```
1  /*************************************************************************
   **
2                            Sortie  -  description
3                              -------------------
4  debut     : 16/03/16
5  copyright : (C) 2016 par Nicolas Gripont
6  e-mail    :  nicolas.gripont@insa-lyon.fr
7
8  *************************************************************************
   */
9
10 //---------- Realisation de la tache <Sortie> (fichier Sortie.cpp) ---
11
12 //////////////////////////////////////////////////////////////////       
   INCLUDE
13 //-------------------------------------------------------- Include      
   systeme
14 #include <sys/types.h>
15 #include <sys/wait.h>
16 #include <sys/msg.h>
17 #include <errno.h>
18 #include <stdio.h>
19 #include <sys/shm.h>
20 #include <sys/sem.h>
21 #include <signal.h>
22 #include <set>
23 #include <unistd.h>
24 #include <time.h>
25
26 //------------------------------------------------------ Include        
   personnel
27 #include "Config.h"
28 #include "Outils.h"
29 #include "Sortie.h"
30
31
32 //////////////////////////////////////////////////////////////////     
   PRIVE
33 //------------------------------------------------------------------      
   Constantes
34
35 //------------------------------------------------------------------------    
   Types
36
37 //------------------------------------------------------ Variables        
   statiques
38 static int mutex_MemoirePartageeVoitures;
39 static int shmId_MemoirePartageeVoitures;
40 static int mutex_MemoirePartageeRequetes;
41 static int shmId_MemoirePartageeRequetes;
42 static int semSyc_MemoirePartageeRequetes;
43 static MemoirePartageeVoitures* memoirePartageeVoitures;
44 static MemoirePartageeRequetes* memoirePartageeRequetes;
45 static set<pid_t> voituriers; // set contenant les pids des             
```

```
    voiturierSortie
46                                  // en cours d'exécution
47
48  //------------------------------------------------------ Fonctions privees
49
50  static void finVoiturier(int numSignal);
51  // Mode d'emploi : fonction permettant de gérer la fin d'un processus
52  // SortieVoiture.
53  // A connecter au signal SIGCHLD
54  //
55  // Contrat : l'accès à la mémoire partagés requêtes et voitures
56  // doit être possible (memoirePartageeRequetes et memoirePartageeVoitures)
57  // L'accès aux différents sémaphores doit être possible
58  // (mutex_MemoirePartageeVoitures,mutex_MemoirePartageeRequetes
59  // semSyc_MemoirePartageeRequetes)
60  //
61
62  static void fin(int numSignal);
63  // Mode d'emploi : fonction permettant de Terminer le processus.
64  // A connecter au signal SIGUSR2
65  //
66  // Contrat : l'accès à la mémoire partagés requêtes et voitures
67  // doit être possible (memoirePartageeRequetes et memoirePartageeVoitures)
68
69
70  static int choixRequete();
71  // Mode d'emploi : Fonction permettant de choisir la requete d'entrée
72  // à traiter. Renvoie l'indice de la requete à traiter (dans la
73  // memoirePartageeRequetes) ou -1 si aucune requete.
74  //
75  // Contrat : l'accès à la mémoire partagés requêtes doit être possible
76  // (memoirePartageeRequetes)
77
78  static bool isV1Prio(Voiture v1, Voiture v2);
79  // Mode d'emploi : Renvoie true si v1 est prioritaire par rapport à v2,
80  // false sinon.
81  //
82  // Contrat : aucun
83
84
85
86  static void finVoiturier(int numSignal)
87  // Algorithme :
88  //
89  {
90      // PHASE MOTEUR :
91      sembuf prendreMutex = {(short unsigned int)0, (short)-1, (short)0};
92      sembuf vendreMutex = {(short unsigned int)0, (short)1, (short)0};
93      pid_t pid_Voiturier;
94      int statut_Voiturier;
95      int numeroPlace;
```

```
 96      Voiture voiture;
 97      int indiceRequete;
 98
 99      if(numSignal == SIGCHLD)
100      {
101          //recuperation du pid du voiturier terminé
102          pid_Voiturier = wait(&statut_Voiturier);
103          numeroPlace = WEXITSTATUS(statut_Voiturier);
104
105          //on prend le mutex memoire partagée voitures pour pouvoir y
             accéder (être seul à y accéder)
106          while(semop(mutex_MemoirePartageeVoitures,&prendreMutex,1) ==
             -1 && errno == EINTR);
107
108          //recupération de la voiture sortie
109          voiture = memoirePartageeVoitures->voitures[numeroPlace-1];
110          //on efface la voiture sortie de la mémoire partagée
111          memoirePartageeVoitures->voitures[numeroPlace-1] =
             {TypeUsager::AUCUN,0,0};
112          //on rend l'accès à la mémoire partagée voitures
113          semop(mutex_MemoirePartageeVoitures,&vendreMutex,1);
114
115          //on efface les infos de la voiture sortie de la zone etat
116          Effacer((TypeZone)numeroPlace);
117          //on affiche les informations de la voiture sortie dans la
             zone sortie
118
             AfficherSortie(voiture.typeUsager,voiture.numero,voiture.arrivee
             ,time(NULL));
119          //on enleve le pid de du voiturierSortie terminé du set
120          voituriers.erase(pid_Voiturier);
121
122          //on prend le mutex memoire partagée requetes pour pouvoir y
             accéder (être seul à y accéder)
123          while(semop(mutex_MemoirePartageeRequetes,&prendreMutex,1) ==
             -1 && errno == EINTR);
124
125          //s'il y a une requete a traiter
126          if( (indiceRequete = choixRequete()) != -1 )
127          {
128              //on efface la requete à traiter de la mémoire partagée
                 requetes
129              memoirePartageeRequetes->requetes[indiceRequete] =
                 {TypeUsager::AUCUN,0,0};
130              //on efface la requete à traiter de la zone requetes
131              Effacer((TypeZone)(TypeZone::REQUETE_R1 + indiceRequete));
132              //on libere le processus entrée en donnant un jeton au
                 semSync correspondant à la requête
133              sembuf vendreSemSync = {(short unsigned int)indiceRequete,
                 (short)1, (short)0};
134
                 while(semop(semSyc_MemoirePartageeRequetes,&vendreSemSync,1)
                  == -1 && errno == EINTR);
135          }
```

```
136             //on rend l'accès à la mémoire partagée requetes
137             semop(mutex_MemoirePartageeRequetes,&vendreMutex,1);
138         }
139     } //----- fin de finVoiturier
140
141
142     static void fin(int numSignal)
143     // Algorithme :
144     //
145     {
146         // PHASE DESTRUCTION :
147         if(numSignal == SIGUSR2)
148         {
149             //on masque le signal SIGCHLD pour ne pas être interrompu par
                la fin
150             //d'un voiturierSortie
151             struct sigaction action;
152             action.sa_handler = SIG_IGN ;
153             sigemptyset(&action.sa_mask);
154             action.sa_flags = 0 ;
155             sigaction(SIGCHLD,&action,NULL);
156
157             //on envoie le signal SIGUSR2 a tous les voiturierSortie en
                cours d'exécution
158             for(set<pid_t>::iterator it = voituriers.begin(); it !=
                voituriers.end(); it++)
159             {
160                 kill(*it,SIGUSR2);
161             }
162             //on attend la fin des voiturierSortie à qui on a envoyé SIGUSR2
163             for(set<pid_t>::iterator it = voituriers.begin(); it !=
                voituriers.end(); it++)
164             {
165                 waitpid(*it,NULL,0);
166             }
167
168             //On se détache des mémoires partagées
169             shmdt(memoirePartageeVoitures);
170             shmdt(memoirePartageeRequetes);
171
172             //On quitte le processus
173             exit(0);
174         }
175     } //----- fin de fin
176
177
178     static bool isV1Prio(Voiture v1, Voiture v2)
179     {
180         if(v1.typeUsager == v2.typeUsager) { return (v1.arrivee <=
            v2.arrivee); }
181         if(v1.typeUsager == TypeUsager::PROF && v2.typeUsager ==
            TypeUsager::AUTRE) { return true; }
182         if(v1.typeUsager == TypeUsager::AUTRE && v2.typeUsager ==
            TypeUsager::PROF) { return false; }
```

```
183         if(v1.typeUsager > v2.typeUsager) { return true; }
184     //    if(v1.typeUsager < v2.typeUsager) { return false; }
185         return false;
186     }
187
188     static int choixRequete()
189     // Algorithme :
190     //
191     {
192         Voiture v1 = memoirePartageeRequetes->requetes[INDICE_ENTREE_BLAISE_PASCALE_PROF];
193         Voiture v2 = memoirePartageeRequetes->requetes[INDICE_ENTREE_BLAISE_PASCALE_AUTRE];
194         Voiture v3 = memoirePartageeRequetes->requetes[INDICE_ENTREE_GASTON_BERGER];
195
196         if(isV1Prio(v1,v2) && isV1Prio(v1,v3) && v1.typeUsager != TypeUsager::AUCUN)
197         {
198             return 0;
199         }
200         if(isV1Prio(v2,v1) && isV1Prio(v2,v3) && v2.typeUsager != TypeUsager::AUCUN)
201         {
202             return 1;
203         }
204         if(isV1Prio(v3,v2) && isV1Prio(v3,v1) && v3.typeUsager != TypeUsager::AUCUN)
205         {
206             return 2;
207         }
208
209         return -1;
210     } //----- fin de choixRequete
211
212     ////////////////////////////////////////////////////////////////  PUBLIC
213     //---------------------------------------------------- Fonctions publiques
214
215
216     void Sortie(int msgid_BAL, int mutex_MPR, int semSyc_MPR, int shmId_MPR, int mutex_MPV, int shmId_MPV)
217     // Algorithme :
218     //
219     {
220         pid_t pid_Voiturier;
221         int msgid_BoiteAuxLettres;
222         MessageDemandeSortie demande;
223
224         // PHASE INITIALISATION
225         //masquage des signaux SIGUSR1, SIGUSR2, SIGCHLD
```

```
226         struct sigaction action;
227         action.sa_handler = SIG_IGN ;
228         sigemptyset(&action.sa_mask);
229         action.sa_flags = 0 ;
230         sigaction(SIGUSR1,&action,NULL);
231         sigaction(SIGUSR2,&action,NULL);
232         sigaction(SIGCHLD,&action,NULL);
233 
234         //Positionnement du handler fin sur SIGUSR2
235         struct sigaction actionFin;
236         actionFin.sa_handler = fin;
237         sigemptyset(&actionFin.sa_mask);
238         actionFin.sa_flags = 0 ;
239         sigaction(SIGUSR2,&actionFin,NULL);
240 
241         //Positionnement du handler finVoiturier sur SIGUSR2
242         struct sigaction actionFinVoiturier;
243         actionFinVoiturier.sa_handler = finVoiturier;
244         sigemptyset(&actionFinVoiturier.sa_mask);
245         actionFinVoiturier.sa_flags = 0 ;
246         sigaction(SIGCHLD,&actionFinVoiturier,NULL);
247 
248         //récupération des paramètres
249         msgid_BoiteAuxLettres = msgid_BAL;
250         semSyc_MemoirePartageeRequetes = semSyc_MPR;
251         mutex_MemoirePartageeRequetes = mutex_MPR;
252         shmId_MemoirePartageeRequetes = shmId_MPR;
253         mutex_MemoirePartageeVoitures = mutex_MPV;
254         shmId_MemoirePartageeVoitures = shmId_MPV;
255 
256         //Attachement des mémoires partagées
257         memoirePartageeVoitures = (MemoirePartageeVoitures*)
            shmat(shmId_MemoirePartageeVoitures,NULL,0);
258         memoirePartageeRequetes = (MemoirePartageeRequetes*)
            shmat(shmId_MemoirePartageeRequetes,NULL,0);
259 
260         // PHASE MOTEUR
261         for(;;)
262         {
263             //lecture de messages dans la boite aux lettres
264 
            while(msgrcv(msgid_BoiteAuxLettres,&demande,sizeof(MessageDemand
            eSortie),0,0) == -1 && errno == EINTR); //sans block
265 
266             // si la création d'un voiturierSortie a fonctionnée
267             if( (pid_Voiturier = SortirVoiture(demande.numeroPlace)) != -1 )
268             {
269                 // ajout du pid au set
270                 voituriers.insert(pid_Voiturier);
271             }
272         }
273     } //----- fin de Sortie
274 
275 
```