**Project 4a**

# Creating a Temperature Aware variant of the Linux Operating System

**Operating Systems (CS110)**

**Spring 2013**

**Guide: Prof. Shrisha Rao**

**Team Members**


| Name | Roll Number | Contact Number | E-mail |
|---|---|---|---|
| Anne Rachel Gotham | MT2012022 | 9663541668 | annerachel.gotham@iiitb.org |
| Avipsa Nayak | MT2012029 | 9980964252 | avipsa.nayak@iiitb.org |
| Brijesh V. Reddy | MT2012037 | 8050736016 | brijesh.v@iiitb.org |
| Dipanjan Sarkar | MT2012043 | 9980960048 | dipanjan.sarkar@iiitb.org |
| Tushar G. Thomas | MT2012152 | 9535092594 | tushar.thomas@iiitb.org |

Team Leader: Dipanjan Sarkar

# Contents

# 1. Introduction

With the advancement of technology in the 21st century, the performance of processors has increased steadily. While processors are better and more faster now with better computational capabilities, due to their superior speeds, it has led to them consuming more and more power. Power consumed by processors gets converted into heat energy, which ultimately leads to exponential rise in the heat density of processors. Consequently, with the increase in power consumption, heating and heat dissipation are really important factors to be considered. Overheated chips can have various detrimental effects like adversely affecting the processor power, damaging the processor or affecting the reliability and reducing the lifespan of the chip. Besides, shortening of battery life in portable devices like PC's, tablets, mobile phones due to increased heating up of processor is another major technical challenge faced these days. Moreover, overheating can lead to instabilities or permanent damage to the hardware of the system.

An obvious solution to this would be to use improved cooling techniques like using larger heat-sinks or air coolers. However, most of the cooling solutions are quite expensive, hence, power-aware techniques were adopted to deal with the problem of high heating of processors. A power-aware ( low-power ) system is one which focuses upon reducing power consumption in order to increase battery-life, hence improving the overall power efficiency of the system. Consider a PC running on battery power being used for a long time. A power-aware PC would reduce the frequency and supply voltage of the processor which would eventually reduce the power consumed by the system, since power consumed is proportional to the frequency of the processor and to the square of the processor supply voltage. Techniques like `Dynamic Voltage and Frequency Scaling` (DVFS) is used in this regard. But, implementing power-aware design alone was not adequate because of many reasons. Firstly, power is dissipated in a non-uniform manner across the processor chip, thus resulting in localized heating or creating `hot spots` in the processor chip rather than chip-wide heating. Power-aware systems focus upon reducing chip-wide power density and have nothing to do with localized heating of processor chip. Secondly, power-aware techniques decrease power consumption even when CPU utilization is quite low, thus just focusing upon decreasing power-density and not on maintaining utilization/performance of the processor.

Hence came the need for Temperature-Aware systems. In contrast to power-aware techniques, temperature-aware design mainly deals with techniques to control localized heating or hotspots in the processor chip. Besides, these

techniques are basically used when the processor utilization is high. Thus, temperature-aware became more popular than power-aware design. An Operating System is said to be temperature-aware, if it has the capability to control all the processes and daemons running on it. A temperature-aware Operating System can monitor all the temperature related parameters of the system and shut down or suspend several processes which cause high heating of the processing chip.

# 2. Project Description

## 2.1 Objective

The main objective of this project is to take an existing kernel of **Linux** and make it temperature-aware by using a proper temperature measuring utility program to retrieve system temperature values for keeping track of the system temperature parameters and use them to control all the processes and daemons running in the system. To exercise control over all the processes several actions can be taken like suspending or shutting down specific processes if the system temperature exceeds a particular threshold.

## 2.2 Description

The operating systems of today usually have a kernel which helps the user processes and daemons interface with the system hardware. Since many processes exist on a system at any particular instance of time, there usually exists a scheduler which grants each process some time when the processor can execute and work on it. Whenever any executing process needs to do some other work like I/O that particular process is preempted and the processor executes some other process which is waiting in the job queue based on some priority among the processes and other parameters. In general a particular timeslice is alloted for each process when it gets dedicated time for execution by the processor. This particular dedicated CPU time for each process keeps switching among the different processes in the job queue and this is known as pre-emptive scheduling. Pre-emptive scheduling occurs so fast that it looks like several processes execute simultaneously. However in reality only one process is executed at a particular instance of time ( may be different for multi-core processors ).

We have described briefly above what a temperature-aware Operating System (OS) can do. The main difference between a normal OS and a temperature-aware OS is that the normal OS cannot perform scheduling and pre-emption of processes based only on simplistic predefined parameters like process priority or resources blocked by processes and other factors. However, it cannot perform process scheduling based on parameters like system temperature, power dissipation and when the heat is crossing some particular threshold and take actions as necessary. In our project, we will deal with building a temperature-aware OS, which should show its temperature awareness capabilities by gathering system temperature information at any instance necessary using some particular temperature gathering utility. It should be able to process that information and compare with a particular threshold value which we assume to be the safety limit and then take some decisions like deciding whether to stop or suspend particular processes running currently in the OS.

Our project is based on developing a temperature-aware version of the existing linux operating system. The operating system in general has a scheduler which allocates processor time to processes from the job queue. We intend to add some new scheduling algorithm in the linux kernel based some definite parameters which will be useful in making it temperature aware. However since playing around with the kernel can be dangerous and complicate things, we plan to keep modifications to a minimum and also develop a user level program which should be able to monitor temperature of the system at any specific time by retrieving temperature specific data from a system temperature measuring utility program ( like HotSpot ) and then used that data to perform some actions which will broadly be carried out in the form of some system calls to the linux kernel and they will mainly consist of either killing or suspending processes which may either be user processes or system processes ( daemons ) which may be responsible for causing more power consumption and dissipation of heat from the system.

# 3. System Architecture

Our project is based on a temperature-aware variant of linux operating system, wherein the OS has the capability to detect temperature variations in the system and can suspend some processes/daemons if the processor temperature exceeds a thermal threshold. The architecture of our system is portrayed in the figure below.

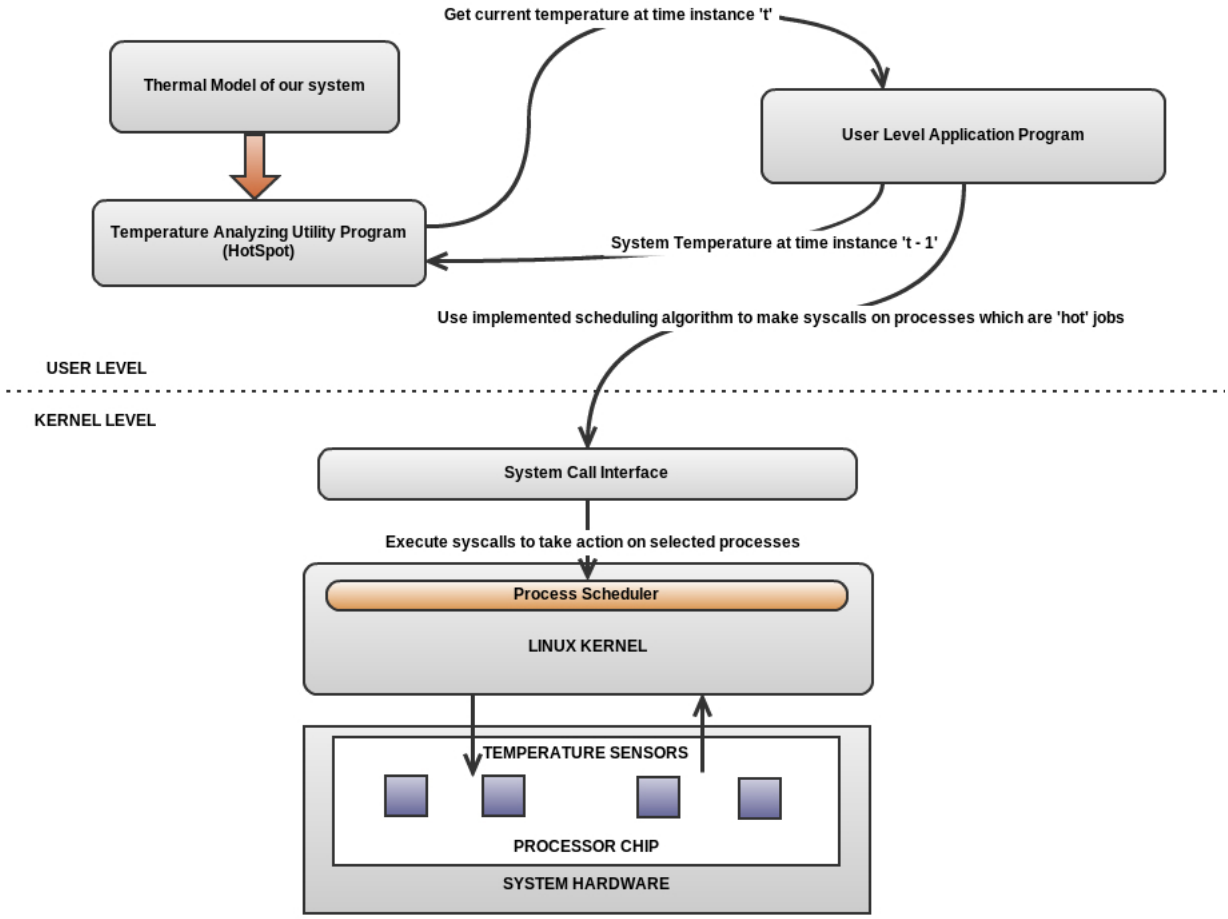


Figure 1: System Architecture of Temperature-Aware Linux OS

The major components of our system's architecture are described in detail below:

- **Temperature Analyzing Utility Program:** The thermal model is constructed first according to the floorplan of our system which would be based on the equivalent circuit which corresponds to the microarchitecture of our processor. We can model this if necessary in the `hotspotUI.xls` file provided by `HotSpot`. We will be using this model in the temperature analyzing utility program ( like `HotSpot` ) to get the temperature of our system dynamically. The utility tool is written in `C` language and it will take the temperature input obtained from the `temperature sensors` using any hardware monitoring tool like `lm-sensors` from our user application program and use the thermal

model along with this temperature to dynamically predict the temperature of the system at any time instance in the future. `HotSpot` basically performs *"temperature tracking"* using techniques like `Dynamic Frequency Scaling` and it can inform if any hot spots occur in our system due to excessive heating or when the temperature crosses a particular threshold.

- **User Level Application Program:** The user level application program will be developed by us using `C` language and it will be running at the user level. The main modules of the program will include the following.
  - Using the hardware monitoring tool to get processor temperature from the sensors in the hardware.
  - Feeding the data obtained as input to `HotSpot`.
  - Retrieving the output from `HotSpot` which will contain the system temperature information at a particular time instance.
  - A scheduling algorithm to determine which processes may be considered as **'hot'** jobs.
  - Some system calls to the linux kernel to take action on processes.
- **System Call Interface:** This basically acts as an interface or bridge between our application program and the linux kernel. All system calls and requests to the kernel pass through this interface.
- **Linux Kernel:** The linux kernel acts as an interface between the user application and the hardware resources. It consists of the `Scheduler` which will be used to to suspend/kill a hot job as directed by our user program and allocate dedicated processing time to the other processes waiting in the job queue.
- **System Hardware:** Consists of the physical/hardware devices in a Linux system like the processor chip and temperature sensors which are used to get the system temperature information necessary to make our system temperature-aware.

Basically we have considered several algorithms for implementation of the process scheduling while making our temperature-aware OS. They have been briefly described below :

- **Random:** Here, the algorithm just picks one random process and executes it. This is usually not very feasible in practical situations.

- **Priority:** Here, the algorithm gives more prority to cool jobs than hot jobs. At the first glance, this sounds to be good but when all the cool jobs are exhausted, the hotter jobs must be executed and hence this violates the thermal conditions and this triggers frequency scaling which lowers the performance of the processor and degrades it.

- **Mintemp:** This algorithm executes the hot job when the temperature is below the threshold otherwise it executes the cold job. Thus, the cold job is executed only when the frequency is scaled which is unfair as it violates the fairness policy of the scheduler.

- **Earliest Deadline First:** This algorithm schedules jobs with earlier deadlines and hence reduces the delay associated with other tasks. This in turn consumes less power and thus, saves a considerable amount of energy.

- **Threshot:** In this algorithm, the hot job is executed first and then the cooler job is executed, provided the hotter job does not exceed the threshold level. Here, the threshold level is usually set either by the manufacturer or by the OS.

We shall be implementing the following steps in our project's architecture :

I. To get the temperature values of the system components, we will be using a utility program which can form an accurate thermal model and provide us the temperature data.

II. To deal with the issue of thermal management, we'll be implementing a temperature-aware scheduling algorithm at the user level in our application program and modify the scheduler in the linux kernel. In order to ensure that the processor's temperature doesn't exceed the thermal threshold, dynamic thermal management techniques are used where speed and frequency scaling take place.

III. Like we mentioned earlier we have considered several potential algorithms which can be used for implementing scheduling as mentioned earlier.

IV. Our main objective in scheduling is to know the temperature of the processes which is usually detected by the thermal sensors and our thermal model at runtime.

V. Whenever the temperature exceeds the threshold values, some system calls should be made so that we can throttle the process which permits the chip to consume less power, for instance by frequency scaling i.e it reduces the speed of the processor.

VI. While our program runs, the temperature of the process during the next interval should be computed at proper intervals of time as the same process which generates some amount of heat at one instance may or may not generate the same amount of heat at every instance of time taking into various factors into account.

VII. The OS obtains system temperature related information from the performance counters of the processor. Those counters record the activities of the current job during the previous interval. Then, they are converted into the power consumption values at the granularity of functional units. Prediction of the power is performed at this point of time. The past power values are then fed into a full-chip thermal model in order to compute the current temperature at the current scheduling interval. For all candidate jobs, their future temperatures are also calculated by using their predicted power values. All these future temperatures are sent to the scheduler to determine the next job selection.

VIII. Our user level program will make use of the above strategies and make appropriate system calls to determine which processes should be killed or suspended.

# 4. Gap Analysis

Most modern operating systems don't have any kind of temperature-awareness except when the temperature usually crosses some critical level, the system is shut down. However most of the time it is unable to detect the imminent dangers of potentially dangerous processes which may cause the processor to heat up suddenly or lead to the formation of hot spots. Hence before any proper action can be taken the hardware is prone to damage.

There exist many ways to implement cooling techniques. One of the ways to cool the computer's processor is by lowering processor clock speed to reduce heat generation. This is known as clock throttling [1] which has seen the most development in recent years. This works because the amount of activity occurring on the processor decreases. This decrease in activity impacts the performace of the processor as fewer number of instructions will be executed

every second. Another way of trying to solve the problem of excess heat generation on the computer chips is to decrease the input supply voltage. The challenge faced here is that the performace of the machine may be hit.

Power-aware design alone has not helped in reducing the cooling costs of the microprocessors. Temperature-aware designs make use of power-management techniques. These power management techniques must directly target the spatial and temporal behavior of operating temperature. However most systems today don't use intelligent temperature detecting systems which keep track of the power consumed by processor while executing different active processes.

The approach in this project is to be able to use an existing version of linux operating system and modify it to be temperature-aware. We intend to use a temperature detecting utility program which can be used to measure the system temperature and retrieve the measured values which is then compared to a dynamic threshold value. Once this value is exceeded, certain services or daemons will be shut down.

# 5. Development Plan

To implement a temperature-aware version of the Linux Operating System, our project will go through the following phases of development :

**Phase I**

- Installation and configuration of Linux OS on the systems.
- Study about various temperature-detection and analyzing utility programs that can be used to provide temperature related information of a system.
- Installation of a temperature analyzing utility tool (like HotSpot) on the system which will be used to feed information to our application.

**Phase II**

- Proper configuration of the temperature analyzing tool using various input parameters needed.

- Detailed study about the temperature-analyzing utility tool to familiarize with the kind of inputs it needs, how the processing is done and how it outputs the temperature related information of each functional block in the system.
- Start formulating the thermal model of our system which will be a crucial input parameter.
- Deliverables:
    - Proper thermal model structure for our system to be used by the temperature analyzing tool.

**Phase III**

- Implementation of our user application which should be capable of retrieving system temperature at any particular time instance.
- Modification of the temperature-detection utility tool to obtain the temperature related data of the system at time-instance **'t'** by feeding the utility tool with the thermal model of our system (as developed in Phase II ) and the system temperature at time-instance **'t-1'** obtained from the processor temperature sensors as discussed in the architechture earlier.
- Further, study how to identify, retrieve and analyze information about the hotspots from the output set obtained.
- Deliverables:
    - User Application capable of retrieving system temperature details at any instance of time.
    - Modified Temperature Utility tool to give dynamic information about temperature.

**Phase IV**

- Develop understanding about the characteristics of the linux kernel and the scheduler used by it.
- Gather information about how scheduling is performed by the scheduler and which system files are responsible for it.

- Detailed study about the various potential scheduling algorithms to make our system temperature aware (as discussed in the architechture section earlier ).

**Phase V**

- Implementation of a scheduling algorithm in the user application ( as developed in Phase III ), which fetches data from the temperature-detection tool, analyzes the data and checks if the temperature of the system exceeds the thermal threshold values.
- Modify the user application to make system calls to the scheduler so as to throttle hot processes, hence permitting the chip to consume less power and maintain the system temperature below the thermal threshold value.
- Deliverables:
  - Working version of a temperature-aware scheduling algorithm.
  - User Application Program capable of making system calls with full scheduling capabilties.

**Phase VI**

- Implementation of modifications to the linux kernel and its scheduler in order to execute the system calls made by the user application to suspend/kill a hot job.
- Compilation of the modified linux kernel.
- Deliverables:
  - Modified version of the successfully compiled linux kernel.

**Phase VII**

- Design test-cases to test the functionality of the developed system.
- Perform testing with different threshold values and make modifications as necessary.

# 6. Milestones

| Milestone | Task | Due Date |
|---|---|---|
| Phase I | • Installation and configuration of linux OS.<br>• Study Temperature detection utility tools and complete installation and configuration. | Feb 4, 2013 |
| Phase II | • Study and formulate a proper thermal model of our system.<br>• First brief presentation of project architecture and plan by the team. | Feb 5, 2013 |
| Phase III | • Detailed study of the linux kernel and its scheduler.<br>• Start developing user application to give system temperature at any instance of time. | Feb 15, 2013 |
| Phase IV | • Study of the various scheduling algorithms to be implemented in the user application.<br>• Test different scheduling algorithms and select the best one. | Feb 27, 2013 |
| Phase V | • Implement a temperature-aware scheduling algorithm in our user application.<br>• Detailed mid-term review of project. | March 4, 2013 |
| Phase VI | • Modify our user application so it can make system calls to the kernel scheduler.<br>• Modify linux kernel scheduler so that it can execute system calls made by the user application.<br>• Make system calls to the kernel scheduler and test the application's scheduling capabilities. | March 25, 2013 |
| Phase VII | • Start testing of the various modules and perform integration testing with various threshold levels.<br>• Completion of project and documentation. | April 18, 2013 |

# References


[1] D. Rajan and P. Yu, "Temperature-aware scheduling: When is system-throttling good enough?," in *The Ninth International Conference on Web-Age Information Management(WAIM'08), Zhangjiajie, China*, pp. 397–404, July 2008.

[2] Y. Zhang, X. Hu, and D. Chen, "Task scheduling and voltage selection for energy minimization," in *39th Design Automation Conference(DAC'02), New Orleans, LA, USA*, pp. 183–188, 2002.

[3] J. Choi, C.-Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose, "Thermal-aware task scheduling at the system software level," in *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'07), Portland, OR, USA*, pp. 213–218, Aug. 2007.

[4] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic thermal management through task scheduling," in *IEEE International Symposium on Performance Analysis of Systems and software(ISPASS'08), Austin, Texas, USA*, pp. 191–201, April 2008.

[5] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *30th Annual International Symposium on Computer Architecture(ISCA'03), San Diego, California, USA*, pp. 2–13, June 2003.