Linux Energy Monitor Application

1.0

Generated by Doxygen 1.9.7

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 raymii::Command Class Reference

Static Public Member Functions

- static CommandResult exec (const std::string &command)
- static CommandResult **execFgets** (const std::string &command)

3.1.1 Member Function Documentation

3.1.1.1 exec()

```
static CommandResult raymii::Command::exec (
            const std::string & command )  [inline], [static]
```

Execute system command and get STDOUT result. Regular system() only gives back exit status, this gives back output as well.

Parameters

command	system command to execute

Returns

commandResult containing STDOUT (not stderr) output & exitstatus of command. Empty if command failed (or has no output). If you want stderr, use shell redirection (2&>1).

The documentation for this class was generated from the following file:

- src/command.h

3.2 raymii::CommandResult Struct Reference

Public Member Functions

- bool **operator==** (const CommandResult &rhs) const
- bool **operator!=** (const CommandResult &rhs) const

Public Attributes

- std::string **output**
- int **exitstatus**

Friends

- std::ostream & **operator**<< (std::ostream &os, const CommandResult &result)

The documentation for this struct was generated from the following file:

- src/command.h

3.3 Memory Class Reference

`#include <memory.h>`

Public Member Functions

- float TotalMemory ()
- float UsedMemory ()
- float Utilisation ()
- void UpdateUsedMemory ()

3.3.1 Detailed Description

This class represents a computer memory and provides methods to retrieve information about its usage.

3.3.2 Member Function Documentation

3.3.2.1 TotalMemory()

```
float Memory::TotalMemory ( )
```

Getter method for total memory.

Returns

The amount of total memory in Kb.

3.3.2.2 UpdateUsedMemory()

```
void Memory::UpdateUsedMemory ( )
```

Update the amount of used memory.

3.3.2.3 UsedMemory()

```
float Memory::UsedMemory ( )
```

Getter method for used_memory.

Returns

The amount of used memory in Kb.

3.3.2.4 Utilisation()

```
float Memory::Utilisation ( )
```

Calculates and returns the memory utilization as a percentage.

Returns

The memory utilization as a percentage.

The documentation for this class was generated from the following files:

- src/memory.h
- src/memory.cpp

3.4 Power Class Reference

```
#include <power.h>
```

Public Member Functions

- double CurrHourEnergyUsage ()
- double CurrPowerUsage ()
- double TotalEnergyUsage ()
- vector< double > HoursEnergyUsages ()
- double HoursEnergyUsages (int hour)
- void ResetLogVector ()
- void SetLogVector (vector< double > datas)
- void UpdateLogVector (int hour)
- map< string, double > LastNDaysEnergyUsage (vector< string > rows, int n)
- void UpdatePrevHoursEnergy ()
- void SetExtra (double value)
- void UpdatePowerAndEnergyUsage ()

3.4.1 Detailed Description

This class represents a computer power monitor and provides methods to retrieve and update information about power and energy usage.

3.4.2 Member Function Documentation

3.4.2.1 CurrHourEnergyUsage()

```
double Power::CurrHourEnergyUsage ( )
```

Getter method for current hour energy usage.

Returns

The energy usage in Wh in current hour.

3.4.2.2 CurrPowerUsage()

```
double Power::CurrPowerUsage ( )
```

Getter method for current power usage.

Returns

The real-time power usage in watts.

3.4.2.3 HoursEnergyUsages() [1/2]

```
vector< double > Power::HoursEnergyUsages ( )
```

Getter method for hourly energy usages in the current day.

Returns

The vector containing energy usage in Wh in every hour.

3.4.2.4 HoursEnergyUsages() [2/2]

```
double Power::HoursEnergyUsages (
            int hour )
```

Getter method for energy usage in particular hour.

Parameters

hour	The given hour.

Returns

The amount of energy usage in Wh in the given hour.

3.4.2.5 LastNDaysEnergyUsage()

```
map< string, double > Power::LastNDaysEnergyUsage (
            vector< string > rows,
            int n )
```

Get the energy usage drawn in the last n days.

Parameters

rows	The unparsed rows read from logging files.
n	The last n days.

Returns

The date and corresponding energy usage.

3.4.2.6 ResetLogVector()

```
void Power::ResetLogVector ( )
```

Reset vector for logging hourly energy usage.

3.4.2.7 SetExtra()

```
void Power::SetExtra (
            double value )
```

Setter method for extra.

Parameters

value	The value will be assigned to extra.

3.4.2.8 SetLogVector()

```
void Power::SetLogVector (
            vector< double > datas )
```

Setter method for hourly energy usages.

Parameters

datas	The vector that will be assigned to log vector.

3.4.2.9 TotalEnergyUsage()

```
double Power::TotalEnergyUsage ( )
```

Get the total energy usage drawn in the current date.

Returns

The total energy usage in Wh.

3.4.2.10 UpdateLogVector()

```
void Power::UpdateLogVector (
            int hour )
```

Update energy usage in a particular hour.

Parameters

hour	The given hour.

3.4.2.11 UpdatePowerAndEnergyUsage()

```
void Power::UpdatePowerAndEnergyUsage ( )
```

Update power and energy usage by reading from the system.

3.4.2.12 UpdatePrevHoursEnergy()

```
void Power::UpdatePrevHoursEnergy ( )
```

Update the energy used in previous hours.

The documentation for this class was generated from the following files:

- src/power.h
- src/power.cpp

3.5 Process Class Reference

```
#include <process.h>
```

Public Member Functions

- bool operator< (Process const &a) const
- bool operator> (Process const &a) const
- int Pid ()
- std::string User ()
- std::string Command ()
- float CpuUtilisation ()
- std::string Ram ()
- long int UpTime ()
- void SetPid (int pid)
- void SetUser (int pid)
- void SetCommand (int pid)
- void SetCpuUtilization (int pid, long curr_total_jiffies)
- void SetRam (int pid)
- void SetUpTime (int pid)

3.5.1 Detailed Description

This class represents a computer process with attrbutes of ID, user name, command, cpu utilisation, memory usage and up time.

3.5.2 Member Function Documentation

3.5.2.1 Command()

```
string Process::Command ( )
```

Getter method for command.

Returns

The command that generated the process.

3.5.2.2 CpuUtilisation()

```
float Process::CpuUtilisation ( )
```

Getter method for cpu utilisation.

Returns

The cpu utilisation of the process as a percentage.

3.5.2.3 operator<()

```
bool Process::operator< (
            Process const & a ) const
```

Overloads the less operator according to cpu utilisation.

Parameters

a	Another Process object.

Returns

True if this process is less than a; otherwise false.

3.5.2.4 operator>()

```
bool Process::operator> (
            Process const & a ) const
```

Overloads the greater operator according to cpu utilisation.

Parameters

a	Another Process object.

Returns

True if this process is greater than a; otherwise false.

3.5.2.5 Pid()

```
int Process::Pid ( )
```

Getter method for pid.

Returns

The process id.

3.5.2.6 Ram()

```
string Process::Ram ( )
```

Getter method for memory.

Returns

The memory used by the process in Mb.

3.5.2.7 SetCommand()

```
void Process::SetCommand (
            int pid )
```

Setter method for process command.

Parameters

pid	The process id.

3.5.2.8 SetCpuUtilization()

```
void Process::SetCpuUtilization (
            int pid,
            long curr_total_jiffies )
```

Setter method for cpu utilisation.

Parameters

pid	The process id.
curr_total_jiffies	The current total cpu jiffies.

3.5.2.9 SetPid()

```
void Process::SetPid (
            int pid )
```

Setter method for process id.

Parameters

pid	The process id.

3.5.2.10 SetRam()

```
void Process::SetRam (
            int pid )
```

Setter method for memory usage.

Parameters

pid	The process id.

3.5.2.11 SetUpTime()

```
void Process::SetUpTime (
            int pid )
```

Setter method for up time of the process.

Parameters

pid	The process id.

3.5.2.12 SetUser()

```
void Process::SetUser (
            int pid )
```

Setter method for process user.

Parameters

pid	The process id.

3.5.2.13 UpTime()

```
long Process::UpTime ( )
```

Getter method for up time.

Returns

The age of the process in seconds.

3.5.2.14 User()

```
string Process::User ( )
```

Getter method for user.

Returns

The user (name) who runs the process.

The documentation for this class was generated from the following files:

- src/process.h
- src/process.cpp

3.6 Processor Class Reference

`#include <processor.h>`

Public Member Functions

- int PhysicalCores ()
- int LogicalCores ()
- int HyperThreadedCores ()
- int ECores ()
- int PCores ()
- std::vector< float > Utilisations ()
- int Temperature ()
- void UpdateUtilisations ()
- void UpdateTemperature ()

3.6.1 Detailed Description

This class represents a computer processor and provides methods to retrieve information about its cores and utilisation.

3.6.2 Member Function Documentation

3.6.2.1 ECores()

`int Processor::ECores ( )`

Getter method for e cores.

Returns

The number of efficiency cores of the processor.

3.6.2.2 HyperThreadedCores()

`int Processor::HyperThreadedCores ( )`

Getter method for hyperthreaded cores.

Returns

The number of hyperthreaded cores of the processor.

3.6.2.3 LogicalCores()

```
int Processor::LogicalCores ( )
```

Getter method for logical cores.

Returns

The number of logical cores of the processor.

3.6.2.4 PCores()

```
int Processor::PCores ( )
```

Getter method for p cores.

Returns

The number of performance cores of the processor.

3.6.2.5 PhysicalCores()

```
int Processor::PhysicalCores ( )
```

Getter method for physical cores.

Returns

The number of physical cores of the processor.

3.6.2.6 Temperature()

```
int Processor::Temperature ( )
```

Getter method for temperature.

Returns

The cpu temperature in degree Celsius.

3.6.2.7 UpdateTemperature()

```
void Processor::UpdateTemperature ( )
```

Update the cpu temperature.

3.6.2.8 UpdateUtilisations()

```
void Processor::UpdateUtilisations ( )
```

Update the utilisation of each cpu core and overall usage.

3.6.2.9 Utilisations()

```
std::vector< float > Processor::Utilisations ( )
```

Getter method for cpu utilisations.

Returns

The vector containing the utilisation in percentage of each cpu core.

The documentation for this class was generated from the following files:

- src/processor.h
- src/processor.cpp

3.7 System Class Reference

```
#include <system.h>
```

Public Member Functions

- string OperatingSystem ()
- string Kernel ()
- long UpTime ()
- int TotalProcesses ()
- int RunningProcesses ()
- float TotalMemory ()
- float UsedMemory ()
- float MemoryUtilisation ()
- int CpuTemperature ()
- vector< float > CpuUtilisations ()
- void UpdateCpuAndMemory ()
- vector< Process > SortedProcesses ()
- void UpdateProcesses ()
- void BindToPCores ()
- void BindToAllCores ()
- void BindToECores ()
- double PowerUsage ()
- vector< double > HoursEnergyUsages ()
- double TotalEnergyUsage ()
- map< string, double > LastWeekEnergyUsage ()
- void UpdateEnergy ()

Static Public Member Functions

- static System * Instance ()

3.7.1 Detailed Description

This class represents a computer system, integrated with essential components such as processor, memory and power. It also provides methods for retrieving essential information about the system.

3.7.2 Member Function Documentation

3.7.2.1 BindToAllCores()

```
void System::BindToAllCores ( )
```

Bind the most cpu consuming processes to all cores.

3.7.2.2 BindToECores()

```
void System::BindToECores ( )
```

Bind the most cpu consuming processes to efficiency cores.

3.7.2.3 BindToPCores()

```
void System::BindToPCores ( )
```

Bind the most cpu consuming processes to performance cores.

3.7.2.4 CpuTemperature()

```
int System::CpuTemperature ( )
```

Getter method for cpu temperature.

Returns

The cpu temperature in degree Celsius.

3.7.2.5 CpuUtilisations()

```
vector< float > System::CpuUtilisations ( )
```

Getter method for cpu utilisations.

Returns

The vector containing the utilisation of each cpu core.

3.7.2.6 HoursEnergyUsages()

```
vector< double > System::HoursEnergyUsages ( )
```

Getter method for hourly energy usages in a day.

Returns

The vector containing energy usage in Wh in every hour.

3.7.2.7 Instance()

```
System * System::Instance ( )  [static]
```

Static method that thread-safely provides a single instance of the class.

Returns

The single instance of System class.

3.7.2.8 Kernel()

```
string System::Kernel ( )
```

Getter method for kernel.

Returns

The kernel of the operating system.

3.7.2.9 LastWeekEnergyUsage()

```
map< string, double > System::LastWeekEnergyUsage ( )
```

Getter method for energy usage drawn in last week.

Returns

The pairs of date and corresponding energy usage in Wh during last week.

3.7.2.10 MemoryUtilisation()

```
float System::MemoryUtilisation ( )
```

Getter method for memory utilisation.

Returns

The live memory utilisation as a percentage.

3.7.2.11 OperatingSystem()

```
string System::OperatingSystem ( )
```

Getter method for operating system.

Returns

The type of the operating system.

3.7.2.12 PowerUsage()

```
double System::PowerUsage ( )
```

Getter method for live power usage.

Returns

The real-time power usage in watts.

3.7.2.13 RunningProcesses()

```
int System::RunningProcesses ( )
```

Getter method for running processes.

Returns

The number of running processes.

3.7.2.14 SortedProcesses()

```
vector< Process > System::SortedProcesses ( )
```

Getter method for processes in descending order on cpu usage.

Returns

The sorted vector containing all processes.

3.7.2.15 TotalEnergyUsage()

```
double System::TotalEnergyUsage ( )
```

Getter method for today's total energy usage.

Returns

The total energy usage in Wh drawn today.

3.7.2.16 TotalMemory()

```
float System::TotalMemory ( )
```

Getter method for total memory.

Returns

The amount of system total memory in Kb.

3.7.2.17 TotalProcesses()

```
int System::TotalProcesses ( )
```

Getter method for total processes.

Returns

The number of total processes.

3.7.2.18 UpdateCpuAndMemory()

```
void System::UpdateCpuAndMemory ( )
```

Update cpu utilisations & temperature and memory usage.

3.7.2.19 UpdateEnergy()

```
void System::UpdateEnergy ( )
```

Keep updating and logging energy and power usage.

3.7.2.20 UpdateProcesses()

```
void System::UpdateProcesses ( )
```

Update the processes.

3.7.2.21 UpTime()

```
long int System::UpTime ( )
```

Getter method for up time.

Returns

The up time of the system.

3.7.2.22 UsedMemory()

```
float System::UsedMemory ( )
```

Getter method for used memory.

Returns

The amount of current used memory in Kb.

The documentation for this class was generated from the following files:

- src/system.h
- src/system.cpp

3.8 View Class Reference

Public Member Functions

- void **ServiceSelect** ()

The documentation for this class was generated from the following files:

- src/view.h
- src/view.cpp

Chapter 4

File Documentation

4.1 command.h

```
00001 #ifndef COMMAND_H
00002 #define COMMAND_H
00003 // Copyright (C) 2021 Remy van Elst
00004 //
00005 //     This program is free software: you can redistribute it and/or modify
00006 //     it under the terms of the GNU General Public License as published by
00007 //     the Free Software Foundation, either version 3 of the License, or
00008 //     (at your option) any later version.
00009 //
00010 //     This program is distributed in the hope that it will be useful,
00011 //     but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 //     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 //     GNU General Public License for more details.
00014 //
00015 //     You should have received a copy of the GNU General Public License
00016 //     along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 #include <array>
00018 #include <ostream>
00019 #include <string>
00020 #include <cstdio>
00021
00022 namespace raymii {
00023
00024     struct CommandResult {
00025         std::string output;
00026         int exitstatus;
00027         friend std::ostream &operator«(std::ostream &os, const CommandResult &result) {
00028             os « "command exitstatus: " « result.exitstatus « " output: " « result.output;
00029             return os;
00030         }
00031         bool operator==(const CommandResult &rhs) const {
00032             return output == rhs.output &&
00033                    exitstatus == rhs.exitstatus;
00034         }
00035         bool operator!=(const CommandResult &rhs) const {
00036             return !(rhs == *this);
00037         }
00038     };
00039
00040     class Command {
00041     public:
00050         static CommandResult exec(const std::string &command) {
00051             int exitcode = 0;
00052             std::array<char, 8192> buffer{};
00053             std::string result;
00054 #ifdef _WIN32
00055 #define popen _popen
00056 #define pclose _pclose
00057 #define WEXITSTATUS
00058 #endif
00059             FILE *pipe = popen(command.c_str(), "r");
00060             if (pipe == nullptr) {
00061                 throw std::runtime_error("popen() failed!");
00062             }
00063             try {
00064                 std::size_t bytesread;
00065                 while ((bytesread = std::fread(buffer.data(), sizeof(buffer.at(0)), sizeof(buffer),
      pipe)) != 0) {
```

```
00066                     result += std::string(buffer.data(), bytesread);
00067                 }
00068             } catch (...) {
00069                 pclose(pipe);
00070                 throw;
00071             }
00072             // Workaround "error: cannot take the address of an rvalue of type 'int'" on MacOS
00073             // see e.g.
      https://github.com/BestImageViewer/geeqie/commit/75c7df8b96592e10f7936dc1a28983be4089578c
00074             int res = pclose(pipe);
00075             exitcode = WEXITSTATUS(res);
00076             return CommandResult{result, exitcode};
00077         }
00078
00079         // Only for reference in the article. Use regular ::exec.
00080         static CommandResult execFgets(const std::string &command) {
00081             int exitcode = 0;
00082             std::array<char, 8192> buffer{};
00083             std::string result;
00084 #ifdef _WIN32
00085 #define popen _popen
00086 #define pclose _pclose
00087 #define WEXITSTATUS
00088 #endif
00089             FILE *pipe = popen(command.c_str(), "r");
00090             if (pipe == nullptr) {
00091                 throw std::runtime_error("popen() failed!");
00092             }
00093             try {
00094                 while (std::fgets(buffer.data(), buffer.size(), pipe) != nullptr) {
00095                     result += buffer.data();
00096                 }
00097             } catch (...) {
00098                 pclose(pipe);
00099                 throw;
00100             }
00101             // Workaround "error: cannot take the address of an rvalue of type 'int'" on MacOS
00102             // see e.g.
      https://github.com/BestImageViewer/geeqie/commit/75c7df8b96592e10f7936dc1a28983be4089578c
00103             int res = pclose(pipe);
00104             exitcode = WEXITSTATUS(res);
00105             return CommandResult{result, exitcode};
00106         }
00107     };
00108
00109 }// namespace raymii
00110 #endif//COMMAND_H
```

4.2 memory.h

```
00001 #ifndef MEMORY_H
00002 #define MEMORY_H
00003
00008 class Memory {
00009
00010     public:
00011         Memory();
00012
00017         float TotalMemory();
00018
00023         float UsedMemory();
00024
00029         float Utilisation();
00030
00034         void UpdateUsedMemory();
00035
00036     private:
00037         float total_memory;
00038         float used_memory;
00039
00040 };
00041
00042 #endif // MEMORY_H
```

4.3 power.h

```
00001 #ifndef POWER_H
00002 #define POWER_H
00003
```

```
00004 #include <string>
00005 #include <vector>
00006 #include <map>
00007
00008 using std::string;
00009 using std::vector;
00010 using std::map;
00011
00016 class Power {
00017     public:
00018         Power();
00019
00024         double CurrHourEnergyUsage();
00025
00030         double CurrPowerUsage();
00031
00036         double TotalEnergyUsage();
00037
00042         vector<double> HoursEnergyUsages();
00043
00049         double HoursEnergyUsages(int hour);
00050
00054         void ResetLogVector();
00055
00060         void SetLogVector(vector<double> datas);
00061
00066         void UpdateLogVector(int hour);
00067
00074         map<string, double> LastNDaysEnergyUsage(vector<string> rows, int n);
00075
00079         void UpdatePrevHoursEnergy();
00080
00085         void SetExtra(double value);
00086
00090         void UpdatePowerAndEnergyUsage();
00091
00092
00093     private:
00094
00099         long long EnergyUsageInUj();
00100
00101         string energy_usage_path = "/sys/class/powercap/intel-rapl/intel-rapl:1/energy_uj";
00102         string mex_energy_path = "/sys/class/powercap/intel-rapl/intel-rapl:1/max_energy_range_uj";
00103
00104         // The energy amount got from the system will
00105         // reset to zero when it exceeds this bound
00106         long long max_energy;
00107
00108         // Record energy usages in every hour in a day
00109         vector<double> hours_energy_usages;
00110
00111         double curr_hour_energy_usage = 0;
00112         double total_energy_usage;
00113         double curr_power_usage = 0;
00114
00115         // The total energy usage (in uj) in the previous hours
00116         // since the pc boots
00117         long long prev_hours_energy = 0;
00118         // The total energy usage since the pc boots
00119         long long accum_energy_usage = 0;
00120         // The current energy amount extracted from the system
00121         long long energy = 0;
00122         // The energy amount extracted in the last logging time
00123         long long prev_energy = 0;
00124         // The number of times when the capped energy amount
00125         // is reached
00126         long long capped_times = 0;
00127         // The extra energy usage in the current hour
00128         // (in case the pc reboots)
00129         double extra = 0;
00130 };
00131
00132 #endif
```

4.4 process.h

```
00001 #ifndef PROCESS_H
00002 #define PROCESS_H
00003
00004 #include <string>
00005
00010 class Process {
00011     public:
```

```
00017         bool operator<(Process const& a) const;
00018
00024         bool operator>(Process const& a) const;
00025
00030         int Pid();
00031
00036         std::string User();
00037
00042         std::string Command();
00043
00048         float CpuUtilisation();
00049
00054         std::string Ram();
00055
00060         long int UpTime();
00061
00066         void SetPid(int pid);
00067
00072         void SetUser(int pid);
00073
00078         void SetCommand(int pid);
00079
00085         void SetCpuUtilization(int pid, long curr_total_jiffies);
00086
00091         void SetRam(int pid);
00092
00097         void SetUpTime(int pid);
00098
00099     private:
00100         int pid;
00101         std::string user;
00102         std::string command;
00103         float cpu_utilisation;
00104         std::string ram;
00105         long up_time;
00106
00107         // Record previous cpu active/total jiffies to calculate
00108         // the live utilisation of each process in a short period
00109         long prev_active_jiffies = 0;
00110         long prev_total_jiffies = 0;
00111 };
00112
00113 #endif // PROCESS_H
```

4.5 processor.h

```
00001 #ifndef PROCESSOR_H
00002 #define PROCESSOR_H
00003
00004 #include <vector>
00005
00010 class Processor {
00011
00012     public:
00013         Processor();
00014
00019         int PhysicalCores();
00020
00025         int LogicalCores();
00026
00031         int HyperThreadedCores();
00032
00037         int ECores();
00038
00043         int PCores();
00044
00049         std::vector<float> Utilisations();
00050
00055         int Temperature();
00056
00060         void UpdateUtilisations();
00061
00065         void UpdateTemperature();
00066
00067     private:
00068         int physical_cores;
00069         int logical_cores;
00070         int hyperthreaded_cores;
00071         int e_cores;
00072         int p_cores;
00073         int temperature;
00074
00075         // First element represents general/overall cpu utilisation,
```

```
00076         // followed by that of all cores with index cid-1
00077         std::vector<float> utilisations;
00078 
00079         // Record previous cpu active/total jiffies to calculate
00080         // the live utilisation of each core in a short period
00081         std::vector<long> prev_active_jiffies;
00082         std::vector<long> prev_total_jiffies;
00083 };
00084 
00085 #endif
```

4.6 system.h

```
00001 #ifndef SYSTEM_H
00002 #define STSTEM_H
00003 
00004 #include <unistd.h>
00005 
00006 #include <string>
00007 #include <vector>
00008 #include <map>
00009 
00010 #include "processor.h"
00011 #include "memory.h"
00012 #include "process.h"
00013 #include "power.h"
00014 
00015 using std::string;
00016 using std::vector;
00017 using std::map;
00018 
00024 class System {
00025 
00026     public:
00032         static System* Instance();
00033 
00038         string OperatingSystem();
00039 
00044         string Kernel();
00045 
00050         long UpTime();
00051 
00056         int TotalProcesses();
00057 
00062         int RunningProcesses();
00063 
00068         float TotalMemory();
00069 
00074         float UsedMemory();
00075 
00080         float MemoryUtilisation();
00081 
00086         int CpuTemperature();
00087 
00092         vector<float> CpuUtilisations();
00093 
00097         void UpdateCpuAndMemory();
00098 
00103         vector<Process> SortedProcesses();
00104 
00108         void UpdateProcesses();
00109 
00113         void BindToPCores();
00114 
00118         void BindToAllCores();
00119 
00123         void BindToECores();
00124 
00129         double PowerUsage();
00130 
00135         vector<double> HoursEnergyUsages();
00136 
00141         double TotalEnergyUsage();
00142 
00147         map<string, double> LastWeekEnergyUsage();
00148 
00152         void UpdateEnergy();
00153 
00154     private:
00155         System();
00156 
00161         vector<int> CpuConsumingProcesses();
00162 
```

```
00169         void BindProcesses(vector<int> pids, int low, int high);
00170
00178         string FormatDate(int year, int mon, int day);
00179
00184         string LastLoggedDate();
00185
00191         void UpdateDaysLogFile(string last_logged_date, double total_usage);
00192
00198         void UpdateHoursLogFile(string curr_date, vector<double> usages);
00199
00204         std::vector<double> ReadHoursFile();
00205
00210         std::vector<string> ReadDaysFile();
00211
00212         static System* instance;
00213         string hours_log_file = "../data/hours_power_usage.csv";
00214         string days_log_file = "../data/days_power_usage.csv";
00215         Processor cpu;
00216         Memory memory;
00217         map<int, Process> processes;
00218         Power power;
00219         string operating_system;
00220         string kernel;
00221         int hour;
00222         string curr_date;
00223 };
00224
00225 #endif // SYSTEM_H
```

4.7 system_parser.h

```
00001 #ifndef SYSTEM_PARSER_H
00002 #define SYSTEM_PARSER_H
00003
00004 #include <string>
00005 #include <vector>
00006
00007 namespace SystemParser {
00008     // Paths
00009     const std::string kProcDirectory{"/proc/"};
00010     const std::string kCmdlineFilename{"/cmdline"};
00011     const std::string kCpuinfoFilename{"/cpuinfo"};
00012     const std::string kStatusFilename{"/status"};
00013     const std::string kStatFilename{"/stat"};
00014     const std::string kUptimeFilename{"/uptime"};
00015     const std::string kMeminfoFilename{"/meminfo"};
00016     const std::string kVersionFilename{"/version"};
00017     const std::string kOSPath{"/etc/os-release"};
00018     const std::string kPasswordPath{"/etc/passwd"};
00019
00020     // Utils
00021     std::string KeyValParser(std::string, std::string);
00022
00023     // System
00024     std::string OperatingSystem();
00025     std::string Kernel();
00026     std::vector<int> Pids();
00027     std::vector<float> MemoryInfo();
00028     float TotalMemory();
00029     float AvalMemory();
00030     int TotalProcesses();
00031     int RunningProcesses();
00032     long UpTime();
00033
00034     // CPU
00035     std::vector<long> CpuTimes(int cid);
00036     long TotalJiffies(int cid);
00037     long IdleJiffiesC(int cid);
00038     long ActiveJiffiesC(int cid);
00039
00040     // Processes
00041     long ActiveJiffiesP(int pid);
00042     std::string Command(int pid);
00043     std::string Ram(int pid);
00044     std::string Uid(int pid);
00045     std::string User(int pid);
00046     long int UpTime(int pid);
00047 };
00048
00049 #endif // SYSTEM_PARSER_H
```

4.8 view.h

```
00001 #ifndef VIEW_H
00002 #define VIEW_H
00003
00004 #include <string>
00005
00006 #include "system.h"
00007 #include "power.h"
00008
00009 class View {
00010
00011     public:
00012         void ServiceSelect();
00013
00014     private:
00015         System* system_ = System::Instance();
00016         // Power* power_ = Power::Instance();
00017
00018         void DisplaySystemInfo();
00019         void DisplayProcesses();
00020         void DisplayTodaysEnergyUsage();
00021         void DisplayLastWeekEnergyUsage();
00022         void DisplayLivePowerUsage();
00023         void PowerUsageSelect();
00024         void PowerModeSelect();
00025
00026 };
00027
00028 #endif
```

Index