UDACITY

‹ Return to "C++" in the classroom

Process Monitor

REVIEW

CODE REVIEW 16

HISTORY

▾ src/linux_parser.cpp 9

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

#include "linux_parser.h"

using std::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") {
```

SUGGESTION

I think you should know this:

```
PRETTY_NAME=
    A pretty operating system name in a format suitable for presentat
    or may not contain a release code name or OS version of some kind
    set, defaults to "PRETTY_NAME="Linux"". Example: "PRETTY_NAME="Fe
    Miracle)"".
```

Source : man pages

```
          std::replace(value.begin(), value.end(), '_', ' ');
```

AWESOME

Nice use of regex to extract values from the lines.

❤️

```
          return value;
        }
      }
    }
  }
  return value;
}

// DONE: An example of how to read data from the filesystem
string LinuxParser::Kernel() {
  string os, kernel;
  string line;
  string version;
  std::ifstream stream(kProcDirectory + kVersionFilename);
  if (stream.is_open()) {
```

SUGGESTION

Most of the students do not close the stream once defined in the function,It does not create any problem using the same stream object to open some other file, If you wish to do the same then you can do two thir

1. Either open the file with some other ifstream object
2. Close the ifstream object first and then use the same stream to open the next file again by stateme

It is a good habit to close the stream once you are done with the file opening.

```
    std::getline(stream, line);
    std::istringstream linestream(line);
    linestream >> os >> version >> kernel;
  }
  return kernel;
}

```

```
// 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;
}

// Read and return the system memory utilization
float LinuxParser::MemoryUtilization() {
   string line, key, value;
   float mem_total, mem_free;
```

SUGGESTION

I use camelCase style of naming variables.

snake_case: In which you separate the words by an underscore
camelCase: the first letter starts with the lower alphabet and then the second word starts with a capital let
ProperCase/PascalCase: capitalising first letter of every word

You must go through this link

```
   std::ifstream filestream(kProcDirectory + kMeminfoFilename);
   if (filestream.is_open()) {
      while (std::getline(filestream, line)) {
    std::istringstream linestream(line);
    while (linestream >> key >> value) {
       if (key == "MemTotal:") {
          mem_total = std::stof(value);
       }
           if (key == "MemFree:") {
          mem_free = std::stof(value);
       }
        }
      }
      return (mem_total - mem_free) / mem_total;
   }
   return -1.0;
}

// Read and return the system uptime
long LinuxParser::UpTime() {
  string line, uptime;
  std::ifstream stream(kProcDirectory + kUptimeFilename);
  if (stream.is_open()) {
    std::getline(stream, line);
    std::istringstream linestream(line);
    linestream >> uptime;
    return std::stol(uptime);
  }
  return -1;
}

// Read and return the jiffies (i.e., clock ticks) for the system. Guest not i
vector<long> LinuxParser::Jiffies() {
   string line, key;
   long user, nice, system, idle, iowait, irq, softirq, steal;
   vector<long> jiffies;
   std::ifstream filestream(kProcDirectory + kStatFilename);
   if (filestream.is_open()) {
      while (std::getline(filestream, line)) {
         std::istringstream linestream(line);
         if (linestream >> key >> user >> nice >> system >> idle >> iowait >>
            if (key == "cpu") {
           jiffies = {user, nice, system, idle, iowait, irq, softirq, steal};
           return jiffies;
        }
         }
      }
   }
   return {};
```

SUGGESTION

Returning NULL is usually the best idea if you intend to indicate that no data is available.

An empty object implies data has been returned, whereas returning null clearly indicates that nothing has

There is a whole lot of discussion on StackOverflow about the same!

PS - You might get some warnings after you use NULL because by default a large domain of warnings turn

```
}

// Read and return the jiffies of a process relative to CPU time
vector<long> LinuxParser::Jiffies(int pid) {
   string line, value;
   int counter = 1;
   // CPU time fields
   vector<int> fields = {14, 15, 16, 17, 22};
   vector<long> jiffies;
   std::ifstream filestream(kProcDirectory + std::to_string(pid) + "/" + kStat
   if (filestream.is_open()) {
      std::getline(filestream, line);
      std::istringstream linestream(line);
      while (linestream >> value) {
         if (std::find(fields.begin(), fields.end(), counter) != fields.end())
            jiffies.push_back(std::stol(value));
     }
         counter++;
      }
      return jiffies;
   }
   return {};
}

// Read and return CPU utilization since boot. Result is in range [0.0, 1.0]
float LinuxParser::CpuUtilization() {
   long total_cpu_time, total_cpu_idle_time, total_cpu_usage_time;
   vector<long> js = Jiffies();
   if (!js.empty()) {
      // 0:user, 1:nice, 2:system, 3:idle, 4:iowait, 5:irq, 6:softirq, 7:steal
      total_cpu_time = js[0] + js[1] + js[2] + js[3] + js[4] + js[5] + js[6] +
      total_cpu_idle_time = js[3] + js[4];
      total_cpu_usage_time = total_cpu_time - total_cpu_idle_time;
      return static_cast<float>(total_cpu_usage_time) / total_cpu_time;
   }
   return -1.0;
}

// Read and return CPU utilization of a process
float LinuxParser::CpuUtilization(int pid) {
   long total_cpu_time, lseconds;
   vector<long> js = Jiffies(pid);
   // Check if process still exists
   if (!js.empty()) {
      // 0:utime, 1:stime, 2:cutime, 3:cstime, 4:starttime
      total_cpu_time = js[0] + js[1] + js[2] + js[3];
      lseconds = LinuxParser::UpTime() - (js[4] / sysconf(_SC_CLK_TCK));
      return static_cast<float>(total_cpu_time) / sysconf(_SC_CLK_TCK) / lseco
   }
   return -1.0;
}

// Read and return the total number of processes
int LinuxParser::TotalProcesses() {
   string line, key, value;
   std::ifstream filestream(kProcDirectory + kStatFilename);
   if (filestream.is_open()) {
      while (std::getline(filestream, line)) {
         std::istringstream linestream(line);
         if (linestream >> key >> value) {
        if (key == "processes") {
               return std::stoi(value);
        }
         }
```

SUGGESTION

I guess Indentation could have been better!

Please look at some Github accounts of some of the famous open-source projects. The code should be in

```
     }
   }
   return -1;
}

// Read and return the number of running processes
int LinuxParser::RunningProcesses() {
   string line, key, value;
   std::ifstream filestream(kProcDirectory + kStatFilename);
   if (filestream.is_open()) {
      while (std::getline(filestream, line)) {
         std::istringstream linestream(line);
         if (linestream >> key >> value) {
            if (key == "procs_running") {
               return std::stoi(value);
        }
         }
     }
   }
   return -1;
}

// Read and return the command associated with a process
string LinuxParser::Command(int pid) {
   string line;
   std::ifstream filestream(kProcDirectory + std::to_string(pid) + "/" + kCmdl
   if (filestream.is_open()) {
      if (std::getline(filestream, line)) {
         return line;
      }
   }
   return string();
}

// Read and return the memory used by a process
string LinuxParser::Ram(int pid) {
  string line, key, value;
  std::ifstream filestream(kProcDirectory + std::to_string(pid) + "/" + kStatu
  if (filestream.is_open()) {
    while (std::getline(filestream, line)) {
      std::istringstream linestream(line);
      while (linestream >> key >> value) {
    if (key == "VmSize:") {
```

SUGGESTION

I understand that you are following the Udacity guidelines and that is why you have extracted value corres

But i should tell you that this will give you memory usage more than your Physical RAM size!

Because VmSize is the sum of all the virtual memory as you can see on the manpages also.
Search for `VmSize` and you will get the following line

```
 * VmSize: Virtual memory size.
```

Whereas when you use VmData then it gives the exact physical memory being used as a part of Physical R
GitHub might not have any idea of Virtual memory and so they will think you have done something wrong

PS - Moreover when you replace then please put a comment stating that you have used `VmData` inste
he/she might make it a required change but once you put the comment with the link to the resources

```
          return value;
```

REQUIRED

Since you will be showing the values in the terminal in MB so you must divide it by 1024. That is why you a

Since variable `value` is of type string so to simplify things you can directly delete the last three character

```
        }
      }
    }
  }
  return string();
}

// Read and return the user ID associated with a process
string LinuxParser::Uid(int pid) {
  string line, key, value;
  std::ifstream filestream(kProcDirectory + std::to_string(pid) + "/" + kStatu
  if (filestream.is_open()) {
    while (std::getline(filestream, line)) {
      std::istringstream linestream(line);
      while (linestream >> key >> value) {
    if (key == "Uid:") {
          return value;
        }
      }
    }
  }
  return string();
```

AWESOME

Normally students forget to return after returning in the middle of the function implementation.
Nice!

```
}

// Read and return the user associated with a process
string LinuxParser::User(int pid) {
  string line, user, placeholder, uid;
  string searched_uid = LinuxParser::Uid(pid);
  std::ifstream filestream(kPasswordPath);
  if (filestream.is_open()) {
    while (std::getline(filestream, line)) {
      std::replace(line.begin(), line.end(), ':', ' ');
      std::istringstream linestream(line);
      while (linestream >> user >> placeholder >> uid) {
    if (uid == searched_uid) {
          return user;
        }
      }
    }
  }
  return string();
}

// Read and return the uptime of a process
long LinuxParser::UpTime(int pid) {
   string line, uptime;
   int uptime_field = 22;
   int counter = 1;
   std::ifstream filestream(kProcDirectory + std::to_string(pid) + "/" + kStat
   if (filestream.is_open()) {
      std::getline(filestream, line);
      std::istringstream linestream(line);
      while (linestream >> uptime) {
    if (counter == uptime_field) {
       //divide by sysconf(_SC_CLK_TCK) to convert clock ticks into seconds
       return std::stol(uptime) / sysconf(_SC_CLK_TCK);
    }
    counter++;
      }
   }
   return -1;
}

```

- src/system.cpp 2
- src/process.cpp 2
- src/processor.cpp 1
- src/format.cpp 1
- include/ncurses_display.h 1
- src/ncurses_display.cpp
- src/main.cpp
- include/system.h
- include/processor.h
- include/process.h
- include/linux_parser.h
- include/format.h
- README.md
- Makefile

▸ CMakeLists.txt

RETURN TO PATH