```
    1: // Copyright 2017 Sam Pickell
    2: #include <boost/regex.hpp>
    3: #include <cstdlib>
    4: #include <iostream>
    5: #include <string>
    6: #include <fstream>
    7: #include <vector>
    8: #include <sstream>
    9: #include "boost/date_time/gregorian/gregorian.hpp"
   10: #include "boost/date_time/posix_time/posix_time.hpp"
   11:
   12: void populate_vectors(std::vector<std::string>& a, std::vector<std::string>&
 b,
   13:                       std::vector<std::string>& c, std::vector<std::string>&
 d);
   14: void print_stats(std::vector<std::string>& a, std::vector<std::string>& b,
   15:                      std::vector<std::string>& c, std::vector<std::string>&
d,
   16:                      std::ofstream& fout, std::ifstream& fin, std::string na
me);
   17:
   18: int main(int argc, char* argv[]) {
   19:   int line_number = 1;
   20:
   21:   // regex
   22:   boost::regex start_up("[0-9]{4}-[0-9]{2}-[0-9]{2}\\s[0-9]{2}:[0-9]{2}:[0-9
]"
   23: "{2}:\\s[(]log[.]c[.]166[)]\\sserver\\sstarted\\s");
   24:   boost::regex success("[0-9]{4}-[0-9]{2}-[0-9]{2}\\s[0-9]{2}:[0-9]{2}:[0-9]
{2}"
   25: "[.][0-9]{3}:INFO:oejs[.]AbstractConnector:Started\\sSelectChannelConnector@
[0-"
   26: "9]{1}[.][0-9]{1}[.][0-9]{1}[.][0-9]{1}:[0-9]{4}");
   27:   boost::regex services("Starting\\sService[.].*");
   28:   boost::regex service_success("Service\\sstarted\\ssuccessfully[.].*");
   29:   boost::regex soft_start("[A-Z]{1}[a-z]{2}\\s[0-9]{2}\\s[0-9]{2}:[0-9]{2}:[
0-9"
   30: "]{2}\\s.*Install\\sstarted\\s");
   31:   boost::regex soft_orig("[A-Z]{1}[a-z]{2}\\s[0-9]{2}\\s[0-9]{2}:[0-9]{2}:[0
-9"
   32: "]{2}\\s[(]none[)]\\s/stagingarea/scripts/install-rollback[.]sh:\\sintouch-a
ppl"
   33: "ication-base-.*[.]armv6jel_vfp[.]rpm\\swas\\spreviously\\sinstalled[,]\\sad
din"
   34: "g\\srpm\\sto\\srollback\\slist");
   35:   boost::regex soft_end("[A-Z]{1}[a-z]{2}\\s[0-9]{2}\\s[0-9]{2}:[0-9]{2}:[0-
9"
   36: "]{2}\\s[(]none[)]\\s/stagingarea/scripts/install-rollback[.]sh:\\sExitValue
\\s"
   37: "from\\sinstall\\scommand\\s:\\s0");
   38:   boost::regex soft_new_ver("[A-Z]{1}[a-z]{2}\\s[0-9]{2}\\s[0-9]{2}:[0-9]{2}
:[0"
   39: "-9]{2}\\s[(]none[)]\\s/stagingarea/scripts/install-rollback[.]sh:\\sProcess
ing"
   40: "\\s43\\sof\\s45\\sintouch-platform-base-.*[.]armv6jel_vfp[.]rpm ...");
   41:
   42:   // vectors
   43:   std::vector<std::string> service_name;
   44:   std::vector<std::string> service_start;
   45:   std::vector<std::string> service_end;
   46:   std::vector<std::string> service_elapsed_time;
```

```
   47:
   48:   // time
   49:   boost::posix_time::time_duration diff;
   50:   std::string time_start;
   51:   std::string time_end;
   52:   std::string s_total_time;
   53:
   54:   std::ifstream fin;
   55:   std::ofstream fout;
   56:
   57:   std::string str_start = " ";
   58:   std::string str_end = " ";
   59:   std::string str_orig_ver = " ";
   60:   std::string str_new_ver = " ";
   61:   std::string str_elapsed = " ";
   62:   std::string s, t;
   63:   std::string service_1;
   64:   std::string input_file = argv[1];
   65:   std::string output_file = input_file;
   66:   output_file.append(".rpt");
   67:
   68:   int start_line;
   69:
   70:   // Open files
   71:   fin.open(input_file.c_str());
   72:   if (fin.fail()) {
   73:       std::cout << "Failed to open file" << std::endl;
   74:       exit(1);
   75:     }
   76:   fout.open(output_file.c_str());
   77:   if (fout.fail()) {
   78:       std::cout << "Failed to open output file" << std::endl;
   79:       exit(1);
   80:     }
   81:   // End opening
   82:
   83:   std::getline(fin, s);
   84:   while (!fin.eof()) {
   85:      // start of service startup
   86:
   87:     if (boost::regex_match(t, start_up) && boost::regex_match(s, services))
{
   88:         std::stringstream out;
   89:
   90:         for (unsigned int i = 0; i < service_name.size(); i++) {
   91:             std::string finder = service_name.at(i);
   92:             std::size_t pos = s.find(finder);
   93:             if (pos != std::string::npos) {
   94:                 service_1 = finder;
   95:                 out << line_number;
   96:                 service_start.at(i) = out.str();
   97:                 break;
   98:               }
   99:           }
  100:       }
  101:
  102:     if (boost::regex_match(t, start_up) &&
  103:         boost::regex_match(s, service_success)) {
  104:         std::stringstream out;
  105:         unsigned int i = 0;
  106:
```

```
  107:             for (i = 0; i < service_name.size(); i++) {
  108:                 std::string finder = service_name.at(i);
  109:                 std::size_t pos = s.find(finder);
  110:                 // if (service_1 == service_name.at(i))
  111:                 if (pos != std::string::npos &&
  112:                     (service_start.at(i) != "Not started")) {
  113:                     out << line_number;
  114:                     service_end.at(i) = out.str();
  115:
  116:                     // grab elapsed time
  117:                     std::size_t my_start = 0;
  118:                     std::size_t my_end = 0;
  119:                     std::string ela_time;
  120:
  121:                     my_start = s.find("(");
  122:                     if (my_start != std::string::npos) {
  123:                         ++my_start;
  124:                         my_end = s.find(")");
  125:                         if (my_end != std::string::npos) {
  126:                             ela_time = s.substr(my_start, my_end - my_start)
;
  127:                           }
  128:                       }
  129:                     service_elapsed_time.at(i) = ela_time;
  130:                     break;
  131:                   }
  132:               }
  133:       }
  134:      // end of service startup
  135:
  136:     // start of software upgrades
  137:
  138:     if (boost::regex_match(s, soft_start)) {
  139:         str_start = s;
  140:         start_line = line_number;
  141:       }
  142:     if (boost::regex_match(s, soft_end)) {
  143:         str_end = s;
  144:       }
  145:     if (boost::regex_match(s, soft_orig)) {
  146:         str_orig_ver = s;
  147:       }
  148:     if (boost::regex_match(s, soft_new_ver)) {
  149:         str_new_ver = s;
  150:       }
  151:
  152:     if (str_start != " " && str_end != " " &&
  153:         str_orig_ver != " " && str_new_ver != " ") {
  154:         // get elapsed time vars
  155:         std::string my_time_start = str_start.substr(0, 15);
  156:         std::string my_time_end = str_end.substr(0, 15);
  157:
  158:         int w, x, y;
  159:
  160:         w = atoi((my_time_start.substr(7, 2)).c_str());
  161:         x = atoi((my_time_start.substr(10, 2)).c_str());
  162:         y = atoi((my_time_start.substr(13, 2)).c_str());
  163:
  164:         boost::posix_time::time_duration time_to_start(w, x, y, 0);
  165:
  166:         w = atoi((my_time_end.substr(7, 2)).c_str());
```

```
  167:         x = atoi((my_time_end.substr(10, 2)).c_str());
  168:         y = atoi((my_time_end.substr(13, 2)).c_str());
  169:
  170:         boost::posix_time::time_duration time_to_end(w, x, y, 0);
  171:         boost::posix_time::time_duration my_diff;
  172:         my_diff = time_to_end - time_to_start;
  173:
  174:         // clean up orig_ver and new_ver
  175:         // grab orig
  176:         std::size_t my_start = 0;
  177:         std::size_t my_end = 0;
  178:         std::string temp;
  179:
  180:         my_start = str_orig_ver.find("base-");
  181:         if (my_start != std::string::npos) {
  182:           my_start+=5;
  183:           my_end = str_orig_ver.find("armv6");
  184:           if (my_end != std::string::npos) {
  185:             temp = str_orig_ver.substr(my_start, (my_end - my_start) - 1);
  186:           }
  187:         }
  188:         str_orig_ver = temp;
  189:
  190:         // now grab new
  191:         my_start = 0;
  192:         my_end = 0;
  193:
  194:         my_start = str_new_ver.find("base-");
  195:         if (my_start != std::string::npos) {
  196:           my_start+=5;
  197:           my_end = str_new_ver.find("armv6");
  198:           if (my_end != std::string::npos) {
  199:             temp = str_new_ver.substr(my_start, (my_end - my_start) - 1);
  200:           }
  201:         }
  202:         str_new_ver = temp;
  203:
  204:         // print out
  205:         fout << "=== Softload ===" << std::endl;
  206:         fout << start_line << "(" << input_file << ") : "
  207:              << str_start.substr(0, 15) << " Softload Start" << std::endl;
  208:         fout << "\tOriginal version ==> " << str_orig_ver << std::endl;
  209:         fout << "\tNew version ==> " << str_new_ver << std::endl;
  210:         fout << "\tElapsed time (sec) ==> " << my_diff.total_seconds()
  211:              << std::endl;
  212:         fout << line_number << "(" << input_file << ") : "
  213:              << str_end.substr(0, 15) << " Softload Completed" << std::endl;
  214:         str_start = " ";
  215:         str_end = " ";
  216:         str_orig_ver = " ";
  217:         str_new_ver = " ";
  218:       }
  219:
  220:     // end of software upgrades
  221:
  222:       if (boost::regex_match(t, start_up) && boost::regex_match(s, success))
 {
  223:           // success
  224:           fout << line_number << "(" << input_file << "): " << s.substr(0, 1
9)
  225:                << " Boot Completed" << std::endl;
```

```
  226:           // boot time
  227:           time_end = s.substr(0, 19);
  228:           boost::posix_time::ptime start =
  229:             boost::posix_time::time_from_string(time_start);
  230:           boost::posix_time::ptime end =
  231:             boost::posix_time::time_from_string(time_end);
  232:           diff = end - start;
  233:           fout << "\tBoot Time: " << diff.total_milliseconds() << "ms"
  234:                << std::endl;
  235:           fout << std::endl;
  236:           t = "empty";
  237:
  238:           // service printout
  239:           print_stats(service_name, service_start, service_end,
  240:                       service_elapsed_time, fout, fin, input_file);
  241:
  242:         } else if (boost::regex_match(t, start_up) &&
  243:                    boost::regex_match(s, start_up)) {
  244:           // failure
  245:           fout << "**** Incomplete boot ****" << std::endl;
  246:           fout << std::endl;
  247:           t = "empty";
  248:
  249:           // service printout
  250:           print_stats(service_name, service_start, service_end,
  251:                       service_elapsed_time, fout, fin, input_file);
  252:         }
  253:
  254:         if (boost::regex_match(s, start_up)) {
  255:           fout << "=== Device boot ===" << std::endl;
  256:           fout << line_number << "(" << input_file << "): " << s.substr(0, 1
9)
  257:                << " Boot Start" << std::endl;
  258:           time_start = s.substr(0, 19);
  259:           t = s;
  260:
  261:           // service startup
  262:           populate_vectors(service_name, service_start, service_end,
  263:                    service_elapsed_time);
  264:         }
  265:       std::getline(fin, s);
  266:       line_number++;
  267:     }
  268:
  269:   fin.close();
  270:   fout.close();
  271:
  272:   return 0;
  273: }
  274:
  275: void populate_vectors(std::vector<std::string>& a, std::vector<std::string>&
 b,
  276:                       std::vector<std::string>& c, std::vector<std::string>&
d) {
  277:   a.push_back(" Logging");
  278:   b.push_back("Not started");
  279:   c.push_back("Not completed");
  280:   d.push_back("");
  281:
  282:   a.push_back(" DatabaseInitialize");
  283:   b.push_back("Not started");
```

```
   c.push_back("Not completed");
   d.push_back("");

   a.push_back(" MessagingService");
   b.push_back("Not started");
   c.push_back("Not completed");
   d.push_back("");

   a.push_back(" HealthMonitorService");
   b.push_back("Not started");
   c.push_back("Not completed");
   d.push_back("");

   a.push_back(" Persistence");
   b.push_back("Not started");
   c.push_back("Not completed");
   d.push_back("");

   a.push_back(" ConfigurationService");
   b.push_back("Not started");
   c.push_back("Not completed");
   d.push_back("");

   a.push_back(" LandingPadService");
   b.push_back("Not started");
   c.push_back("Not completed");
   d.push_back("");

   a.push_back(" PortConfigurationService");
   b.push_back("Not started");
   c.push_back("Not completed");
   d.push_back("");

   a.push_back(" CacheService");
   b.push_back("Not started");
   c.push_back("Not completed");
   d.push_back("");

   a.push_back(" ThemingService");
   b.push_back("Not started");
   c.push_back("Not completed");
   d.push_back("");

   a.push_back(" StagingService");
   b.push_back("Not started");
   c.push_back("Not completed");
   d.push_back("");

   a.push_back(" DeviceIOService");
   b.push_back("Not started");
   c.push_back("Not completed");
   d.push_back("");

   a.push_back(" BellService");
   b.push_back("Not started");
   c.push_back("Not completed");
   d.push_back("");

   a.push_back(" GateService");
   b.push_back("Not started");
   c.push_back("Not completed");
```

```
  345:   d.push_back("");
  346:
  347:   a.push_back(" ReaderDataService");
  348:   b.push_back("Not started");
  349:   c.push_back("Not completed");
  350:   d.push_back("");
  351:
  352:   a.push_back(" BiometricService");
  353:   b.push_back("Not started");
  354:   c.push_back("Not completed");
  355:   d.push_back("");
  356:
  357:   a.push_back(" StateManager");
  358:   b.push_back("Not started");
  359:   c.push_back("Not completed");
  360:   d.push_back("");
  361:
  362:   a.push_back(" OfflineSmartviewService");
  363:   b.push_back("Not started");
  364:   c.push_back("Not completed");
  365:   d.push_back("");
  366:
  367:   a.push_back(" AVFeedbackService");
  368:   b.push_back("Not started");
  369:   c.push_back("Not completed");
  370:   d.push_back("");
  371:
  372:   a.push_back(" DatabaseThreads");
  373:   b.push_back("Not started");
  374:   c.push_back("Not completed");
  375:   d.push_back("");
  376:
  377:   a.push_back(" SoftLoadService");
  378:   b.push_back("Not started");
  379:   c.push_back("Not completed");
  380:   d.push_back("");
  381:
  382:   a.push_back(" WATCHDOG");
  383:   b.push_back("Not started");
  384:   c.push_back("Not completed");
  385:   d.push_back("");
  386:
  387:   a.push_back(" ProtocolService");
  388:   b.push_back("Not started");
  389:   c.push_back("Not completed");
  390:   d.push_back("");
  391:
  392:   a.push_back(" DiagnosticsService");
  393:   b.push_back("Not started");
  394:   c.push_back("Not completed");
  395:   d.push_back("");
  396: }
  397:
  398: void print_stats(std::vector<std::string>& a, std::vector<std::string>& b,
  399:                 std::vector<std::string>& c, std::vector<std::string>& d,
  400:                 std::ofstream& fout, std::ifstream& fin, std::string name)
{
  401:   std::vector<std::string> boot_failures;
  402:   unsigned int my_size = a.size();
  403:
  404:   fout << "Services" << std::endl;
```

```
405:
406:   for (unsigned int i = 0; i < my_size; i++) {
407:       a.at(i).erase(0, 1);
408:       fout << "\t" << a.at(i) << std::endl;
409:       fout << "\t\tStart: " << b.at(i)
410:            << "(" << name << ")" << std::endl;
411:       fout << "\t\tCompleted: " << c.at(i)
412:            << "(" << name << ")" <<  std::endl;
413:       if (c.at(i) == "Not completed") {
414:           boot_failures.push_back(a.at(i));
415:         }
416:       fout << "\t\tElapsed Time: " << d.at(i) << std::endl;
417:     }
418:   if (boot_failures.size() > 0) {
419:       fout << "\t*** Services not successfully started: ";
420:       for (unsigned int i = 0; i < boot_failures.size(); i++) {
421:           if (i != (boot_failures.size() - 1)) {
422:               fout << boot_failures.at(i) << ", ";
423:             } else {
424:               fout << boot_failures.at(i) << std::endl;
425:             }
426:         }
427:     }
428:   a.clear();
429:   b.clear();
430:   c.clear();
431:   d.clear();
432: }
```

```
 1: C=g++ -g -Wall --std=c++98 -Werror
 2: E=.cpp
 3: O=main.o
 4: P=ps7b
 5: BOOST= -lboost_regex -lboost_date_time
 6: all: $(P)
 7: $(P):$(O)
 8:         $(C) -o $(P) $(O) $(BOOST)
 9:
10: $(E).o:
11:         $(C) -c $<  -o $@
12:
13: clean:
14:         rm $(O) $(P)
```