

TP MULTITACHE :

SPECIFICATION

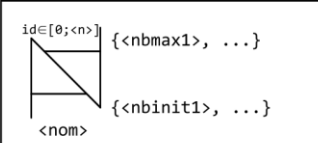
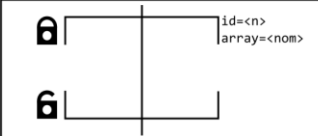
B3330

Paul-Emmanuel SOTIR

Victoire CHAPELLE

La principale différence entre l'architecture générale envisagée et les choix qui ont finalement été pris est que nous avons choisi d'utiliser un sémaphore pour faire attendre les barrières le temps qu'une place se libère et que la tâche de gestion de la sortie libère le sémaphore. Nous avons également pu préciser plus en détail les différents mécanismes de communication entre processus comme par exemple le détail de la mémoire partagée qui manquait à l'architecture générale.

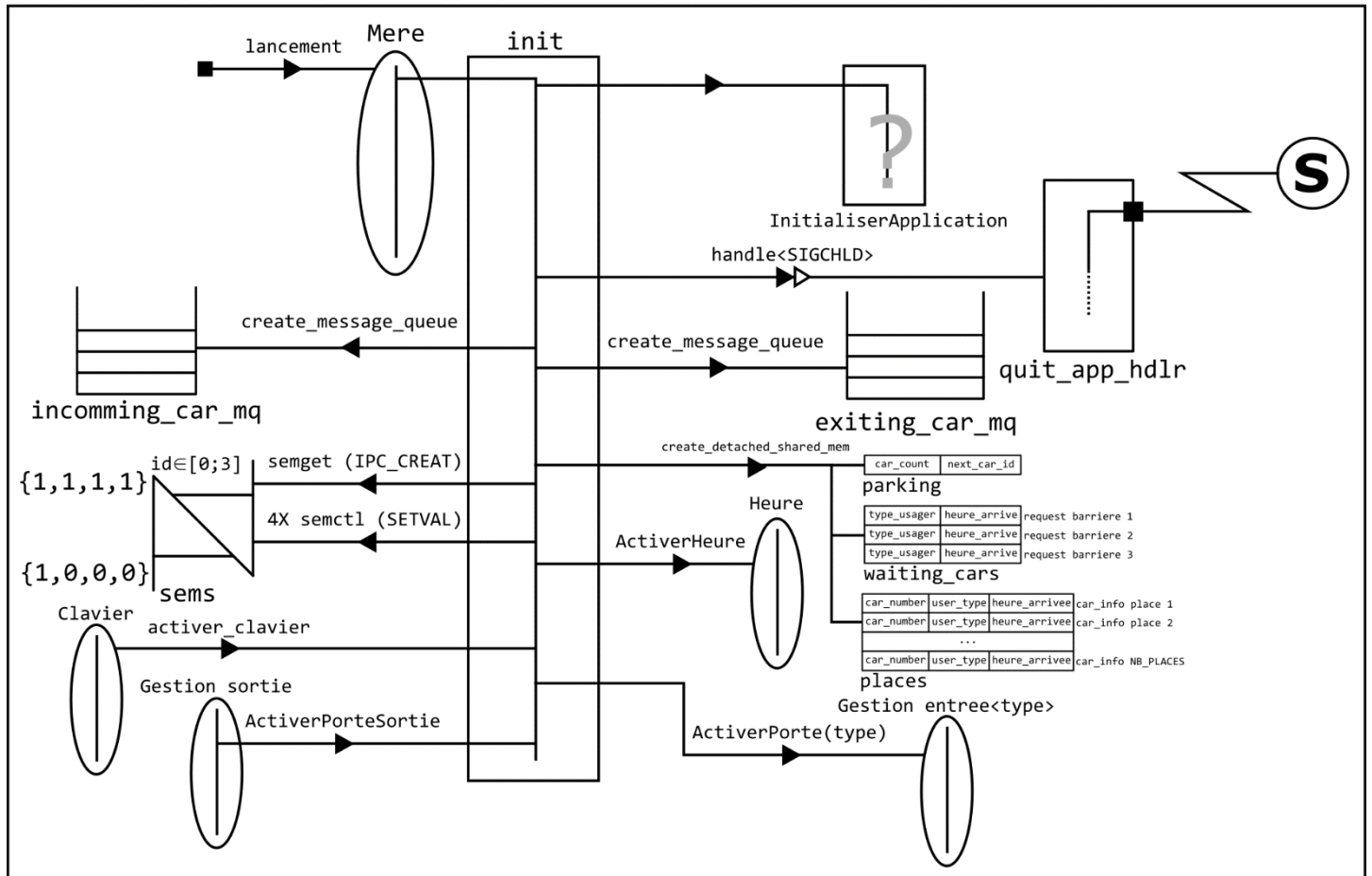
Voici la Légende des desseins de la communication interprocessus des tâches de l'application :

| LEGENDE | |
|---|---|
|  | Symbole représentant un tableau de $\langle n \rangle$ sémaphores ayant pour valeurs initiales $\langle nbinit1 \rangle, \dots, \langle nbinitn \rangle$. Une opération (sempv/lock) sur un sémaphore doit donc préciser un id de sémaphore (e.g. 'id=1'). |
|  | Symbole représentant un appel à la fonction lock (process_utils.h) protégeant du code (ligne entre les deux symboles) avec la sémaphore ayant pour id $\langle n \rangle$ dans le tableau de sémaphores $\langle nom \rangle$. |

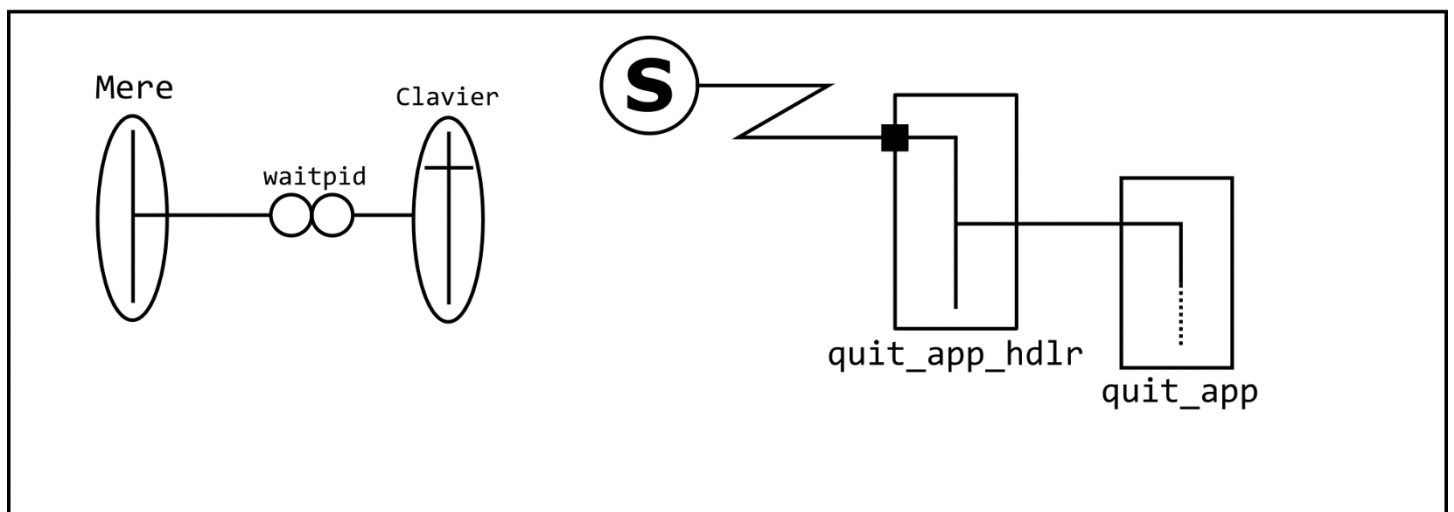
Les appels à la destruction de la tâche en cas d'erreur sont omis par soucis de simplicité.
La tâche de gestion d'entrée est créée 3 fois avec le paramètre $\langle type \rangle$ qui identifie de quelle barrière il s'agit (TypeBarriere).
La sémaphore d'id=0 dans le table de sémaphores est le mutex utilisé pour protéger la mémoire partagée.
Les sémaphores d'id $\in [1;3]$ permettent aux trois tâches de gestion d'entrée d'attendre la libération de place si le parking est plein.

TACHE MERE

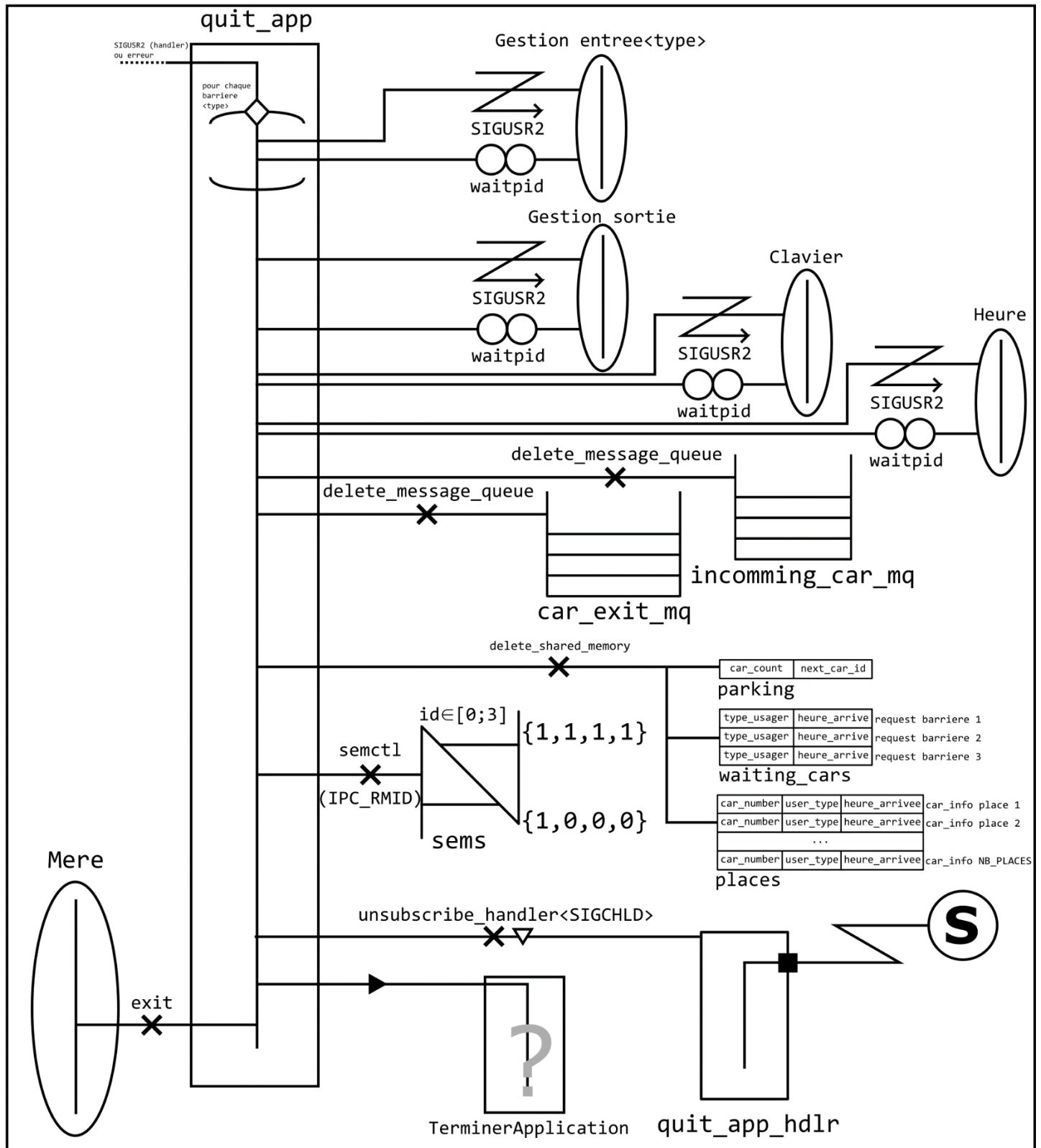
INITIALISATION MERE



PHASE MOTEUR MERE

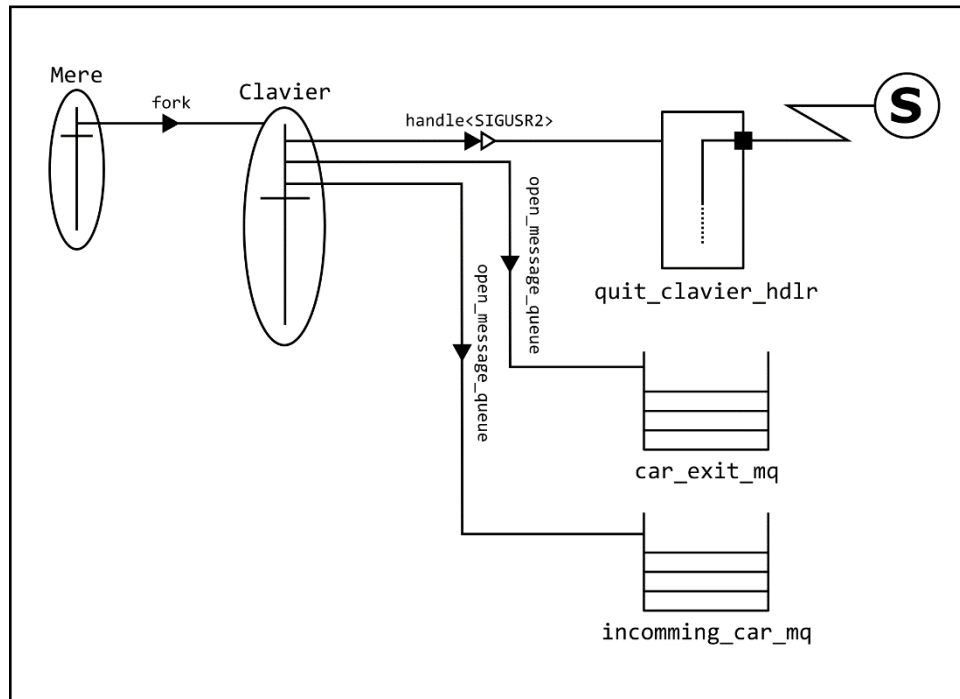


DESTRUCTION MERE

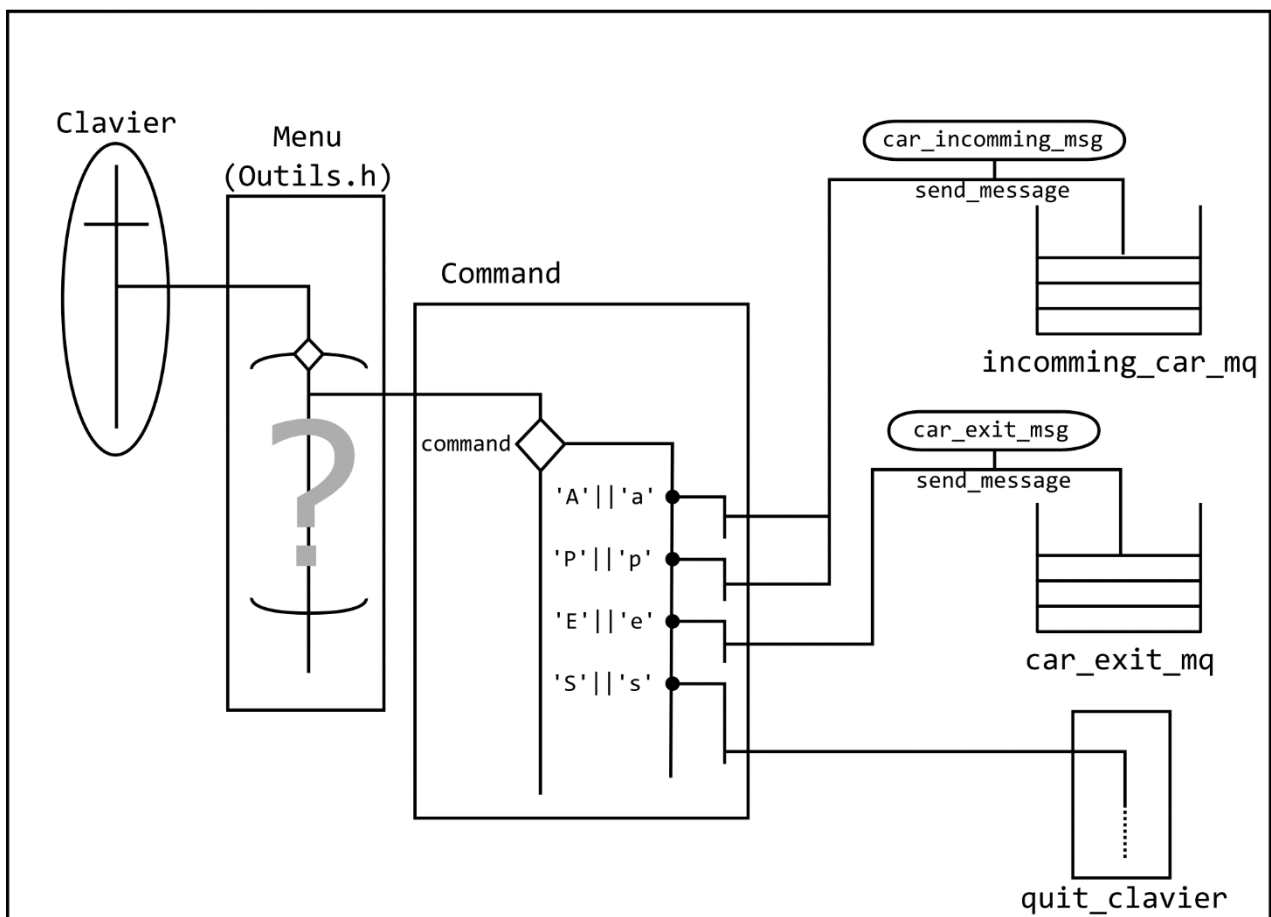


TACHE CLAVIER

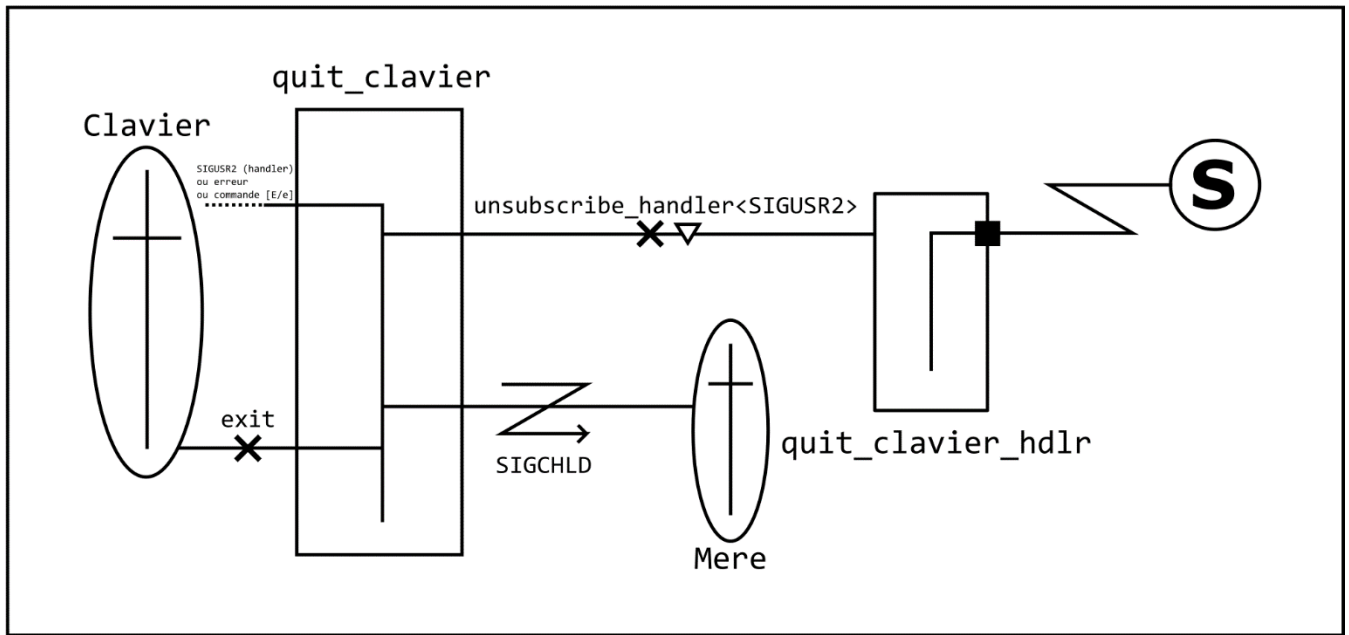
INITIALISATION CLAVIER



PHASE MOTEUR CLAVIER

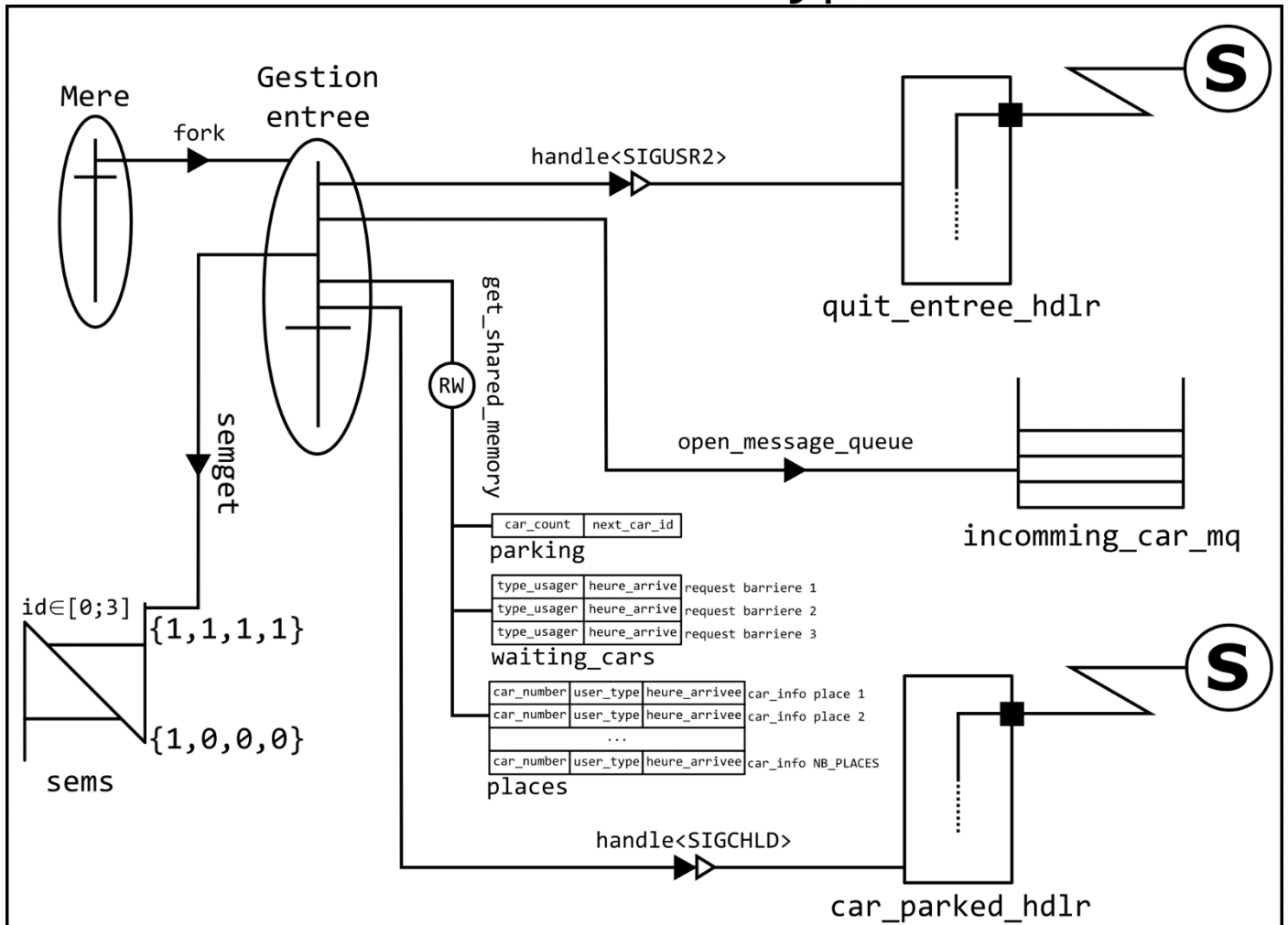


DESTRUCTION CLAVIER



TACHE GESTION ENTREE

INITIALISATION ENTREE <type>



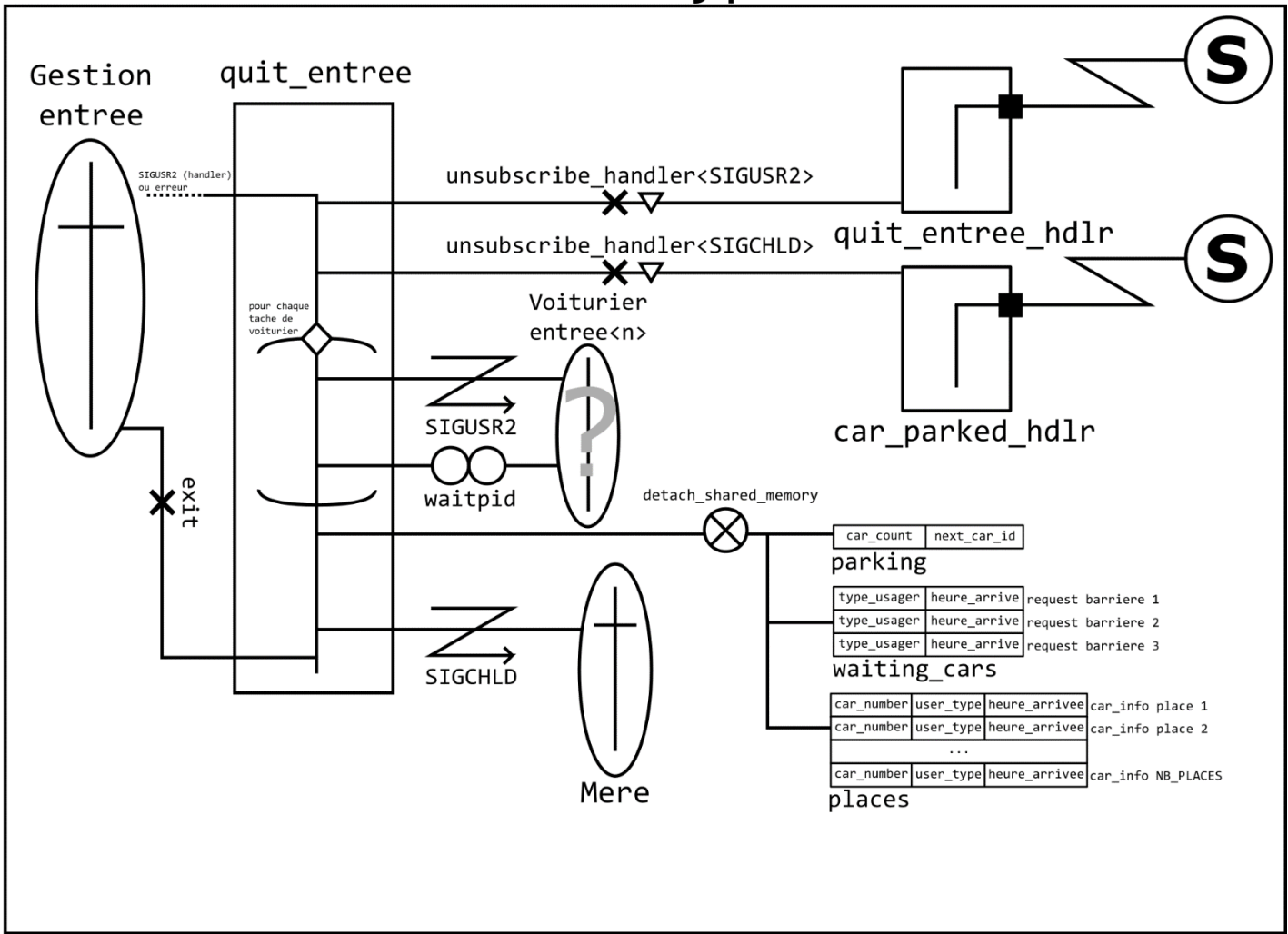
```

sequenceDiagram
    participant G as Gestion entree
    participant DV as DessinerVoitureBarriere
    participant AR as AfficherRequete
    participant GV as GarerVoiture
    participant VE as Voiturier entree
    participant E as Effacer
    participant CP as car_parked_hdlr
    participant AP as AfficherPlace

    G->>DV: read_message
    G->>AR: car_count++  
next_car_id++
    G->>AR: next_car_id
    G->>AR: request[<type>-1]
    G->>AR: request[<type>-1] = { AUCUN, 0 }
    G->>GV: efface la requete
    G->>GV: 1s
    G->>VE: waitpid
    G->>VE: met la 'carInfo' dans la place
    G->>VE: car_info place 1  
car_info place 2  
...  
car_info NB_PLACES
    G->>AP: AfficherPlace

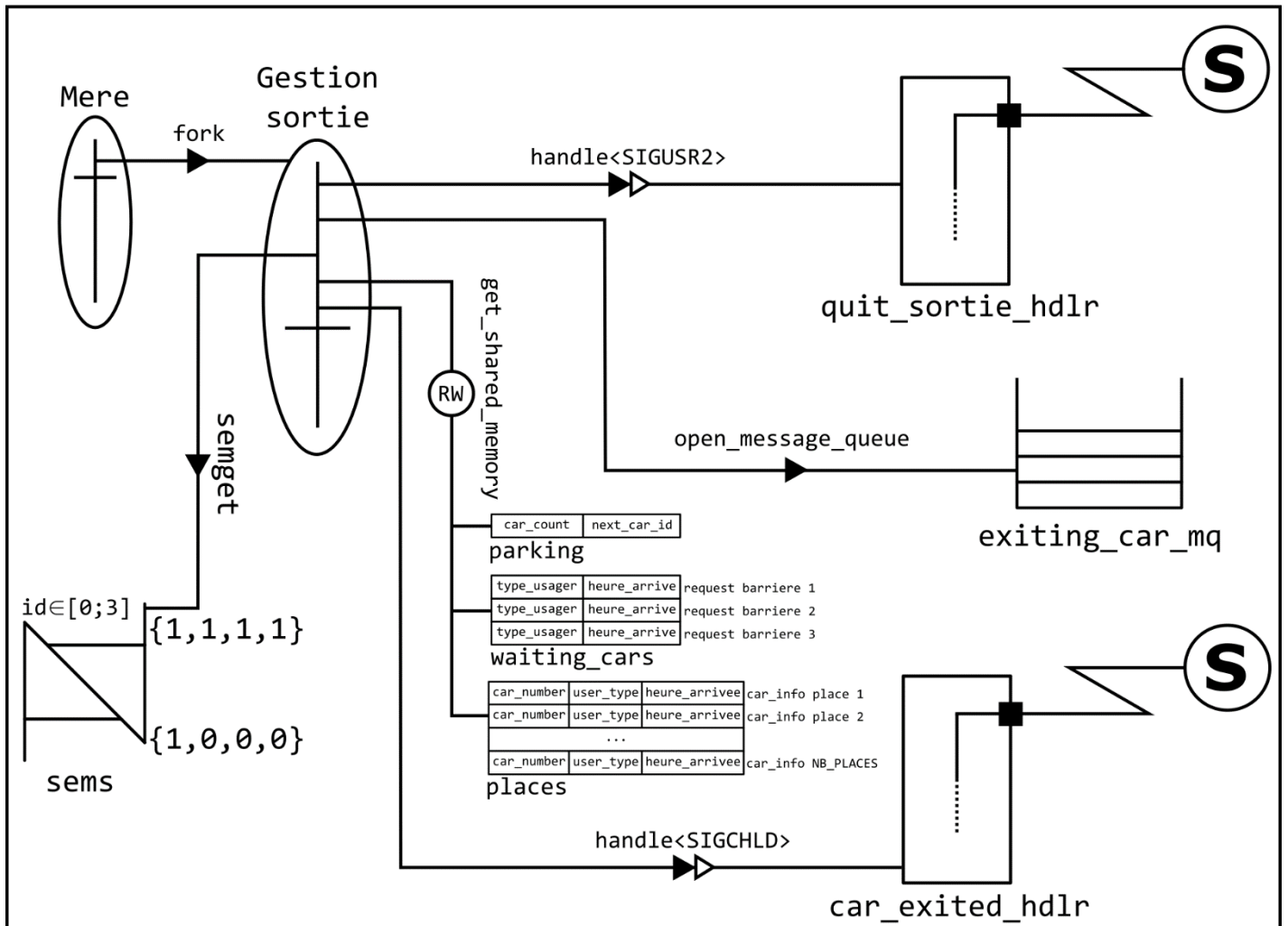
    Note over G: id=0  
array=sems
    Note over G: parking plein?
    Note over G: parking plein?
    Note over G: id=0  
array=sems
    Note over G: id∈[0;3]
    Note over G: {1,1,1,1}
    Note over G: {1,0,0,0}
    Note over G: sempv -1(P) id=<type>  
(attente place libre)
    
```

DESTRUCTION ENTREE <type>

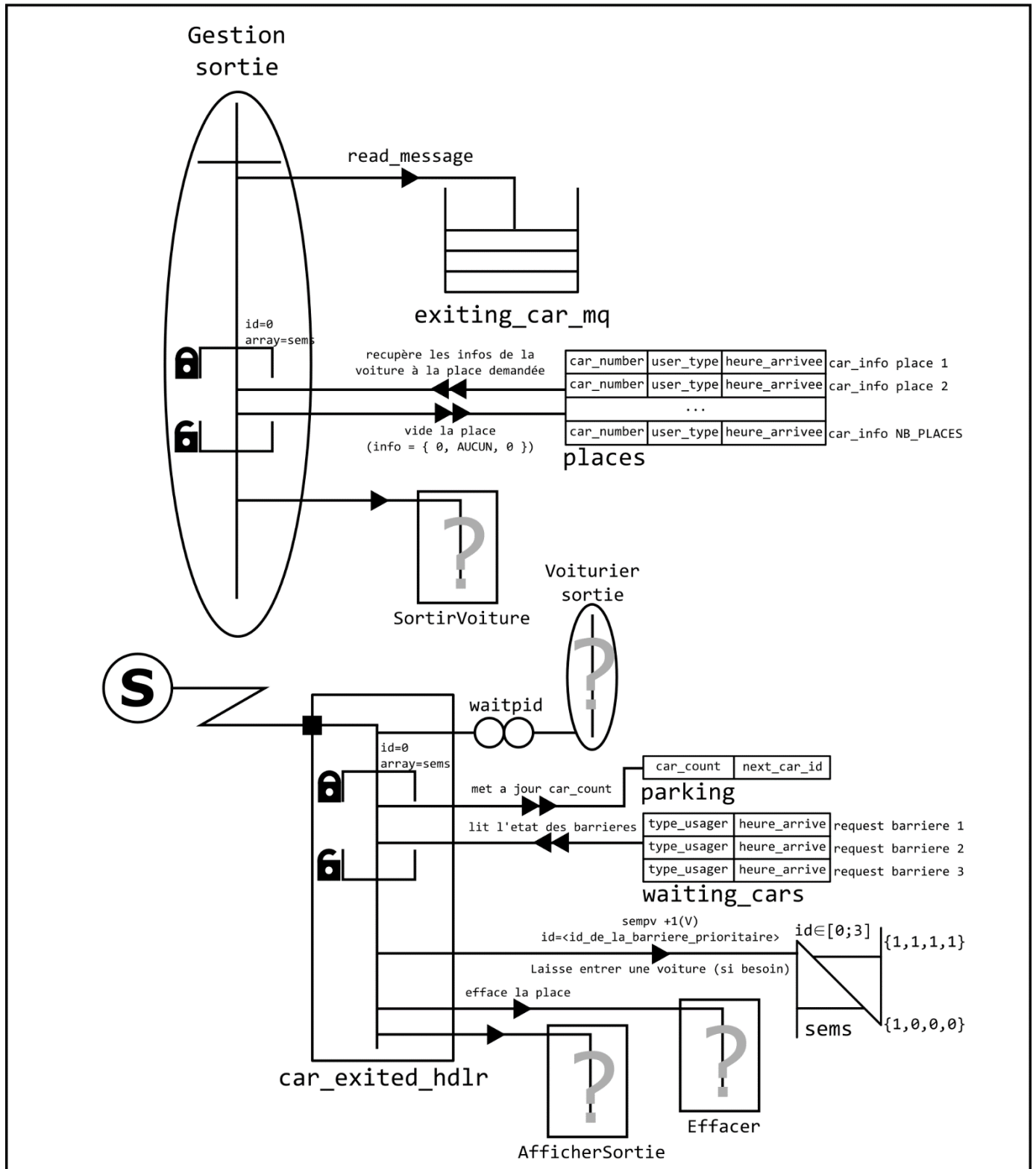


TACHE SORTIE

INITIALISATION SORTIE



PHASE MOTEUR SORTIE



DESTRUCTION SORTIE

