

TP Multitache

Gestion d'un Parking Automobile

Document de Réalisation

Listing du code source

ConfigParking.h
Mere.h
Mere.cpp
Entree.h
Entree.cpp
Sortie.h
Sortie.cpp
Clavier.h
Clavier.cpp

Makefile

```

/*****
                                ConfigParking - description
                                -----
    début                        :
    copyright                    : (C) 2014 par Mehdi Kitane
*****/

//----- Interface de la tâche <Clavier> (fichier Clavier.h) -----
#if ! defined ( ConfigParking_H )
#define ConfigParking_H

//-----
// Rôle du module ConfigParking
// Ce module regroupe les différentes configurations utilisées par
// l'application
//-----

////////////////////// INCLUDE
//----- Interfaces utilisées
#include </public/tp/tp-multitache/0utils.h>

//----- Constantes
#define TERMINALUTILISE XTERM
#define DROITS_CANAL 0660
#define DROITS_MEMOIRE 0660
#define DROITS_SEMAPHORE 0660

//Nom des différents canaux
#define CANAL_PROF_BP "canalProfBP"
#define CANAL_AUTRE_BP "canalAutreBP"
#define CANAL_GB "canalGB"
#define CANAL_SORTIE "canalSortie"

const int NB_SEM = 5; //Nombre de sémaphores elementaires
const int TEMPO = 1 ; //Temporisation d'attente avant l'arrivée d'un nouveau
    véhicule

#define CHEMIN_EXE  "./Parking" //Fichier utilisé pour batir la clé publique
//----- Types

//définition de la structure utilisée pour modéliser une voiture
// Une voiture est représentée par la personne la conduisant, son numero
    d'immatriculation
//et son instant d'arrivée
typedef struct Voiture {
    enum TypeUsager typeUsager;
    int numeroPlaque;
    time_t instantArrivee;
} Voiture;

//définition de la structure utilisée par la mémoire partagée
//Elle contient les 8 places de parking et les 3 requetes
typedef struct memStruct{
    Voiture placesParking[NB_PLACES];

    Voiture requetes[NB_BARRIERES_ENTREE];
} memStruct;

```

```
//definition des différents sémaphores élémentaires utilisés par l'application
typedef enum Semaphore
{
    SemaphoreCompteurPlaces = 0,
    SynchroPorteBPPROF = 1,
    SynchroPorteBPAUTRE = 2,
    SynchroPorteGB = 3,
    MutexMP = 4
}Semaphore;
/////////////////////////////////////// PUBLIC
//----- Fonctions publiques

#endif
```

```

/*****
                                     Mere - description
                                     -----
    début                               :
    copyright                           : (C) 2014 par Mehdi Kitane
*****/

//----- Interface de la tâche <Mere> (fichier Mere.h) -----
#if ! defined ( Mere_H )
#define Mere_H

//-----
// Rôle de la tâche <Mere>
// Le role de la tache mère est de coordiner la création et la destruction
// de ses processus fils et de mettre en place et detruire les
// differents ressources systemes créés
//-----

////////////////////// INCLUDE
//----- Interfaces utilisées

//----- Constantes

//----- Types

////////////////////// PUBLIC
//----- Fonctions publiques
int main();
//Mode d'emploi
// Processus principal Mere

#endif // Mere_H
```

```

/*****
                                Mere - description
                                -----
    début                        :
    copyright                    : (C) 2014 par Mehdi Kitane
*****/

//----- Réalisation de la tâche <Mere> (fichier Mere.cpp) ---

//////////////////////////////////// INCLUDE
//----- Include système

#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <sys/stat.h>
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
//----- Include personnel
#include "Mere.h"
#include "Clavier.h"
#include "Entree.h"
#include "Sortie.h"
#include "/public/tp/tp-multitache/Heure.h"
//////////////////////////////////// PRIVE
//----- Constantes

//----- Types

//----- Variables statiques

//----- Fonctions privées

//////////////////////////////////// PUBLIC
//----- Fonctions publiques
int main()
// Processus rincipal Mere
{

    int memID;
    int semID;

    //Declaration des differents pids des processus déclarés
    pid_t noClavier;
    pid_t noHeure;
    pid_t noEntreeUn;
    pid_t noEntreeDeux;
    pid_t noEntreeTrois;
    pid_t noSortie;

    //Mise en place du canal de communication entre les entrees et le clavier
    //Creation des differents canaux de communication nommes
    if(mkfifo(CANAL_PROF_BP,DROITS_CANAL) == -1){
        cerr<< "erreur creation du canal entre entree et clavier" << endl;
        return -1 ;
    }

```

```

}
if(mkfifo(CANAL_AUTRE_BP,DROITS_CANAL) == -1){
    cerr<< "erreur creation du canal entre entree et clavier" << endl;
    return -1 ;
}
if(mkfifo(CANAL_GB,DROITS_CANAL) == -1){
    cerr<< "erreur creation du canal entre entree et clavier" << endl;
    return -1 ;
}
if(mkfifo(CANAL_SORTIE,DROITS_CANAL) == -1){
    cerr<< "erreur creation du canal entre sortie et clavier" << endl;
    return -1 ;
}

```

```

//Creation de la memoire partagee
memID = shmget(ftok(CHEMIN_EXE,1), sizeof(memStruct), IPC_CREAT | IPC_EXCL
    | DROITS_MEMOIRE);

```

```

if(memID == -1){
    cerr << "erreur creation memoire Partagee" << endl;
    return -1;
}

```

```

//Initialisation de la memoire partagee
memStruct *a = (memStruct *) shmat(memID, NULL, 0); //attachement
for(int i=0; i<(int)NB_PLACES ; i++){
    a->placesParking[i] = {AUCUN, 0,0};
}
for(int i=0; i<(int)NB_BARRIERES_ENTREE ; i++){
    a->requetes[i] = {AUCUN, 0,0};
}
shmdt(a); //detachement

```

```

//Creation des semaphores
semID = semget(ftok(CHEMIN_EXE,2), NB_SEM , IPC_CREAT | IPC_EXCL |
    DROITS_SEMAPHORE);

```

```

if(semID == -1){
    cerr << "erreur creation semaphore" << endl;
    return -1;
}

```

```

//Initialisation des semaphores
//Initialisation du semaphore compteur de places
semctl(semID,SemaphoreCompteurPlaces,SETVAL,NB_PLACES);
//Initialisation des mutexs et sem synchro
semctl(semID,SynchroPorteBPPROF,SETVAL,0);
semctl(semID,SynchroPorteBPAUTRE,SETVAL,0);
semctl(semID,SynchroPorteGB,SETVAL,0);
semctl(semID,MutexMP,SETVAL,1);

```

```

InitialiserApplication(TERMINALUTILISE);

```

```

noHeure = ActiverHeure();

```

```

//Lancement des differents processus Fils

```

```
if( (noClavier = fork() ) == 0 ){
    /*Code du fils */
    Clavier();
}else if( (noEntreeUn = fork() ) == 0 ){
    /*Code du fils */
    Entree(PROF_BLAISE_PASCAL,memID,semID);

}else if( (noEntreeDeux = fork() ) == 0 ){
    /*Code du fils */
    Entree(AUTRE_BLAISE_PASCAL,memID,semID);

}else if( (noEntreeTrois = fork() ) == 0 ){
    /*Code du fils */
    Entree(ENTREE_GASTON_BERGER,memID,semID);

}else if( (noSortie = fork()) == 0 ){
    /*Code du fils*/
    Sortie(memID,semID);
}else{
    /*Code du pere */

    //On attend la reception de Q du clavier
    //Synchro avec le clavier
    waitpid(noClavier, NULL, 0);

    //Demande de fin avec envoi de SIGUSR2
    kill(noSortie,SIGUSR2);
    kill(noEntreeTrois,SIGUSR2);
    kill(noEntreeDeux,SIGUSR2);
    kill(noEntreeUn,SIGUSR2);
    kill(noHeure, SIGUSR2);

    //attente de fin des processus fils
    waitpid(noSortie,NULL,0);
    waitpid(noEntreeTrois,NULL,0);
    waitpid(noEntreeDeux,NULL,0);
    waitpid(noEntreeUn,NULL,0);
    waitpid(noHeure, NULL,0);

    //fermeture des canaux de communication
    unlink(CANAL_SORTIE);
    unlink(CANAL_GB);
    unlink(CANAL_AUTRE_BP);
    unlink(CANAL_PROF_BP);

    //Suppression memoire partagee
    shmctl(memID, IPC_RMID,0);

    //Suppression du semaphore
    semctl(semID, 0, IPC_RMID, 0);

    TerminerApplication();
    exit(0);
}
```

```
    }  
  
    return 0;  
}
```



```

/*****
                                Entree - description
                                -----
    début                        :
    copyright                    : (C) 2014 par Mehdi Kitane
*****/

//----- Interface de la tâche <Entree> (fichier Entree.h) -----
#if ! defined ( Entree_H )
#define Entree_H

//-----
// Rôle de la tâche <Entree>
// La tâche Entrée permet de gérer les voituriers qui rentrent dans le
// Parking
//-----

////////////////////// INCLUDE
//----- Interfaces utilisées
#include "ConfigParking.h"
//----- Constantes

//----- Types

////////////////////// PUBLIC
//----- Fonctions publiques
void Entree(TypeBarriere parametrage, int pmemID, int psemID);
// Mode d'emploi :
// Processus fils Entree
// Contrat :
//

#endif // Entree_H
```

```

/*****
                                Entree - description
                                -----
    début                        :
    copyright                    : (C) 2014 par Mehdi Kitane
*****/

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

//////////////////////////////////// INCLUDE
//----- Include système

//----- Include personnel
#include "Entree.h"
#include <signal.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <iostream>
#include <map>
#include <stdio.h>
#include <string.h>

#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <errno.h>
//////////////////////////////////// PRIVE
//----- Constantes

//----- Types

//----- Variables statiques
static int descR; //descripteur du canal qui relie le clavier a l'entree
static map<pid_t,Voiture> mapVoiture; //Map qui stocke les voituriers entrain
    de se garer
static int memID;
static int semID;
//----- Fonctions privées
static void destruction(int noSignal);
static void receptionMortVoyeurier(int noSignal);

static void initialisation(TypeBarriere parametrage)
//Mode d'emploi :
// Phase d'initialisation du processus Entree
{
    //Installation du handler destruction
    struct sigaction action;
    action.sa_handler = destruction ;
    sigemptyset(&action.sa_mask);
    action.sa_flags = 0 ;
    sigaction(SIGUSR2,&action,NULL); //armer SIGUSR2 sur destruction;

    //Installation du handler actionFinVoyeurier
    struct sigaction actionFinVoyeurier;

```

```

actionFinVoiturier.sa_handler = receptionMortVoiturier ;
sigemptyset(&actionFinVoiturier.sa_mask);
actionFinVoiturier.sa_flags = 0 ;
sigaction(SIGCHLD,&actionFinVoiturier,NULL); //armer SIGCHLD sur
    actionFinVoiturier;

switch(parametrage)
{
    case(PROF_BLAISE_PASCAL):
        descR = open(CANAL_PROF_BP,0_RDONLY);    //Ouverture Canal
        break;
    case(AUTRE_BLAISE_PASCAL):
        descR = open(CANAL_AUTRE_BP,0_RDONLY);    //Ouverture Canal
        break;
    case(ENTREE_GASTON_BERGER):
        descR = open(CANAL_GB,0_RDONLY);    //Ouverture Canal
        break;
    default:
        break;
}
}

static void moteur(TypeBarriere parametrage)
//Mode d'emploi :
// Phase moteur du processus Entree
{
    Voiture voiture;
    struct sembuf reserver = {MutexMP, -1,0};    //p Operation --> Reservation
    struct sembuf liberer = {MutexMP, 1, 0};    //v Operation --> liberation

    if(read(descR,&voiture,sizeof(Voiture)) > 0){ //lecture canal

        DessinerVoitureBarriere(parametrage,voiture.typeUsager);

        if( semctl(semID,SemaphoreCompteurPlaces,GETVAL,0) <= 0){ //
            Recuperation valeur du semaphore a compte
            //Si il ne reste plus de place
            //On place en liste d'attente !
            AfficherRequete(parametrage, voiture.typeUsager, voiture.
                instantArrivee);

            //On ecrit dans la mémoire partagée que l'on a une requete !

            while(semop(semID,&reserver,1)==-1 && errno==EINTR); //
                Reservation de la memoire partagee

            //Ecrire la voiture sur la mémoire partagée
            memStruct *a = (memStruct *) shmat(memID, NULL, 0) ; //
                attachement
            a->requetes[parametrage-1] = voiture ;
            shmdt(a); //detachement

            semop(semID,&liberer,1); //Liberation de la memoire partagee

```

```

        struct sembuf p0p = {parametrage,-1,0}; //p Operation sur le
            mutex de synchronisation
        while(semop(semID,&p0p,1)==-1 && errno==EINTR);

        Effacer((TypeZone)(ETAT_P8+parametrage));
    }

    //On met a jour le Semaphore compteur de place
    struct sembuf p0p = {SemaphoreCompteurPlaces,-1,0};
    semop(semID,&p0p,1);

    // garage voiture ajout du pid voiturier dans la list
    pid_t voiturier=GarerVoiture(parametrage);
    mapVoiture.insert(pair<pid_t,Voiture>(voiturier,voiture));

    //sleep 1s
    sleep(TEMPO);
}

static void destruction(int noSignal)
//Mode d'emploi :
// Phase de destruction du processus Entree
{
    if(noSignal == SIGUSR2){
        //On masque SIGCHLD avant de kill !
        struct sigaction action;
        action.sa_handler = SIG_IGN ;
        sigemptyset(&action.sa_mask);
        action.sa_flags = 0 ;
        sigaction(SIGCHLD,&action,NULL);

        for(map<pid_t,Voiture>::iterator it=mapVoiture.begin(); it!=
            mapVoiture.end() ; it++){
            kill(it->first,SIGUSR2); //Envoi du signal SIGUSR2 aux voitures en
                train de se garer
        }
        for(map<pid_t,Voiture>::iterator it=mapVoiture.begin(); it!=
            mapVoiture.end() ; it++){
            waitpid(it->first,NULL,0); //Attente de la fin des voitures a
                lesquelles on a envoyé un signal
        }

        close(descR); //fermeture canal
        exit(0);
    }
}

static void receptionMortVoiturier(int noSignal)
//Mode d'emploi
// Handler pour le signal SIGCHLD
{

```

```

if(noSignal == SIGCHLD){
    struct sembuf reserver = {MutexMP, -1,0};    //p Operation -->
        Reservation
    struct sembuf liberer = {MutexMP, 1, 0};    //v Operation -->
        liberation

    int status;
    //Recuperer le fils qui a envoye le SIGCHLD
    pid_t filsFini = wait(&status);

    //Recuperer la bonne voiture qui a lancé le signal
    map<pid_t,Voiture>::iterator itLE = mapVoiture.find(filsFini);
    Voiture v = itLE ->second ;

    //Afficher ses caractéristiques dans l'endroit indique
    AfficherPlace(WEXITSTATUS(status),v.typeUsager,v.numeroPlaque,v.
        instantArrivee);

    while(semop(semID,&reserver,1)==-1 && errno==EINTR); //Reservation de
        la memoire

    //Ecrire la voiture sur la mémoire partagée
    memStruct *a = (memStruct *) shmat(memID, NULL, 0) ; //attachement
    a->placesParking[WEXITSTATUS(status)-1] = v ;
    shmdt(a); //detachement

    semop(semID,&liberer,1); //Liberation de la memoire

    //Supprimer la bonne voiture de la map des voitures en train de
        stationner
    mapVoiture.erase(itLE);
}
}
//////////////////////////////////////////////////////////////////////////////////////////////////////////////// PUBLIC
//----- Fonctions publiques

void Entree(TypeBarriere parametrage,int pmemID, int psemID)
//Mode d'emploi :
//Processus Fils Entree
{
    memID = pmemID; //Récupération de l'identifiant de la mémoire partagée
    semID = psemID; //Récupération de l'identifiant du sémaphore général

    initialisation(parametrage);

    for(;;){

```

```
        moteur(parametrage);  
    }  
}
```

```

/*****
                                Sortie - description
                                -----
    début                        :
    copyright                    : (C) 2014 par Mehdi Kitane
*****/

//----- Interface de la tâche <Sortie> (fichier Sortie.h) -----
#if ! defined ( Sortie_H )
#define Sortie_H

//-----
// Rôle de la tâche <Sortie>
// La tâche Sortie permet de gérer les voituriers qui sortent du Parking
//-----

////////////////////// INCLUDE
//----- Interfaces utilisées

//----- Constantes

//----- Types

////////////////////// PUBLIC
//----- Fonctions publiques
void Sortie(int pmemID, int psemID);
// Mode d'emploi :
// Processus fils Sortie
// Contrat :
//

#endif // Sortie_H
```

```

/*****
                                Sortie - description
                                -----
    début                        :
    copyright                    : (C) 2014 par Mehdi Kitane
*****/

//----- Réalisation de la tâche <Sortie> (fichier Sortie.cpp) ---

////////////////////////////////////// INCLUDE
//----- Include système
#include <signal.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <vector>
#include <algorithm>
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>

#include <errno.h>
//----- Include personnel
#include "ConfigParking.h"
////////////////////////////////////// PRIVE

//----- Constantes

//----- Types

//----- Variables statiques
static vector<pid_t> voituresEnSortie; //Vecteur qui stocke les pid des
//voituriers toujours en train de se garer
static int descR;
static int memID;
static int semID;
//----- Fonctions privées
static bool operator < (enum TypeUsager lhs, enum TypeUsager rhs)
//Mode d'emploi :
// Redefinition de l'opérateur < pour le type TypesUsager
{
    //On inverse l'ordre car ici on veut
    //AUCUN<AUTRE<PROF
    //Or notre enum nous donne AUCUN<PROF<AUTRE
    if(lhs == 2 && rhs == 1) return true;
    if(lhs == 1 && rhs == 2) return false;
    return (int)lhs<(int)rhs;
}
static int gererPriorite(Voiture const &a, Voiture const &b, Voiture const &c)
//Mode d'emploi :
// Permet de gerer les priorités entre prof et autre
// Un prof est toujours prioritaire
// Entre deux usagers de meme type, celui qui est arrivé en premier est
// servi en premier
{
    Voiture requetePrio = a;

```



```
    if(requetePrio.typeUsager < b.typeUsager){
        requetePrio= b;
    }else if(b.typeUsager == requetePrio.typeUsager){
        if(b.instantArrivee < requetePrio.instantArrivee){
            requetePrio= b;
        }
    }

    if(requetePrio.typeUsager < c.typeUsager){
        requetePrio= c;
    }else if(c.typeUsager == requetePrio.typeUsager){
        if(c.instantArrivee < requetePrio.instantArrivee){
            requetePrio= c;
        }
    }

    if(requetePrio.typeUsager == AUCUN){
        return 0;
    }
    if(requetePrio.typeUsager == a.typeUsager && requetePrio.instantArrivee ==
        a.instantArrivee){
        return 1;
    }
    if(requetePrio.typeUsager == b.typeUsager && requetePrio.instantArrivee ==
        b.instantArrivee){
        return 2;
    }
    if(requetePrio.typeUsager == c.typeUsager && requetePrio.instantArrivee ==
        c.instantArrivee){
        return 3;
    }

    return 0;
}

static void destruction(int noSignal);
static void receptionMortVoiturier(int noSignal);

static void initialisation()
//Mode d'emploi :
// Phase d'initialisation du processus Sortie
{
    //Installation du handler
    struct sigaction action;
    action.sa_handler = destruction ;
    sigemptyset(&action.sa_mask);
    action.sa_flags = 0 ;
    sigaction(SIGUSR2,&action,NULL); //armer sigusr2 sur destruction;

    struct sigaction actionMortVoiturier;
    actionMortVoiturier.sa_handler = receptionMortVoiturier ;
    sigemptyset(&actionMortVoiturier.sa_mask);
    actionMortVoiturier.sa_flags = 0 ;
    sigaction(SIGCHLD,&actionMortVoiturier,NULL); //armer SigCHLD sur
```

```
        receptionMortVoiturier;

    //Ouverture du canal de la sortie
    descR = open(CANAL_SORTIE, O_RDONLY);
}

static void moteur()
//Mode d'emploi
// Phase moteur du processus Sortie
{
    int numeroPlace;

    //Lecture sur le canal du numero de la place
    if(read(descR, &numeroPlace, sizeof(int)) > 0){
        pid_t voiturierSortie = SortirVoiture(numeroPlace);

        //on stocke les voituriers en sortie pour pouvoir les supprimer si on
        appuie sur Q
        voituriersEnSortie.push_back(voiturierSortie);
    }
}

static void destruction(int noSignal)
//Mode d'emploi :
// Phase de destruction du processus Sortie
{
    if(noSignal == SIGUSR2){
        //On masque SIGCHLD avant de kill !
        struct sigaction action;
        action.sa_handler = SIG_IGN ;
        sigemptyset(&action.sa_mask);
        action.sa_flags = 0 ;
        //armer sigusr2 sur handlerEntree;
        sigaction(SIGCHLD, &action, NULL);

        for(vector<pid_t>::iterator itLE = voituriersEnSortie.begin(); itLE !=
            voituriersEnSortie.end(); itLE++){
            kill(*itLE, SIGUSR2); //Envoi du signal SIGUSR2 aux voitures en
            train de se garer
        }
        for(vector<pid_t>::iterator itLE = voituriersEnSortie.begin(); itLE !=
            voituriersEnSortie.end(); itLE++){
            waitpid(*itLE, NULL, 0); //Attente de la fin des voitures a
            laquelle on a envoyé un signal
        }

        close(descR); //Fermeture du canal
        exit(0);
    }
}

static void receptionMortVoiturier(int noSignal)
//Mode d'emploi
// --Handler pour le signal SIGCHLD
{
    if(noSignal == SIGCHLD){
```

```

struct sembuf reserver = {MutexMP, -1,0};    //p Operation -->
    Reservation
struct sembuf liberer = {MutexMP, 1, 0};    //v Operation -->
    liberation
struct sembuf vOp = {SemaphoreCompteurPlaces,1,0};

int status;
Voiture requetePorteBPPROF;
Voiture requetePorteBPAUTRE;
Voiture requetePorteGB;

pid_t filsFini = wait(&status); //Recuperer le fils qui a envoye le
    SIGCHLD

Effacer((TypeZone)WEXITSTATUS(status)); //Efface la bonne place sur
    l'ecran

while(semop(semID,&reserver,1)==-1 && errno==EINTR); //Reservation de
    la memoire

//Recuperer la voiture et les demandes d'entree sur la memoire
    partagee
memStruct *a = (memStruct *) shmat(memID, NULL, 0) ; //Attachement
Voiture v = a->placesParking[WEXITSTATUS(status)-1] ;
requetePorteBPPROF = a->requetes[(int)PROF_BLAISE_PASCAL -1];
requetePorteBPAUTRE = a->requetes[(int)AUTRE_BLAISE_PASCAL -1];
requetePorteGB = a->requetes[(int)ENTREE_GASTON_BERGER -1];
shmdt(a); //Detachment

semop(semID,&liberer,1); //Liberation de la memoire

AfficherSortie(v.typeUsager,v.numeroPlaque,v.instantArrivee, time(NULL
));

vector<pid_t>::iterator itSorti = std::find(voituriersEnSortie.begin()
    ,voituriersEnSortie.end(),filsFini);
voituriersEnSortie.erase(itSorti); //On efface le voiturier car plus
    besoin de le stocker

semop(semID,&vOp,1); //on effectue l'operation v pour le semaphore à
    compte i.e On incremente le nombre de places

unsigned short int prio = gererPriorite(requetePorteBPPROF,
    requetePorteBPAUTRE, requetePorteGB);

if(prio!=0){
    //Si une requete est en attente, on la satisfait!

```

```

while(semop(semID,&reserver,1)==-1 && errno==EINTR); //Reservation
    de la memoire

memStruct *a = (memStruct *) shmat(memID, NULL, 0) ;
a->requetes[prio-1] = {AUCUN, 0,0}; //On efface la requete de la
    memoire
shmdt(a);

semop(semID,&liberer,1); //Liberation de la memoire


struct sembuf p0p = {prio,1,0};
semop(semID,&p0p,1); //On relache le bon semaphore de
    synchronisation i.e on liberer la bonne porte
}

}

}
//////////////////////////////////// PUBLIC
//----- Fonctions publiques

void Sortie(int pmemID , int psemID)
//Mode d'emploi
// Processus fils Sortie
{
    memID = pmemID;
    semID = psemID;

    initialisation();

    for(;;){
        moteur();
    }
}

```

```

/*****
                                Clavier - description
                                -----
début                          :
copyright                      : (C) 2014 par Mehdi Kitane
*****/

//----- Interface de la tâche <Clavier> (fichier Clavier.h) -----
#if ! defined ( Clavier_H )
#define Clavier_H

//-----
// Rôle de la tâche <Clavier>
// Cette tâche permet de prendre en charge les entrées clavier de
// l'utilisateur et de les traiter et effectuer les actions adéquates.
//-----

////////////////////// INCLUDE
//----- Interfaces utilisées
#include "/public/tp/tp-multitache/Menu.h"
#include "ConfigParking.h"
#include <stdlib.h>
#include <time.h>
//----- Constantes

//----- Types

////////////////////// PUBLIC
//----- Fonctions publiques
void Clavier();
// Mode d'emploi :
//   Processus fils Clavier
// Contrat :
//

void Commande(char code, unsigned int valeur);
// Mode d'emploi :
//   Cette procédure est appelé par Menu() et permet de gerer les entrées
//   clavier
// de l'utilisateur
// Contrat :
//

#endif // Clavier_H

```

```

/*****
                                Clavier - description
                                -----
début                          :
copyright                      : (C) 2014 par Mehdi Kitane
*****/

//----- Réalisation de la tâche <Clavier> (fichier Clavier.cpp) ---

////////////////////////////////////// INCLUDE
//----- Include système

//----- Include personnel
#include "Clavier.h"
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
////////////////////////////////////// PRIVE
//----- Constantes

//----- Types

//----- Variables statiques
static int descProfBPW ;
static int descSortieW;
static int descGBW ;
static int descAutreBPW ;
static int compteurVoiture = 0;

//----- Fonctions privées
static void initialisation()
//Mode d'emploi :
// Phase d'initialisation du processus Clavier
{
    //Ouverture des canaux
    descProfBPW = open(CANAL_PROF_BP,0_WRONLY);
    descAutreBPW = open(CANAL_AUTRE_BP,0_WRONLY);
    descGBW = open(CANAL_GB,0_WRONLY);
    descSortieW = open(CANAL_SORTIE, 0_WRONLY);
}

//---Fin de initialisation

static void destruction()
//Mode d'emploi :
// Phase de destruction du processus Clavier
{
    //Fermeture des canaux
    close(descSortieW);
    close(descGBW);
    close(descAutreBPW);
    close(descProfBPW);

    exit(0);
}

//---Fin de destruction

////////////////////////////////////// PUBLIC
//----- Fonctions publiques

void Clavier()
//Mode d'emploi:

```

```
// Processus Fils Clavier
//
{
    initialisation();

    for(;;){
        Menu();
    }
} //----- fin de Clavier

void Commande(char code, unsigned int valeur)
// Mode d'emploi :
//
{
    if(code == 'Q'){
        destruction();
    }else if(code == 'P'){
        compteurVoiture++;
        if(compteurVoiture > 999){
            compteurVoiture=0; //Reinitialisation du compteur
        }

        Voiture voiture;
        voiture.numeroPlaque = compteurVoiture;
        voiture.instantArrivee = time(NULL);
        voiture.typeUsager = PROF;

        if(valeur == 1){
            //Prof Blaise Pascal
            write(descProfBPW,&voiture,sizeof(voiture)); //Ecriture de la
                voiture arrivée dans le canal
        }
        if(valeur == 2){
            //Prof Gaston Berger
            write(descGBW,&voiture,sizeof(voiture)); //Ecriture de la voiture
                arrivée dans le canal
        }
    }else if(code == 'A'){
        compteurVoiture++;
        if(compteurVoiture > 999){
            compteurVoiture=0; //Reinitialisation du compteur
        }

        Voiture voiture;
        voiture.numeroPlaque = compteurVoiture;
        voiture.instantArrivee = time(NULL);
        voiture.typeUsager = AUTRE;

        if(valeur ==1){
            //Autre Blaise Pascal
            write(descAutreBPW,&voiture,sizeof(voiture)); //Ecriture de la
                voiture arrivée dans le canal
        }
        if(valeur == 2){
            //Autre Gaston Berger
            write(descGBW,&voiture,sizeof(voiture)); //Ecriture de la voiture
                arrivée dans le canal
        }
    }
}
```

```
    }  
    }else if(code == 'S'){  
        write(descSortieW,&valeur,sizeof(int)); //Ecriture du numero de la  
        place a sortir  
    }  
} //--Fin de commande
```



```
COMP = g++
EDL = g++
RM = rm
EXE = Parking
CLEAN = efface
CPPFLAGS = -std=c++11 -c -Wall -Wextra
RMFLAGS = -f
EDLFLAGS = -std=c++11
LIBS = -l tp -l ncurses -l tcl
LIBSPATH = -L/public/tp/tp-multitache
INTERFACE = Mere.h Clavier.h Entree.h Sortie.h ConfigParking.h
REAL = $(INTERFACE:.h=.cpp)
OBJ = $(INTERFACE:.h=.o)

.PHONY : $(CLEAN)

$(EXE) : $(OBJ)
    $(EDL) -o $(EXE) $(OBJ) $(LIBSPATH) $(LIBS)

%.o : %.cpp
    $(COMP) $(CPPFLAGS) $<

$(CLEAN) :
    $(RM) $(RMFLAGS) *.o $(EXE) core
```