Haute Ecole de la Province de Liège

Informatique industrielle: Labo 2 Suite

Rahali Nassim & Schmidt Sébastien - M18

Liège, le 22 février 2015

Table des matières

1 Organisation du projet

Pour la réalisation de ce projet, nous avons essayé au maximum de scinder le programme en parties. Nous aurions pu créer un seul fichier de code source mais la compréhension du code en aurait pâti. C'est la raison pour laquelle une série de processus a été créés afin de rendre le code compréhensible.

De plus, les processus vont se créer automatiquement, il n'y aura pas besoin de lancer les processus manuellement. On peut considérer le processus Gestion comme le père des processus. Il aura pour tâche de lancer X processus Bateau et Y processus Port. Le port créera ensuite ses propres processus Quai et son processus GenVehicle. Le nombre de processus bateau et Port à lancer sera spécifier dans un fichier de configuration qui sera lu au démarrage de l'application.

2 Structures de données

Pour ce projet, nous avons défini deux structures de données : Une pour les bateaux et une pour les quais. Ces structures seront contenues dans des mémoire partagées accessibles par tous les processus.

2.1 Structure pour les bateaux

```
1
2  #define MQ_TRUCKS     "/MQT"
3  #define MQ_CARS_VANS   "/MQCV"
4
5  #define MQ_MAXSIZE      50
6  #define MQ_MSGSIZE      50
7
8  typedef enum {SEA, ENTERS_PORT, DOCK, LEAVES_PORT} boat_p;
9  typedef enum {UNDEFINED, DOVER, CALAIS, DUNKERQUE} boat_d;
10
11 typedef struct Boat_t
12 {
13   pid_t pid;
```

Dans un premier temps, on a défini deux énumération pour la direction ainsi que la position du bateau. Ces énumération seront utilisées dans la structure. Par rapport au point de vue théorique, nous avons rajouté deux données : l'index du bateau ainsi qu'un indicateur pour savoir si les données ont été modifiées ou pas. Cette données est uniquement utilisées pour l'affichage et n'a donc pas grand intérêt. La mémoire partagée aura une taille égale à 6 fois la taille de cette structure.

2.2 Structure pour les quais

```
1    boat_p position;
2    boat_d direction;
3    int state_changed;
4    MessageQueue mq1;
5    MessageQueue mq2;
```

Chaque port possède X quais, les informations essentielles de ces quais seront contenues dans une mémoire partagée par port. Dans cette structure, on spécifiera l'index du quai ainsi que le bateau qui y est actuellement accosté. Si aucun bateau n'est présent le boat_index vaudra -1 et sera libre.

3 Les différents processus

3.1 Processus Gestion

Comme spécifié précédemment, ce processus va créé des processus fils grâce à la fonction fork(). On va ensuite exécuter un fichier grâce a la fonction execl qui peut également passer des arguments à ce processus. Aux processus Bateau, on leur fournira leur numéro d'index : Comme il y a 6 bateaux, chaque processus aura un index compris entre 0 et 5. Ce numéro sera utilisé par le processus Quai et Port afin de savoir quel bateau est sur le point de rentrer dans le port ou d'accoster. Quant au processus Port, on lui fournir le nom de son port (Calais, Dunkerque ou Douvre) afin de notamment savoir le nombre de quais que le port possède mais également pour créer des sémaphores. En effet les ports ont besoin de sémaphores uniques : on va dans ce cas utiliser le nom du port concaténé à un nom de sémaphore commun pour avoir un nom de sémaphore unique et facilement retrouvable pour les autres processus qui en auront besoin (comme le bateau lors de son entrée).

```
1  for (i = 0; i < nb_boats; i++)
2  {
3    if ((child_pid = fork()) < 0)
4    {
5      perror("fork failure");
6      exit(1);
7    }
8
9    if (child_pid == 0)
10   {
11     char* p = malloc(sizeof(p));
12     sprintf(p, "%d", i);
13     execl("Boat", "BOAT", p, NULL);
14   }
15 }
16
17
18 // Création des ports
19 for (i = 0; i < nb_ports; i++)
20 {
21   if ((child_pid = fork()) < 0)
22   {
23     perror("fork failure");
24     exit(1);
25   }
26
27   if (child_pid == 0)
28   {
29     char* p = malloc(sizeof(p));
30     sprintf(p, "%d", (i == 0) ? 3 : 2);
31     execl("Port", "PORT", ports_name[i], p, NULL);
32   }
33 }
```

3.2 Processus Bateau

Le rôle du bateau dans un premier temps est d'initialiser sa propre structure contenue dans la mémoire partagée des bateaux avec une position au milieu de la mer. On a vu au point 2.1 que l'énumération des directions contenait une valeur supplémentaire : UNDEFINED qui sera utilisée pour initialiser la direction du bateau.

Une fois l'initialisation terminée, on rentre dans la boucle qui va lire l'état du bateau dans la mémoire partagée et va effectuer une série de tâche en fonction de sa position.

```
boat.mq2.oflag = (O_CREAT | O_EXCL | O_RDWR ) | O_NONBLOCK;
boat.mq2.mode  = 0644;
sprintf(boat.mq2.name,"%s%d", MQ_CARS_VANS, index);
```

L'index correspondant à l'identifiant du bateau. Il permet de récupérer la structure correspondant à ce dit bateau. Quand les différentes modifications en fonction de la position du bateau, une mise à jour de la structure de la mémoire partagée est effectuée.

3.2.1 Au milieu de l'eau

```
    wait_sem(mutex_boat);
    memcpy(&boat, shm_boat.pShm + (index * sizeof(Boat)), sizeof(Boat));
    signal_sem(mutex_boat);

    char* port_name;

    switch(boat.position)
    {
      case SEA:
        // Premier voyage
        if (boat.direction == UNDEFINED)
          boat.direction = rand() % 3 + 1;
        // Les bateaux viennent d'un port
        else
        {
```

S'il s'agit du premier voyage, on définit une destination aléatoirement. Par contre si ce n'est pas le premier voyage, les bateaux venant de France (Calais ou Dunkerque) doivent impérativement aller vers Douvre. Au contraire s'ils proviennent d'Angleterre, une destination est tiré de manière aléatoire entre Calais et Dunkerque. On effectue ensuite une simulation de la traversée.

3.2.2 Entrée dans le port

```
        print_boat(index, msg);
        sleep(duration);

        boat.state_changed = 1;
        boat.position = ENTERS_PORT;
        break;

      case ENTERS_PORT:
      {
        // Récupération des ressources du port
        open_port_ressources(&sem_port, &mutex_dep, &mutex_arr, &shm_dep, &
    shm_arr, port_name);

        wait_sem(mutex_arr);
        memcpy(&cpt_arr, shm_arr.pShm, sizeof(int));
        cpt_arr++;
```

Lorsque le bateau arrive à l'entrée d'un port, il doit prévenir le port et incrémenté le compteur d'arrivée de ce port. Tout d'abord il doit récupérer le mutex et la mémoire partagée appartenant à ce port. Pour cela on utilise le nom du port qui est stocké dans la structure du bateau et on reconstruit le nom du mutex concerné. Le même travaille est effectué pour la mémoire partagée et le sémaphore permettant d'avertir le port d'une arrivée.

Le bateau doit ensuite attendre l'autorisation du port pour pouvoir rentrer. Cette synchronisation se fait grâce à un sémaphore débloquer par le port.

3.2.3 A quai

3.2.4 Sortie du port

```
        sprintf(msg, "Sem : %s", sem_dock.semname);
        print_boat(index, msg);
        open_sem(&sem_dock);

        // Debloque le quai
        signal_sem(sem_dock);

        // Autorisation pour la sortie du port
        wait_sem(mutex_sync);

        sprintf(msg, "Quitte le quai");
        print_boat(index, msg);
```

La sortie dun port se fait exactement de la même manière que l'entrée à la différence qu'ici on incrémente le compteur de départ du port. Il faut donc récupérer les ressources correspondantes.

3.3 Processus Port

Les ports commencent par la création des ressources qui leur sont propres et initialisent les compteurs de départ et d'arrivée à zéro. Ils créent également deux ou trois quais en fonction du port qu'ils représentent ainsi que processus permettant de générer un embarquement/débarquement de véhicules. Pour la création des processus fils, on utilise la même méthode que pour la création des processus depuis le processus Gestion : C'est-à-dire une combinaison de fork() et de execl. On passe en paramètres aux processus Quai le nom du port ainsi que l'index du quai. Ces informations seront utiles pour créer des ressources uniques.

Ils se mettent ensuite dans l'attente d'un bateau grâce à une sémaphore initialisée à zéro également :

```
  {
    // En attente de bateau
    printf("\tPort %s > En attente de bateau\n", port_name);
```

Quand le processus Port est débloqué, il commence par traité les départs des bateaux si il y'en a. Il va devoir lire le compteur de départ écrit dans une mémoire partagée. Si des bateaux désirent quitter, il doit décrémenter le compteur et avertir le bateau concerné qu'il peut quitter le port.

```
    memcpy(&cpt_dep, shm_dep.pShm, sizeof(int));
    if (cpt_dep > 0)
    {
      // Recherche du bateau
      wait_sem(mutex_boat);
      boat = get_actual_boat(LEAVES_PORT, port_name, nb_boats, shm_boat);
      signal_sem(mutex_boat);

      sprintf(msg, "Bateau %d sort", boat.index);
      print_boat(port_name, boat.index, msg);

      // Décrémente le compteur
      cpt_dep--;
```

```
14        memcpy(shm_dep.pShm, &cpt_dep, sizeof(int));
15
16        signal_sem(mutex_dep);
17
18        // Autorise le bateau à sortir
19        mutex_sync.oflag = 0;
20        sprintf(mutex_sync.semname,"%s%d", MUTEX_SYNC, boat.index);
21        open_sem(&mutex_sync);
22        signal_sem(mutex_sync);
23        close_sem(mutex_sync);
24      }
25      else
26      {
27        signal_sem(mutex_dep);
```

Sinon il aucun bateau n'est prêt à quitter, c'est qu'un bateau est sur le point d'entrer. Il faut donc lui trouver un quai libre auquel il peut venir accoster et il faut ensuite décrémenter le compteur d'arrivée et lui prévenir d'entrer. Quand le quai est réservé, le numéro du bateau est placé dans la structure du quai en question. Le bateau devra allé lire la mémoire partagée du port contenant l'ensemble des quais pour savoir quel quai lui est destiné.

```
1
2         // Recherche du bateau
3         wait_sem(mutex_boat);
4         boat = get_actual_boat(ENTERS_PORT, port_name, nb_boats, shm_boat);
5         signal_sem(mutex_boat);
6
7         sprintf(msg, "Bateau %d entre", boat.index);
8         print_boat(port_name, boat.index, msg);
9
10        // TODO Reservation du quai
11        int found = 0;
12        wait_sem(mutex_dock);
13        for (i = 0; i < nb_docks && !found; i++)
14        {
15          Dock tmpDock;
16          memcpy(&tmpDock, shm_dock.pShm + (i * sizeof(Dock)), sizeof(Dock));
17          //printf("Port %s > Bateau - %d Quai %d - %d\n", port_name, boat.
    index, tmpDock.index, tmpDock.boat_index);
```

Il ne reste plus qu'à s'occuper des entrées après la réservation du quai et prévenir le bateau qu'il peut entrer.

3.4 Processus Quai

Un processus quai commence par la création des ressources qui lui sont propres :
- Une sémaphore quai.
- Une mutex pour l'accès relatif à la SHM des quais.
- Une SHM relative aux quais.
- Une sémaphore pour le processus de générations de véhicules.

Il entre ensuite dans une boucle dans laquelle il commence par attendre sur sa sémaphore quai. Celle ci sera signalée par un bateau en temps voulu.

```
1   while (!stop)
2   {
3     // Attente d'un bateau
4     printf("\t\t Quai %s %d > En attente %s\n", port_name, dock_index,
      sem_dock.semname);
5     wait_sem(sem_dock);
```

Quand un processus Quai est signalé, le bateau est à quai et la première chose que le processus va réaliser est d'ouvrir les Messages Queues de ce bateau.

```
mqd_trucks = mq_open(mq1_name, O_RDONLY | O_NONBLOCK);
mqd_cars_vans = mq_open(mq2_name, O_RDONLY | O_NONBLOCK);
```

De manière logique, il s'agit alors de réaliser le débarquement. Ceci est réalisé de la manière suivante : on récupère d'abord le nombre de messages dans les Messages Queues, puis une fois ces valeurs obtenues, on consomme les messages des deux Messages Queues (une pour les Camions et l'autre pour les Voitures des Camionnettes donc).

```
if (mq_getattr(mqd_trucks, &attr1) == -1 || mq_getattr(mqd_cars_vans, &
attr2) == -1)
{
  perror("Erreur when mq_getattr\n");
}
```

```
if(attr1.mq_curmsgs > 0)
{
  while(num_read != -1)
  {
    num_read = mq_receive(mqd_trucks, buffer, attr1.mq_msgsize, NULL);
    printf("Sortie de %s\n", (char *)buffer);
    nanosleep((struct timespec[]){{0, 250000000}}, NULL);
  }
}
num_read = 0;
printf("CURMSGS CARS & VANS: %ld\n", attr2.mq_curmsgs);
if(attr2.mq_curmsgs > 0)
{
  while(num_read != -1)
  {
    num_read = mq_receive(mqd_cars_vans, buffer, attr2.mq_msgsize, NULL);
    printf("Sortie de %s\n", (char *)buffer);
    nanosleep((struct timespec[]){{0, 250000000}}, NULL);
  }
}
```

Avant de repartir sur un nouveau tour de boucle, le processus Quai va signaler la sémaphore relative à la génération de véhicules ce qui va permettre de remplir à nouveau les Messages Queues du bateau avant son départ. Notons aussi que le débarquement réalisé par un processus Quai peut prendre un certain temps (0.25 secondes par véhicules).

```
sem_gen_v.oflag = 0;
sprintf(sem_gen_v.semname,"%s%s", SEM_GEN_V, argv[1]);
open_sem(&sem_gen_v);
signal_sem(sem_gen_v);
```

3.5 Processus GenVehicle

Il y a une version de ce processus associée à chaque port. Comme le système que nous mettons en place nécessite de la synchronisation, ce processus va aussi devoir utiliser les sémaphores et autres structures présentées précédemment.

- La sémaphore concernant la génération de véhicule (qui lui est propre).
- La mutex protégeant l'accès à la SHM des bateaux.
- La sémaphore permettant la synchronisation avec le bateau qui a terminé son embarquement (et donc le prévenir qu'il peut partir).
- La SHM des bateaux.

On définit également une variable de type Boat pour stocker le bateau recherché par la fonction `get_actual_boat()` ainsi que les deux descripteurs qui permettront l'accès aux Messages Queues du bateau.

```
Semaphore sem_gen_v;
Semaphore mutex_boat;
Semaphore mutex_sync;
Shm shm_boat;
Boat boat;
mqd_t mqd_trucks;
mqd_t mqd_cars_vans;
```

Une fois l'ouverture des ressources nécessaires terminées, ce processus entre également dans une boucle. Il commence par attendre sur sa propre sémaphore concernant la génération de véhicule et comme celle-ci est initialisée avec une valeur de 0, le processus est en attente.

```
while(1)
{
  // Waiting signal_sem on sem_gen_v from Docks processes.
  wait_sem(sem_gen_v);
```

Cette sémaphore sera donc signalée par un processus quai associé au même port que le processus *GenVehicle*. Le processus de génération de véhicule peut alors réaliser ce à quoi il est destiné : générer des voitures, des camionnettes et des camions pour le bateau qui est en attente ! Il va donc commencer par rechercher de quel bateau il s'agit.

```
    // Waiting for access on shm_boat
    wait_sem(mutex_boat);
    boat = get_actual_boat(DOCK, argv[1], nb_boats, shm_boat);
    signal_sem(mutex_boat);

    // MUTEX_SYNC
    mutex_sync.oflag = 0;
    sprintf(mutex_sync.semname,"%s%d", MUTEX_SYNC, boat.index);
    open_sem(&mutex_sync);
```

Une fois celui obtenu (et donc l'accès aux Messages Queues), nous utilisons `srand(getpid())` et `rand()` pour fournir des nombres aléatoires de camions, voitures et camionnettes.

```
    // Ouverture MQs
    mqd_trucks = mq_open(boat.mq1.name, O_WRONLY);
    mqd_cars_vans = mq_open(boat.mq2.name, O_WRONLY);

    nb_cars = rand() % MAX_N_CARS + 1;
    nb_vans = rand() % MAX_N_VANS + 1;
    nb_trucks = rand()% MAX_N_TRUCKS + 1;
```

Les constantes sont définies dans le header de *GenVehicle.c*.

```
#define MAX_N_TRUCKS 5
#define MAX_N_CARS 10
#define MAX_N_VANS 15
```

Quand ces valeurs sont générées, il reste donc à remplir les Messages Queues ce qui est donc fait par la fonction `mq_send()`. Si on prend par exemple le remplissage de la Message Queue représentant les camionnettes et les voitures :

```
    for(i = 0; i < nb_cars; i++)
    {
      sprintf(buffer, "Car %d", i + 1);
      if(mq_send(mqd_cars_vans, buffer, strlen(buffer), CAR_PRIORITY) == -1)
      {
```

```
6           mq_close(mqd_cars_vans);
7           mq_unlink(boat.mq1.name);
8           perror("Error occured when mq_send (cars & vans)\n");
9           exit(EXIT_FAILURE);
10        }
11        printf("%s on board\n", buffer);
12        // Sleep 1/4s -- TODO Paramétrable.
13        nanosleep((struct timespec[]){{0, 250000000}}, NULL);
14      }
15      printf("\t%d cars entered the boat %d.\n", nb_cars, boat.index);
16      for(i = 0; i < nb_vans; i++)
17      {
18        sprintf(buffer, "Van %d", i);
19        if(mq_send(mqd_cars_vans, buffer, strlen(buffer), VAN_PRIORITY) == -1)
20        {
21          mq_close(mqd_cars_vans);
22          mq_unlink(boat.mq1.name);
23          perror("Error occured when mq_send (cars & vans)\n");
24          exit(EXIT_FAILURE);
25        }
26        printf("%s on board\n", buffer);
27        // Sleep 1/4s
28        nanosleep((struct timespec[]){{0, 250000000}}, NULL);
29      }
```

On peut constater que les voitures entrent d'abord et que les camionnettes suivent. On remarquera également les deux priorités diffèrent et sont en fait des constantes définies dans le header de *GenVehicle.c*.

```
1 #define CAR_PRIORITY 1
2 #define VAN_PRIORITY 2
```

Et les camionnettes ayant une priorité plus elevée seront bien celles qui débarqueront en premier dans le quai. La dernière étape est donc de prévenir le bateau qui l'embarquement est terminé et lui permettre donc de prendre son état suivant :

```
1     // Récupération de la mutex_sync
2     mutex_sync.oflag = 0;
3     sprintf(mutex_sync.semname,"%s%d", MUTEX_SYNC, boat.index);
4     // Signal le bateau qu'il peut y aller
5     signal_sem(mutex_sync);
```

Après quoi le processus repart pour une boucle.

4 Test du programme

5 Code source

5.1 Config.cfg

```
1 nb_ports=3
2 nb_boats=3
```

5.2 Common.h

```
1 #ifndef COMMON_H
2 #define COMMON_H
3
4 #include "Ressources.h"
```

```
5
6  #define PROP_FILE "../Config.cfg"
7
8  #define MUTEX_BOAT     "mutexBoat"
9  #define SHM_BOAT       "shmBoat"
10 #define SHM_ARR        "shmArr"
11 #define SHM_DEP        "shmDep"
12 #define SHM_DOCK       "shmDock"
13 #define SEM_PORT       "semPort"
14 #define SEM_DOCK       "semDock"
15 #define SEM_GEN_V      "semGenV"
16 #define MUTEX_DEP      "mutexDep"
17 #define MUTEX_DOCK     "mutexDock"
18 #define MUTEX_ARR      "mutexArr"
19 #define MUTEX_SYNC     "mutexSync"
20 #define MUTEX_DOCK     "mutexDock"
21
22 #define MQ_TRUCKS      "/MQT"
23 #define MQ_CARS_VANS   "/MQCV"
24
25 #define MQ_MAXSIZE     50
26 #define MQ_MSGSIZE     50
27
28 typedef enum {SEA, ENTERS_PORT, DOCK, LEAVES_PORT} boat_p;
29 typedef enum {UNDEFINED, DOVER, CALAIS, DUNKERQUE} boat_d;
30
31 typedef struct Boat_t
32 {
33   pid_t pid;
34   int index;
35   boat_p position;
36   boat_d direction;
37   int state_changed;
38   MessageQueue mq1;
39   MessageQueue mq2;
40 } Boat;
41
42 typedef struct Dock_t
43 {
44   int index;
45   int boat_index;
46 } Dock;
47
48 #endif /* COMMON_H */
```

5.3 Gestion.h

```
1  #ifndef GESTION_H
2  #define GESTION_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <errno.h>
7  #include <string.h>
8  #include <unistd.h>
9
10 #include "Ressources.h"
11 #include "Common.h"
12
13 char* getProp(const char *fileName, const char *propName);
14 void init_ressources(int nb_boats);
15
```

```
16 Semaphore mutex_boat;
17
18 Shm shm_boat;
19
20 #endif /* GESTION_H */
```

5.4 Gestion.c

```
1  #include "Gestion.h"
2
3  int main()
4  {
5    int i;
6    int stop = 0;
7    int nb_ports = atoi(getProp(PROP_FILE, "nb_ports"));
8    int nb_boats = atoi(getProp(PROP_FILE, "nb_boats"));
9    char* ports_name[] = {"Douvre", "Calais", "Dunkerque"};
10   pid_t child_pid;
11
12   printf("Nb_boats : %d \n", nb_boats);
13   printf("Nb_ports : %d \n", nb_ports);
14
15   mutex_boat.oflag = (O_CREAT | O_RDWR);
16     mutex_boat.mode  = 0600;
17     mutex_boat.value = 1;
18     strcpy(mutex_boat.semname, MUTEX_BOAT);
19
20   sem_unlink(mutex_boat.semname);
21
22   open_sem(&mutex_boat);
23
24   shm_boat.sizeofShm = sizeof(Boat) * nb_boats;
25   shm_boat.mode = O_CREAT | O_RDWR;
26   strcpy(shm_boat.shmName, SHM_BOAT);
27
28   open_shm(&shm_boat);
29   mapping_shm(&shm_boat, sizeof(Boat) * nb_boats);
30
31   /*for (i = 0; i < 10; i++)
32   {
33     wait_sem(mutex_boat);
34     printf("Test de section\n");
35     signal_sem(mutex_boat);
36   }*/
37
38
39   // Création des ressources nécessaires
40   //init_ressources(nb_boats);
41
42   // Création des bateaux
43   for (i = 0; i < nb_boats; i++)
44   {
45     if ((child_pid = fork()) < 0)
46     {
47       perror("fork failure");
48       exit(1);
49     }
50
51     if (child_pid == 0)
52     {
53       char* p = malloc(sizeof(p));
54       sprintf(p, "%d", i);
```

```
55        execl("Boat", "BOAT", p, NULL);
56      }
57    }
58
59
60    // Création des ports
61    for (i = 0; i < nb_ports; i++)
62    {
63      if ((child_pid = fork()) < 0)
64      {
65        perror("fork failure");
66        exit(1);
67      }
68
69      if (child_pid == 0)
70      {
71        char* p = malloc(sizeof(p));
72        sprintf(p, "%d", (i == 0) ? 3 : 2);
73        execl("Port", "PORT", ports_name[i], p, NULL);
74      }
75    }
76
77    // Lecture des données
78    Boat tmpBoat;
79    while (!stop)
80    {
81      for (i = 0; i < nb_boats; i++)
82      {
83        wait_sem(mutex_boat);
84        memcpy(&tmpBoat, shm_boat.pShm + (i * sizeof(Boat)), sizeof(Boat));
85        if (tmpBoat.state_changed == 1)
86        {
87          //printf("Boat %d - pid = %d - position : %d - direction %d - state %
      d\n", i, tmpBoat.pid, tmpBoat.position, tmpBoat.direction, tmpBoat.
      state_changed);
88
89          tmpBoat.state_changed = 0;
90          memcpy(shm_boat.pShm + (i * sizeof(Boat)), &tmpBoat, sizeof(Boat));
91        }
92        signal_sem(mutex_boat);
93      }
94    }
95
96    return EXIT_SUCCESS;
97  }
98
99  void init_ressources(int nb_boats)
100 {
101   // MUTEX_BATEAU
102   /*mutex_boat.oflag = (O_CREAT | O_RDWR);
103     mutex_boat.mode  = 0600;
104     mutex_boat.value = 1;
105     strcpy(mutex_boat.semname, MUTEX_BOAT);
106
107   open_sem(&mutex_boat);
108
109   // SHM_BATEAU
110   shm_boat.sizeofShm = sizeof(Boat) * nb_boats;
111   shm_boat.mode = O_CREAT | O_RDWR;
112   strcpy(shm_boat.shmName, SHM_BOAT);
113
114   open_shm(&shm_boat);
115   mapping_shm(&shm_boat, sizeof(Boat) * nb_boats);*/
```

```
116 }
117
118 char* getProp(const char *fileName, const char *propName)
119 {
120   FILE*    file = NULL;
121   char*    token = NULL;
122   char  line[128];
123   char  sep[2] = "=";
124   int    i;
125   int    loginFound = 0;
126
127   if ((file = fopen(fileName, "r")) == NULL)
128   {
129     perror("Opening file\n");
130     exit(errno);
131   }
132   else
133   {
134         while (fgets(line, sizeof line, file) != NULL)
135         {
136         token = strtok(line, sep);
137       i = 0;
138
139         while(token != NULL)
140         {
141         if (i == 0)
142         {
143           if (strcmp(token, propName) == 0)
144             loginFound++;
145         }
146         else if (i != 0 && loginFound == 1)
147         {
148           char *password = malloc(sizeof(char *) * 30);
149           strcpy(password, token);
150           fclose(file);
151           return password;
152         }
153             token = strtok(NULL, sep);
154         i++;
155         }
156       }
157   }
158
159   fclose(file);
160   return NULL;
161 }
```

5.5 Boat.h

```
1  #ifndef BATEAU_H
2  #define BATEAU_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <time.h>
8  #include <signal.h>
9
10 #include "Ressources.h"
11 #include "Common.h"
12
```

```
13 void init_ressources(Semaphore* mutex_sync, Semaphore* mutex_boat, Shm*
       shm_boat, int index);
14 void open_port_ressources(Semaphore* sem_port, Semaphore* mutex_dep,
       Semaphore* mutex_arr, Shm* shm_dep, Shm* shm_arr, char* port_name);
15 void open_dock_ressources(Semaphore* mutex_dock, Shm* shm_dock, char*
       port_name, int nb_docks);
16 void handler(int sig);
17 void print_boat(int index, char* msg);
18
19 #endif /* BATEAU_H */
```

5.6 Boat.c

```
1  #include "Boat.h"
2
3  int main(int argc, char* argv[])
4  {
5    Semaphore mutex_boat;
6    Semaphore sem_port;
7    Semaphore mutex_dep;
8    Semaphore sem_dock;
9    Semaphore mutex_arr;
10   Semaphore mutex_sync;
11   Semaphore mutex_dock;
12
13   Shm shm_dock;
14   Shm shm_dep;
15   Shm shm_arr;
16   Shm shm_boat;
17   char* ports_name[] = {"Douvre", "Calais", "Dunkerque"};
18   int index;
19   int cpt_arr;
20   int cpt_dep;
21   sscanf(argv[1], "%d", &index);
22   int stop = 0;
23   Boat boat;
24   struct mq_attr attr;
25   struct sigaction act;
26   char* msg = malloc(sizeof(msg));
27   act.sa_handler = handler;
28
29   srand(getpid());
30
31   // Initialisation des ressources
32   init_ressources(&mutex_sync, &mutex_boat, &shm_boat, index);
33
34   // Placement de l'état par défaut du bateau
35   boat.pid = getpid();
36   boat.index = index;
37   boat.position = SEA;
38   boat.direction = UNDEFINED;
39   boat.state_changed = 0;
40
41   attr.mq_curmsgs = 0;
42   attr.mq_flags = 0;
43   attr.mq_maxmsg = MQ_MAXSIZE;
44   attr.mq_msgsize = MQ_MSGSIZE;
45
46   // MQ CAMIONS
47   boat.mq1.oflag = (O_CREAT | O_EXCL | O_RDWR ) | O_NONBLOCK;
48   boat.mq1.mode  = 0644;
49   sprintf(boat.mq1.name,"%s%d", MQ_TRUCKS, index);
```

```
50
51    // MQ VOITURES ET CAMIONNETTES
52    boat.mq2.oflag = (O_CREAT | O_EXCL | O_RDWR ) | O_NONBLOCK;
53    boat.mq2.mode  = 0644;
54    sprintf(boat.mq2.name,"%s%d", MQ_CARS_VANS, index);
55
56    mq_unlink(boat.mq1.name);
57    mq_unlink(boat.mq2.name);
58    open_mq(&boat.mq1, &attr);
59    open_mq(&boat.mq2, &attr);
60
61    wait_sem(mutex_boat);
62    memcpy(shm_boat.pShm + (index * sizeof(Boat)), &boat, sizeof(Boat));
63    signal_sem(mutex_boat);
64
65    while (!stop)
66    {
67      // Lecture de l'état du bateau
68      wait_sem(mutex_boat);
69      memcpy(&boat, shm_boat.pShm + (index * sizeof(Boat)), sizeof(Boat));
70      signal_sem(mutex_boat);
71
72      char* port_name;
73
74      switch(boat.position)
75      {
76        case SEA:
77          // Premier voyage
78          if (boat.direction == UNDEFINED)
79            boat.direction = rand() % 3 + 1;
80          // Les bateaux viennent d'un port
81          else
82          {
83            if (boat.direction == CALAIS || boat.direction == DUNKERQUE)
84              boat.direction = DOVER;
85            else
86              boat.direction =  rand() % (3 - 2 + 1) + 2;
87          }
88
89          port_name = ports_name[boat.direction - 1];
90
91          // Simulation de la traversée
92          int duration = rand() % 5 + 10;
93          sprintf(msg, "Traversée vers %s (%d secondes)", port_name, duration);
94          print_boat(index, msg);
95          sleep(duration);
96
97          boat.state_changed = 1;
98          boat.position = ENTERS_PORT;
99          break;
100
101       case ENTERS_PORT:
102       {
103         // Récupération des ressources du port
104         open_port_ressources(&sem_port, &mutex_dep, &mutex_arr, &shm_dep, &
      shm_arr, port_name);
105
106         wait_sem(mutex_arr);
107         memcpy(&cpt_arr, shm_arr.pShm, sizeof(int));
108         cpt_arr++;
109         memcpy(shm_arr.pShm, &cpt_arr, sizeof(int));
110         signal_sem(mutex_arr);
111         signal_sem(sem_port);
```

```
112
113         sprintf(msg, "Devant l'entrée de %s", port_name);
114         print_boat(index, msg);
115
116         wait_sem(mutex_sync);
117         //printf("### BOAT %d ENTERS_PORT [%s]\n", index, port_name);
118         sprintf(msg, "Entree dans le port de %s", port_name);
119         print_boat(index, msg);
120         boat.position = DOCK;
121         break;
122       }
123       case DOCK:
124       {
125         int nb_docks = (strcmp(port_name, "Douvre") == 0) ? 3 : 2;
126         // Création ressouces du quai
127         mutex_dock.oflag = O_RDWR;
128         mutex_dock.mode  = 0644;
129         mutex_dock.value = 1;
130         sprintf(mutex_dock.semname,"%s%s", MUTEX_DOCK, port_name);
131         open_sem(&mutex_dock);
132
133         // SHM_DOCK
134         shm_dock.sizeofShm = sizeof(Dock) * nb_docks;
135         shm_dock.mode = O_RDWR;
136         sprintf(shm_dock.shmName,"%s%s", SHM_DOCK, port_name);
137
138         open_shm(&shm_dock);
139         mapping_shm(&shm_dock, sizeof(Dock) * nb_docks);
140         open_dock_ressources(&mutex_dock, &shm_dock, port_name, nb_docks);
141
142         // Recherche de l'id du quai
143         int i;
144         int dock_index;
145         int found = 0;
146         Dock dock;
147         wait_sem(mutex_dock);
148         for (i = 0; i < nb_docks && !found; i++)
149         {
150           memcpy(&dock, shm_dock.pShm + (i * sizeof(Dock)), sizeof(Dock));
151           //printf("#[%s]# BOAT_INDEX [%d] == DOCK.BOAT_INDEX [%d] ?\n",
    port_name, index, dock.boat_index);
152           if (dock.boat_index == index)
153           {
154             dock_index = dock.index;
155             //printf("Index trouvé : %d\n", dock_index);
156             found = 1;
157           }
158         }
159         signal_sem(mutex_dock);
160
161         // Création de la sémaphore correspondante
162         sem_dock.oflag = O_RDWR;
163         sem_dock.mode  = 0644;
164         sem_dock.value = 0;
165         sprintf(sem_dock.semname, "%s%s%d", SEM_DOCK, port_name, dock_index);
166         sprintf(msg, "Sem : %s", sem_dock.semname);
167         print_boat(index, msg);
168         open_sem(&sem_dock);
169
170         // Debloque le quai
171         signal_sem(sem_dock);
172
173         // Autorisation pour la sortie du port
```

```
174             wait_sem(mutex_sync);
175
176             sprintf(msg, "Quitte le quai");
177             print_boat(index, msg);
178             boat.position = LEAVES_PORT;
179             break;
180           }
181           case LEAVES_PORT:
182
183             wait_sem(mutex_dep);
184             memcpy(&cpt_dep, shm_dep.pShm, sizeof(int));
185             cpt_dep++;
186             memcpy(shm_dep.pShm, &cpt_dep, sizeof(int));
187             signal_sem(mutex_dep);
188             signal_sem(sem_port);
189
190             sprintf(msg, "Devant la sortie de %s", port_name);
191             print_boat(index, msg);
192
193             //pause();
194             wait_sem(mutex_sync);
195
196             sprintf(msg, "Sortie du port de %s", port_name);
197             print_boat(index, msg);
198
199             // Fermeture des ressources du port
200             close_sem(sem_port);
201             close_sem(mutex_dep);
202             close_sem(mutex_arr);
203             boat.position = SEA;
204             break;
205       }
206
207       // Copie de la structure
208       wait_sem(mutex_boat);
209       memcpy(shm_boat.pShm + (index * sizeof(Boat)), &boat, sizeof(Boat));
210       signal_sem(mutex_boat);
211     }
212     return EXIT_SUCCESS;
213 }
214
215 void init_ressources(Semaphore* mutex_sync, Semaphore* mutex_boat, Shm*
        shm_boat, int index)
216 {
217     // MUTEX_SYNC
218     mutex_sync->oflag = (O_CREAT | O_RDWR);
219     mutex_sync->mode  = 0644;
220     mutex_sync->value = 0;
221     sprintf(mutex_sync->semname,"%s%d", MUTEX_SYNC, index);
222
223     sem_unlink(mutex_sync->semname);
224     open_sem(mutex_sync);
225
226     // MUTEX_BATEAU
227     mutex_boat->oflag = O_RDWR;
228     mutex_boat->mode  = 0644;
229     mutex_boat->value = 1;
230     strcpy(mutex_boat->semname, MUTEX_BOAT);
231
232     open_sem(mutex_boat);
233
234     shm_boat->sizeofShm = sizeof(Boat) * 6;
235     shm_boat->mode = O_RDWR;
```

```
236     strcpy(shm_boat->shmName, SHM_BOAT);
237
238     open_shm(shm_boat);
239     mapping_shm(shm_boat, sizeof(Boat) * 6);
240 }
241
242 void open_port_ressources(Semaphore* sem_port, Semaphore* mutex_dep,
        Semaphore* mutex_arr, Shm* shm_dep, Shm* shm_arr, char* port_name)
243 {
244     sem_port->oflag = O_RDWR;
245     sem_port->mode  = 0644;
246     sem_port->value = 0;
247     sprintf(sem_port->semname,"%s%s", SEM_PORT, port_name);
248
249     // MUTEX_DEP
250     mutex_dep->oflag = O_RDWR;
251     mutex_dep->mode  = 0644;
252     mutex_dep->value = 1;
253     sprintf(mutex_dep->semname,"%s%s", MUTEX_DEP, port_name);
254
255     // MUTEX_ARR
256     mutex_arr->oflag = O_RDWR;
257     mutex_arr->mode  = 0644;
258     mutex_arr->value = 1;
259     sprintf(mutex_arr->semname,"%s%s", MUTEX_ARR, port_name);
260
261     // SHM_DEP
262     shm_dep->sizeofShm = sizeof(int);
263     shm_dep->mode = O_RDWR;
264     sprintf(shm_dep->shmName,"%s%s", SHM_DEP, port_name);
265
266     // SHM_ARR
267     shm_arr->sizeofShm = sizeof(int);
268     shm_arr->mode = O_RDWR;
269     sprintf(shm_arr->shmName,"%s%s", SHM_ARR, port_name);
270
271     open_sem(sem_port);
272     open_sem(mutex_dep);
273     open_sem(mutex_arr);
274
275     open_shm(shm_dep);
276     mapping_shm(shm_dep, sizeof(int));
277
278     open_shm(shm_arr);
279     mapping_shm(shm_arr, sizeof(int));
280 }
281
282 void open_dock_ressources(Semaphore* mutex_dock, Shm* shm_dock, char*
        port_name, int nb_docks)
283 {
284     mutex_dock->oflag = O_RDWR;
285     mutex_dock->mode  = 0644;
286     mutex_dock->value = 1;
287     sprintf(mutex_dock->semname,"%s%s", MUTEX_DOCK, port_name);
288
289     open_sem(mutex_dock);
290
291     // SHM_DOCK
292     shm_dock->sizeofShm = sizeof(Dock) * nb_docks;
293     shm_dock->mode = O_RDWR;
294     sprintf(shm_dock->shmName,"%s%s", SHM_DOCK, port_name);
295
296     open_shm(shm_dock);
```

```
297   mapping_shm(shm_dock, sizeof(Dock) * nb_docks);
298 }
299
300 void print_boat(int index, char* msg)
301 {
302   char* color[] = {"\x1B[31m", "\x1B[32m", "\x1B[33m", "\x1B[34m", "\x1B[35m"
        , "\x1B[36m"};
303   char* reset = "\033[0m";
304
305   printf("Bateau %d> %s%s%s\n", index, color[index], msg, reset);
306 }
307
308 void handler(int sig)
309 {
310   printf("Signal recu %d\n", sig);
311 }
```

5.7 Port.h

```
1  #ifndef PORT_H
2  #define PORT_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <signal.h>
8  #include <unistd.h>
9  #include <string.h>
10
11 #include "Common.h"
12 #include "Ressources.h"
13
14 void create_processes(int nb_docks, char* port_name);
15 void init_ressources(Semaphore* mutex_boat, Semaphore* sem_port, Semaphore*
       mutex_dep, Semaphore* mutex_dock, Semaphore* mutex_arr, Shm* shm_dep, Shm
       * shm_arr, Shm* shm_dock, Shm* shm_boat, char* port_name, int nb_docks,
       int nb_boats);
16 void print_boat(char* port_name, int boat_index, char* msg);
17 Boat get_actual_boat(boat_p position, char* port, int nb_boats, Shm shm_boat)
       ;
18 char* getProp(const char *fileName, const char *propName);
19
20 #endif /* PORT_H */
```

5.8 Port.c

```
1  #include "Port.h"
2
3  int main(int argc, char** argv)
4  {
5    Semaphore sem_port;
6    Semaphore mutex_boat;
7    Semaphore mutex_dep;
8    Semaphore mutex_dock;
9    Semaphore mutex_arr;
10   Semaphore mutex_sync;
11
12   Shm shm_dep;
13   Shm shm_arr;
14   Shm shm_boat;
```

```
15  Shm shm_dock;
16
17  Boat boat;
18  char* port_name = argv[1];
19  char* msg = malloc(sizeof(msg));
20  int cpt_arr = 0;
21  int cpt_dep = 0;
22  int stop = 0;
23  int nb_boats = atoi(getProp(PROP_FILE, "nb_boats"));
24  int nb_docks = 0;
25    int i;
26
27    sscanf(argv[2], "%d", &nb_docks);
28
29  // Création des processus fils
30  create_processes(nb_docks, port_name);
31
32  // Initialisation des ressources
33  init_ressources(&mutex_boat, &sem_port, &mutex_dep, &mutex_dock, &mutex_arr
      , &shm_dep, &shm_arr, &shm_dock, &shm_boat, port_name, nb_docks, nb_boats
      );
34
35  // Mise a 0 des compteurs
36  wait_sem(mutex_dep);
37  memcpy(shm_dep.pShm, &cpt_dep, sizeof(int));
38  signal_sem(mutex_dep);
39
40  wait_sem(mutex_arr);
41  memcpy(shm_arr.pShm, &cpt_arr, sizeof(int));
42  signal_sem(mutex_arr);
43
44  while (!stop)
45  {
46    // En attente de bateau
47    printf("\tPort %s > En attente de bateau\n", port_name);
48    wait_sem(sem_port);
49
50    // Compteur de depart
51    wait_sem(mutex_dep);
52    memcpy(&cpt_dep, shm_dep.pShm, sizeof(int));
53    if (cpt_dep > 0)
54    {
55      // Recherche du bateau
56      wait_sem(mutex_boat);
57      boat = get_actual_boat(LEAVES_PORT, port_name, nb_boats, shm_boat);
58      signal_sem(mutex_boat);
59
60      sprintf(msg, "Bateau %d sort", boat.index);
61      print_boat(port_name, boat.index, msg);
62
63      // Décrémente le compteur
64      cpt_dep--;
65      memcpy(shm_dep.pShm, &cpt_dep, sizeof(int));
66
67      signal_sem(mutex_dep);
68
69      // Autorise le bateau à sortir
70      mutex_sync.oflag = 0;
71      sprintf(mutex_sync.semname,"%s%d", MUTEX_SYNC, boat.index);
72      open_sem(&mutex_sync);
73      signal_sem(mutex_sync);
74      close_sem(mutex_sync);
75    }
```

```
76      else
77      {
78        signal_sem(mutex_dep);
79
80        // Compteur d'arrivée
81        wait_sem(mutex_arr);
82        memcpy(&cpt_arr, shm_arr.pShm, sizeof(int));
83        if (cpt_arr > 0)
84        {
85          // Décrémente le compteur
86          cpt_arr--;
87          memcpy(shm_arr.pShm, &cpt_arr, sizeof(int));
88
89          signal_sem(mutex_arr);
90
91          // Recherche du bateau
92          wait_sem(mutex_boat);
93          boat = get_actual_boat(ENTERS_PORT, port_name, nb_boats, shm_boat);
94          signal_sem(mutex_boat);
95
96          sprintf(msg, "Bateau %d entre", boat.index);
97          print_boat(port_name, boat.index, msg);
98
99          // TODO Reservation du quai
100         int found = 0;
101         wait_sem(mutex_dock);
102         for (i = 0; i < nb_docks && !found; i++)
103         {
104           Dock tmpDock;
105           memcpy(&tmpDock, shm_dock.pShm + (i * sizeof(Dock)), sizeof(Dock));
106           //printf("Port %s > Bateau - %d Quai %d - %d\n", port_name, boat.
    index, tmpDock.index, tmpDock.boat_index);
107           sprintf(msg, "Quai %d", tmpDock.index);
108           print_boat(port_name, boat.index, msg);
109           // Recherche du premier quai disponible
110           if (tmpDock.boat_index == -1)
111           {
112             tmpDock.boat_index = boat.index;
113             memcpy(shm_dock.pShm + (i * sizeof(Dock)), &tmpDock, sizeof(Dock)
    );
114             found = 1;
115           }
116         }
117         signal_sem(mutex_dock);
118
119         // Envoie d'un signal au bateau
120         //kill(boat.pid, SIGUSR1);
121         mutex_sync.oflag = O_RDWR;
122         mutex_sync.mode  = 0644;
123         mutex_sync.value = 1;
124         sprintf(mutex_sync.semname,"%s%d", MUTEX_SYNC, boat.index);
125
126         open_sem(&mutex_sync);
127         sleep(1);
128         signal_sem(mutex_sync);
129         close_sem(mutex_sync);
130       }
131       else
132         signal_sem(mutex_arr);
133     }
134   }
135
136   return EXIT_SUCCESS;
```

```
137 }
138
139 void create_processes(int nb_docks, char* port_name)
140 {
141   pid_t child_pid;
142   int i;
143
144   // Création des quais
145   for (i = 0; i < nb_docks; i++)
146   {
147     if ((child_pid = fork()) < 0)
148     {
149       perror("fork failure");
150       exit(1);
151     }
152
153     if (child_pid == 0)
154     {
155       char* p = malloc(sizeof(p));
156       char* d = malloc(sizeof(d));
157       sprintf(p, "%d", i);
158       sprintf(d, "%d", nb_docks);
159       execl("Dock", "DOCK", port_name, p, d, NULL);
160     }
161   }
162
163   // Création de GenVehicle
164
165   if ((child_pid = fork()) < 0)
166   {
167     perror("fork failure");
168     exit(1);
169   }
170
171   if (child_pid == 0)
172   {
173     execl("GenVehicle", "GENVEHICLE", port_name, NULL);
174   }
175 }
176
177 void init_ressources(Semaphore* mutex_boat, Semaphore* sem_port, Semaphore*
      mutex_dep, Semaphore* mutex_dock, Semaphore* mutex_arr, Shm* shm_dep, Shm
      * shm_arr, Shm* shm_dock, Shm* shm_boat, char* port_name, int nb_docks,
      int nb_boats)
178 {
179   // MUTEX_BATEAU
180   mutex_boat->oflag = O_RDWR;
181     mutex_boat->mode  = 0644;
182     mutex_boat->value = 1;
183     strcpy(mutex_boat->semname, MUTEX_BOAT);
184
185   // SEM_PORT
186   sem_port->oflag = (O_CREAT | O_RDWR);
187     sem_port->mode  = 0644;
188     sem_port->value = 0;
189     sprintf(sem_port->semname,"%s%s", SEM_PORT, port_name);
190
191   // MUTEX_DEP
192   mutex_dep->oflag = (O_CREAT | O_RDWR);
193     mutex_dep->mode  = 0644;
194     mutex_dep->value = 1;
195     sprintf(mutex_dep->semname,"%s%s", MUTEX_DEP, port_name);
196
```

```
197     // MUTEX_DOCK
198     mutex_dock->oflag = (O_CREAT | O_RDWR);
199       mutex_dock->mode  = 0644;
200       mutex_dock->value = 1;
201       sprintf(mutex_dock->semname,"%s%s", MUTEX_DOCK, port_name);
202
203     // MUTEX_ARR
204     mutex_arr->oflag = (O_CREAT | O_RDWR);
205       mutex_arr->mode  = 0644;
206       mutex_arr->value = 1;
207       sprintf(mutex_arr->semname,"%s%s", MUTEX_ARR, port_name);
208
209     // SHM_DEP
210     shm_dep->sizeofShm = sizeof(int);
211     shm_dep->mode = O_CREAT | O_RDWR;
212     sprintf(shm_dep->shmName,"%s%s", SHM_DEP, port_name);
213
214     // SHM_ARR
215     shm_arr->sizeofShm = sizeof(int);
216     shm_arr->mode = O_CREAT | O_RDWR;
217     sprintf(shm_arr->shmName,"%s%s", SHM_ARR, port_name);
218
219     // SHM_DOCK
220     shm_dock->sizeofShm = sizeof(Dock) * nb_docks;
221     shm_dock->mode = O_CREAT | O_RDWR;
222     sprintf(shm_dock->shmName,"%s%s", SHM_DOCK, port_name);
223
224     shm_boat->sizeofShm = sizeof(Boat) * nb_boats;
225     shm_boat->mode = O_RDWR;
226     strcpy(shm_boat->shmName, SHM_BOAT);
227
228     sem_unlink(sem_port->semname);
229     sem_unlink(mutex_dep->semname);
230     sem_unlink(mutex_dock->semname);
231     sem_unlink(mutex_arr->semname);
232     sem_unlink(mutex_dock->semname);
233
234     open_sem(sem_port);
235     open_sem(mutex_dep);
236     open_sem(mutex_dock);
237     open_sem(mutex_arr);
238     open_sem(mutex_boat);
239     open_sem(mutex_dock);
240
241     open_shm(shm_boat);
242     mapping_shm(shm_boat, sizeof(Boat) * nb_boats);
243
244     open_shm(shm_dock);
245     mapping_shm(shm_dock, sizeof(Dock) * nb_docks);
246
247     open_shm(shm_dep);
248     mapping_shm(shm_dep, sizeof(int));
249
250     open_shm(shm_arr);
251     mapping_shm(shm_arr, sizeof(int));
252 }
253
254 void print_boat(char* port_name, int boat_index, char* msg)
255 {
256     char* color[] = {"\x1B[31m", "\x1B[32m", "\x1B[33m", "\x1B[34m", "\x1B[35m"
        , "\x1B[36m"};
257     char* reset = "\033[0m";
258
```

```
259   printf("\tPort %s > %s%s%s\n", port_name, color[boat_index], msg, reset);
260 }
261
262 Boat get_actual_boat(boat_p position, char* port, int nb_boats, Shm shm_boat)
263 {
264   // Parcours de la shm pour trouver le bateau concerné
265   int i;
266   int found;
267   char* ports_name[] = {"Douvre", "Calais", "Dunkerque"};
268   Boat *tmp = malloc(sizeof(Boat));
269   boat_d direction;
270
271   // Recherche le nom du port pour l'enum
272   for (i = 0, found = 0; i < 3 && !found; i++)
273   {
274     if (strcmp(port, ports_name[i]) == 0)
275     {
276       found = 1;
277       direction = i + 1;
278     }
279   }
280
281   // Recherche le bateau aux portes du port
282   for (i = 0, found = 0; i < nb_boats && !found; i++)
283   {
284     memcpy(tmp, shm_boat.pShm + (i * sizeof(Boat)), sizeof(Boat));
285     printf("Recherche: Bateau : %d - Port %d - Position %d\n", tmp->index,
      tmp->direction, tmp->position);
286     if (tmp->position == position && tmp->direction == direction)
287       found = 1;
288   }
289
290   return *tmp;
291 }
292
293 char* getProp(const char *fileName, const char *propName)
294 {
295   FILE*    file = NULL;
296   char*    token = NULL;
297   char  line[128];
298   char  sep[2] = "=";
299   int    i;
300   int    loginFound = 0;
301
302   if ((file = fopen(fileName, "r")) == NULL)
303   {
304     perror("Opening file\n");
305     exit(errno);
306   }
307   else
308   {
309         while (fgets(line, sizeof line, file) != NULL)
310         {
311         token = strtok(line, sep);
312       i = 0;
313
314         while(token != NULL)
315         {
316         if (i == 0)
317         {
318           if (strcmp(token, propName) == 0)
319             loginFound++;
320         }
```

```
321         else if (i != 0 && loginFound == 1)
322         {
323           char *password = malloc(sizeof(char *) * 30);
324           strcpy(password, token);
325           fclose(file);
326           return password;
327         }
328             token = strtok(NULL, sep);
329         i++;
330         }
331       }
332   }
333
334   fclose(file);
335   return NULL;
336 }
```

5.9 Dock.h

```
1  #ifndef DOCK_H
2  #define DOCK_H
3
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <unistd.h>
7  #include <string.h>
8  #include <time.h>
9
10 #include "Ressources.h"
11 #include "Common.h"
12
13 void init_ressources(Semaphore* sem_dock, Semaphore* mutex_dock, Shm*
       shm_dock, char* port_name, int dock_index, int nb_docks);
14 void print_boat(char* port_name, int dock_index, int boat_index, char* msg);
15
16 #endif /* DOCK_H */
```

5.10 Dock.c

```
1  #include "Dock.h"
2
3  int main(int argc, char** argv)
4  {
5    Semaphore sem_dock;
6    Semaphore mutex_dock;
7    Semaphore sem_gen_v;
8    Shm shm_dock;
9    struct mq_attr attr1;
10   struct mq_attr attr2;
11   mqd_t mqd_trucks;
12   mqd_t mqd_cars_vans;
13   char mq1_name[MQ_NAME_LENGTH];
14   char mq2_name[MQ_NAME_LENGTH];
15   char* port_name = argv[1];
16   char* msg = malloc(sizeof(msg));
17   int dock_index;
18   int nb_docks;
19   int stop = 0;
20   int num_read = 0;
21   void *buffer;
```

```
22
23    sscanf(argv[2], "%d", &dock_index);
24    sscanf(argv[3], "%d", &nb_docks);
25
26    init_ressources(&sem_dock, &mutex_dock, &shm_dock, port_name, dock_index,
        nb_docks);
27
28    // Mise a zero des info de la Shm
29    Dock dock;
30    dock.index = dock_index;
31    dock.boat_index = -1;
32    wait_sem(mutex_dock);
33    memcpy(shm_dock.pShm + (dock_index * sizeof(Dock)), &dock, sizeof(Dock));
34    signal_sem(mutex_dock);
35
36    while (!stop)
37    {
38      // Attente d'un bateau
39      printf("\t\t Quai %s %d > En attente %s\n", port_name, dock_index,
        sem_dock.semname);
40      wait_sem(sem_dock);
41
42      wait_sem(mutex_dock);
43      memcpy(&dock, shm_dock.pShm + (dock_index * sizeof(Dock)), sizeof(Dock));
44      signal_sem(mutex_dock);
45
46      //printf("Quai %s %d > Bateau %d a quai \n", port_name, dock_index, dock.
        boat_index);
47      sprintf(msg, "Bateau %d a quai", dock.boat_index);
48      print_boat(port_name, dock_index, dock.boat_index, msg);
49      // Ouverture MQ --- TODO: refactor with open_mq, ...
50      sprintf(mq1_name, "%s%d", MQ_TRUCKS, dock.boat_index);
51      sprintf(mq2_name, "%s%d", MQ_CARS_VANS, dock.boat_index);
52      mqd_trucks = mq_open(mq1_name, O_RDONLY | O_NONBLOCK);
53      mqd_cars_vans = mq_open(mq2_name, O_RDONLY | O_NONBLOCK);
54
55      if (mqd_trucks == (mqd_t) -1 || mqd_cars_vans == (mqd_t) -1)
56      {
57        perror("Error when opening MQs (trucks, cars, vans)");
58      }
59      if (mq_getattr(mqd_trucks, &attr1) == -1 || mq_getattr(mqd_cars_vans, &
        attr2) == -1)
60      {
61        perror("Erreur when mq_getattr\n");
62      }
63      buffer = malloc(attr1.mq_msgsize);
64      printf("[DEBARQUEMENT]\n");
65      printf("CURMSGS TRUCKS: %ld\n", attr1.mq_curmsgs);
66      if(attr1.mq_curmsgs > 0)
67      {
68        while(num_read != -1)
69        {
70          num_read = mq_receive(mqd_trucks, buffer, attr1.mq_msgsize, NULL);
71          printf("Sortie de %s\n", (char *)buffer);
72          nanosleep((struct timespec[]){{0, 250000000}}, NULL);
73        }
74      }
75      num_read = 0;
76      printf("CURMSGS CARS & VANS: %ld\n", attr2.mq_curmsgs);
77      if(attr2.mq_curmsgs > 0)
78      {
79        while(num_read != -1)
80        {
```

```
81          num_read = mq_receive(mqd_cars_vans, buffer, attr2.mq_msgsize, NULL);
82          printf("Sortie de %s\n", (char *)buffer);
83          nanosleep((struct timespec[]){{0, 250000000}}, NULL);
84        }
85      }
86      sem_gen_v.oflag = 0;
87      sprintf(sem_gen_v.semname,"%s%s", SEM_GEN_V, argv[1]);
88      open_sem(&sem_gen_v);
89      signal_sem(sem_gen_v);
90    }
91    return 0;
92  }
93
94  void init_ressources(Semaphore* sem_dock, Semaphore* mutex_dock, Shm*
        shm_dock, char* port_name, int dock_index, int nb_docks)
95  {
96    // SEM_DOCK
97    sem_dock->oflag = (O_CREAT | O_RDWR);
98      sem_dock->mode  = 0644;
99      sem_dock->value = 0;
100     sprintf(sem_dock->semname,"%s%s%d", SEM_DOCK, port_name, dock_index);
101
102   // MUTEX_DOCK
103   mutex_dock->oflag = O_RDWR;
104     mutex_dock->mode  = 0644;
105     mutex_dock->value = 1;
106     sprintf(mutex_dock->semname,"%s%s", MUTEX_DOCK, port_name);
107
108   // SHM_DOCK
109   shm_dock->sizeofShm = sizeof(Dock) * nb_docks;
110   shm_dock->mode = O_RDWR;
111   sprintf(shm_dock->shmName,"%s%s", SHM_DOCK, port_name);
112
113   sem_unlink(sem_dock->semname);
114
115   open_sem(sem_dock);
116   open_sem(mutex_dock);
117
118   open_shm(shm_dock);
119   mapping_shm(shm_dock, sizeof(Dock) * nb_docks);
120 }
121
122 void print_boat(char* port_name, int dock_index, int boat_index, char* msg)
123 {
124   char* color[] = {"\x1B[31m", "\x1B[32m", "\x1B[33m", "\x1B[34m", "\x1B[35m"
        , "\x1B[36m"};
125   char* reset = "\033[0m";
126
127   printf("\t\tQuai %s %d > %s%s%s\n", port_name, dock_index, color[boat_index
        ], msg, reset);
128 }
```

5.11 GenVehicle.h

```
1  #ifndef GENVEHICLE_H
2  #define GENVEHICLE_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <time.h>
8
```

```
9  #include "Ressources.h"
10 #include "Common.h"
11
12 #define TRUCK_PRIORITY 2
13 #define CAR_PRIORITY 1
14 #define VAN_PRIORITY 2
15
16 #define MAX_N_TRUCKS 5
17 #define MAX_N_CARS 10
18 #define MAX_N_VANS 15
19
20 Boat get_actual_boat(boat_p position, char* port, int nb_boats, Shm shm_boat)
       ;
21 char* getProp(const char *fileName, const char *propName);
22
23 #endif /* GENVEHICLE_H */
```

5.12 GenVehicle.c

```
1  #include "GenVehicle.h"
2
3  // argv[1] = port name
4  int main(int argc, char** argv)
5  {
6    Semaphore sem_gen_v;
7    Semaphore mutex_boat;
8    Semaphore mutex_sync;
9    Shm shm_boat;
10   Boat boat;
11   mqd_t mqd_trucks;
12   mqd_t mqd_cars_vans;
13
14   char buffer[MQ_MSGSIZE];
15   int i = 0;
16   int nb_trucks = 0, nb_cars = 0, nb_vans = 0;
17   int nb_boats = atoi(getProp(PROP_FILE, "nb_boats"));
18
19   srand(getpid());
20
21   // SEM_GEN_V
22   sem_gen_v.oflag = (O_CREAT | O_RDWR);
23   sem_gen_v.mode  = 0644;
24   sem_gen_v.value = 0;
25   sprintf(sem_gen_v.semname,"%s%s", SEM_GEN_V, argv[1]);
26   sem_unlink(sem_gen_v.semname);
27   open_sem(&sem_gen_v);
28
29   // Preparing mutex for shm_boat access
30   mutex_boat.oflag = O_RDWR;
31   mutex_boat.mode  = 0644;
32   mutex_boat.value = 1;
33   strcpy(mutex_boat.semname, MUTEX_BOAT);
34   open_sem(&mutex_boat);
35
36   // Preparing shm_boat access
37   shm_boat.sizeofShm = sizeof(Boat) * 6;
38   shm_boat.mode = O_RDWR;
39   strcpy(shm_boat.shmName, SHM_BOAT);
40   open_shm(&shm_boat);
41   mapping_shm(&shm_boat, sizeof(Boat) * 6);
42
43   while(1)
```

```
44  {
45    // Waiting signal_sem on sem_gen_v from Docks processes.
46    wait_sem(sem_gen_v);
47    printf("----> GEN VEHICLE FOR %s UNLOCKED\n", argv[1]);
48    // Waiting for access on shm_boat
49    wait_sem(mutex_boat);
50    boat = get_actual_boat(DOCK, argv[1], nb_boats, shm_boat);
51    signal_sem(mutex_boat);
52
53    // MUTEX_SYNC
54    mutex_sync.oflag = 0;
55    sprintf(mutex_sync.semname,"%s%d", MUTEX_SYNC, boat.index);
56    open_sem(&mutex_sync);
57
58    // Ouverture MQs
59    mqd_trucks = mq_open(boat.mq1.name, O_WRONLY);
60    mqd_cars_vans = mq_open(boat.mq2.name, O_WRONLY);
61
62    nb_cars = rand() % MAX_N_CARS + 1;
63    nb_vans = rand() % MAX_N_VANS + 1;
64    nb_trucks = rand()% MAX_N_TRUCKS + 1;
65
66    memset(buffer, 0, MQ_MSGSIZE);
67    printf("[BEGINNING BOARDING] > Boat %d\n", boat.index);
68    for(i = 0; i < nb_cars; i++)
69    {
70      sprintf(buffer, "Car %d", i + 1);
71      if(mq_send(mqd_cars_vans, buffer, strlen(buffer), CAR_PRIORITY) == -1)
72      {
73        mq_close(mqd_cars_vans);
74        mq_unlink(boat.mq1.name);
75        perror("Error occured when mq_send (cars & vans)\n");
76        exit(EXIT_FAILURE);
77      }
78      printf("%s on board\n", buffer);
79      // Sleep 1/4s -- TODO Paramétrable.
80      nanosleep((struct timespec[]){{0, 250000000}}, NULL);
81    }
82    printf("\t%d cars entered the boat %d.\n", nb_cars, boat.index);
83    for(i = 0; i < nb_vans; i++)
84    {
85      sprintf(buffer, "Van %d", i);
86      if(mq_send(mqd_cars_vans, buffer, strlen(buffer), VAN_PRIORITY) == -1)
87      {
88        mq_close(mqd_cars_vans);
89        mq_unlink(boat.mq1.name);
90        perror("Error occured when mq_send (cars & vans)\n");
91        exit(EXIT_FAILURE);
92      }
93      printf("%s on board\n", buffer);
94      // Sleep 1/4s
95      nanosleep((struct timespec[]){{0, 250000000}}, NULL);
96    }
97    printf("\t%d vans entered the boat %d.\n", nb_cars, boat.index);
98    for(i = 0; i < nb_trucks; i++)
99    {
100     sprintf(buffer, "Truck %d", i + 1);
101     if(mq_send(mqd_trucks, buffer, strlen(buffer), TRUCK_PRIORITY) == -1)
102     {
103       mq_close(mqd_trucks);
104       mq_unlink(boat.mq2.name);
105       perror("Error occured when mq_send (trucks)\n");
106       exit(EXIT_FAILURE);
```

```
107         }
108         printf("%s on board\n", buffer);
109         nanosleep((struct timespec[]){{0, 250000000}}, NULL);
110       }
111       printf("\t%d trucks entered the boat %d.\n", nb_trucks, boat.index);
112       printf("[ENDING BOARDING] for Boat [%d]", boat.index);
113       // Récupération de la mutex_sync
114       mutex_sync.oflag = 0;
115       sprintf(mutex_sync.semname,"%s%d", MUTEX_SYNC, boat.index);
116       // Signal le bateau qu'il peut y aller
117       signal_sem(mutex_sync);
118     }
119
120     return 0;
121 }
122
123 // NB : Dupliquées de Port.c
124 Boat get_actual_boat(boat_p position, char* port, int nb_boats, Shm shm_boat)
125 {
126     int i;
127     int found;
128     char* ports_name[] = {"Douvre", "Calais", "Dunkerque"};
129     Boat tmp;
130     boat_d direction;
131
132     for (i = 0, found = 0; i < 3 && !found; i++)
133     {
134       if (strcmp(port, ports_name[i]) == 0)
135       {
136         found = 1;
137         direction = i + 1;
138         //printf("Port %s > Nom port %d\n", port, direction);
139       }
140     }
141
142     for (i = 0, found = 0; i < nb_boats && !found; i++)
143     {
144       memcpy(&tmp, shm_boat.pShm + (i * sizeof(Boat)), sizeof(Boat));
145       if (tmp.position == position && tmp.direction == direction)
146       {
147         found = 1;
148         printf("GenVehicle %s > Bateau trouve %d\n", port, tmp.pid);
149       }
150     }
151
152     return tmp;
153 }
154
155 char* getProp(const char *fileName, const char *propName)
156 {
157     FILE*    file = NULL;
158     char*    token = NULL;
159     char  line[128];
160     char  sep[2] = "=";
161     int    i;
162     int    loginFound = 0;
163
164     if ((file = fopen(fileName, "r")) == NULL)
165     {
166       perror("Opening file\n");
167       exit(errno);
168     }
169     else
```

```
170   {
171     while (fgets(line, sizeof line, file) != NULL)
172     {
173       token = strtok(line, sep);
174       i = 0;
175
176       while(token != NULL)
177       {
178         if (i == 0)
179         {
180           if (strcmp(token, propName) == 0)
181           loginFound++;
182         }
183         else if (i != 0 && loginFound == 1)
184         {
185           char *password = malloc(sizeof(char *) * 30);
186           strcpy(password, token);
187           fclose(file);
188           return password;
189         }
190         token = strtok(NULL, sep);
191         i++;
192       }
193     }
194   }
195
196   fclose(file);
197   return NULL;
198 }
```

6 Conclusion