UDACITY

‹ Return to Classroom

Process Monitor

REVIEW

CODE REVIEW 5

HISTORY

▼ CppND-System-Monitor/src/linux_parser.cpp 2

```cpp
#include <dirent.h>
#include <unistd.h>
#include <string>
#include <vector>

#include "linux_parser.h"
#include "my_utility.h"

using std::stof;
using std::string;
using std::to_string;
using std::vector;

// DONE: An example of how to read data from the filesystem
string LinuxParser::OperatingSystem() {
  string line;
  string key;
  string value;
  std::ifstream filestream(kOSPath);
  if (filestream.is_open()) {
    while (std::getline(filestream, line)) {
      std::replace(line.begin(), line.end(), ' ', '_');
      std::replace(line.begin(), line.end(), '=', ' ');
      std::replace(line.begin(), line.end(), '"', ' ');
      std::istringstream linestream(line);
      while (linestream >> key >> value) {
        if (key == "PRETTY_NAME") {
          std::replace(value.begin(), value.end(), '_', ' ');
          return value;
        }
      }
    }
  }
  return value;
}


// DONE: An example of how to read data from the filesystem
```

```
string LinuxParser::Kernel() {
  string os, kernel;
  string line, output;
  vector<string> split_str;
  std::ifstream stream(kProcDirectory + kVersionFilename);
  if (stream.is_open()) {
    std::getline(stream, line);
    //---------------------------------------
    if (1){
      std::istringstream linestream(line);
      linestream >> os >> kernel;
      output = kernel;
    }else{
      split_str = split(line, ' ');
      output = (split_str[8] + split_str[9]);
    }
```

REQUIRED

Your kernel version is not able to show in the output. You need to fix this part.

```
    //---------------------------------------
  }
  return output;
}


// BONUS: Update this to use std::filesystem
vector<int> LinuxParser::Pids() {
  vector<int> pids;
  DIR* directory = opendir(kProcDirectory.c_str());
  struct dirent* file;
  while ((file = readdir(directory)) != nullptr) {
    // Is this a directory?
    if (file->d_type == DT_DIR) {
      // Is every character of the name a digit?
      string filename(file->d_name);
      if (std::all_of(filename.begin(), filename.end(), isdigit)) {
        int pid = stoi(filename);
        pids.push_back(pid);
      }
    }
  }
  closedir(directory);
  return pids;
}


// TODO: Read and return the system memory utilization
float LinuxParser::MemoryUtilization() {
  string line;
  vector<string> split_str;
  float mem_total, mem_available, mem_used;
  float mem_utilization;

  // std::ifstream filestream(kMeminfoFilename); // open file
  std::ifstream filestream("/proc/meminfo"); // open file

  if (filestream.is_open()) {
    // if file open is ok, do while.
    while(std::getline(filestream, line)){
      int pos = line.find(":");

      if (pos != (int)string::npos){
          // delete space from line
          line = del_space(line);

          // split line with ":"
          split_str = split(line, ':');

```

```cpp
        // get key & value
        if (split_str.size() == 2){
          if (split_str[0]=="MemTotal"){
            mem_total = std::stof(trim_rear(split_str[1],2));
          }
          if (split_str[0]=="MemAvailable"){
            mem_available = std::stof(trim_rear(split_str[1],2));
            mem_used      = mem_total - mem_available;
            mem_utilization = (float)(mem_used/mem_available);

            return mem_utilization;
          }
        }
      }
    }
  }
  return 0.0;
}


// TODO: Read and return the system uptime
long LinuxParser::UpTime(){
  string line, uptime, idletime;
  long output;
  vector<string> split_str;

  // std::ifstream filestream(kMeminfoFilename); // open file
  std::ifstream filestream("/proc/uptime"); // open file

  if (filestream.is_open()) {
    // if file open is ok, do while.
    while(std::getline(filestream, line)){
      split_str = split(line, ' ');

      // get key & value
      if (split_str.size() == 2){
        uptime   = split_str[0];
        idletime = split_str[1];
        output   = (long)std::stoi(uptime);
        return output;
      }
    }
  }

  return 0;
}



// TODO: Read and return the number of jiffies for the system
long LinuxParser::Jiffies() {
  return LinuxParser::UpTime() * sysconf(_SC_CLK_TCK);
}

// TODO: Read and return the number of active jiffies for a PID
// REMOVE: [[maybe_unused]] once you define the function
long LinuxParser::ActiveJiffies(int pid) {
    vector<string> split_str;
    string line;
    string file_path = "/proc/" + std::to_string(pid) + "/stat";
    long utime, stime, cutime, cstime, active_jiffies;
    std::ifstream filestream(file_path);

    if (filestream.is_open()){
        while(std::getline(filestream, line)){
            split_str = split(line, ' ');

            if (split_str.size() >= 17 ){
                utime  = std::stol(split_str[13]);
                stime  = std::stol(split_str[14]);
                cutime = std::stol(split_str[15]);
                cstime = std::stol(split_str[16]);
                active_jiffies = utime + stime + cutime + cstime;

                return active_jiffies;
            }
        }
    }

    return 0;
}

// TODO: Read and return the number of active jiffies for the system
long LinuxParser::ActiveJiffies() {
    vector<string> split_str;
    string line;
    long active_jiffies = 0;
    std::ifstream filestream("/proc/stat");

    if(filestream.is_open()){
        while(std::getline(filestream, line)){
            split_str = split(line, ' ');
            if (split_str.size() == 11){

                if (split_str[0] == "cpu"){
                    for(int i=1; i<=10; i++ ){
                        // sum of all values without idle & iowait
                        if ( (i!=4) && (i!=5)){
                            active_jiffies += std::stol(split_str[i]);
                        }
                    }
                    return active_jiffies;
                }
            }
        }
    }
    return 0;
}

// TODO: Read and return the number of idle jiffies for the system
long LinuxParser::IdleJiffies() {
    vector<string> split_str;
    string line;
    long active_jiffies = 0;
    std::ifstream filestream("/proc/stat");

    if(filestream.is_open()){
        while(std::getline(filestream, line)){
            split_str = split(line, ' ');
            if (split_str.size() == 11){

                if (split_str[0] == "cpu"){
                    for(int i=1; i<=10; i++ ){
                        // sum of 2 values which are idle & iowait
                        if ( (i==4) || (i==5)){
                            active_jiffies += std::stol(split_str[i]);
                        }
                    }
                    return active_jiffies;
                }
            }
        }
    }
    return 0;
}




// TODO: Read and return CPU utilization
float LinuxParser::CpuUtilization() {
  string line;
  vector<string> split_str;
  float user, nice, system, idle, iowait, irq, softirq, steal;
```

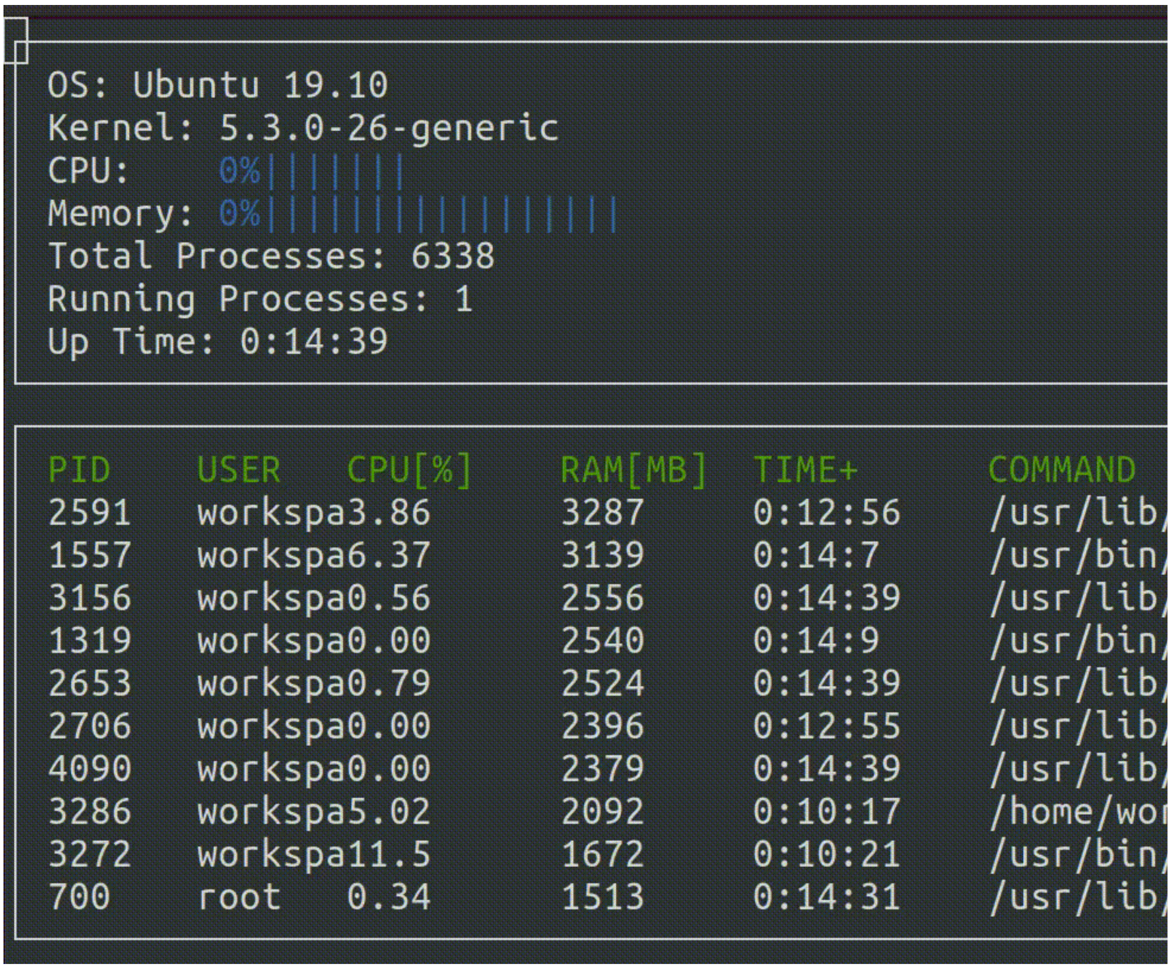
```
//float guest, guest_nice;
  float sum_idle, sum_non_idle, total;
  float cpu_util;

  std::ifstream filestream("/proc/stat");

  if (filestream.is_open()){
    while(std::getline(filestream, line)){
      split_str = split(line, ' ');
      if (split_str[0] == "cpu"){
        user   = std::stof(split_str[1]);
        nice   = std::stof(split_str[2]);
        system = std::stof(split_str[3]);
        idle   = std::stof(split_str[4]);
        iowait = std::stof(split_str[5]);
        irq    = std::stof(split_str[6]);
        softirq= std::stof(split_str[7]);
        steal  = std::stof(split_str[8]);
      //guest  = std::stof(split_str[9]);
      //guest_nice= std::stof(split_str[10]);
        //
        sum_idle     = idle + iowait;
        sum_non_idle = user + nice + system + irq + softirq + steal;
        total        = sum_idle + sum_non_idle;
        cpu_util     = (total - sum_idle) / total;

        return cpu_util;
      }
    }
  }
  return 0.0;
}

// TODO: Read and return the total number of processes
int LinuxParser::TotalProcesses() {
    vector<string> split_str;
    string line;
    std::ifstream filestream("/proc/stat");

    if(filestream.is_open()){
        while(std::getline(filestream, line)){
            int pos = line.find("processes ");
            if (pos != (int)string::npos){
                split_str = split(line, ' ');
                return std::stoi(split_str[1]);
            }
        }
    }
    return 0;
}

// TODO: Read and return the number of running processes
int LinuxParser::RunningProcesses() {
    vector<string> split_str;
    string line;
    std::ifstream filestream("/proc/stat");

    if(filestream.is_open()){
        while(std::getline(filestream, line)){
            int pos = line.find("procs_running ");
            if (pos != (int)string::npos){
                split_str = split(line, ' ');
                return std::stoi(split_str[1]);
            }
        }
    }
    return 0;
}


// TODO: Read and return the command associated with a process
// REMOVE: [[maybe_unused]] once you define the function
```

```cpp
string LinuxParser::Command(int pid) {
    string line;
    string file_path = "/proc/" + std::to_string(pid) + "/cmdline";
    std::ifstream filestream(file_path);
    if (std::getline(filestream, line)){
        return line;
    }
    return " ";
}

// TODO: Read and return the memory used by a process
// REMOVE: [[maybe_unused]] once you define the function
float LinuxParser::Ram(int pid) {
    string line;
    vector<string> split_str;
    float sum_rss = 0.0;
    string file_path = "/proc/" + std::to_string(pid) + "/smaps";
    std::ifstream filestream(file_path);

    if (filestream.is_open()) {
        while(std::getline(filestream, line)){
            line = del_space(line);
            split_str = split(line, ':');
            if (split_str[0] == "Rss"){
                sum_rss += std::stof( trim_rear(split_str[1], 2) );
            }
        }
        return sum_rss;
    }
    return 0.0;
}

// TODO: Read and return the user ID associated with a process
// REMOVE: [[maybe_unused]] once you define the function
int LinuxParser::Uid(int pid) {
    vector<string> split_str;
    string line;
    string file_path = "/proc/" + std::to_string(pid) + "/status";
    std::ifstream filestream(file_path);

    if(filestream.is_open()){
        while(std::getline(filestream, line)){
         // split_str = split(line, ' ');
            char str_tab = '¥t';
            split_str = split(line, str_tab);

            if(split_str.size() >= 2){
                if (split_str[0] == "Uid:"){
                    return std::stoi(split_str[1]);
                }
            }
        }
    }

    return 0;
}

// TODO: Read and return the user associated with a process
// REMOVE: [[maybe_unused]] once you define the function
string LinuxParser::User(int pid) {
    vector<string> split_str;
    string line;
    string uid = std::to_string(LinuxParser::Uid(pid));
    string file_path = "/etc/passwd";
    std::ifstream filestream(file_path);

    if(filestream.is_open()){
        while(std::getline(filestream, line)){
            // cout << "line = " << line << "¥n";
            split_str = split(line, ':');

            if (split_str.size() >= 3){
                if (split_str[2] == uid){
                    return split_str[0];
                }
            }
        }
    }

    return "_";
}

// TODO: Read and return the uptime of a process
// REMOVE: [[maybe_unused]] once you define the function
long LinuxParser::UpTime(int pid) {
```

SUGGESTION

Uptime was showing static output for processes. It should be updated timely as known from its function name.

Expected Output:



```
OS: Ubuntu 19.10
Kernel: 5.3.0-26-generic
CPU:     0%||||||
Memory: 0%|||||||||||||||||
Total Processes: 6338
Running Processes: 1
Up Time: 0:14:39

PID    USER   CPU[%]    RAM[MB]  TIME+      COMMAND
2591   workspa3.86      3287     0:12:56    /usr/lib
1557   workspa6.37      3139     0:14:7     /usr/bin
3156   workspa0.56      2556     0:14:39    /usr/lib
1319   workspa0.00      2540     0:14:9     /usr/bin
2653   workspa0.79      2524     0:14:39    /usr/lib
2706   workspa0.00      2396     0:12:55    /usr/lib
4090   workspa0.00      2379     0:14:39    /usr/lib
3286   workspa5.02      2092     0:10:17    /home/wo
3272   workspa11.5      1672     0:10:21    /usr/bin
700    root   0.34      1513     0:14:31    /usr/lib
```

```
    vector<string> split_str;
    string line;
    string file_path = "/proc/" + std::to_string(pid) + "/stat";
    std::ifstream filestream(file_path);
    if (filestream.is_open()){
        while(std::getline(filestream, line)){
            split_str = split(line, ' ');
            if (split_str.size() >= 22){
                return std::stol(split_str[21]);
            }
        }
    }
    return 0;
}
```

▸ CppND-System-Monitor/src/system.cpp 1

▸ CppND-System-Monitor/src/process.cpp 1

▸ CppND-System-Monitor/src/format.cpp 1

▸ CppND-System-Monitor/src/test07.cpp

▸ CppND-System-Monitor/src/test06.cpp

▸ CppND-System-Monitor/src/test05.cpp

▸ CppND-System-Monitor/src/test04.cpp

▸ CppND-System-Monitor/src/test03.cpp

▸ CppND-System-Monitor/src/test02.cpp

▸ CppND-System-Monitor/src/test01.cpp

▸ CppND-System-Monitor/src/test00.cpp

▸ CppND-System-Monitor/src/processor.cpp

▸ CppND-System-Monitor/src/ncurses_display.cpp

▸ CppND-System-Monitor/src/my_utility.h

▸ CppND-System-Monitor/src/my_utility.cpp

▸ CppND-System-Monitor/src/main.cpp

▸ CppND-System-Monitor/include/system.h

▸ CppND-System-Monitor/include/processor.h

▸ CppND-System-Monitor/include/process.h

▸ CppND-System-Monitor/include/ncurses_display.h

▸ CppND-System-Monitor/include/linux_parser.h

▸ CppND-System-Monitor/include/format.h

▸ CppND-System-Monitor/README.md

▸ CppND-System-Monitor/Makefile

▸ **CppND-System-Monitor/CMakeLists.txt**

Learn the best practices for revising and resubmitting your project.