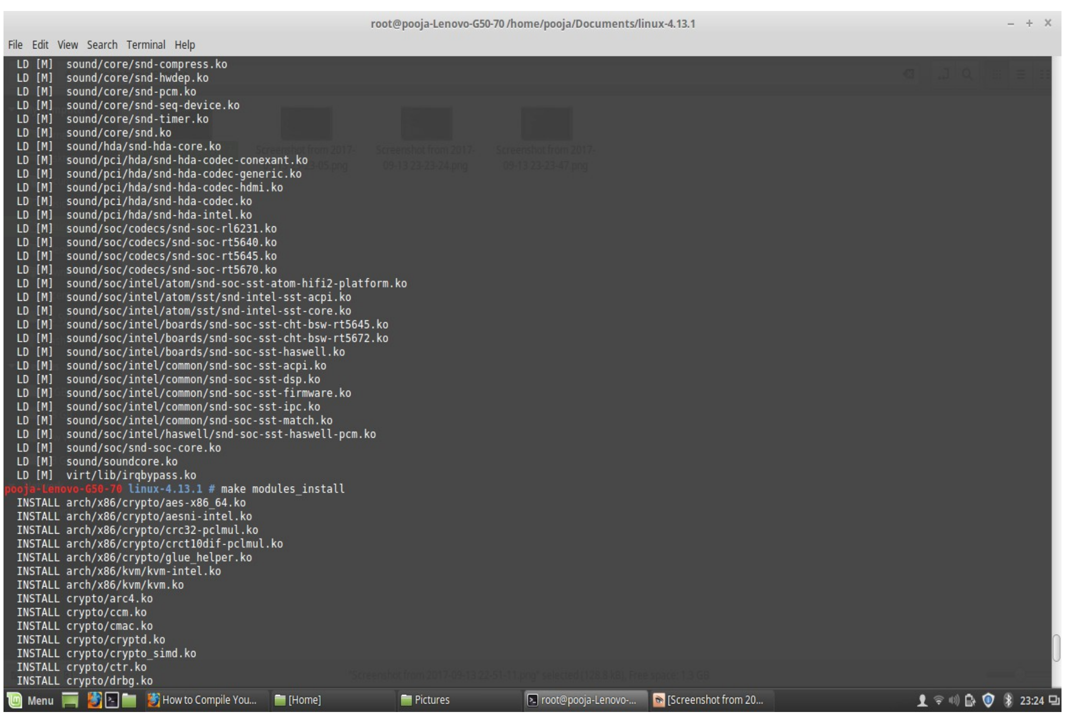
- **“Sudo make-j$(nproc—all)”**

  This command is used to compile the kernel. “$” represents number of processors



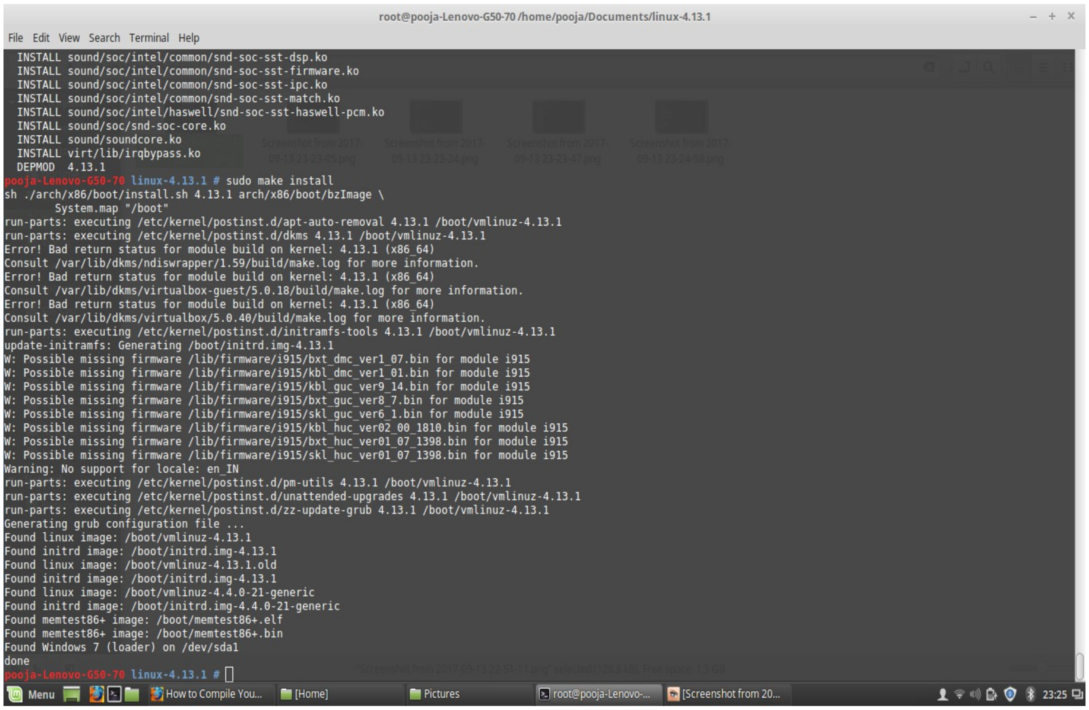
- **“make modules_install”**

This command is used to finish compiling the rest of the kernel

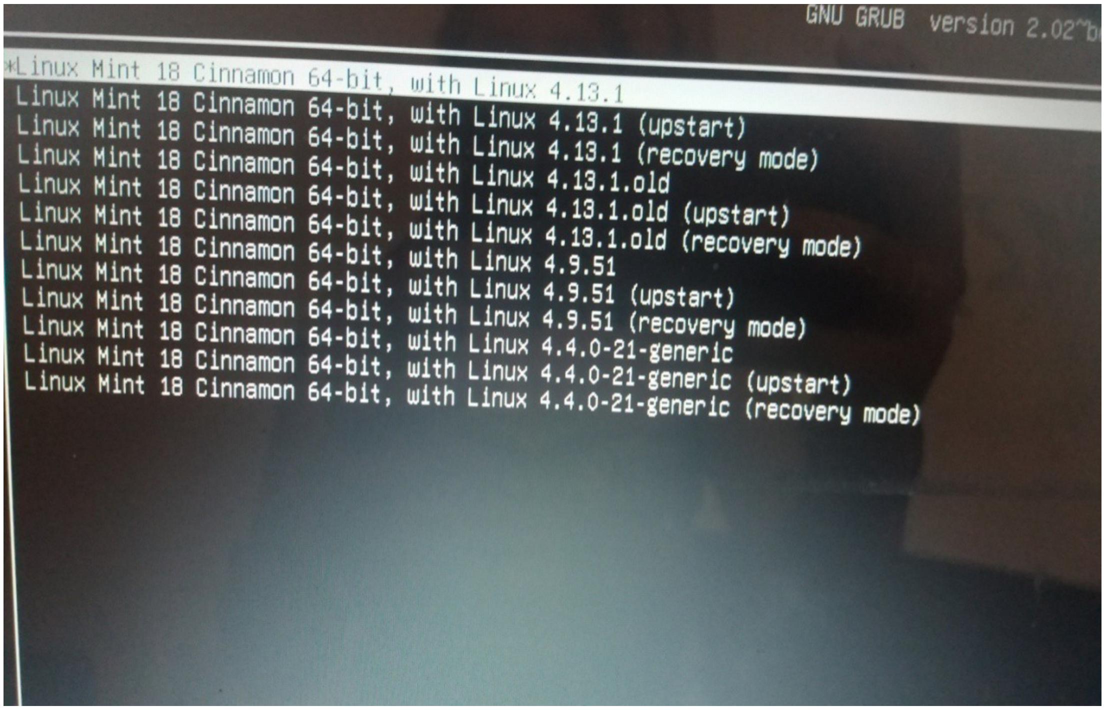


- **“make install”**

  This will automatically copy the kernel to your / boot folder and generate the appropriate files to make it work.



Available kernels after compilation:



9. Functional Testing:

Test for Xenomai:

Before performing the functional tests, the first step is to search for the Xenomai whether it is present in our kernel or not?

After opening the newly built kernel we need to 1st check for xenomai,

Xenomai installation steps:

make –j8

make modules install

make install

Then navigate to the xenomai folder

: *cd Xenomai/*

Now select the xenomai version which is present the folder

: *cd xenomai-3.0.5/*

Now test for xenomai by typing

dmesg tail |

dmesg | grep xenomai

dmesg | grep xenomai-i

Now navigate to the usr file in xenomai-xx xx using ***cd /usr***

Now list the files present in usr file by typing ls and select xenomai from the list of files

cd xenomai/

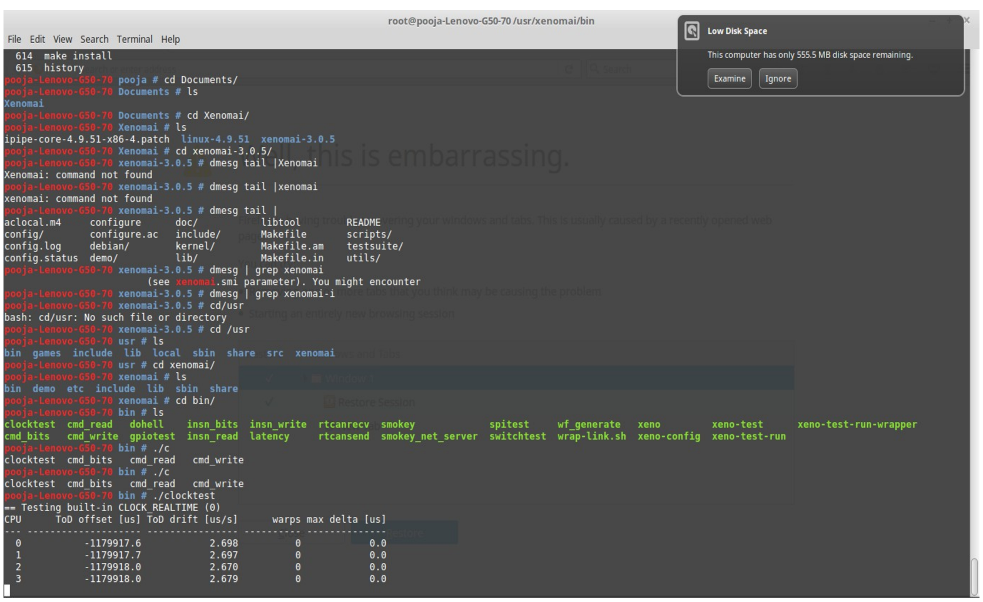
List the files and navigate to bin file using *cd bin/*

Now from bin file perform the latency, clock and switch tests using

./clocktest

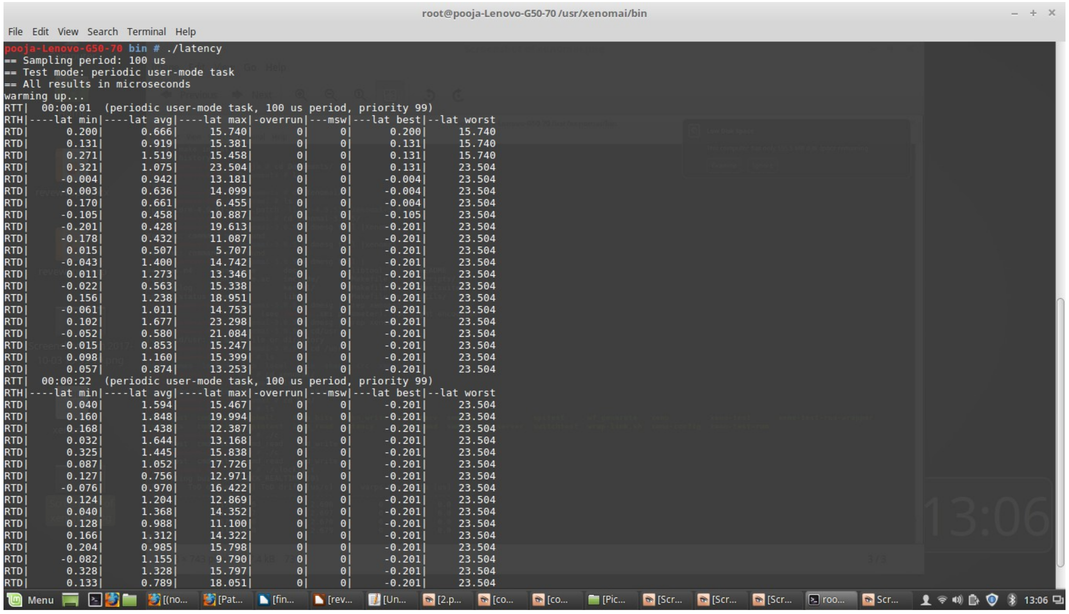
./latency

./switchtest



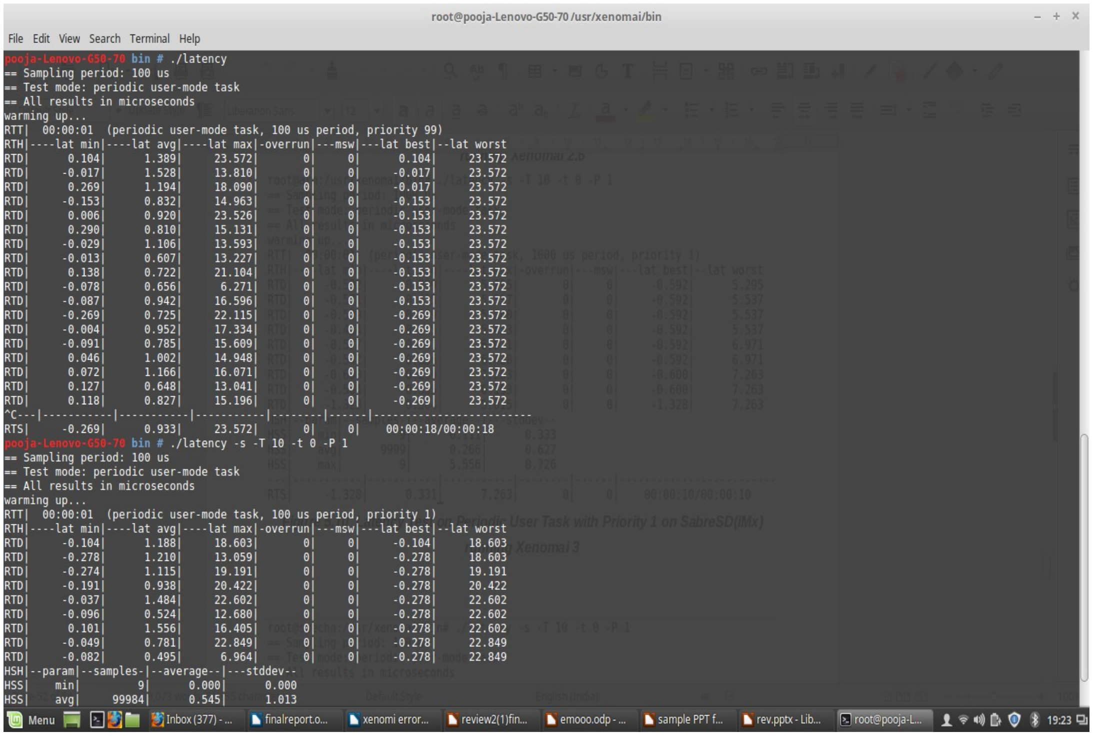
Latency Test:

Performing Latency test using ./latency



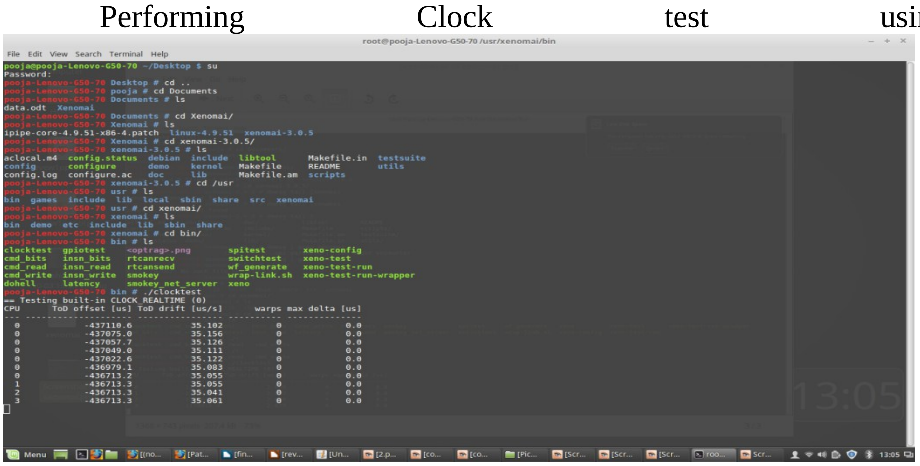
Arguments for Latency Test

-s : Prints Statistics of min, max and average latencies
-T : Test duration in seconds
-t : 0 = User task
1= Kernel Task
2 = Timer Task
-P : Priority of the Task



Clock test:

Performing Clock test using ./clocktest



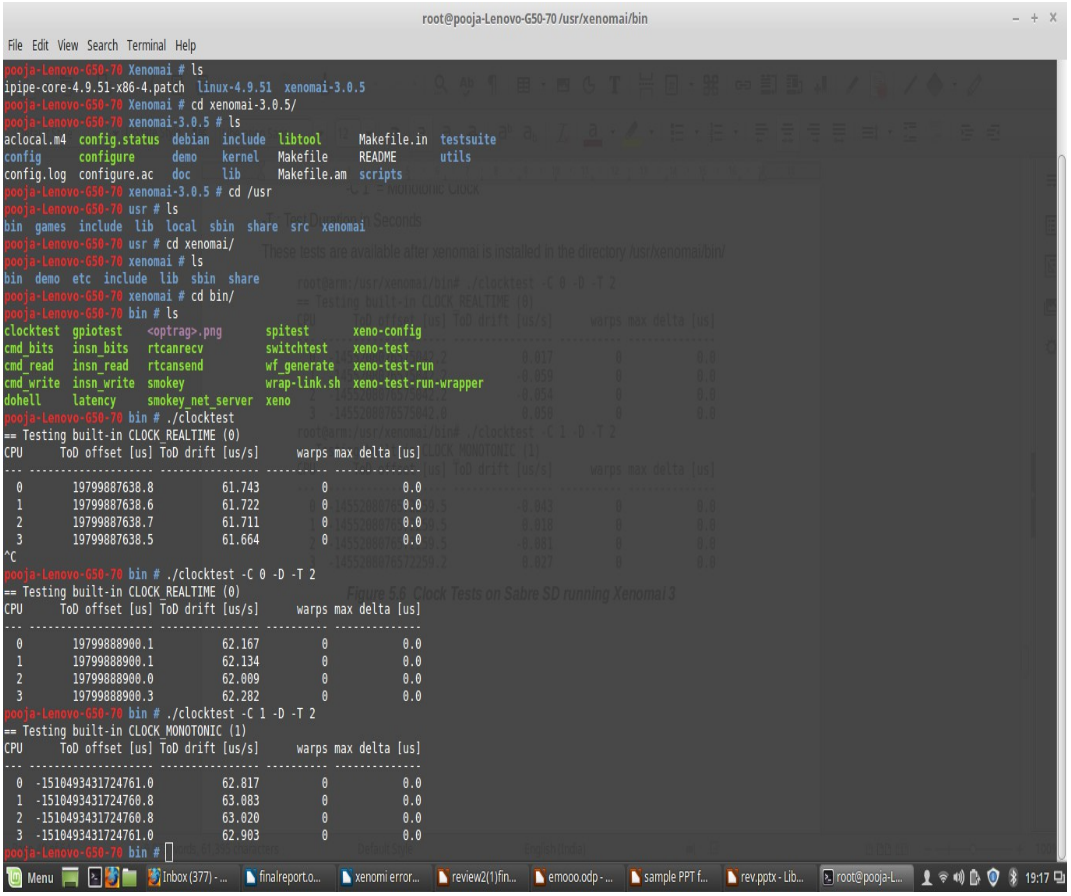
Arguments for Clock test

-C : Clock ID

Usage: -C0 = Real time Clock

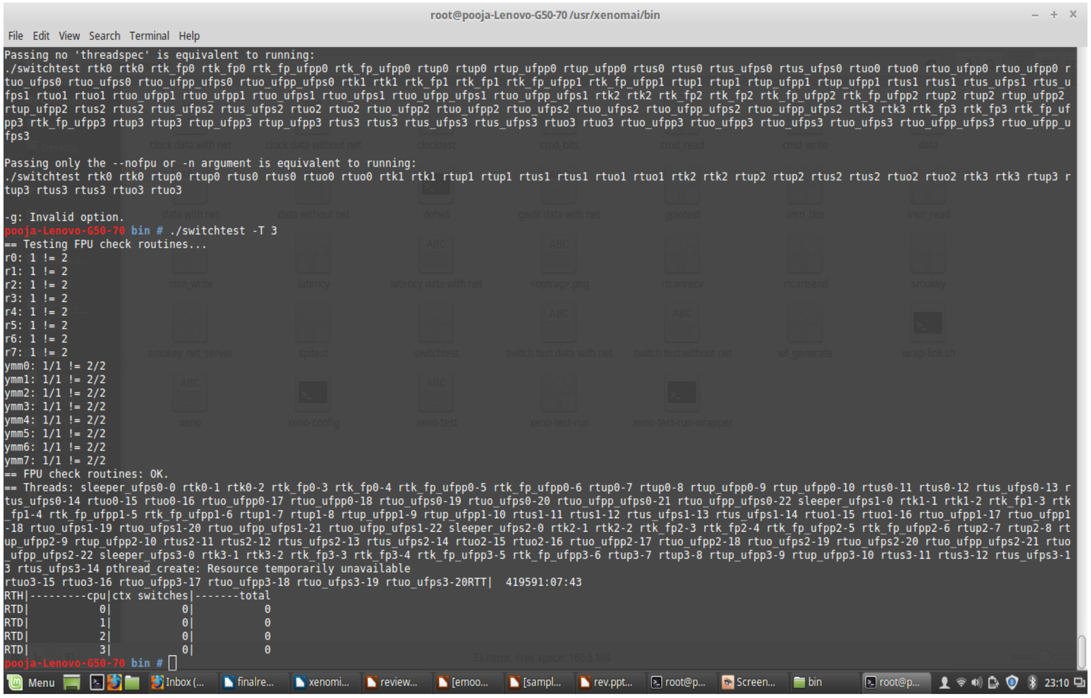
-C1 = Monotonic Clock

-T : Test Duration in Seconds



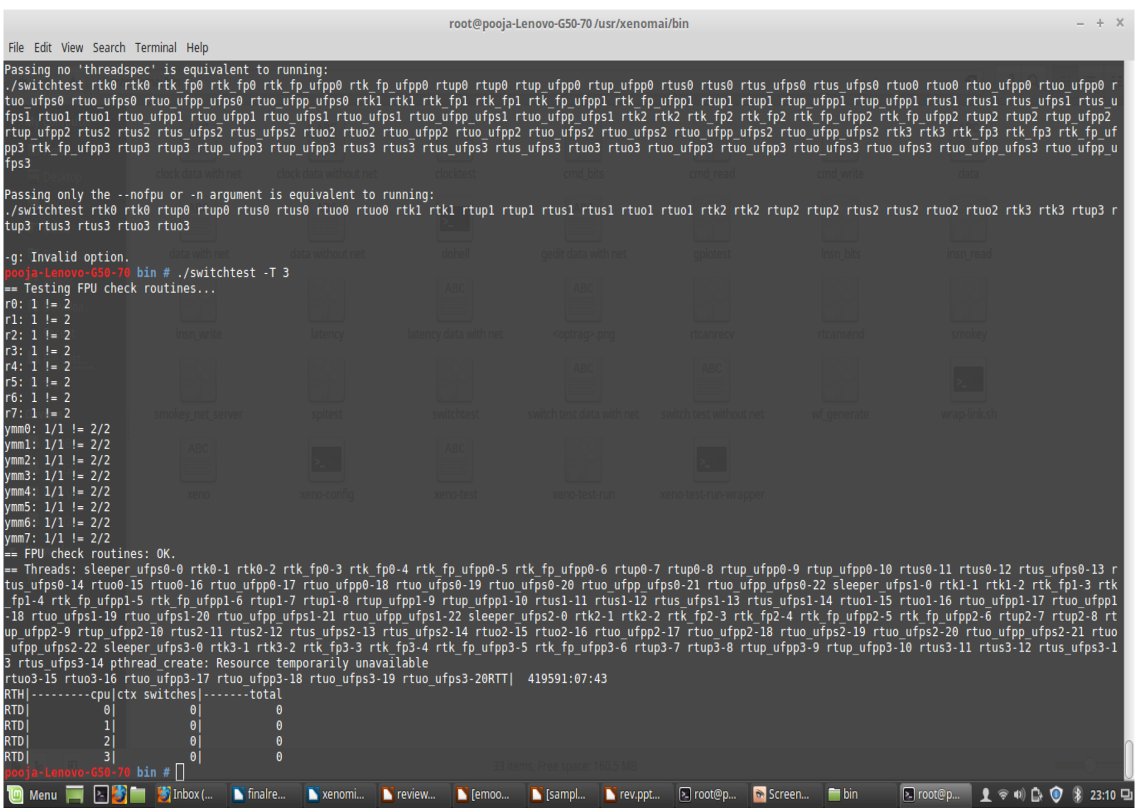
Switch Test:

Performing switch test using ./switchtest.



Arguments for Switch test

T- Time duration for every second.



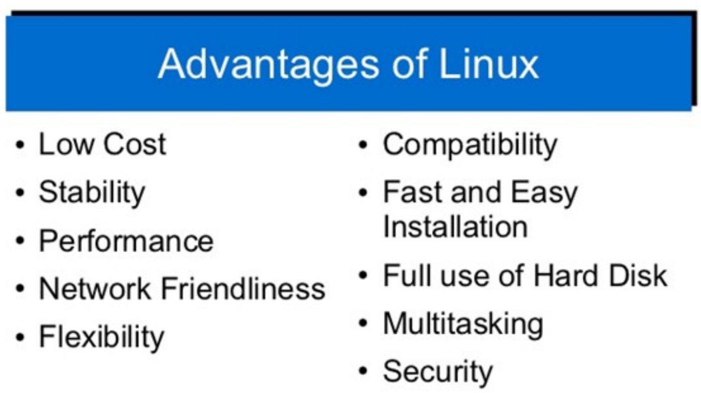
10. Implementing Asymmetric Multiprocessing :

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

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

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 .



Selection of CPU:

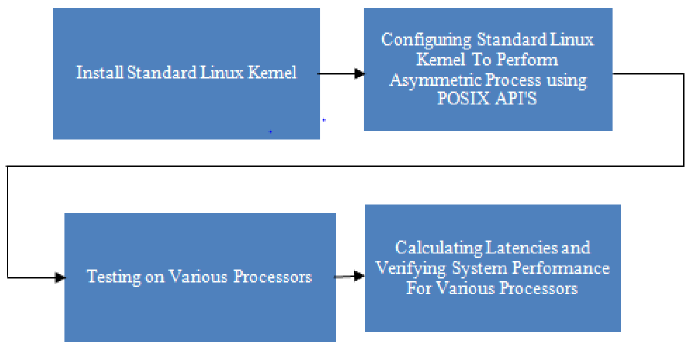
We have implemented asymmetric multi processing on various multi core processors like a. x86 i3 processor with Frequency- 1900 MHz. b. AMD II X3 720 Processor, Frequency- 800 MHz and c. x8 i5 Processor, Frequency- 1400.158 MHz. The system performance and latency was measure for the same. We chose general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

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 ample 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.

Methodology:



Installing Standard Linux Kernel.

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

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

Testing on various processors

Nt 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.

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

Test for performance:

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 performance metrics of the kernel, we need to carry some standard test benches like latency test.

Asymmetric Processing on Multi Core Processors

Various threads were created and each thread is given a specific task.

Creating threads and assigning them specific tasks is showed in the code please refer the aymmetricprocessing.c Save this source code in some file let it me xyz.c and then run the file using the command

cc –pthread xyz.c (For compiling)

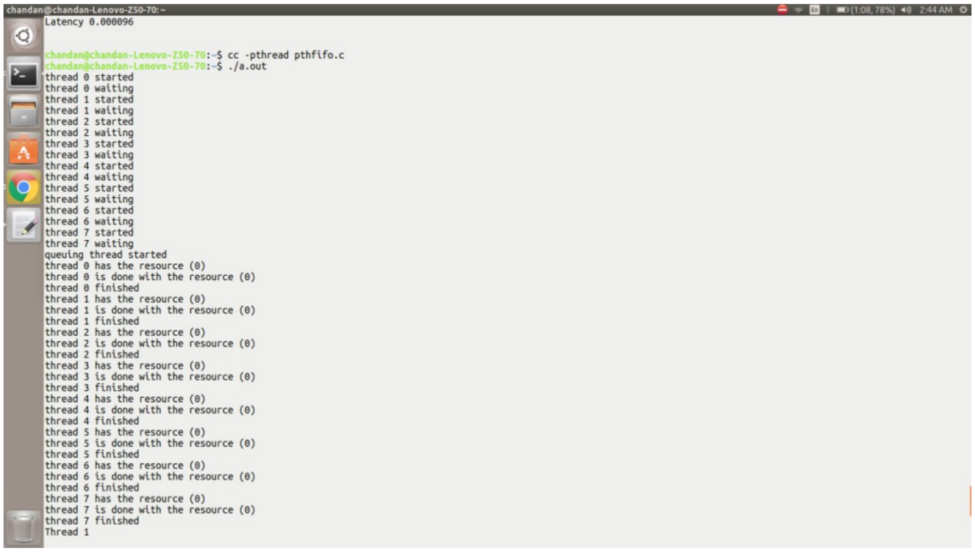
./a.out (For viewing the output)

After running this code four files will be created such as cpu1.txt, cpu2.txt, cpu3.txt, cpu4.txt.

All these files will be created in the same directory from were u executed the source code.

These files gives us the latency values of each individual cpu. Now to check the over all latency of a processor we plot the python plot.

To get the python plot save the code in the same directory with some file name, and execute the code in the terminal using the command *python filename.py*



Creation of threads latency time and end time of each thread.

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

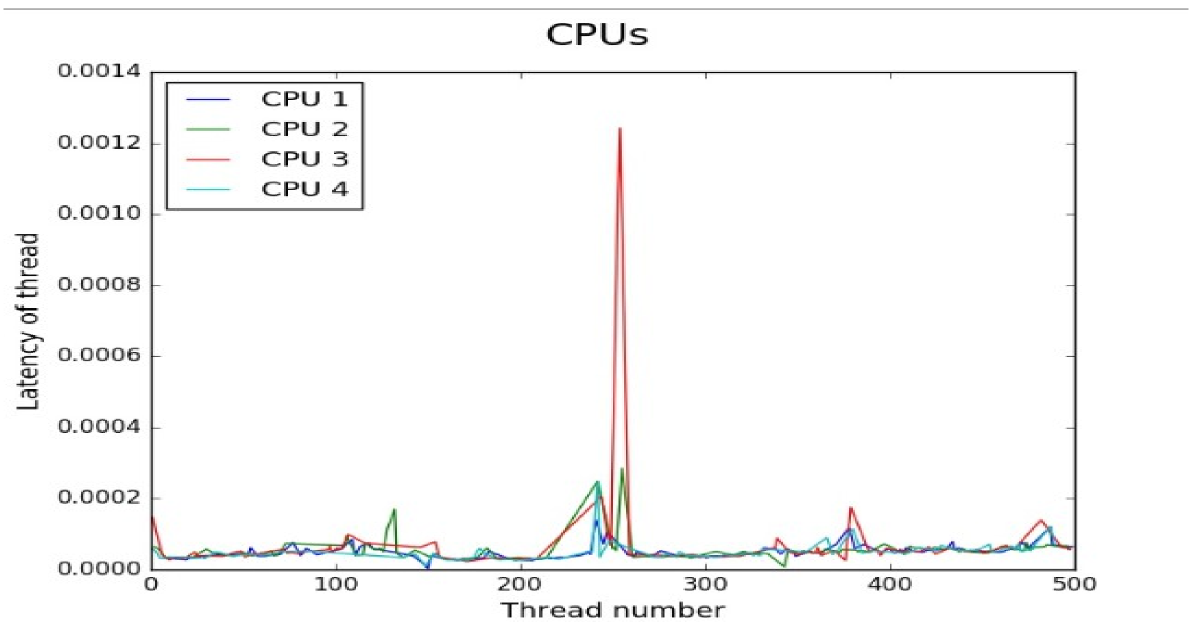
Open system monitor in your Linux system and navigate to the processors and view gnome-system-monitor.

Now navigate to the resources tab and see the out put

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

Run the python code to view the latency plot of the processor, refer to the python code latencyplot.py

After you execute the plot we will get a image in the same directory, that images shows us the plot of latency.

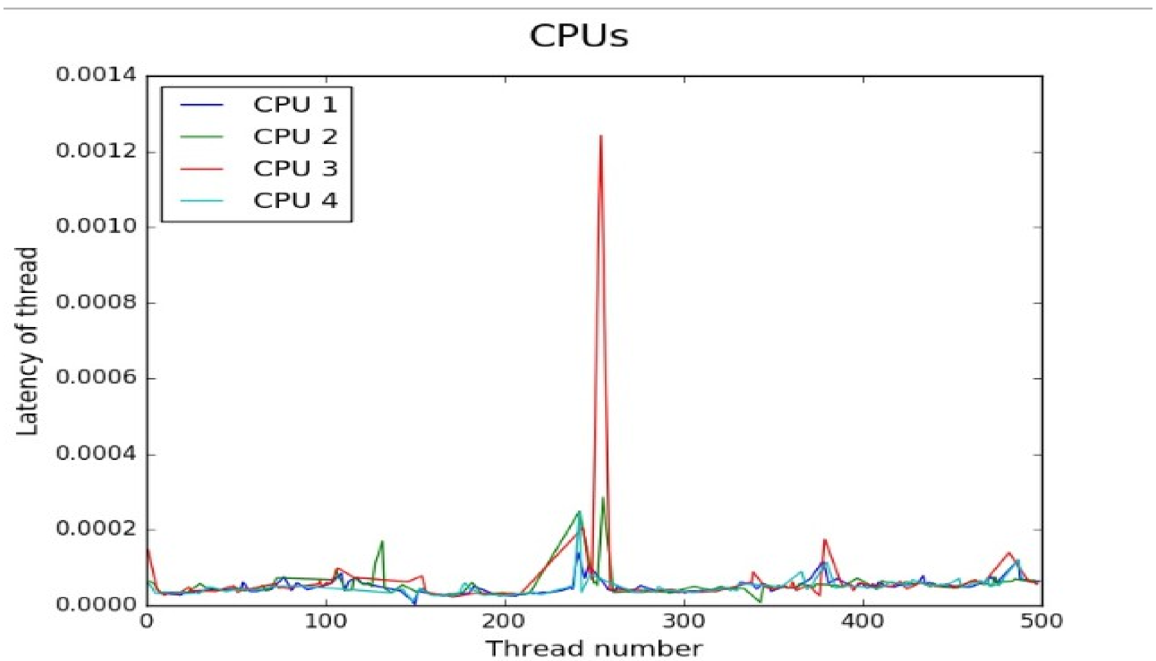


Asynchronous process on standard Linux kernel (x86 i3 processor, Frequency- 1900 MHz)

Repeat the same as above mentioned for i3 processor

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

Run the same python code in i3 processor.

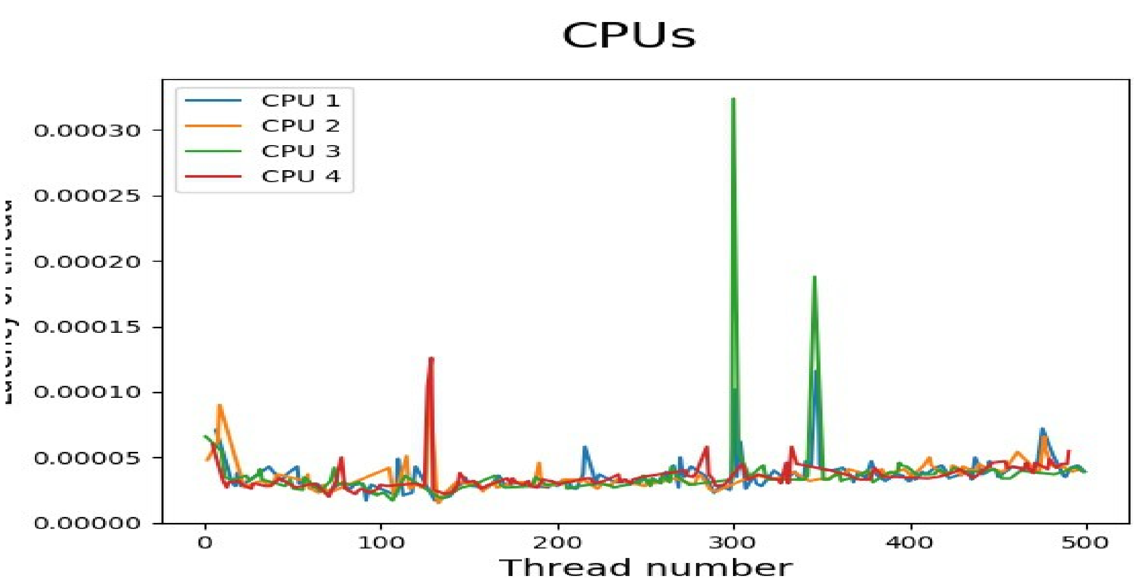


Asynchronous Processing plot on standard Linux kernel (x86 i5 processor, Frequency-1400.158 MHz).

Check the system monitor performance of i5 processor

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

Run the python code in i5 processor



11.Discussion on Results:

We took a stable Linux Kernel and downloaded stable Xenomai version, collected a stable I-PIPE core to patch with Xenomai. Essential packages were built in the stable Linux kernel. Later Linux kernel was patched with Xenomai and configuration was done. Then compiled it and functional tests like Latency, Switch and clock tests were performed to measure the performance.

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.

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 ample 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.