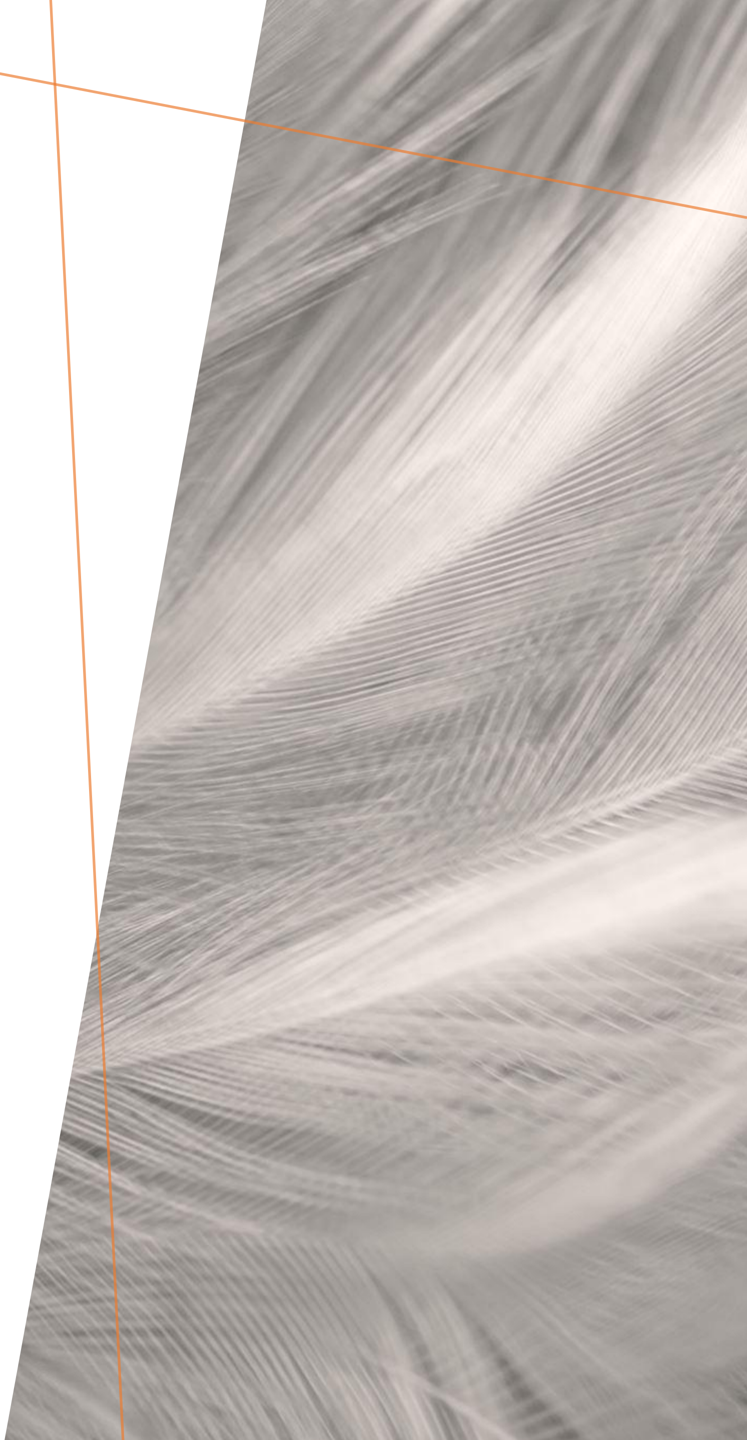
CS3500 Course Project: Energy Aware Scheduling



Literature Review

1. We have familiarized ourselves with the paper given by Nikhilesh Sir. The paper can be found here.
2. We read the chapters related to scheduling in Professional Linux Kernel Architecture Wolfgang Maurer.
3. The block diagram of the paper can be found to the right
4. Successfully installed the linux kernel in one of our systems.

BLOCK DIAGRAM OF THE PAPER

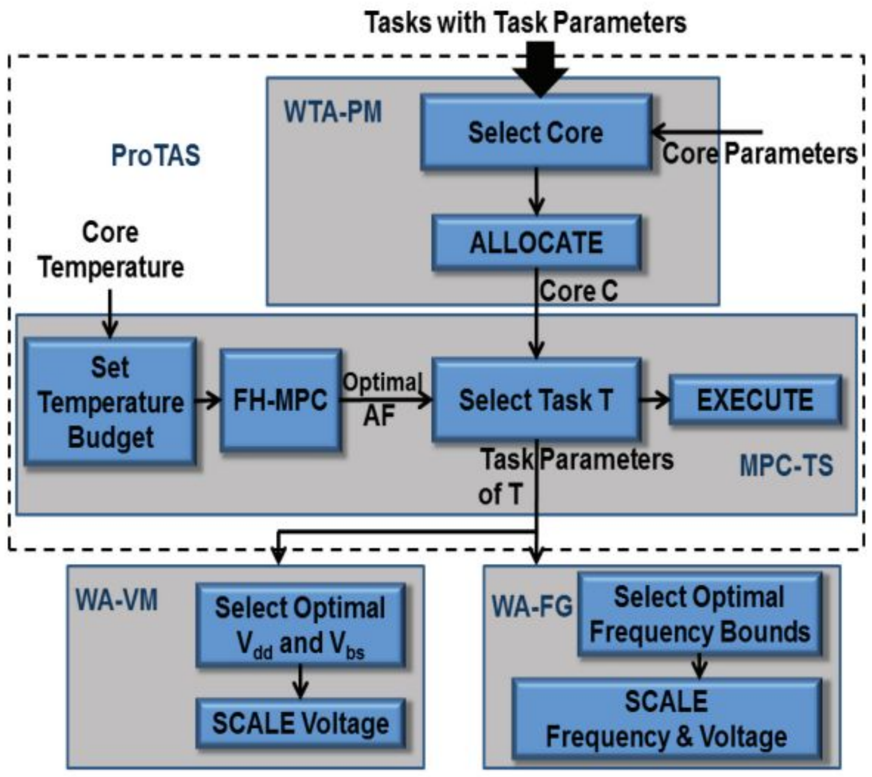
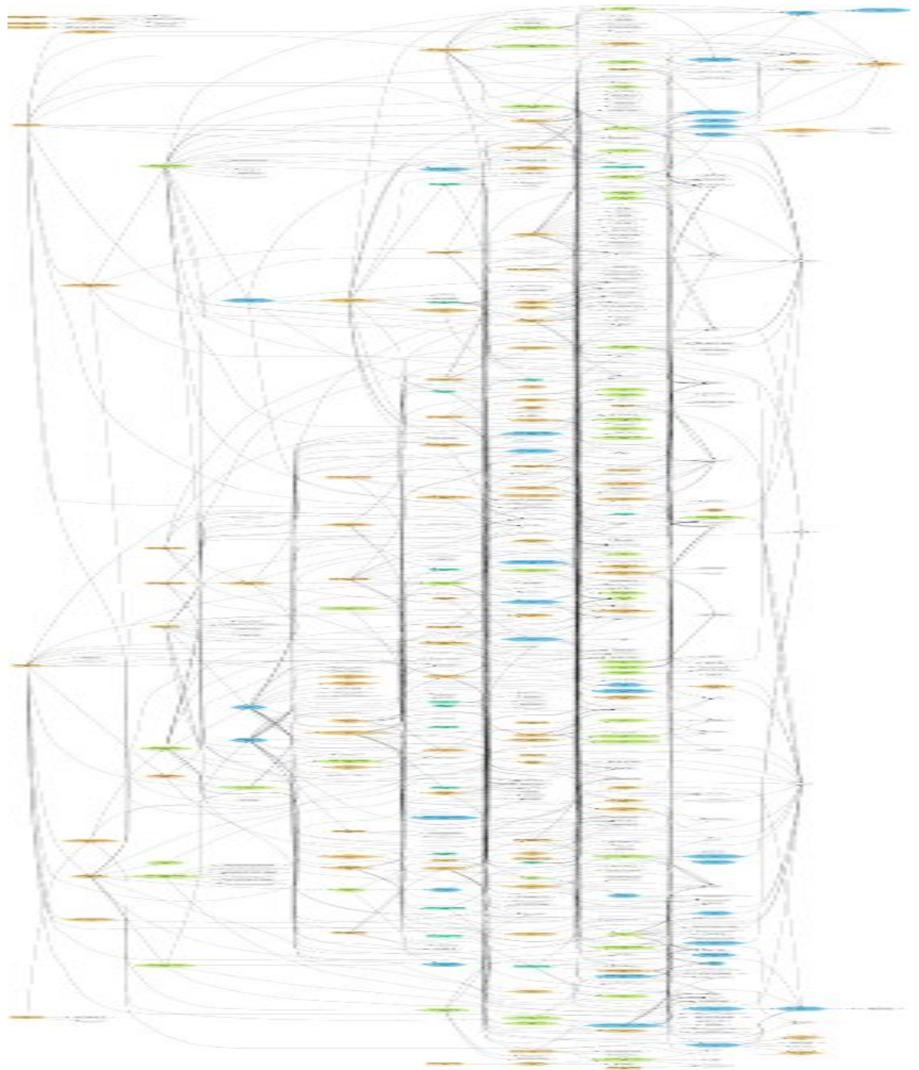


Fig. 1. Block diagram: ProWATCh.

PROJECT REPORT - ENERGY AWARE SCHEDULING

Understanding the code and building blocks



To get a broad level idea of fair.c in kernel/sched we created a control flow graph that can be found herewith.

We developed iteratively and incrementally following basic building blocks. These are:-

- Getting the frequency of CPU.
- Setting the frequency of CPU
- Getting the temperature of the CPU.

Arnhav Datar(CS18B003) *Shriram Chandran(CS18B007)*

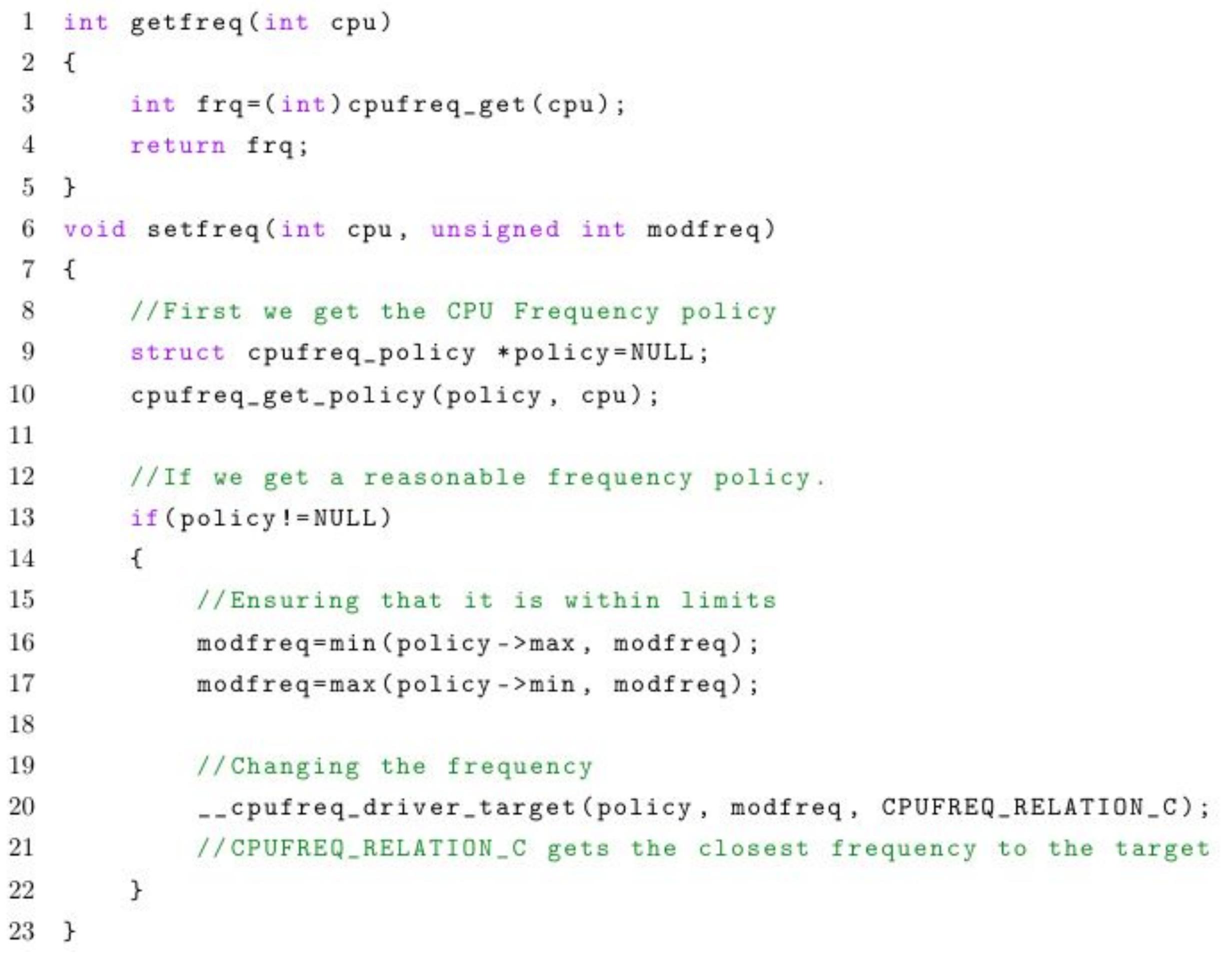
Set and Get Frequency functions

We use the linux/sched.h to create the get and set frequency functions they are given attached herewith

The get frequency function is easily created using a simple function from sched.h

The set frequency function

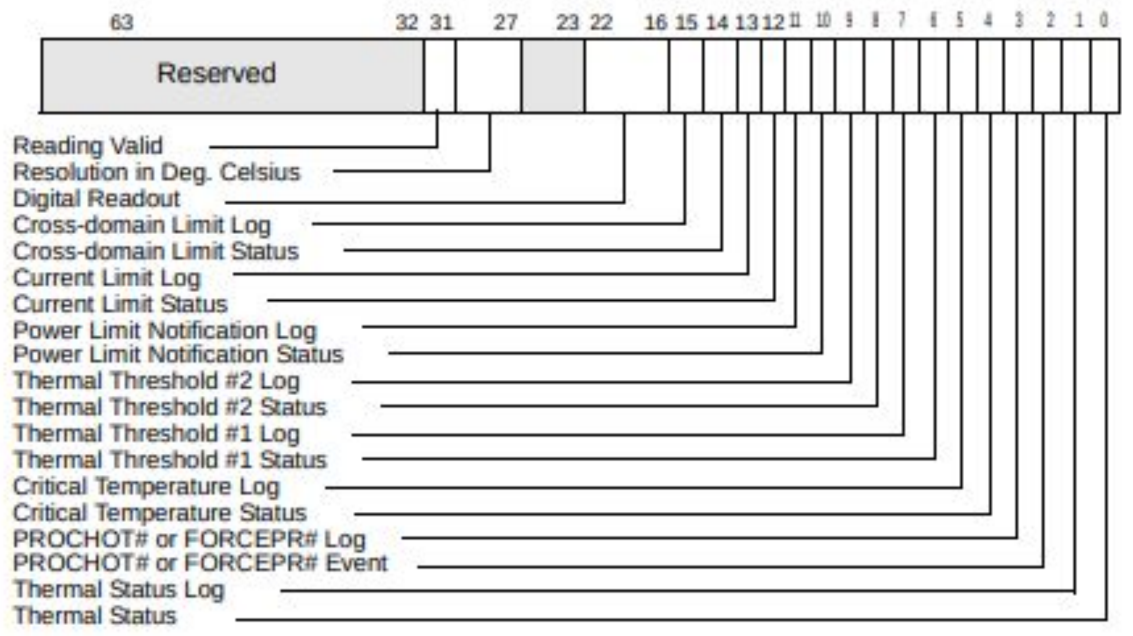
1. Finds the cpu frequency policy
2. Ensure the given frequency to be set is within bounds
3. Changes the frequency to the closest frequency using the CPUFREQ_RELATION_C flag which assigns the closest possible frequency to the CPU.



```c
int getfreq(int cpu)
{
    int frq=(int)cpufreq_get(cpu);
    return frq;
}
void setfreq(int cpu, unsigned int modfreq)
{
    //First we get the CPU Frequency policy
    struct cpufreq_policy *policy=NULL;
    cpufreq_get_policy(policy, cpu);

    //If we get a reasonable frequency policy.
    if(policy!=NULL)
    {
        //Ensuring that it is within limits
        modfreq=min(policy->max, modfreq);
        modfreq=max(policy->min, modfreq);

        //Changing the frequency
        __cpufreq_driver_target(policy, modfreq, CPUFREQ_RELATION_C);
        //CPUFREQ_RELATION_C gets the closest frequency to the target
    }
}
```

Get Temperature function

```c
int gettemp(int cpu)
{
    //Get the temperature
    uint64_t temp=0, vfl, msk;
    rdmsrl_safe_on_cpu(cpu,
    MSR_IA32_THERM_STATUS, &temp);

    // A mask to check validity
    vfl=1<<31;
    //A mask to get only the temperature
    msk=1111111;

    //Check for validity
    if(temp&vfl){
        //Take bits 22:16
        temp=temp>>16;
        temp= temp&msk;
        // return the temperature
        return temp;
    }
    else
        return -1;
}
```



IA32_THERM_STATUS Register

Experimentation with different heuristics

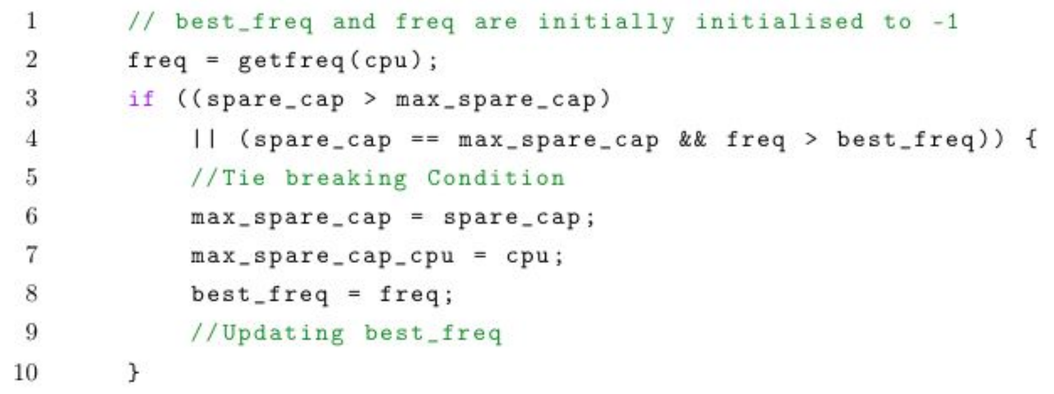
Some heuristics we worked on:-

1. Using frequency as a tie-breaker when two cpus have the same performance domain.
2. Using Temperature as a tie-breaker when two cpus have the same performance domain.
3. When the CPUs are switched we reduce the frequency of the CPU getting the new process by a small fraction
4. When the CPUs are switched we increase the frequency of the CPU which had the process i.e. prev_cpu.

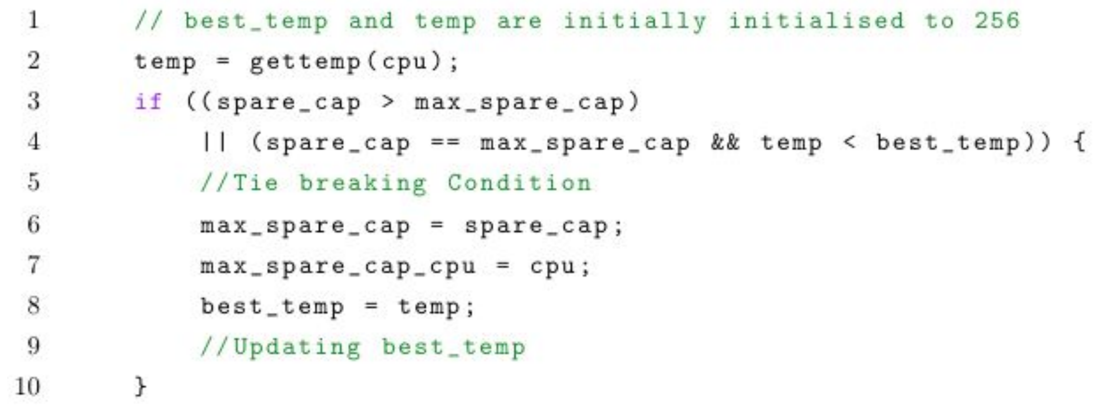
Tie Breaking Heuristics

Spare Capacity of the CPU is the criterion for choosing optimal CPU. We try two different tie-breaking criterion.

Tie-breaking code based on frequency. Here best_freq was initialised to -1. In case two CPUs have equal performance domain we give control to the CPU with higher frequency.



```
// best_freq and freq are initially initialised to -1
freq = getfreq(cpu);
if ((spare_cap > max_spare_cap)
    || (spare_cap == max_spare_cap && freq > best_freq)) {
    //Tie breaking Condition
    max_spare_cap = spare_cap;
    max_spare_cap_cpu = cpu;
    best_freq = freq;
    //Updating best_freq
}
```



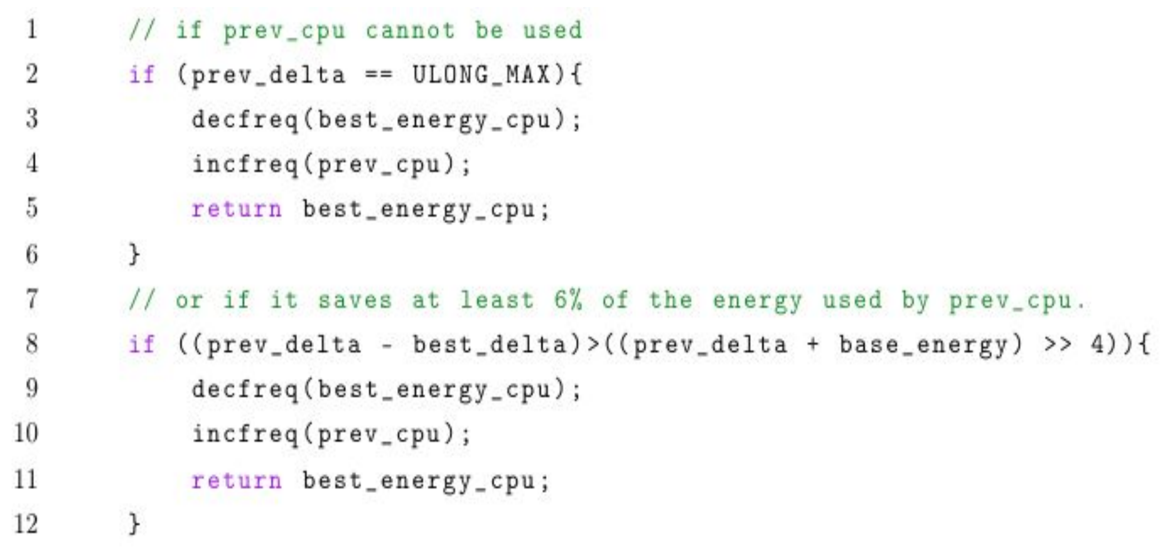
```
// best_temp and temp are initially initialised to 256
temp = gettemp(cpu);
if ((spare_cap > max_spare_cap)
    || (spare_cap == max_spare_cap && temp < best_temp)) {
    //Tie breaking Condition
    max_spare_cap = spare_cap;
    max_spare_cap_cpu = cpu;
    best_temp = temp;
    //Updating best_temp
}
```

Tie-breaking code based on temperature. Here best_temp was initialised to 256. In case two CPUs have equal performance domain we give control to the CPU with lower temperature.

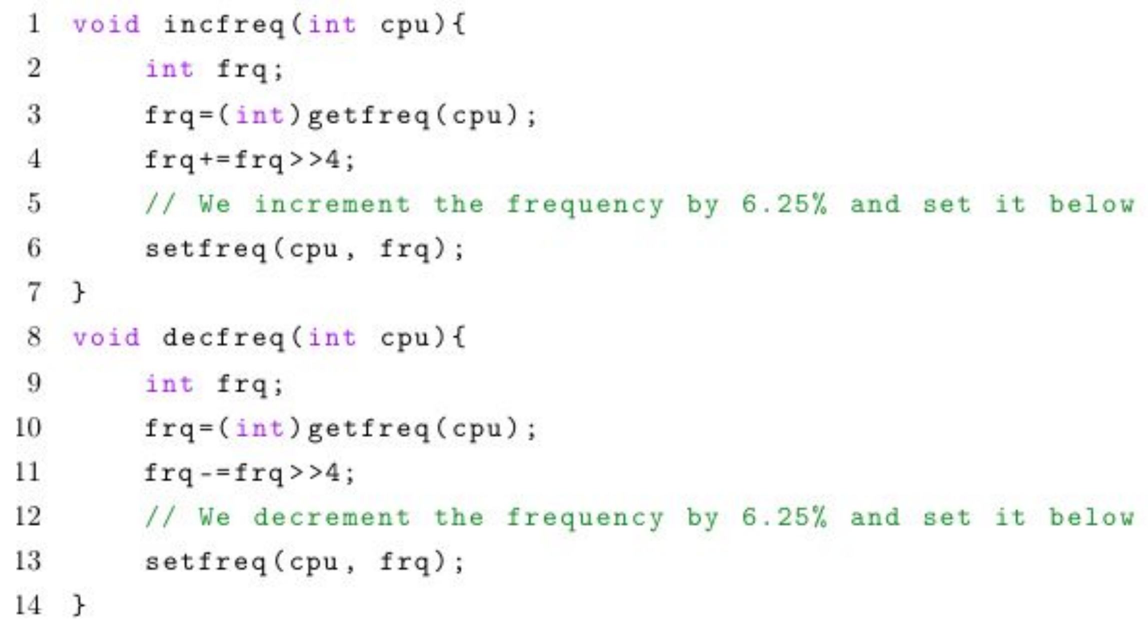
PROJECT REPORT - ENERGY AWARE SCHEDULING

Frequency Modification Heuristics

Frequency updates while switching processes. We mainly use the helper functions here. We update if the previous CPU cannot be used or the there is at least 6.25% lesser energy used by the new CPU. We increment the frequency of the CPU losing the process and decrement the frequency of the CPU that gets the new process.

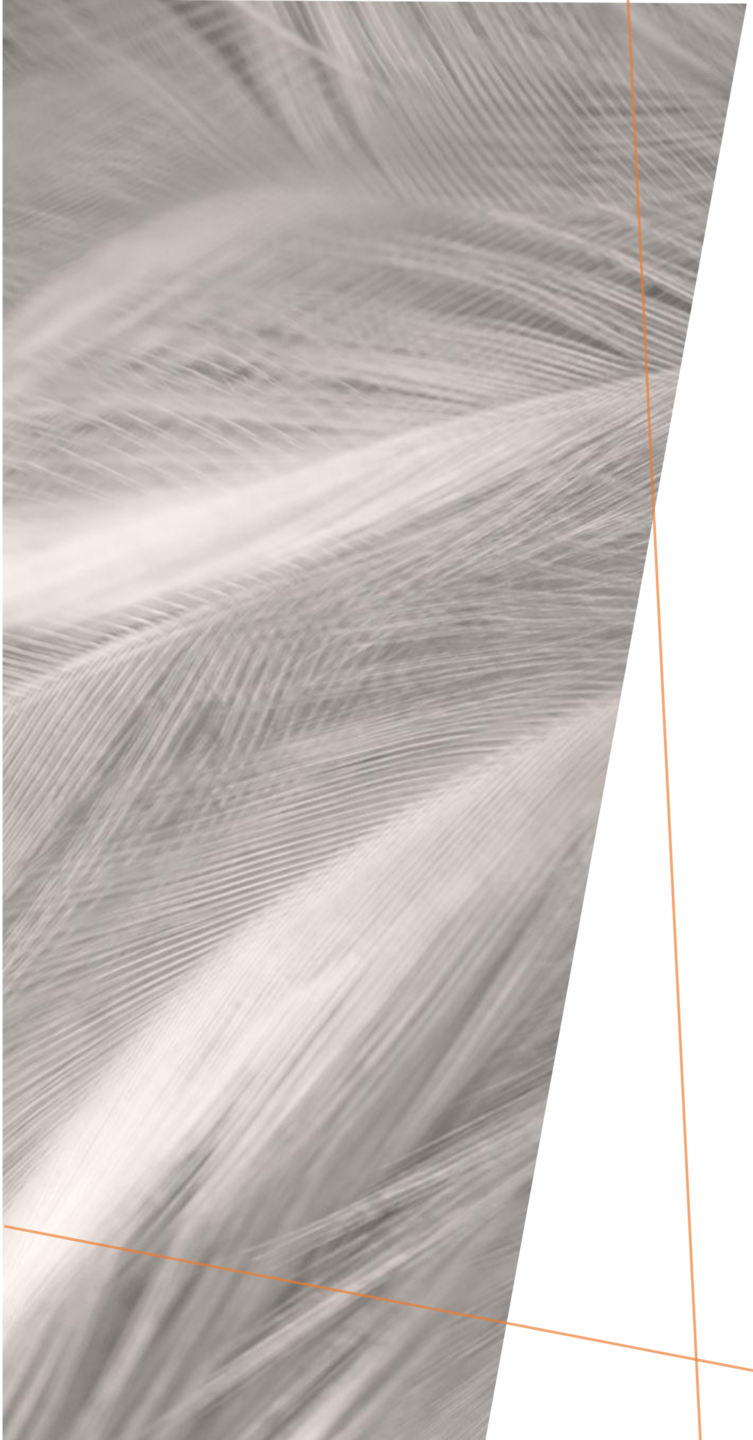


```
// if prev_cpu cannot be used
if (prev_delta == ULONG_MAX){
    decfreq(best_energy_cpu);
    incfreq(prev_cpu);
    return best_energy_cpu;
}
// or if it saves at least 6% of the energy used by prev_cpu.
if ((prev_delta - best_delta)>((prev_delta + base_energy) >> 4)){
    decfreq(best_energy_cpu);
    incfreq(prev_cpu);
    return best_energy_cpu;
}
```



```
void incfreq(int cpu){
    int frq;
    frq=(int)getfreq(cpu);
    frq+=frq>>4;
    // We increment the frequency by 6.25% and set it below
    setfreq(cpu, frq);
}
void decfreq(int cpu){
    int frq;
    frq=(int)getfreq(cpu);
    frq-=frq>>4;
    // We decrement the frequency by 6.25% and set it below
    setfreq(cpu, frq);
}
```

Arnhav Datar(CS18B003) Shriram Chandran(CS18B007)



Thank You