

```

1  //*****
2  // SensorDevices.cc Messtation
3  // Author:      M. Thaler
4  // Date:        3/2011
5  //*****
6
7  //*****
8  // system includes
9
10 #include <unistd.h>
11 #include <signal.h>
12 #include <stdio.h>
13 #include <stdlib.h>
14
15 #include <sys/socket.h>
16 #include <netdb.h>
17 #include <string.h>
18
19 //*****
20 // local includes
21
22 #include "defs.h"
23
24 //*****
25 // local constants
26
27 #define MAX_ITERATIONS 100000
28 #define THE_OVERLAP    2
29
30 //*****
31 // local data
32
33 int overlap = THE_OVERLAP;
34 int globalK = 0;
35
36 //*****
37 // local procedures
38
39 int connToServer(char *hostname, int port);
40
41 void SignalHandler(int sig) {
42     printf("\nSensors process receiving termination signal\n");
43     globalK = MAX_ITERATIONS + 1;
44 }
45
46 //*****
47
48 int numofsensors(int num) {
49     static int numofs = 0;
50     if (num > numofs) {
51         if (num <= SENSOR_MAX_NUM)
52             numofs = num;
53         else
54             numofs = -1;
55     }
56     return(numofs);
57 }
58
59 //*****
60 // equally distributed random values in the range low ... high
61
62 int intRand(int low, int high) {

```

```

63     int    lp, res;
64     double lo, hi, dif, ran;
65
66     if (low <= 0)
67         lp = (-1)*low + 1;
68     else
69         lp = 0;
70     lo = low + lp - 0.5;
71     hi = high + lp + 0.499;    // make sure not to round too much
72     dif = hi - lo;
73     ran = random();
74     ran = lo + dif * ran/RAND_MAX;
75     res = (int)(ran + 0.5);
76     res -= lp;
77     if (res > high)
78         res = high;           // make sure not to round too much
79     if (res < low)
80         res = low;
81     return res;
82 }
83
84 //*****
85 // generate sequence of temp devices
86
87 void RandomSequence(int *seq, int number) {
88     static int reservation[2*SENSOR_MAX_NUM];
89     static int firstRun = 1;
90     int devCount[SENSOR_MAX_NUM];
91
92     int idx, i, tmp;
93
94     if (number > numOfSensors(0)) {
95         printf("sequence: too many devices\n");
96         exit(0);
97     }
98
99     if (firstRun == 1) {
100         firstRun = 0;
101         for (i = 0; i < numOfSensors(0); i++)
102             devCount[i] = 0;
103         idx = 0;
104         while (idx < number) {
105             i = intRand(0, number-1);
106             if (devCount[i] < overlap) {
107                 devCount[i]++;
108                 reservation[idx] = i;
109                 idx++;
110             }
111         }
112     }
113
114     for (i = 0; i < number; i++)
115         devCount[i] = 0;
116
117     for (i = 0; i < number; i++) {
118         reservation[i+number] = reservation[i];
119         devCount[reservation[i]]++;
120     }
121
122     idx = 0;
123     while (idx < number) {
124         i = intRand(0, number-1);

```

```

125     if (devCount[i] < 2) {                // if not yet twice in list
126         if (devCount[i] < 1) {            // if not in list
127             devCount[i]++;
128             reservation[idx] = i;
129             idx++;
130         }
131         else {
132             tmp = intRand(0, 9);           // if in list
133             if (tmp > 6) {                  // do only for 5%
134                 devCount[i]++;
135                 reservation[idx] = i;
136                 idx++;
137             }
138         }
139     }
140 }
141
142 for (i = 0; i < number; i++) {
143     *seq = reservation[i+number];
144     seq++;
145 }
146 }
147
148
149 //*****
150 // Function: main(), parameter: hostname or IP address in dot format
151 //*****
152
153 int main(int argc, char *argv[]) {
154
155     struct sigaction sig;
156
157     int      StationSeq[SENSOR_MAX_NUM];
158
159     int      sfd, maxWait, i, j, rand;
160 int      anzSensors;
161     char      buf[BUF_SIZE];
162     SensorData sensor;
163     float      deltaT;
164
165     float      tempPreset[8] = {20, 45, 30, 20, 15, 10, 15, 20};
166     float      startup[8]    = {0, 0, 0, 0, 0, 0, 0, 0};
167     int        sequenceNr[8] = {0, 0, 0, 0, 0, 0, 0, 0};
168
169     //*** check for hostname ... a kind of hack
170
171     if (argc < 4) {
172         printf("Need number of devices, hostname or IP address and port number\n");
173         exit(-1);
174     }
175
176     if ((anzSensors = numOfSensors(atoi(argv[1]))) < 0) {
177         printf("\n*** invalid number of sensor devices ***\n\n");
178         exit(0);
179     }
180
181     // set up signal handlers
182     sigemptyset(&sig.sa_mask);
183     sig.sa_handler = SignalHandler;
184     sig.sa_flags = 0;
185     sigaction(SIGTERM, &sig, NULL);
186     sigaction(SIGKILL, &sig, NULL);

```

```

187 sigaction(SIGINT, &sig, NULL);
188
189 sleep(2);
190 printf("Sensor device starting up\n");
191
192 globalK = 0;
193 while (globalK < MAX_ITERATIONS) {
194
195     RandomSequence(StationSeq, anzSensors);
196
197     for (i = 0; i < anzSensors; i++) { // for all devices
198         deltaT = intRand(-2, 2);
199         sensor.deviceID = StationSeq[i];
200         sensor.sequenceNr = sequenceNr[sensor.deviceID];
201         sensor.valIS = deltaT + startup[sensor.deviceID];
202         sensor.valREF = tempPreset[sensor.deviceID];
203         sensor.status = 0;
204
205         sequenceNr[sensor.deviceID]++;
206         sfd = connToServer(argv[2], atoi(argv[3]));
207         write(sfd, (char *)&sensor, sizeof(SensorData));
208         close(sfd);
209         maxWait = 4000000;
210         maxWait = maxWait / anzSensors;
211         rand = intRand(maxWait/3, maxWait);
212         usleep(rand);
213     }
214     for (j = 0; j < anzSensors; j++) {
215         if (startup[j] < tempPreset[j])
216             startup[j] += 2;
217         else
218             startup[j] = tempPreset[j];
219     }
220     globalK++;
221 }
222 exit(0);
223
224 } // end main
225
226 //*****
227 // socket client
228
229 int connToServer(char *hostname, int port) {
230     int sfd, sysRet;
231     char stringPort[8];
232     struct addrinfo hints, *aiList, *aiPtr = NULL;
233
234     sprintf(stringPort, "%d", port);
235
236     memset(&hints, '\0', sizeof(hints));
237     hints.ai_flags = AI_CANONNAME;
238     hints.ai_family = AF_UNSPEC;
239     hints.ai_socktype = SOCK_STREAM;
240
241     sysRet = getaddrinfo(hostname, stringPort, &hints, &aiList);
242     if (sysRet != 0) {
243         printf("error getting network address %s\n", gai_strerror(sysRet));
244         return(-1);
245     }
246
247     aiPtr = aiList;
248     while (aiPtr != 0) {

```

```
249     sfd = socket(aiPtr->ai_family, aiPtr->ai_socktype, aiPtr->ai_protocol);
250     if (sfd >= 0) {
251         sysRet = connect(sfd, aiPtr->ai_addr, aiPtr->ai_addrlen);
252         if (sysRet == 0)
253             break;
254         else
255             close(sfd);
256     }
257     aiPtr = aiPtr->ai_next;
258 }
259 if (aiPtr == NULL) {
260     return(-1);
261 }
262 else
263     return(sfd);
264 }
265
266 //*****
```