INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Rim EL IDRISSI-MOKDAD
Nicolas GRIPONT

Document de Spécification et Conception TP Multitache

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Rim EL IDRISSI-MOKDAD
Nicolas GRIPONT

TP Multitâche

La modélisation de la gestion informatique d'un parking avec 3 barrière d'entrée et une barrière de sortie nous est confiée dans ce TP. Le but est de permettre la gestion des entrées et sorties des voitures avec la contrainte d'un nombre de places limité à 8 ainsi que des priorité d'accès selon le type d'usager (prof ou autre).

Variations entre la conception initiale et la réalisation finale :

Les faiblesses majeures de notre conception concernaient notre définition de la mémoire partagée ainsi que notre compréhension du fonctionnement des boites aux lettres. Nous avons aussi relevé certaines imprécisions concernant la modélisation de la fin des processus. Une lecture approfondie de l'annexe a facilité la compréhension de ce qui été attendu.
Nous allons détailler dans ce qui suit les divergences entre la conception et la réalisation :

Mémoire partagée :
On avait initialement prévu une seule mémoire partagée. Cela essentiellement parce que nous avions une vision floue de ce que nous allions y stocker. De plus, nous n'avions pas pris en compte les requêtes lors de la première conception. Nous avons donc fini par créer deux mémoires partagées : une pour stocker les informations concernant les voitures garées et une pour les informations concernant les requêtes.

Boites aux lettres :
Les messages transitant entre les boites aux lettres n'étaient pas bien définis, nous avions rajouté un attribut n°Voiture qui n'est avéré inutile.

Signaux :
La modélisation de la fin des processus à l'initiative des tâches mères par l'envoie du signal SIGUSR2 figurait dans la conception. Il manquait simplement de modéliser l'envoie des SIGCHLD envoyé par les processus crées par GarerVoiture() et SortirVoiture().

Affichage :
Certains affichages comme DessinerVoitureBarriere() et AfficherPlace() n'avait pas été pris e compte dans la conception.

Définition des différentes structures :

(*X) se réfèrent aux créations des IPCs du schémas de l'initialisation de la mère.

Définition d'un type voiture :

```
typedef struct voiture {
    TypeUsager typeUsager;
    unsigned int numero;
    time_t arrivee;
} Voiture;
```

Cette structure permet de stocker les différentes informations caractérisant une voiture.

`TypeUsager typeUsager` : Stocke le type de l'usager.

`unsigned int numero` : Stocke le numéro de la voiture (plaque minéralogique)

`time_t arrivée` : Permet d'abord de stocker l'instant où la voiture arrive à la barrière puis l'instant où la voiture passe la barrière.

Messages transitant par les boites aux lettres d'entrée (*1):

```
typedef struct messsageDemandeEntree {
    long type;
    Voiture voiture;
} MessageDemandeEntree;
```

Cette structure permet de définir la structure d'un message envoyée dans les boites aux lettres d'entrée.

`long type` : Type du message, champs obligatoire, mis à 1 dans notre cas.

`Voiture voiture` : Element de type Voiture contenant les informations sur la voiture faisant une demande d'entrée.

Messages transitant par la boite aux lettres de sortie (*2):

```
typedef struct messsageDemandeSortie {
    long type;
    int numeroPlace;
} MessageDemandeSortie;
```

Cette structure permet de définir la structure d'un message envoyée dans la boite aux lettres de sortie.

`long type` : Type du message, champs obligatoire, mis à 1 dans notre cas.

`int numeroPlace` : Numéro de la place que le processus Sortie doit libérer.

Mémoire partagée voitures (*3):

```
typedef struct memoirePartageeVoitures {
    Voiture voitures[NB_PLACES];
} MemoirePartageeVoitures;
```

Cette structure permet de définir la mémoire partagée contenant les informations sur les voitures garées.

`Voiture voitures[NB_PLACES]` : Tableau de `NB_PLACES`, dans notre cas 8, éléments de type Voiture. La voiture garée à la place i est stockée dans la case i-1 du tableau. L'attribut `arrivee` de chaque voiture correspond au temps de passage de la barrière de cette voiture.

Mémoire partagée requêtes (*4):

```
typedef struct memoirePartageeRequetes {
    Voiture requetes[NB_BARRIERES_ENTREE];
} MemoirePartageeRequetes;
```

Cette structure permet de définir la mémoire partagée contenant les informations sur les requêtes de demandes d'entrée ne pouvant être traiter (attente de la voiture car le parking est plein).

`Voiture requetes[NB_BARRIERES_ENTREE]` : Tableau de `NB_BARRIERES_ENTREE`, dans notre cas 3, éléments de type Voiture. Chaque case du tableau correspond à la requête d'une des entrées (une case par entrée). L'attribut `arrivee` de chaque voiture émettant une requête correspond au temps d'arrivée à la barrière de cette voiture et donc au temps où elle a émis la requête.

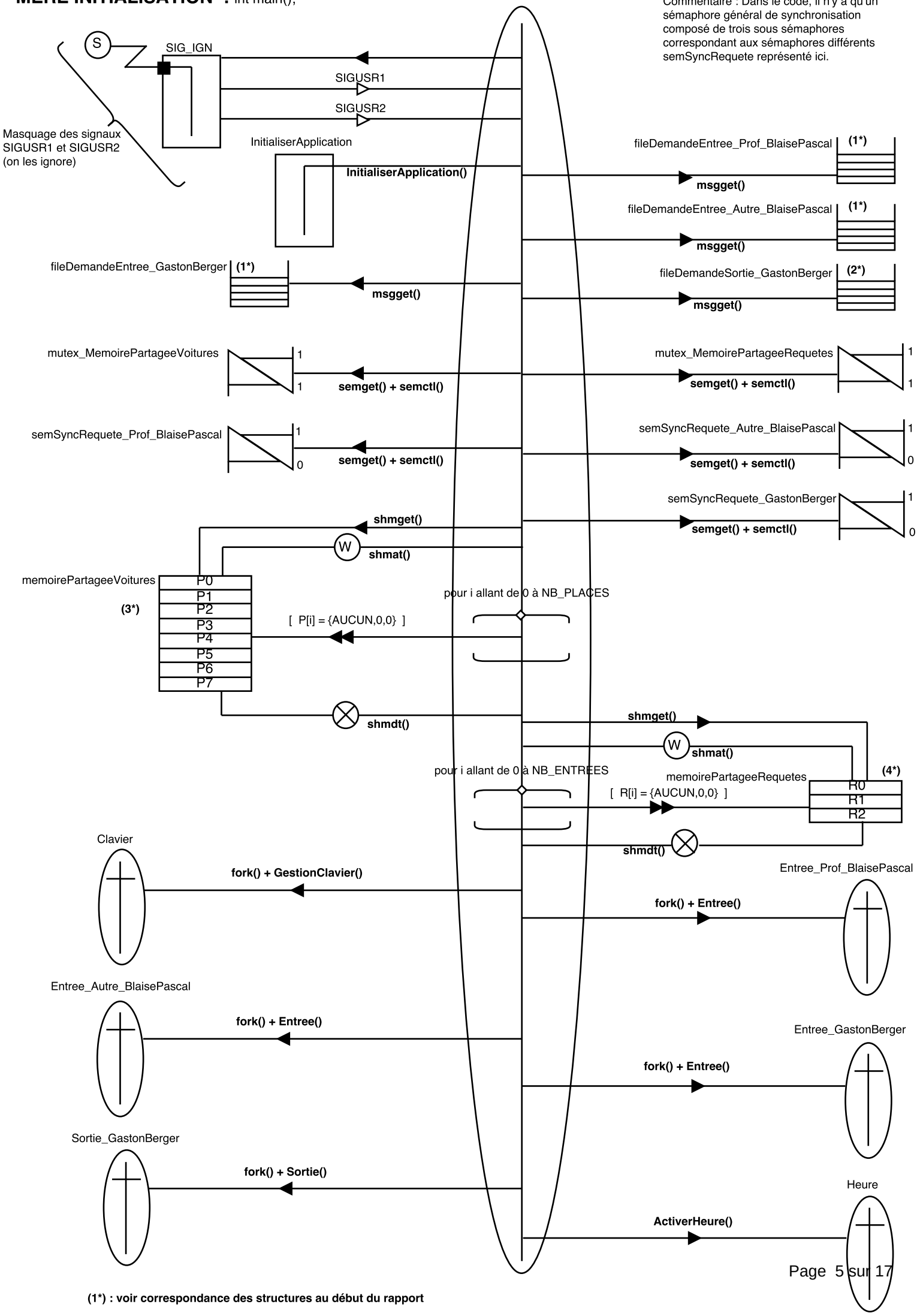
Point sur l'avancé du code :

Le code est terminé. Nous n'avons trouvé aucun bug.

Schémas des différentes tâches:

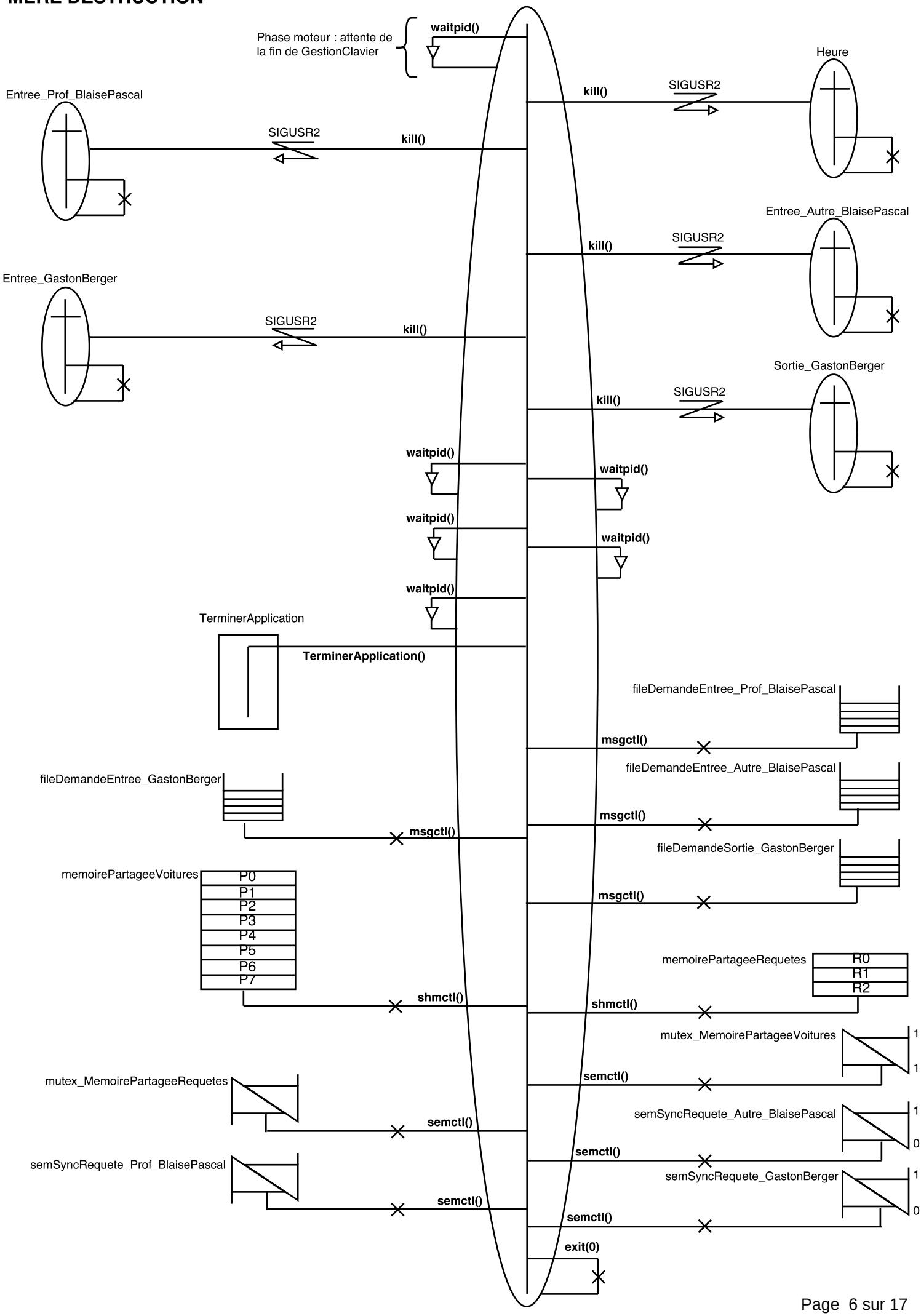
MERE INITIALISATION : int main();

Commentaire : Dans le code, il n'y a qu'un sémaphore général de synchronisation composé de trois sous sémaphores correspondant aux sémaphores différents semSyncRequete représenté ici.

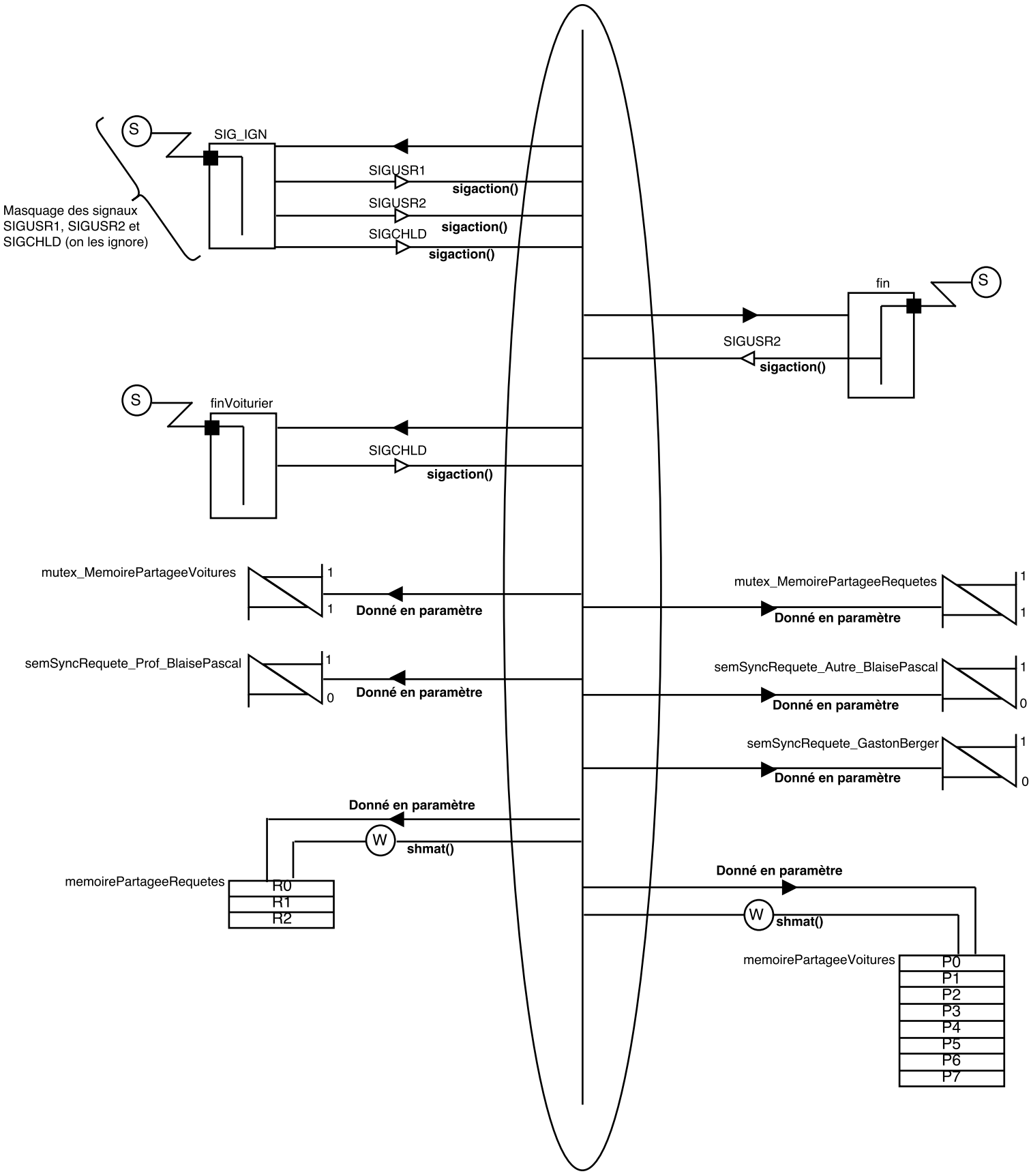


(1*) : voir correspondance des structures au début du rapport

MERE DESTRUCTION

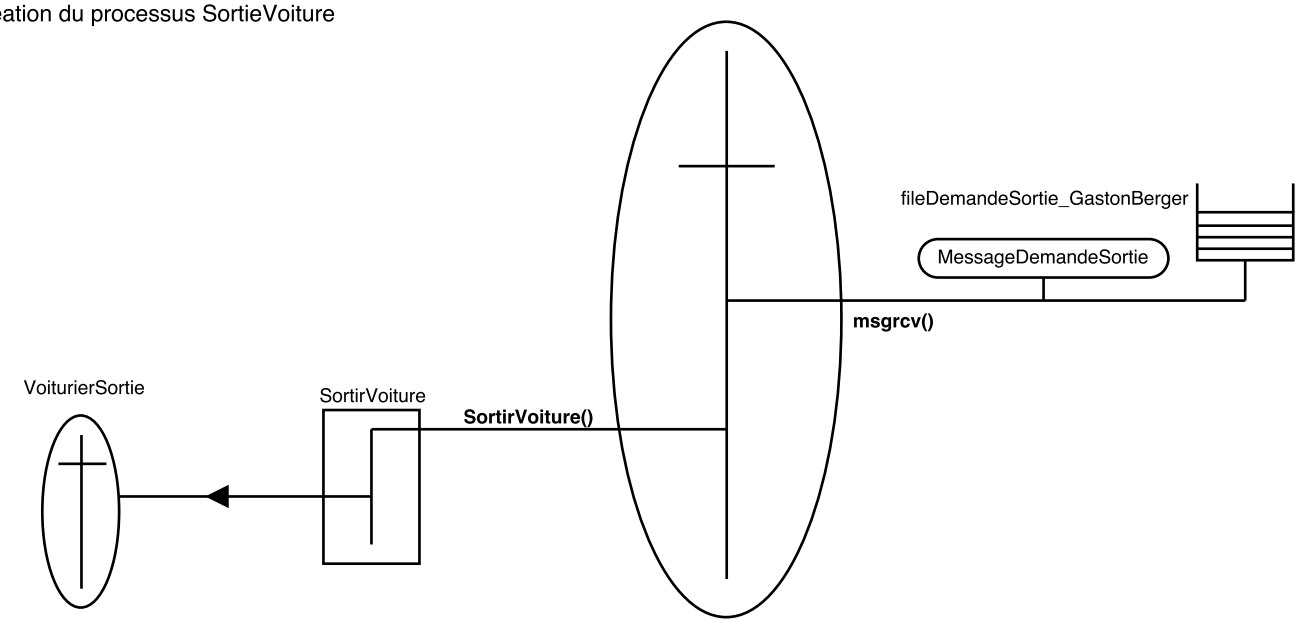


SORTIE INITIALISATION voidSortie(int msgid_BAL,int mutex_MPR,int semSyc_MPR,int shmId_MPR,int mutex_MPV,int shmId_MPV)
msgid_BAL : identifiant de la boîte aux lettres contenant les demandes d'entrées
mutex_MPR : identifiant du mutex de la mémoire partégée des requêtes
semSync_MPR : identifiant du sémaphore général contenant les 3 sémaphores de synchonisation des requêtes.
shmid_MPR : identifiant de la mémoire partagée des requetes
mutex_MPV : identifiant du mutex de la mémoire partagée des voitures
shmid_MPV : identifiant de la mémoire partagée des voitures
Commentaire : Le choix des paramètres nous évite de refaire les appels systèmes permettant d'identifier les différents IPCs.

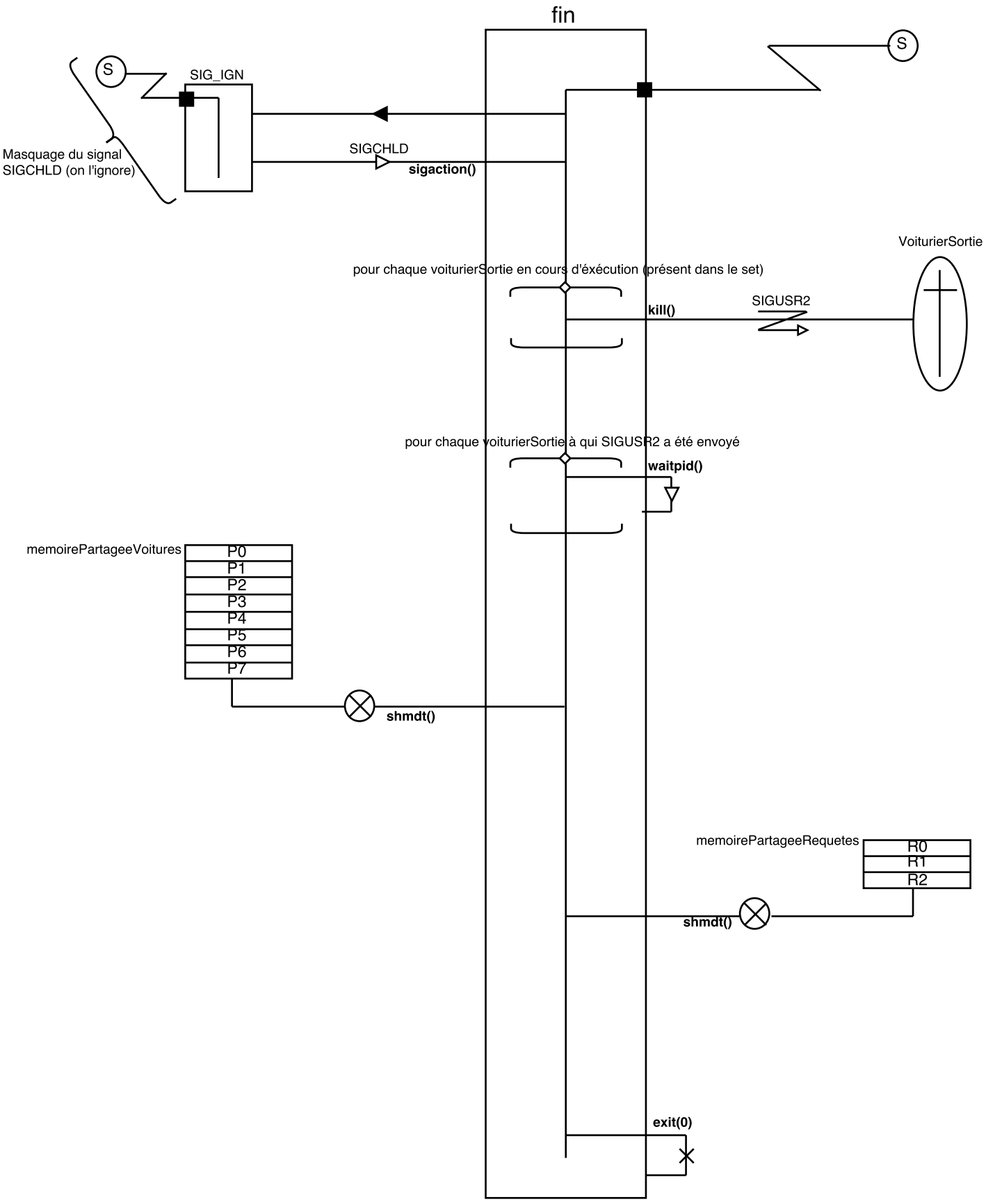


SORTIE MOTEUR

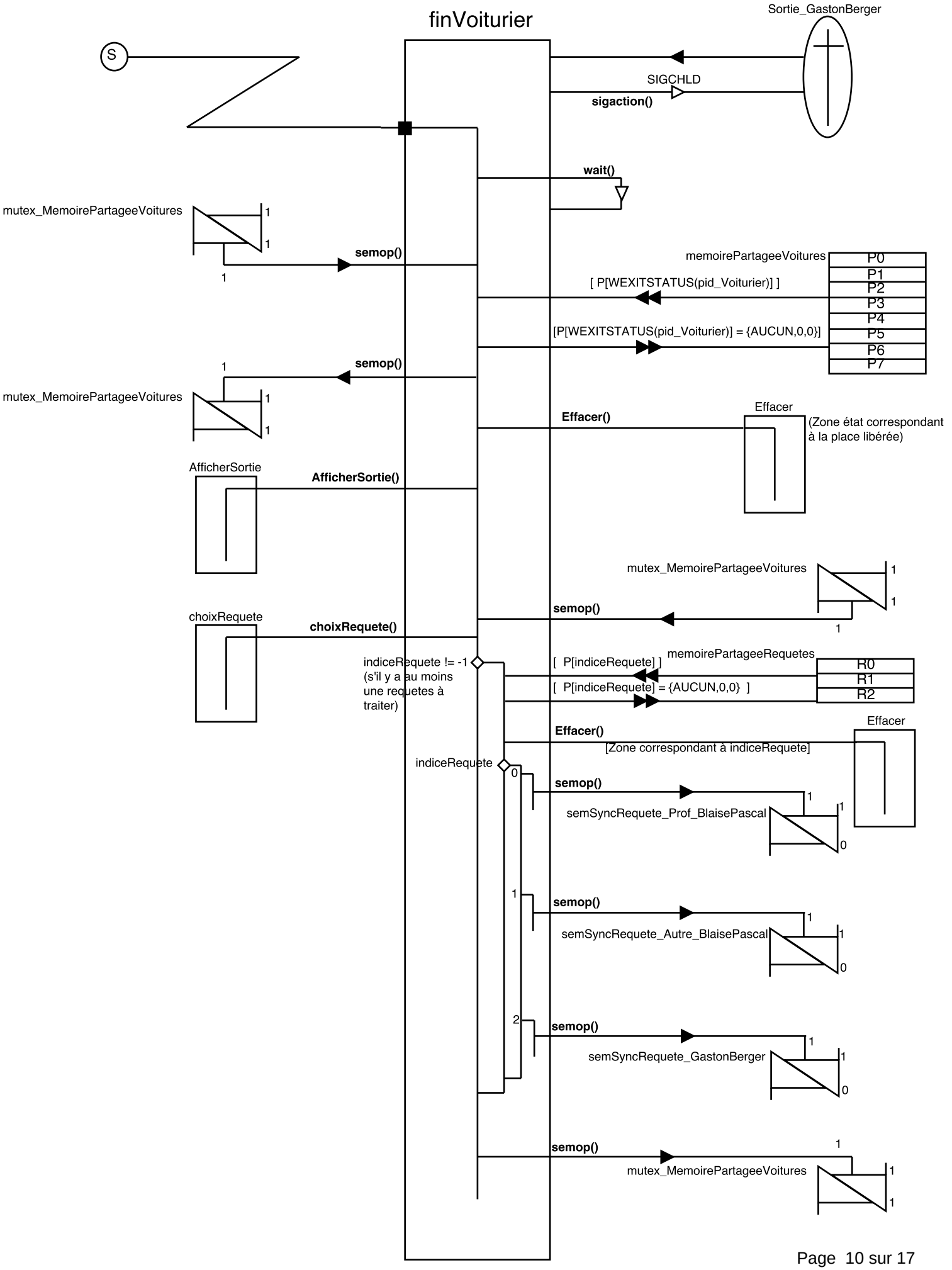
Commentaire : On stocke les pids des voiturierSorties en cours dans un set. On ajout le pid du voturierSorie après la création du processus SortieVoiture



SORTIE DESTRUCTION : static void fin(int numSignal)
numSignal : Numéro du signal reçu
Commentaire : On stocke les pids des voiturierSortie en cours dans un set.

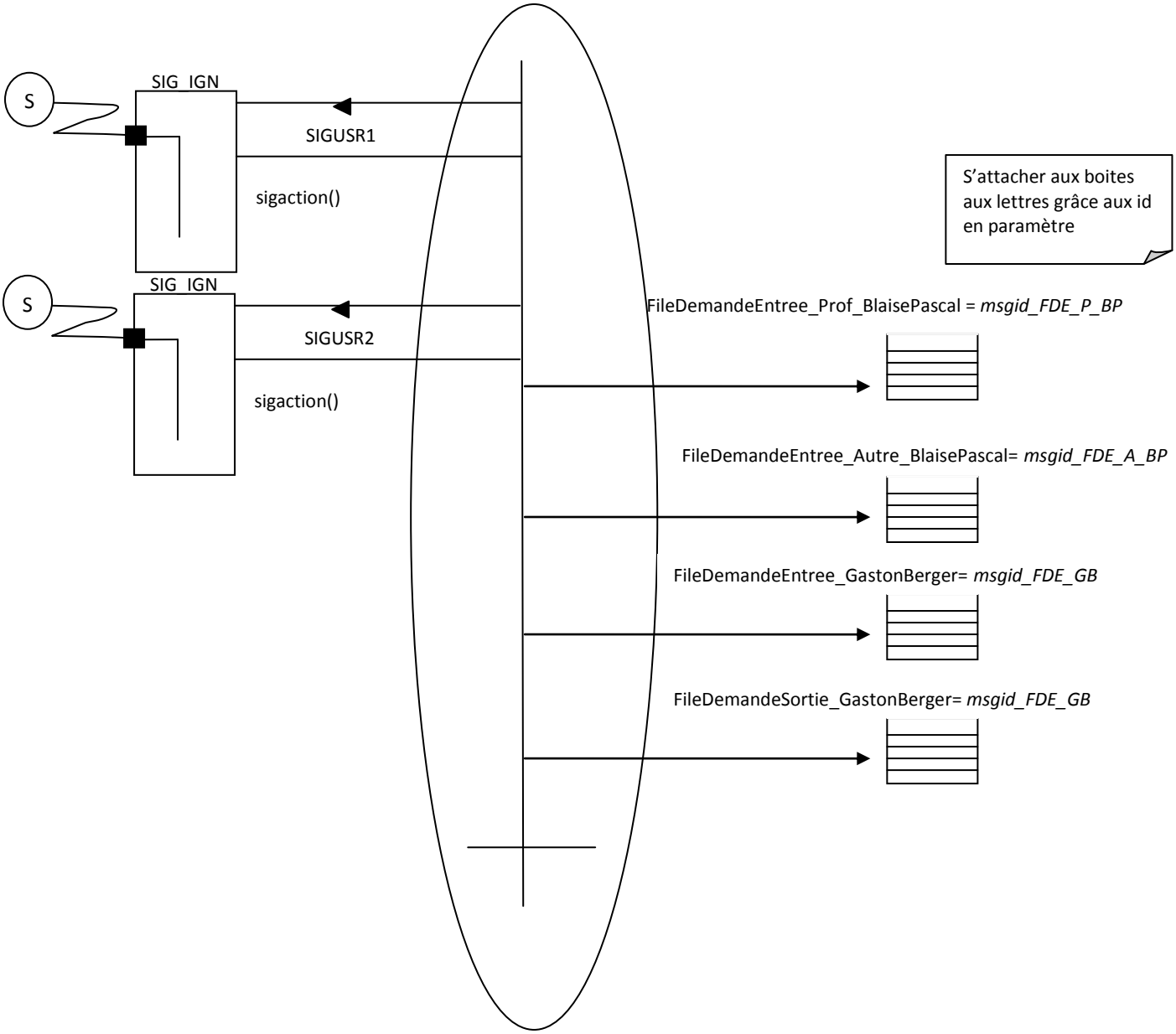


SORTIE FIN_VOITURIER static void finVoiturier(int numSignal)
numSignal : Numéro du signal reçu
Commentaire : Une fois le wait() terminé, on supprime le pid retournée par cette appel du set qui contient les pids des voiturierSortie en cours d'exécution.

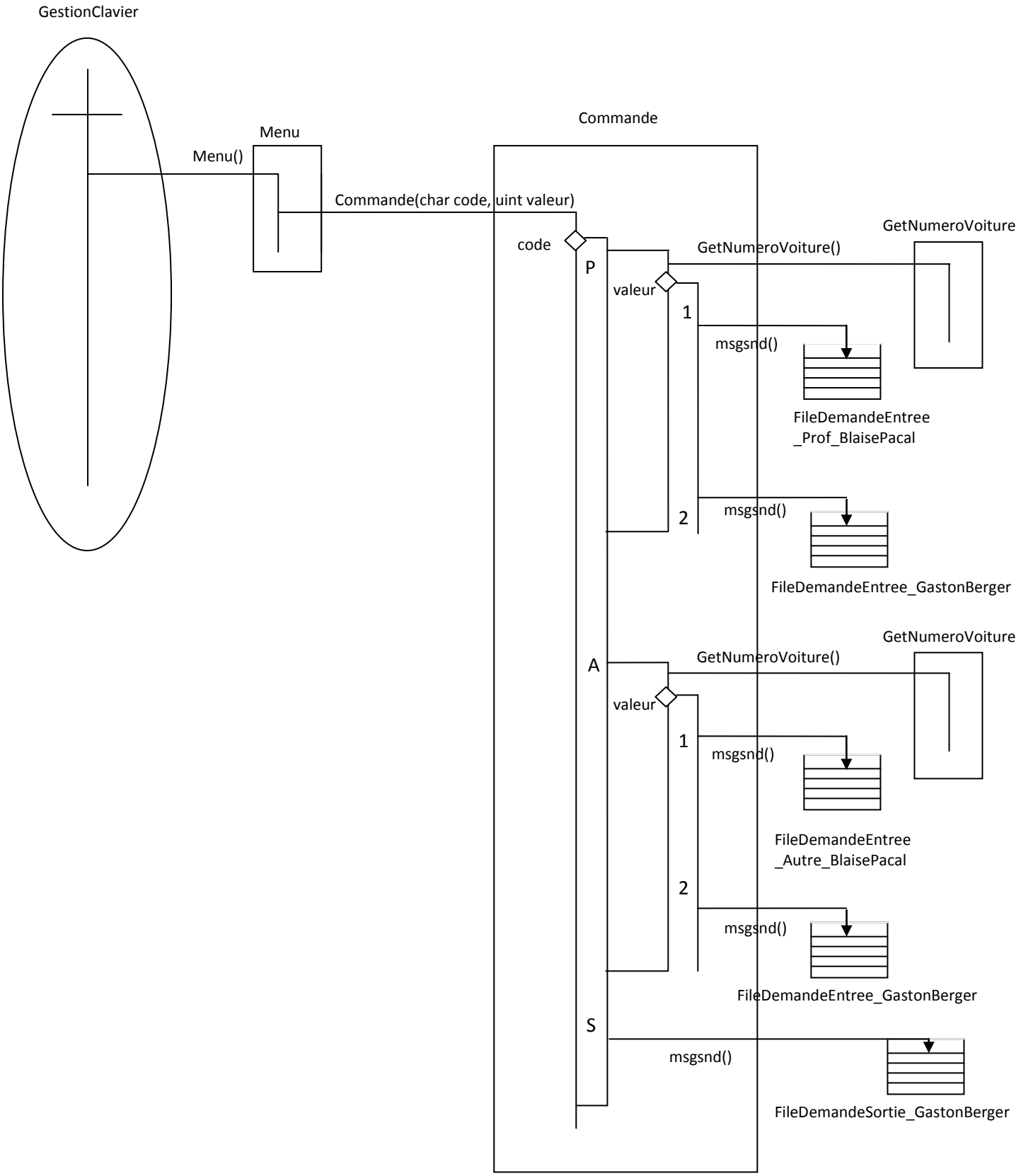


Initialisation de Simulation : *GestionClavier(int msgid_FDE_P_BP, int msgid_FDE_A_BP, int msgid_FDE_GB, int msgid_FDS_GB)*

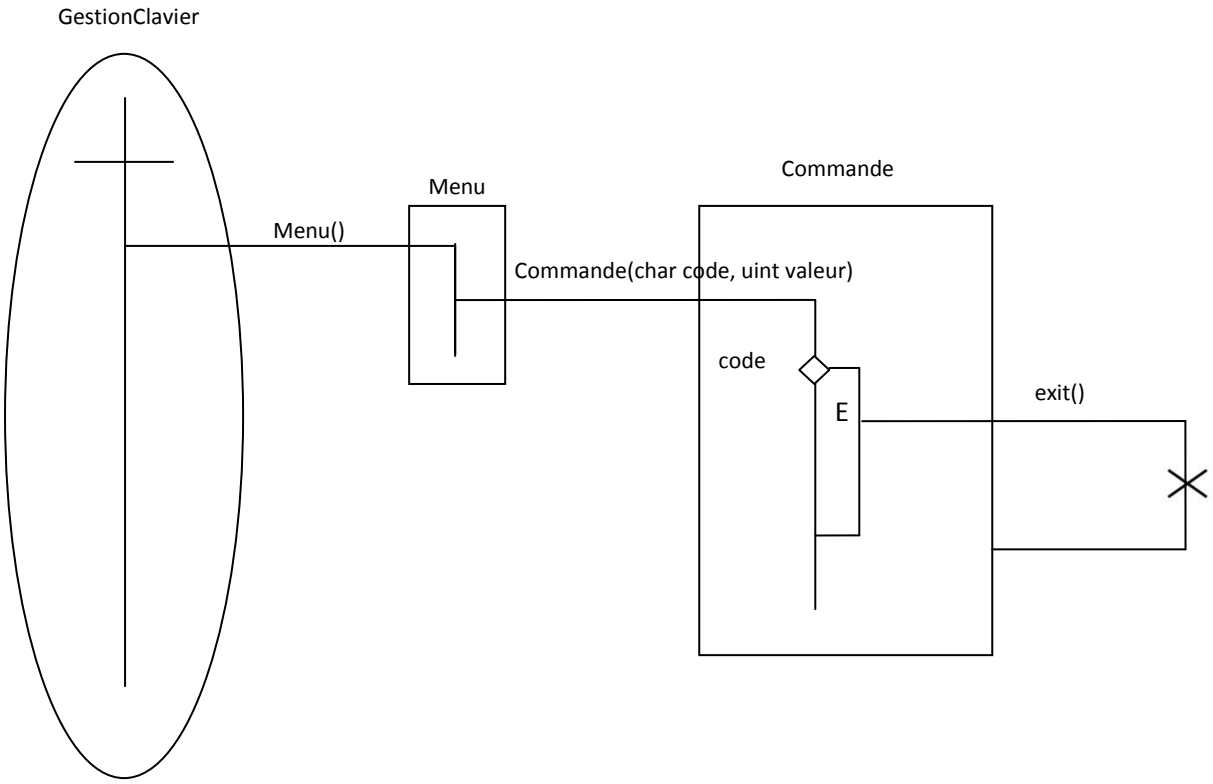
Remarque : le choix de ces paramètres nous évite de refaire les appels systèmes.



Phase moteur de Simulation : *GestionClavier(int msgid_FDE_P_BP, int msgid_FDE_A_BP, int msgid_FDE_GB, int msgid_FDS_GB)*

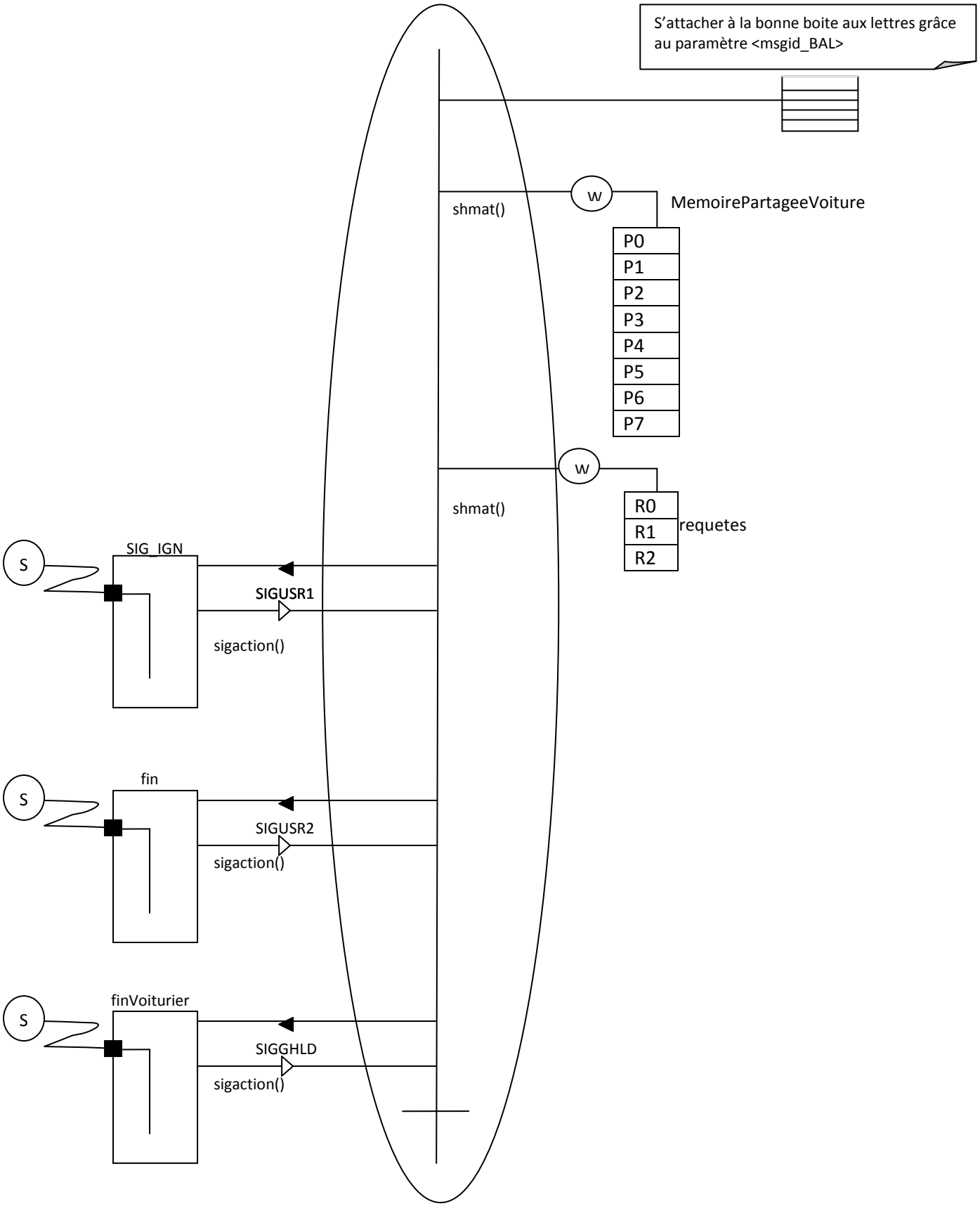


Destruction de Simulation : *GestionClavier(int msgid_FDE_P_BP, int msgid_FDE_A_BP, int msgid_FDE_GB, int msgid_FDS_GB)*

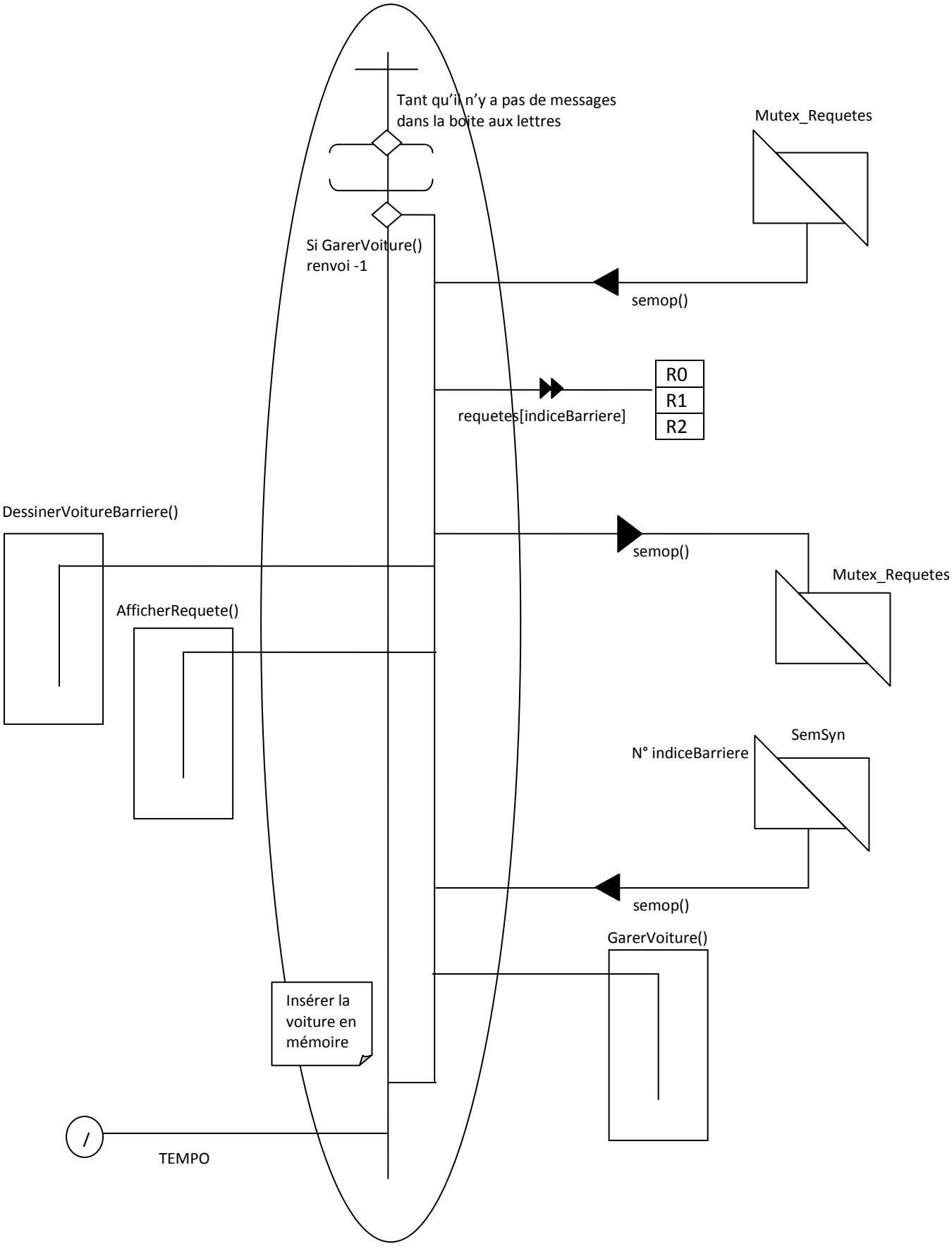


Initialisation de Entree : *Entree(TypeBarriere type, int indiceBarriere, int msgid_BAL, int mutex_R, int semSyc_R, int shmId_R, int mutex_MPV, int shmId_MPV)*

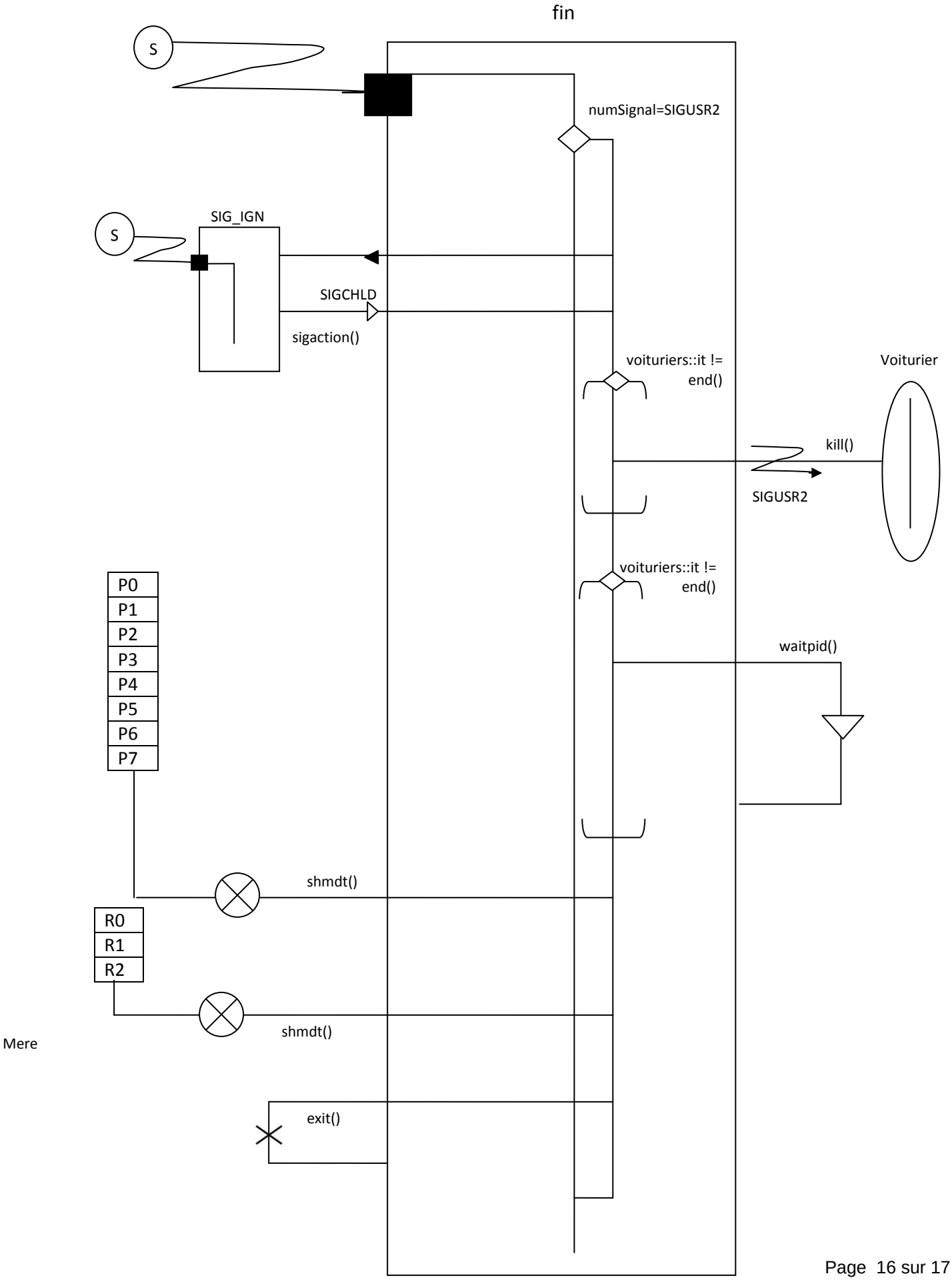
Remarque : le choix de ces paramètres nous évite de refaire les appels systèmes et d'identifier directement la boite aux lettres et le type de barrière adéquate.



Phase moteur de Entree : *Entree(TypeBarriere type, int indiceBarriere, int msgid_BAL, int mutex_R, int semSyc_R, int shmId_R, int mutex_MPV, int shmId_MPV)*



Destruction Entrée : *fin(int numSignal)*



Mere

Destruction du voiturier : *FinVoiturier(int numSignal)*