

```
1 //----- Réalisation <Sortie> -----
2
3 //-----
4 //-----
5 #include <sys/types.h>
6 #include <sys/wait.h>
7 #include <sys/ipc.h>
8 #include <sys/msg.h>
9 #include <unistd.h>
10 #include <time.h>
11 #include <signal.h>
12 #include <stdlib.h>
13 #include <vector>
14 #include <fcntl.h>
15 #include <time.h>
16 #include <errno.h>
17 #include <sys/sem.h>
18 #include <sys/shm.h>
19 #include <map>
20 #include <iostream>
21 //-----
22 #include "libs/Utils.h"
23 #include "libs/Heure.h"
24 #include "Mere.h"
25 #include "Sortie.h"
26 #include "Clavier.h"
27 #include "Voiture.h"
28
29 //-----
30 //-----
31 //-----
32 //-----
33 //-----
34
35 static int idMemoirePartagee;
36 static int idFileVoituresSortie;
37 static void * memPartagee;
38 static int idSemMemPartagee;
39 static int idSemEntree;
40 static std::map<pid_t,int> pidSortie;
41 static char buff[60];
42
43 //-----
44 static void finSortieVoiture(int numSignal)
45 // Mode d'emploi :
46 //
47 {
48     int status;
49     pid_t pidFile = wait(&status);
50     unsigned int numS-WEXITSTATUS(status);
51
52     struct sembuf opP;
53     opP.sem_num=0;
54     opP.sem_op=-1;
55     opP.sem_flg=0;
56
57     struct sembuf opV;
58     opV.sem_num=0;
59     opV.sem_op=1;
60     opV.sem_flg=0;
61 }
```

```
62 while(semop(idSemMemPartagee, &opP, 1) == -1 && errno==EINTR);
63 Voiture uneVoiture = ((VoituresParking*)memPartagee)->voituresGarees[numS-1];
64 ((VoituresParking*)memPartagee)->placesLibres++;
65 semop(idSemMemPartagee, &opV, 1);
66 Effacer((TypeZone)numS);
67 Effacer(MESSAGE);
68 Afficher(MESSAGE);
69 AfficherSortie(uneVoiture.type, uneVoiture.numero, uneVoiture.arrivee, time(NULL));
70
71 while(semop(idSemMemPartagee, &opP, 1) == -1 && errno==EINTR);
72 Voiture * lesVoitures = ((VoituresParking*)memPartagee)->voituresEnAttente;
73
74 char buff[50];
75 printf(buff, "\t[[Sortie]]\tPas de file ouverte\n");
76 if(lesVoitures[FILE_PROF_BP].type==PROF && lesVoitures[FILE_GB].type!=PROF)
77 {
78     opV.sem_num=FILE_PROF_BP;
79     semop(idSemEntree, &opV, 1);
80     sprintf(buff, "\t[[Sortie]]\touvrir FILE_PROF_BP\n");
81
82     printf(buff, "\t[[Sortie]]\touvrir FILE_PROF_BP\n");
83
84 }
85 else if(lesVoitures[FILE_PROF_BP].type==PROF && lesVoitures[FILE_GB].type==PROF)
86 {
87     if(lesVoitures[FILE_PROF_BP].arrivee <= lesVoitures[FILE_GB].arrivee)
88     {
89         opV.sem_num=FILE_PROF_BP;
90         semop(idSemEntree, &opV, 1);
91         sprintf(buff, "\t[[Sortie]]\touvrir FILE_PROF_BP\n");
92     }
93     else
94     {
95         opV.sem_num=FILE_GB;
96         semop(idSemEntree, &opV, 1);
97         sprintf(buff, "\t[[Sortie]]\touvrir FILE_GB\n");
98     }
99 }
100 else if(lesVoitures[FILE_PROF_BP].type!=PROF && lesVoitures[FILE_GB].type==PROF)
101 {
102     opV.sem_num=FILE_GB;
103     semop(idSemEntree, &opV, 1);
104     sprintf(buff, "\t[[Sortie]]\touvrir FILE_GB\n");
105 }
106 else if(lesVoitures[FILE_AUTRE_BP].type==AUTRE && lesVoitures[FILE_GB].type==
AUTRE)
107 {
108     if(lesVoitures[FILE_AUTRE_BP].arrivee <= lesVoitures[FILE_GB].arrivee)
109     {
110         opV.sem_num=FILE_AUTRE_BP;
111         semop(idSemEntree, &opV, 1);
112         sprintf(buff, "\t[[Sortie]]\touvrir FILE_AUTRE_BP\n");
113     }
114     else
115     {
116         opV.sem_num=FILE_GB;
117         semop(idSemEntree, &opV, 1);
118         sprintf(buff, "\t[[Sortie]]\touvrir FILE_GB\n");
119     }
120 }
121 else if(lesVoitures[FILE_AUTRE_BP].type==AUTRE && lesVoitures[FILE_GB].type==
```

```
122 AUCUN
123 {
124     opV.sem_num=FILE_AUTRE_BP;
125     semop.idSemEntree=&opV.1;
126     printf(buff, "\t[[Sortie]]\touvrir FILE_AUTRE_BP\n");
127 }
128 else if (lesVoitures[FILE_AUTRE_BP].type==AUCUN && lesVoitures[FILE_GB].type==
129 AUTRE)
130 {
131     opV.sem_num=FILE_GB;
132     printf(buff, "\t[[Sortie]]\touvrir FILE_GB\n");
133     semop.idSemEntree=&opV.1;
134 }
135 opV.sem_num=0;
136 ecritureLog(buff);
137 semop.idSemMemPartagee=&opV.1;
138 pidSortie.erase(pidFille);
139 }
140 //-----Fin de finSortieVoiture
141
142 static void finTache(int numSignal)
143 // Mode d'emploi :
144 //
145 {
146     for (std::pair<const int, int> children : pidSortie)
147     {
148         kill(children.first, SIGUSR2);
149     }
150     shmdt(memPartagee);
151     exit(0);
152 }
153 //-----Fin de finTache
154
155 static void Initialisation(int idMP, int idFVS, int idSemMP, int idSemEnt)
156 {
157     printf(buff, "\t[[Sortie]]\tInitialisation\n");
158     ecritureLog(buff);
159
160     struct sigaction handlerUSR2;
161     handlerUSR2.sa_handler=finTache;
162     sigemptyset(&handlerUSR2.sa_mask);
163     handlerUSR2.sa_flags=0;
164     sigaction(SIGUSR2, &handlerUSR2, NULL);
165
166     struct sigaction handlerCHLD;
167     handlerCHLD.sa_handler=finSortieVoiture;
168     sigemptyset(&handlerCHLD.sa_mask);
169     handlerCHLD.sa_flags=0;
170     sigaction(SIGCHLD, &handlerCHLD, NULL);
171
172     idMemoirePartagee=idMP;
173     idFileVoitureSortie=idFVS;
174     idSemMemPartagee=idSemMP;
175     idSemEntree=idSemEnt;
176     memPartagee=shmat(idMemoirePartagee, NULL, 0);
177 }
178
179 static void Moteur()
180 {
```

```
181 msgInt numS = {0};
182 MY_SA_RESTART(msgrcv(idFileVoitureSortie, (void *) &numS, sizeof(msgInt), 0, 0));
183 pid_t pid = SortirVoiture(numS.numero);
184 if (pid > 0){
185     pidSortie.insert(std::pair<pid_t, int>(pid, numS.numero));
186 }
187 }
188
189 //-----Fonctions publiques
190
191 void Sortie(int idMP, int idFVS, int idSemMP, int idSemEnt)
192 {
193     Initialisation(idMP, idFVS, idSemMP, idSemEnt);
194
195     for(;;)
196     {
197         Moteur();
198     }
199 }
200 //-----Fin de Sortie
201
```