PS7: Kronos Log Parsing

Assignment Overview

This project involved parsing and analyzing logs from a Kronos InTouch device to detect and report all device boot events. Each boot sequence begins with the message (log.c.166) server started and is considered complete if it is followed by oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080. The goal was to scan the entire log file, identify valid boot start and end pairs, calculate the elapsed boot time using Boost date/time libraries, and report both successful and incomplete boot events in a structured output. The output file (.rpt) mirrors the filename of the input log with added diagnostics for each boot sequence.

Key Algorithms / Data Structures / OO Designs

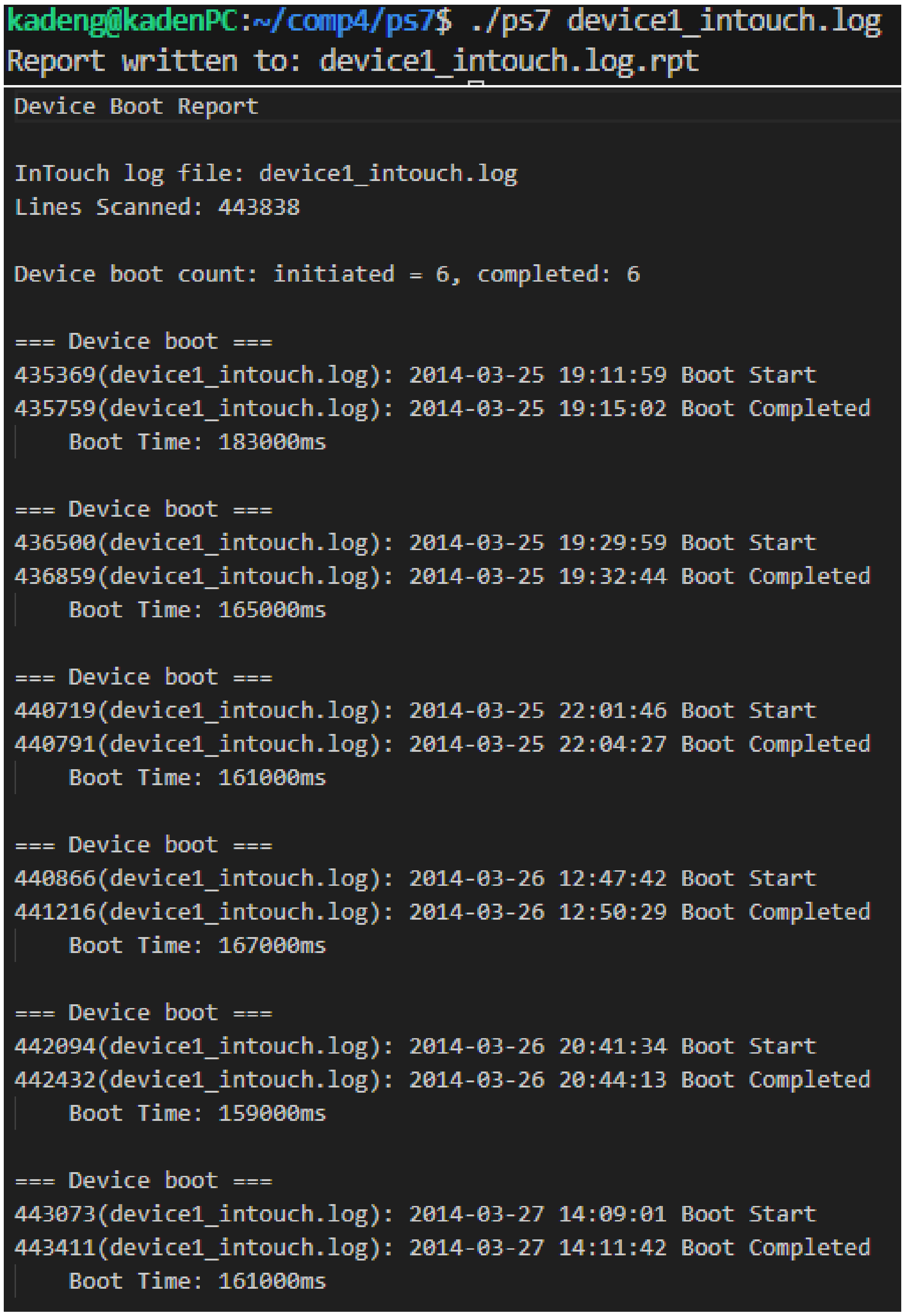
The core logic was based on regular expression pattern matching using the <regex> library. The following design choices were made: two distinct regular expressions were used|one to match server startup lines, and one to match boot completion lines. Timestamps were parsed and stored using Boost's posix_time and ptime objects for precise duration calculations. A stack-like structure was used to track unpaired boot starts, with each valid boot start followed by a search for the next matching boot completion. The elapsed time between matched entries was reported in milliseconds. Incomplete boots (boot starts with no matching end) were logged with appropriate flags and line numbers. Output was structured and timestamped according to a required .rpt format. The program was structured for modularity, with clear separation between log parsing, time computation, and reporting. It accepted the input log filename as a command-line argument and produced a matching output report file with a .rpt extension.

What I Learned

Through PS7, I learned how to use C++ regular expressions to extract structured data from real-world logs, and how to parse and manipulate timestamps with the Boost Date-Time library. I also learned to design logic for paired-event detection (e.g., matching boot start and end) while handling malformed or incomplete cases. Additionally, I handled file input/output and formatting in a way that adheres to strict autograder requirements (lots of edge cases), detected, logged, and reported incomplete processes without program crashes or missed data, and wrote robust logic that correctly tracks multiple device states across very large and noisy datasets.

What Doesn't Work

While this program is fully functional there are some discrepancies in the formatting that dont directly reflect the assignment details. This unfortunately lead to a certain point reduction but the program, like said before, remains functional and representative of the over arching assignment details.



```
kadeng@kadenPC:~/comp4/ps7$ ./ps7 device1_intouch.log
Report written to: device1_intouch.log.rpt
```

```
Device Boot Report

InTouch log file: device1_intouch.log
Lines Scanned: 443838

Device boot count: initiated = 6, completed: 6

=== Device boot ===
435369(device1_intouch.log): 2014-03-25 19:11:59 Boot Start
435759(device1_intouch.log): 2014-03-25 19:15:02 Boot Completed
    Boot Time: 183000ms

=== Device boot ===
436500(device1_intouch.log): 2014-03-25 19:29:59 Boot Start
436859(device1_intouch.log): 2014-03-25 19:32:44 Boot Completed
    Boot Time: 165000ms

=== Device boot ===
440719(device1_intouch.log): 2014-03-25 22:01:46 Boot Start
440791(device1_intouch.log): 2014-03-25 22:04:27 Boot Completed
    Boot Time: 161000ms

=== Device boot ===
440866(device1_intouch.log): 2014-03-26 12:47:42 Boot Start
441216(device1_intouch.log): 2014-03-26 12:50:29 Boot Completed
    Boot Time: 167000ms

=== Device boot ===
442094(device1_intouch.log): 2014-03-26 20:41:34 Boot Start
442432(device1_intouch.log): 2014-03-26 20:44:13 Boot Completed
    Boot Time: 159000ms

=== Device boot ===
443073(device1_intouch.log): 2014-03-27 14:09:01 Boot Start
443411(device1_intouch.log): 2014-03-27 14:11:42 Boot Completed
    Boot Time: 161000ms
```

Figure 8: Example Output

Source Code

```
Makefile:
CXX = g++
CXXFLAGS = -Wall -Werror -pedantic -g --std=c++20

TARGET = ps7
```

```
SRC = ps7.cpp

.PHONY: all ps7 lint clean

all: $(TARGET)

$(TARGET): $(SRC)
	$(CXX) $(CXXFLAGS) -o $(TARGET) $(SRC) -lboost_date_time

lint:
	cpplint $(SRC)

clean:
	rm -f $(TARGET) *.o *.rpt

ps7.cpp:
// COPYRIGHT 2025 Kaden Gardiner
#include <iostream>
#include <fstream>
#include <string>
#include <regex>
#include <vector>
#include <sstream>
#include <iomanip>
#include <boost/date_time/posix_time/posix_time.hpp>

using boost::posix_time::ptime;
using boost::posix_time::time_duration;
using boost::posix_time::time_from_string;
using boost::gregorian::to_iso_extended_string;

struct BootReport {
    int startLine;
    std::string startDate;
    std::string startClock;
    int endLine;
    std::string endDate;
    std::string endClock;
    int duration;
    bool success;
};

ptime extractTimestamp(const std::string& line) {
    std::smatch match;
    std::regex tsRegex(R"((\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}(?:\.\d
       {1,3})?))");
    if (std::regex_search(line, match, tsRegex)) {
        return time_from_string(match[1]);
    } else {
        return ptime();
    }
}
```

```cpp

std::string formatClock(const ptime& time) {
    auto tod = time.time_of_day();
    std::ostringstream oss;
    oss << std::setw(2) << std::setfill('0') << tod.hours() << ":"
        << std::setw(2) << std::setfill('0') << tod.minutes() << ":"
        << std::setw(2) << std::setfill('0') << tod.seconds();
    return oss.str();
}

int main(int argc, char* argv[]) {
    if (argc < 2) {
        std::cerr << "Usage: " << argv[0] << " <log_filename>" << std::endl;
        return 1;
    }

    std::ifstream infile(argv[1]);
    if (!infile) {
        std::cerr << "Error: Could not open file " << argv[1] << std::endl;
        return 1;
    }

    std::string line;
    int line_number = 1;

    std::regex bootStartPattern(R"(\(log\.c\.166\) server started)");
    std::regex bootEndPattern("oejs\\.AbstractConnector:Started SelectChannelConnector");

    std::vector<BootReport> reportLines;
    ptime startTime;
    int startLine = -1;
    int bootStartCount = 0;
    int bootEndCount = 0;

    while (std::getline(infile, line)) {
        if (std::regex_search(line, bootStartPattern)) {
            ++bootStartCount;
            if (!startTime.is_not_a_date_time()) {
                BootReport failed;
                failed.startLine = startLine;
                failed.startDate = to_iso_extended_string(startTime.date());
                failed.startClock = formatClock(startTime);
                failed.endLine = -1;
                failed.endDate = "**** Incomplete boot ****";
                failed.endClock = "";
                failed.duration = -1;
                failed.success = false;
                reportLines.push_back(failed);
            }
            startTime = extractTimestamp(line);
            startLine = line_number;
        } else if (std::regex_search(line, bootEndPattern)) {
            ptime endTime = extractTimestamp(line);

            if (!startTime.is_not_a_date_time()) {
                time_duration dur = endTime - startTime;

                BootReport report;
                report.startLine = startLine;
                report.startDate = to_iso_extended_string(startTime.date()
                   );
                report.startClock = formatClock(startTime);
                report.endLine = line_number;
                report.endDate = to_iso_extended_string(endTime.date());
                report.endClock = formatClock(endTime);
                report.duration = static_cast<int>(dur.total_seconds());
                report.success = true;
                reportLines.push_back(report);
                ++bootEndCount;
                startTime = ptime();
                startLine = -1;
            }
        }
        ++line_number;
    }

    if (!startTime.is_not_a_date_time()) {
        BootReport failed;
        failed.startLine = startLine;
        failed.startDate = to_iso_extended_string(startTime.date());
        failed.startClock = formatClock(startTime);
        failed.endLine = -1;
        failed.endDate = "**** Incomplete boot ****";
        failed.endClock = "";
        failed.duration = -1;
        failed.success = false;
        reportLines.push_back(failed);
    }
    infile.close();

    std::string reportFile = argv[1];
    reportFile += ".rpt";
    std::string filename = reportFile.substr(reportFile.find_last_of
       ("/\\") + 1);
    filename = filename.substr(0, filename.find(".rpt"));

    std::ofstream outfile(reportFile);
    if (!outfile) {
        std::cerr << "Error: Could not create report file " << reportFile
           << std::endl;
        return 1;
    }

    outfile << "Device Boot Report\n\n";
    outfile << "InTouch log file: " << filename << "\n";
    outfile << "Lines Scanned: " << (line_number - 1) << "\n\n";
    outfile << "Device boot count: initiated = " << bootStartCount
    << ", completed: " << bootEndCount << "\n\n";

    for (const BootReport& report : reportLines) {
        outfile << "=== Device boot ===\n";
        outfile << report.startLine << "(" << filename << "): "
        << report.startDate << " " << report.startClock << " Boot Start\n
           ";
        if (report.success) {
            outfile << report.endLine << "(" << filename << "): "
            << report.endDate << " " << report.endClock << " Boot
               Completed\n";
            outfile << "\tBoot Time: " << (report.duration * 1000) << "ms
               \n\n";
        } else {
            outfile << report.endDate << " \n\n";
        }
    }

    outfile.close();
    std::cout << "Report written to: " << reportFile << std::endl;

    return 0;
}
```