```cpp
//---------------------- Réalisation <Sortie> --------------------------

////////////////////////////////////////////////////////////////  INCLUDE
//-------------------------------------------------------- Include système
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.h>
#include <time.h>
#include <signal.h>
#include <stdlib.h>
#include <vector>
#include <fcntl.h>
#include <time.h>
#include <errno.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <map>
#include <iostream>
//------------------------------------------------------ Include personnel
#include "libs/Outils.h"
#include "libs/Heure.h"
#include "Mere.h"
#include "Sortie.h"
#include "Clavier.h"
#include "Voiture.h"
////////////////////////////////////////////////////////////////  PRIVE
//---------------------------------------------------------------- Constantes

//------------------------------------------------------------------------ Types

//------------------------------------------------------ Variables statiques

static int idMemoirePartagee;
static int idFileVoitureSortie;
static void * memPartagee;
static int idSemMemPartagee;
static int idSemEntree;
static std::map<pid_t,int> pidSortie;
static char buff[60];

//------------------------------------------------------ Fonctions privées
static void finSortieVoiture(int numSignal)
// Mode d'emploi :
//
{
    int status;
    pid_t pidFille = wait(&status);
    unsigned int numS=WEXITSTATUS(status);

    struct sembuf opP;
    opP.sem_num=0;
    opP.sem_op=-1;
    opP.sem_flg=0;

    struct sembuf opV;
    opV.sem_num=0;
    opV.sem_op=1;
    opV.sem_flg=0;

```



```cpp
    while(semop(idSemMemPartagee, &opP, 1) ==-1 && errno==EINTR);
    Voiture uneVoiture = ((VoituresParking*)memPartagee)->voituresGarees[numS-1];
    ((VoituresParking*)memPartagee)->placesLibres++;
    semop(idSemMemPartagee, &opV, 1);
    Effacer((TypeZone)numS);
    Effacer(MESSAGE);
    Afficher(MESSAGE, "Une voiture vient de sortir");
    AfficherSortie(uneVoiture.type,uneVoiture.numero,uneVoiture.arrivee,time(NULL));

    while(semop(idSemMemPartagee, &opP, 1) ==-1 && errno==EINTR);
    Voiture * lesVoitures = ((VoituresParking*)memPartagee)->voituresEnAttente;

    char buff[50];
    sprintf(buff,"\t[[Sortie]]\tPas de file ouverte\n");
    if(lesVoitures[FILE_PROF_BP].type==PROF && lesVoitures[FILE_GB].type!=PROF)
    {
        opV.sem_num=FILE_PROF_BP;
        semop(idSemEntree,&opV, 1);


        sprintf(buff,"\t[[Sortie]]\tOuvrir FILE_PROF_BP\n");

    }
    else if(lesVoitures[FILE_PROF_BP].type==PROF && lesVoitures[FILE_GB].type==PROF)
    {
        if(lesVoitures[FILE_PROF_BP].arrivee <= lesVoitures[FILE_GB].arrivee)
        {
            opV.sem_num=FILE_PROF_BP;
            semop(idSemEntree,&opV, 1);
            sprintf(buff,"\t[[Sortie]]\tOuvrir FILE_PROF_BP\n");
        }
        else
        {
            opV.sem_num=FILE_GB;
            semop(idSemEntree,&opV, 1);
            sprintf(buff,"\t[[Sortie]]\tOuvrir FILE_GB\n");
        }
    }
    else if(lesVoitures[FILE_PROF_BP].type!=PROF && lesVoitures[FILE_GB].type==PROF)
    {
        opV.sem_num=FILE_GB;
        semop(idSemEntree,&opV, 1);
        sprintf(buff,"\t[[Sortie]]\tOuvrir FILE_GB\n");
    }
    else if(lesVoitures[FILE_AUTRE_BP].type==AUTRE && lesVoitures[FILE_GB].type==
AUTRE)
    {
        if(lesVoitures[FILE_AUTRE_BP].arrivee <= lesVoitures[FILE_GB].arrivee)
        {
            opV.sem_num=FILE_AUTRE_BP;
            semop(idSemEntree,&opV, 1);
            sprintf(buff,"\t[[Sortie]]\tOuvrir FILE_AUTRE_BP\n");
        }
        else
        {
            opV.sem_num=FILE_GB;
            semop(idSemEntree,&opV, 1);
            sprintf(buff,"\t[[Sortie]]\tOuvrir FILE_GB\n");
        }
    }
    else if(lesVoitures[FILE_AUTRE_BP].type==AUTRE && lesVoitures[FILE_GB].type==
```

```
    AUCUN)
    {
        opV.sem_num=FILE_AUTRE_BP;
        semop(idSemEntree,&opV, 1);
        sprintf(buff,"\t[[Sortie]]\tOuvrir FILE_AUTRE_BP\n");
    }
    else if(lesVoitures[FILE_AUTRE_BP].type==AUCUN && lesVoitures[FILE_GB].type==
    AUTRE)
    {
        opV.sem_num=FILE_GB;
        sprintf(buff,"\t[[Sortie]]\tOuvrir FILE_GB\n");
        semop(idSemEntree,&opV, 1);
    }
    opV.sem_num=0;
    ecrireLog(buff);
    semop(idSemMemPartagee, &opV, 1);
    pidSortie.erase(pidFille);
}
//-----Fin de finSortieVoiture


static void finTache(int numSignal)
// Mode d'emploi :
//
{
    for (std::pair<const int, int> children : pidSortie)
    {
        kill(children.first, SIGUSR2);
    }
    shmdt(memPartagee);
    exit(0);
}//-----Fin de finTache


static void Initialisation(int idMP, int idFVS, int idSemMP, int idSemEnt)
{
    sprintf(buff,"\t[[Sortie]]\tInitialisation\n");
    ecrireLog(buff);

    struct sigaction handlerUSR2;
    handlerUSR2.sa_handler=finTache;
    sigemptyset(&handlerUSR2.sa_mask);
    handlerUSR2.sa_flags=0;
    sigaction(SIGUSR2, &handlerUSR2, NULL);

    struct sigaction handlerCHLD;
    handlerCHLD.sa_handler=finSortieVoiture;
    sigemptyset(&handlerCHLD.sa_mask);
    handlerCHLD.sa_flags=0;
    sigaction(SIGCHLD, &handlerCHLD, NULL);

    idMemoirePartagee=idMP;
    idFileVoitureSortie=idFVS;
    idSemMemPartagee=idSemMP;
    idSemEntree=idSemEnt;
    memPartagee=shmat(idMemoirePartagee,NULL,0);

}

static void Moteur()
{
```



```
    msgInt numS = {0};
    MY_SA_RESTART(msgrcv(idFileVoitureSortie, (void *) &numS, sizeof(msgInt),0,0));
    pid_t pid = SortirVoiture(numS.numero);
    if (pid > 0){
        pidSortie.insert(std::pair<pid_t,int>(pid, numS.numero));
    }
}

////////////////////////////////////////////////////////////////  PUBLIC
//---------------------------------------------------- Fonctions publiques
void Sortie(int idMP, int idFVS, int idSemMP, int idSemEnt)
{
    Initialisation(idMP, idFVS, idSemMP, idSemEnt);

    for(;;)
    {
        Moteur();
    }
}
//-----Fin de Sortie
```