

```

1  //*****
2  // File:      semaphore.cc
3  // Author:   M. Thaler    15.01.2003
4  //
5  // Semaphor operations
6  //*****
7
8  #include "semaphore.h"
9
10 //*****
11 /* Static class variables
12
13 int    Semaphore::numOfSems = 0;
14 int    Semaphore::semID = 0;
15 char*  Semaphore::keyFilename = NULL;
16 int    Semaphore::projectID = 0;
17
18 //*****
19 /* Constructor & Destructor
20 /* if number > 0 then create array with num semaphore
21 /*      else get existing semaphore
22
23 Semaphore::Semaphore(int num) {
24     numOfSems = num;
25     createSemaphorArray();
26 }
27
28 Semaphore::Semaphore(int num, const char* keyFile, int projID) {
29     numOfSems = num;
30     keyFilename = (char *)keyFile;
31     projectID = projID;
32     createSemaphorArray();
33 }
34
35 Semaphore::~Semaphore() {}    // do nothing
36
37 //*****
38 /* release semaphore
39
40 int
41 Semaphore::up(int semaphor) {
42     struct sembuf buf;
43     buf.sem_num = semaphor;    // semaphor number
44     buf.sem_op = 1;    // add 1 to value
45     buf.sem_flg = 0;    // SEM_UNDO ist explicitly not
46         // set, since otherwise a full
47         // reset is made on exit, which
48         // complicates termination
49     return semop(semID, &buf, 1); // do it
50 }
51
52 //*****
53 /* wait for sempahore to be release (if closed)
54
55 int
56 Semaphore::down(int semaphor) {
57     struct sembuf buf;    // semaphor number
58     buf.sem_num = semaphor;    // semaphor number
59     buf.sem_op = -1;    // sub 1 from value
60     buf.sem_flg = 0;    // SEM_UNDO ist explicitly not
61         // set, since otherwise a full
62         // reset is made on exit, which

```

```

63         // complicates termination
64     return semop(semID, &buf, 1); // do it
65 }
66
67 //*****
68 /* get value of semaphor
69
70 int
71 Semaphore::getValue(int semaphor) {
72     semun sem_union;
73     return semctl(semID, semaphor, GETVAL, sem_union);
74 }
75
76 //*****
77 /* set value of semaphor
78
79 int
80 Semaphore::setValue(int semaphor, int value) {
81     semun sem_union;
82     sem_union.val = value;
83     return semctl(semID, semaphor, SETVAL, sem_union);
84 }
85
86 //*****
87 /* cleanup: destroy semaphor array and delete key file
88
89 void
90 Semaphore::removeSemaphore(void) {
91     semctl(semID, 0, IPC_RMID);
92     if (keyFilename != NULL)
93         unlink(keyFilename);
94 }
95
96 //*****
97 /* Local procedures
98 /* obtain an array of Semaphores
99
100 int
101 Semaphore::createSemaphoreArray(void) {
102     key_t key = IPC_PRIVATE;
103     semun sem_union;
104     int flags, semErr;
105
106     if (numOfSems > 0)
107         flags = 0664 | IPC_CREAT;
108     else
109         flags = 0;
110
111     if (keyFilename == NULL) {
112         cout << "sem: not implemented feature\n";
113         exit(-1);
114     }
115     // create key file, if not available
116     int fd = open(keyFilename, O_RDWR | O_CREAT, 0770);
117     close(fd);
118
119     // get key by key file and ID
120     key = ftok(keyFilename, projectID);
121
122     // obtain semaphor array and initilaize to 0
123     semID = semget(key, numOfSems, flags);
124     semErr = semID;

```

```
125  if (semErr > -1) {
126      sem_union.val = 0;
127      for (int j = 0; j < numOfSems; j++) {
128          if ((semErr = semctl(semID, j, SETVAL, sem_union)) < 0)
129              break;
130      }
131  }
132  if (semErr < 0) {
133      cout << "failed to allocate semaphore array\n";
134      exit(-1);
135  }
136  return semID;
137 }
138
139 /**/
```