```
 1: //  Copyright 2015 Zheondre Calcano
 2: //  PS7b
 3: #include <boost/regex.hpp>
 4: #include <boost/date_time.hpp>
 5: #include <exception>
 6: #include <stdexcept>
 7: #include <sstream>
 8: #include <fstream>
 9: #include <iostream>
10: #include <string>
11: #include <vector>
12: #include "Services.hpp"
13:
14: using namespace std; //NOLINT
15: using namespace boost; //NOLINT
16:
17: void efname(string &name) { name += ".rpt";}
18: void parse(string fn) {
19:   int linenum, completeboot, startS, i, SoftLoadfound, ngvalf, serf;
20:   vector< int > holdval;
21:   services s;
22:   holdval.push_back(0);
23:   holdval.push_back(0);
24:   holdval.push_back(0);
25:   string ufn, filename, lif, rs, rsa, temp, boottime, isnll;
26:   isnll = "";
27:   ufn = fn;
28:   efname(fn);
29:   std::fstream outfile;
30:   outfile.open(fn.c_str(), fstream::out);
31:   rs = ".*log.c.166.*";
32:   rsa = ".*oejs.AbstractConnector:Started SelectChannelConnector.*";
33:   string t = "(\\d{2}):(\\d{2}):(\\d{2})";
34:   string tmm = "(\\d{2}):(\\d{2}):(\\d{2})\\.(\\d{3})";
35:   string gd = "(\\d{4})-(\\d{2})-(\\d{2})";
36:   boottime = "Boot Time: ";
37:   std::ifstream infile(ufn.c_str());
38:   smatch sm, sn, so, sp;
39:   regex e = regex(rs);
40:   regex ea = regex(rsa);
41:   regex etime(t);
42:   regex f(tmm);
43:   regex getdate(gd);
44:   regex getdatea(gd);
45:   std::ostringstream ss;
46:   linenum = completeboot = startS = SoftLoadfound = 0;
47:   while (getline(infile, lif)) {
48:     linenum++;
49:     if (regex_match(lif, e)) {
50:       if (completeboot == 1) {
51:         outfile << "**** Incomplete boot ****\n\n";
52:         completeboot = 0;
53:         outfile << "Services \n";
54:         outfile << s.Sformat(ufn);
55:         outfile << s.LFS();
56:         outfile << "\n";
57:       }
58:       if (SoftLoadfound == 1) {
59:         outfile << s.getL1();
60:         outfile << s.getL2();
61:         outfile << s.getL3();
```

```
   62:         outfile << s.getL4();
   63:         outfile << s.getL5();
   64:         s.makeLsNull();
   65:       }
   66:       outfile << "=== Device boot ===\n";
   67:       regex_search(lif, sm, etime);
   68:       regex_search(lif, so, getdate);
   69:       holdval[0] = boost::lexical_cast<int>(sm[1]);
   70:       holdval[1] = boost::lexical_cast<int>(sm[2]);
   71:       holdval[2] = boost::lexical_cast<int>(sm[3]);
   72:       ss.str("");
   73:       ss << linenum;
   74:       temp = ss.str();
   75:       temp += "(" + ufn + "): ";
   76:       temp += so[0] + " " + sm[0] + " Boot Start \n";
   77:       outfile << temp;
   78:       completeboot = 1;
   79:       startS = 1;
   80:       temp.clear();
   81:     }
   82:     if (startS == 1) {
   83:       s.ServiceStart(lif, linenum);
   84:       s.ServiceSuccess(lif, linenum);
   85:     }
   86:     if (s.SoftloadS(lif, linenum, ufn)) {
   87:       SoftLoadfound = 1;
   88:     }
   89:     if (SoftLoadfound == 1) {
   90:       s.findOV(lif);
   91:       s.findNV(lif);
   92:       SoftLoadfound = s.SoftloadEnd(lif, linenum, ufn);
   93:     }
   94:     if (regex_match(lif, ea)) {
   95:       ss.str("");
   96:       ss << linenum;
   97:       temp = ss.str();
   98:       temp += "(" + ufn + "): ";
   99:       regex_search(lif, sn, f);
  100:       regex_search(lif, sp, getdatea);
  101:       boost::posix_time::time_duration ta(holdval[0], holdval[1], holdval[2]
);
  102:       boost::posix_time::time_duration tb(boost::lexical_cast<int>(sn[1]),
  103:                                           boost::lexical_cast<int>(sn[2]),
  104:                                           boost::lexical_cast<int>(sn[3]));
  105:       // tb += boost::posix_time::millisec(boost::lexical_cast<int>(sn[4]));
  106:       tb = tb - ta;
  107:       temp += sp[0] + " " + sn[1] +":" + sn[2] + ":" + sn[3]
  108:       + " " + "Boot Completed\n";
  109:       outfile << temp;
  110:       ss.str("");
  111:       ss << tb.total_milliseconds();
  112:       outfile <<"\t"+ boottime + ss.str() + "ms\n\n";
  113:       completeboot = 0;
  114:       startS = 0;
  115:       temp.clear();
  116:       outfile << "Services\n";
  117:       outfile << s.Sformat(ufn);
  118:       isnll = s.LFS();
  119:       outfile << isnll;
  120:       if (isnll != "") {
  121:         outfile << "\n";
```

```
  122:       }
  123:       s.setNegvalues();
  124:     }
  125:   }
  126:   outfile.close();
  127: }
  128: int main(int argc, char *argv[]) {
  129:   string filename;
  130:   filename = argv[1];
  131:   if (filename.size() < 1)
  132:     throw
  133:       std::runtime_error("Null string for file name");
  134:   parse(filename);
  135:   return 0;
  136: }
```

```
 1: //  Copyright 2015 Zheondre Calcano
 2: //  PS7b
 3: #include <boost/regex.hpp>
 4: #include <boost/date_time.hpp>
 5: #include <exception>
 6: #include <stdexcept>
 7: #include <sstream>
 8: #include <fstream>
 9: #include <iostream>
10: #include <string>
11: #include <vector>
12: #include "Services.hpp"
13:
14: using namespace std; //NOLINT
15: using namespace boost; //NOLINT
16:
17: void services::makeLsNull() {
18:   l1 = l2 = l3 = l4 = l5 = "";
19: }
20: string services::getL1() {
21:   return l1;
22: }
23: string services::getL2() {
24:   return l2;
25: }
26: string services::getL3() {
27:   return l3;
28: }
29: string services::getL4() {
30:   return l4;
31: }
32: string services::getL5() {
33:   return l5;
34: }
35: regex services::getRS() { return rs; }
36: string services::AFail() { return allfs; }
37: string services::getsr(int x) {
38:   if (x < 0)
39:     throw std::invalid_argument("Value is less than 0");
40:   if (x > sofV)
41:     throw std::invalid_argument("Value is greater than vector size");
42:   return sname[x];
43: }
44: string services::getfSM() { return fSM ; }
45: string services::getSta() { return startservice; }
46: string services::getGS() { return GoodStart; }
47: string services::getCompleteLN(int x) { return CompleteLN[x]; }
48: string services::getStartLN(int x) { return StartLN[x]; }
49: string services::getElapsedT(int x) { return ElapsedT[x]; }
50: int services::sz() { return sofV; }
51: void services::setNegvalues() {
52:   for (int i = 0; i < sofV; i++) {
53:     StartLN[i] = "-1";
54:     CompleteLN[i]= "-1";
55:     ElapsedT[i] = "-1";
56:   }
57: }
58: void services::ServiceStart(string line, int linenum) {
59:   int i;
60:   std::ostringstream so;
61:   for (i = 0; i < sofV; i++) {
```

```
   62:     regex e(getSta() + getsr(i) + ".*");
   63:     if (regex_match(line, e)) {  // Start service found
   64:       so.str("");
   65:       so << linenum;
   66:       StartLN[i] = so.str();
   67:     }
   68:   }
   69: }
   70: void services::ServiceSuccess(string line, int linenum) {
   71:   int i; smatch sm; std::ostringstream so;
   72:   for (i = 0; i < sofV; i++) {
   73:     regex e(getGS() + getsr(i) + ".*");
   74:     if (regex_match(line, e)) {  // Success service found
   75:       regex dig(rs);
   76:       if (regex_search(line, sm, dig)) {
   77:         so.str("");
   78:         so << linenum;
   79:         ElapsedT[i] = boost::lexical_cast<string>(sm[1]);
   80:         CompleteLN[i] = so.str();
   81:       }
   82:     }
   83:   }
   84: }
   85: void services::findOV(string line) {
   86:   smatch sm; std::ostringstream so; string ltp;
   87:   regex e(".*intouch-application-base-.*");
   88:   if (regex_match(line, e)) {
   89:     regex rge(".*: removing.*");
   90:     if (regex_match(line, rge)) {
   91:       regex dig("(?=-[0-9])(.*?)(?=\\.armv)");
   92:       if (regex_search(line, sm, dig)) {
   93:         ltp = boost::lexical_cast<string>(sm[0]);
   94:         ltp.erase(0, 1);
   95:         l2 = "\tOriginal version ==> " + ltp + "\n";
   96:       }
   97:     }
   98:   }
   99: }
  100: void services::findNV(string line) {
  101:   smatch sm; std::ostringstream so; string ltp;
  102:   regex e(".*intouch-application-base-.*");
  103:   if (regex_match(line, e)) {
  104:     regex rge(".*: Processing.*");
  105:     if (regex_match(line, rge)) {
  106:       regex dig("(?=-[0-9])(.*?)(?=\\.armv)");
  107:       if (regex_search(line, sm, dig)) {
  108:         ltp = boost::lexical_cast<string>(sm[0]);
  109:         ltp.erase(0, 1);
  110:         l3 = "\tNew version ==> "+ltp +"\n";
  111:       }
  112:     }
  113:   }
  114: }
  115: bool services::SoftloadS(string line, int ln, string fn) {
  116:   smatch sm; std::ostringstream so; string ltp;
  117:   regex e(startSoftload);
  118:   if (regex_match(line, e)) {
  119:     regex rge("(\\s*\\w{3}\\s*[0-9]{1,2})");
  120:     regex rgd("([0-9]{2}):([0-9]{2}):([0-9]{2})");
  121:     if (regex_search(line, sm, rge)) {
  122:       so.str("");
```

```
  123:       so << ln;
  124:       ltp = "=== Softload ===\n";
  125:       ltp += so.str() + "(" + fn + ") : " + boost::lexical_cast<string>(sm[0
]);
  126:       ltp += " ";
  127:     }
  128:     if (regex_search(line, sm, rgd)) {
  129:       ltp += boost::lexical_cast<string>(sm[0]) + " Softload Start\n";
  130:       start[0] = boost::lexical_cast<int>(sm[1]);
  131:       start[1] = boost::lexical_cast<int>(sm[2]);
  132:       start[2] = boost::lexical_cast<int>(sm[3]);
  133:       l1 = ltp;
  134:       return true;
  135:     }
  136:   }
  137:   return false;
  138: }
  139: int services::SoftloadEnd(string line, int ln, string fn) {
  140:   // if valid line drab date
  141:   smatch sm; std::ostringstream so; string ltp;
  142:   regex e(EndSoftload);
  143:   if (regex_match(line, e)) {
  144:     regex rge("(\\s*\\w{3}\\s*[0-9]{1,2})");
  145:     regex rgd("([0-9]{2}):([0-9]{2}):([0-9]{2})");
  146:     if (regex_search(line, sm, rge)) {
  147:       so.str("");
  148:       so << ln;
  149:       ltp = so.str() + "(" + fn + ") : " + boost::lexical_cast<string>(sm[0]
);
  150:       ltp += " ";
  151:     }
  152:     if (regex_search(line, sm, rgd)) {
  153:       ltp += boost::lexical_cast<string>(sm[0]) + " Softload Completed\n";
  154:       end[0] = boost::lexical_cast<int>(sm[1]);
  155:       end[1] = boost::lexical_cast<int>(sm[2]);
  156:       end[2] = boost::lexical_cast<int>(sm[3]);
  157:       l5 = ltp;
  158:       GetEtime();
  159:     }
  160:   }
  161:   return 1;
  162: }
  163: void services::GetEtime() {
  164:   string temp;
  165:   std::ostringstream so;
  166:   boost::posix_time::time_duration st(start[0], start[1], start[2]);
  167:   boost::posix_time::time_duration et(end[0], end[1], end[2]);
  168:   et = et - st;
  169:   so.str("");
  170:   so << et.total_seconds();
  171:   temp = "\tElapsed time (sec) ==> " + so.str() + "\n";
  172:   l4 = temp;
  173: }
  174: string services::Sformat(string ufn) {
  175:   string temp = "";
  176:     for (int i = 0; i < sofV; i++) {
  177:       if (getCompleteLN(i) != "-1") {
  178:         temp +=  "\t" + getsr(i) + "\n\t\tStart: "
  179:         + getStartLN(i) + "(" + ufn + ")\n";
  180:         temp += "\t\tCompleted: " + getCompleteLN(i) + "(" + ufn + ")\n";
  181:         temp += "\t\tElapsed Time: " + getElapsedT(i) + "\n";
```

```
182:       } else {
183:         if (getStartLN(i) == "-1") {
184:           temp += "\t" + getsr(i) + "\n\t\tStart: "
185:           + "Not started(" + ufn + ")\n";
186:           temp += "\t\tCompleted: Not completed("
187:           + ufn +")\n\t\tElapsed Time:\n";
188:         } else {
189:           temp += "\t" + getsr(i) + "\n\t\tStart: "
190:           + getStartLN(i) +"(" + ufn + ")\n";
191:           temp += "\t\tCompleted: Not completed("
192:           + ufn +")\n\t\tElapsed Time:\n";
193:         }
194:       }
195:     }
196:     return temp;
197: }
198: string services::LFS() {
199:   string temp, t2;
200:   temp = t2 = "";
201:   int ngvalf = 0;
202:   for (int i = 0; i < sofV; i++) {
203:     if (getCompleteLN(i) == "-1") {
204:       if (ngvalf== 0) {
205:         t2 += fSM;
206:         temp += getsr(i);
207:         ngvalf++;
208:         continue;
209:       }
210:       if (ngvalf == 1) {
211:         temp +=", "+ getsr(i);
212:       }
213:     }
214:   }
215:   t2 += temp;
216:   return t2;
217: }
```