```cpp
/*************************************************************************
                           Entree  -  description
                             -------------------
    début                : 2016
    copyright            : (C) 2016 par cespeute
*************************************************************************/

//---------- Réalisation de la classe <Entree> (fichier Entree.cpp)---

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include système
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <iostream>
#include <signal.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/msg.h>


#include <sys/wait.h>

#include <map>

//------------------------------------------------------ Include personnel
#include "Entree.h"
#include "Voiture.h"
#include "libs/Outils.h"

//------------------------------------------------------------------ Prototypes

/** Execute la phase d'initialisation de la tâche
*/
static void EntreePhaseInitialisation();

/** Execute la phase moteur de la tâche
*/
static void EntreePhaseMoteur();

/** Execute la phase de destruction de la tâche
*/
static void EntreePhaseDestruction();

static void EnregisterHandlers();
static void QuitterHandler(int sig);
static void ChildHandler(int sig);

//------------------------------------------------------------------ Constantes

//------------------------------------------------------ Variables fichier

static int shmVoituresParking;  // Shared memory pour les voitures dans
                                // le parking
static int semVoituresParking;  // Sémaphore pour les voitures
static int msgFileVoiture;      // File de messages pour les voitures
                                // en attente
static int semOuvrirPortes;     // Sémaphores pour indiquer qu'une voiture
                                // à quitté le parking
```



```cpp
static int idBariere;           // Le numéro d'identification de la barrière

static VoituresParking *parking;

static std::map<pid_t,Voiture> pidGarage;

static char buff[60];           // Buffer pour la manipulation de strings c

//-----------------------------------------  Outils pour les sémaphores
static sembuf reserver = {0,-1,0};
static sembuf liberer  = {0,1,0};
//------------------------------------------------------------------ Fonctions

//------------------------------------------ Define Constantes
#define TEMPO 1

//------------------------------------------------------ Corps des methodes

int CreerEntree(
                int _shmVoituresParking,
                int _semVoituresParking,
                int _msgFileVoiture,
                int _semOuvrirPortes,
                int _idBariere
                )
{
    pid_t pid = fork();
    if(pid == 0)
    {
        // Initialiser toutes les variables fichier
        shmVoituresParking = _shmVoituresParking;
        semVoituresParking = _semVoituresParking;
        msgFileVoiture = _msgFileVoiture;
        semOuvrirPortes = _semOuvrirPortes;
        idBariere = _idBariere;

        EntreePhaseInitialisation();
        EntreePhaseMoteur();

        // Au cas où quelquechose d'étrange se passerait dans PhaseMoteur
        EntreePhaseDestruction();
    }

    sprintf(buff, "Processus Entree crée avec pid : %d\n", pid);
    ecrireLog(buff);
    return pid;
}

static void EntreePhaseInitialisation()
{
    sprintf(buff,"\t[[Entree n°%d]]\tInitialisation\n",idBariere);
    ecrireLog(buff);

    parking = (VoituresParking*) shmat( shmVoituresParking, NULL, 0);

    EnregisterHandlers();
}

static void EntreePhaseMoteur()
{
    Voiture voitureBarriere = {AUCUN,0,0};
```

```
123         for(;;)
124         {
125             // Eviter que le ChildHandler ne fasse sauter cette commande
126             MY_SA_RESTART(msgrcv(msgFileVoiture, (void *) &voitureBarriere, sizeof(
                Voiture),0, 0));
127 
128             // Dessiner la voiture
129             DessinerVoitureBarriere((TypeBarriere) (idBariere+1),voitureBarriere.type);
130             voitureBarriere.arrivee = time(NULL);
131             MY_SA_RESTART(semop(semVoituresParking,&reserver, 1));
132             // Si il n'y à plus de places disponibles
133             if (parking->placesLibres <= 0)
134             {
135                 parking->voituresEnAttente[idBariere].type      = voitureBarriere.type;
136                 parking->voituresEnAttente[idBariere].numero    = voitureBarriere.numero;
137                 parking->voituresEnAttente[idBariere].arrivee   = voitureBarriere.
                    arrivee;
138                 semop(semVoituresParking,&liberer, 1);
139 
140                 sprintf(buff,"\t[[Entree n°%d]]\tParking plein, attente\n",idBariere);
141                 ecrireLog(buff);
142 
143                 // Afficher la requête dans l'interface
144 
145                 AfficherRequete((TypeBarriere) (idBariere+1), voitureBarriere.type,
                    time(NULL));
146                 // Attendre la sortie d'une nouvelle voiture
147 
148                 struct sembuf reserverPorte = {(short unsigned int) idBariere, -1,0};
149                 MY_SA_RESTART(semop(semOuvrirPortes,&reserverPorte, 1));
150 
151                 //struct sembuf libererPorte = {(short unsigned int) idBariere, 1,0};
152                 semctl (semOuvrirPortes, idBariere, SETVAL, 0);
153                 Effacer((TypeZone)(REQUETE_R1 + idBariere));
154 
155 
156             }
157             else
158             {
159                 semop(semVoituresParking,&liberer, 1);
160             }
161             voitureBarriere.arrivee = time(NULL);
162 
163             sprintf(buff,"\t[[Entree n°%d]]\tArrivée de la voiture n°%d\n",idBariere,
                parking->voituresEnAttente[idBariere].numero);
164             ecrireLog(buff);
165 
166             MY_SA_RESTART(semop(semVoituresParking,&reserver, 1));
167             parking->voituresEnAttente[idBariere].type      = AUCUN;
168             parking->voituresEnAttente[idBariere].numero    = -1;
169             parking->voituresEnAttente[idBariere].arrivee   = 0;
170             parking->placesLibres --;
171             pid_t pid = GarerVoiture((TypeBarriere) (idBariere+1));
172             if (pid > 0)
173             {
174                 pidGarage.insert(std::pair<pid_t,Voiture>(pid, voitureBariere));
175             }
176             semop(semVoituresParking,&liberer, 1);
177 
178 
179 
```



```
180             // Attente d'une seconde pour laisser passer la voiture
181             while(sleep(TEMPO) != 0);
182 
183 
184         }
185     }
186 
187     static void EntreePhaseDestruction()
188     {
189         sprintf(buff,"\t[[Entree n°%d]]\tDestruction\n",idBariere);
190         ecrireLog(buff);
191 
192         // Désactiver ChildHandler pour eviter qu'il se déclence
193 
194         struct sigaction ignorer;
195         ignorer.sa_handler = SIG_IGN;
196         sigemptyset(&ignorer.sa_mask);
197         ignorer.sa_flags = 0;
198 
199         sigaction(SIGCHLD,&ignorer,NULL);
200 
201         for (std::pair<const int, Voiture> couple : pidGarage)
202         {
203             // Dire aux process fils de se terminer
204             kill(couple.first, SIGUSR2);
205             MY_SA_RESTART(waitpid(couple.first, NULL, 0));
206         }
207         shmdt(parking);
208 
209         exit(0);
210     }
211 
212 
213     static void EnregisterHandlers()
214     {
215         struct sigaction QuitterSigaction;
216         QuitterSigaction.sa_handler = &QuitterHandler;
217         sigaction(SIGUSR2,&QuitterSigaction,NULL);
218 
219         struct sigaction ChildSigaction;
220         ChildSigaction.sa_handler = &ChildHandler;
221         ChildSigaction.sa_flags = SA_RESTART | SA_NOCLDSTOP;
222         sigaction(SIGCHLD,&ChildSigaction,NULL);
223     }
224 
225     static void QuitterHandler(int sig)
226     {
227         sprintf(buff,"\t[[Entree n°%d]]\t\tSIGUSR2 recu\n",idBariere);
228         ecrireLog(buff);
229 
230         EntreePhaseDestruction();
231     }
232 
233     static void ChildHandler(int sig)
234     {
235 
236         int status;
237         pid_t pid;
238         // Récuperer absolument tous les enfants qui ont exit
239         while (( pid = waitpid((pid_t) -1, &status, WNOHANG))>0) {
240             int place = WEXITSTATUS(status);
            Voiture* voiture = &(pidGarage[pid]);
```

```
241
242
243             AfficherPlace(place, voiture->type, voiture->numero, voiture->arrivee);
244             MY_SA_RESTART(semop(semVoituresParking,&reserver, 1));
245             parking->voituresGarees[place-1].type      = voiture->type;
246             parking->voituresGarees[place-1].numero    = voiture->numero;
247             parking->voituresGarees[place-1].arrivee   = voiture->arrivee;
248             semop(semVoituresParking,&liberer, 1);
249
250             sprintf(buff,"\t[[Entree n°%d]]\tVoiture garée en place %d\n",idBariere,
                place);
251             ecrireLog(buff);
252             pidGarage.erase(pid);
253
254         }
255     }
256
```