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

```
typedef enum {SEA, ENTERS_PORT, DOCK, LEAVES_PORT} boat_p;
typedef enum {UNDEFINED, DOVER, CALAIS, DUNKERQUE} boat_d;

typedef struct Boat_t
{
  pid_t        pid;
  int        index;
  boat_p       position;
  boat_d       direction;
  int        state_changed;
  MessageQueue  mq1;
  MessageQueue  mq2;
} Boat;
```

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

```
typedef struct Dock_t
{
  int index;
  int boat_index;
} Dock;
```

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.

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

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

```
        // Premier voyage
        if (boat.direction == UNDEFINED)
          boat.direction = rand() % 3 + 1;
        // Les bateaux viennent d'un port
        else
        {
          if (boat.direction == CALAIS || boat.direction == DUNKERQUE)
            boat.direction = DOVER;
          else
            boat.direction =  rand() % (3 - 2 + 1) + 2;
        }

        port_name = ports_name[boat.direction - 1];

        // Simulation de la traversée
```

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

```
        // 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++;
        memcpy(shm_arr.pShm, &cpt_arr, sizeof(int));
        signal_sem(mutex_arr);
        signal_sem(sem_port);

        sprintf(msg, "Devant l'entrée de %s", port_name);
        print_boat(index, msg);

        //pause();
        wait_sem(mutex_sync);
```

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

```
        wait_sem(mutex_dep);
        memcpy(&cpt_dep, shm_dep.pShm, sizeof(int));
        cpt_dep++;
        memcpy(shm_dep.pShm, &cpt_dep, sizeof(int));
        signal_sem(mutex_dep);
        signal_sem(sem_port);

        sprintf(msg, "Devant la sortie de %s", port_name);
        print_boat(index, msg);

        //pause();
        wait_sem(mutex_sync);
```

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("Port %s > En attente de bateau\n", port_name);
    wait_sem(sem_port);
```

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.

```
    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);

      printf("Port %s > Bateau %d sort\n", port_name, boat.pid);

      // Décrémente le compteur
      cpt_dep--;
      memcpy(shm_dep.pShm, &cpt_dep, sizeof(int));

```

```
14        signal_sem(mutex_dep);
15
16        // Envoie d'un signal au bateau
17        //kill(boat.pid, SIGUSR2);
18        mutex_sync.oflag = O_RDWR;
19        mutex_sync.mode  = 0644;
20        mutex_sync.value = 1;
21        sprintf(mutex_sync.semname,"%s%d", MUTEX_SYNC, boat.index);
22
23        open_sem(&mutex_sync);
24        sleep(1);
25        signal_sem(mutex_sync);
26        close_sem(mutex_sync);
27      }
```

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

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

3.5 Processus GenVehicle

4 Test du programme

5 Code source

5.1 Config.cfg

```
1 nb_ports=3
2 nb_boats=6
```

5.2 Common.h

```c
#ifndef COMMON_H
#define COMMON_H

#include "Ressources.h"

#define PROP_FILE "../Config.cfg"

#define MUTEX_BOAT "mutexBoat"
#define SHM_BOAT "shmBoat"
#define SHM_ARR "shmArr"
#define SHM_DEP "shmDep"
#define SHM_DOCK "shmDock"
#define SEM_PORT "semPort"
#define SEM_DOCK "semDock"
#define MUTEX_DEP "mutexDep"
#define MUTEX_DOCK "mutexDock"
#define MUTEX_ARR "mutexArr"
#define MUTEX_SYNC "mutexSync"
#define MUTEX_DOCK "mutexDock"

typedef enum {SEA, ENTERS_PORT, DOCK, LEAVES_PORT} boat_p;
typedef enum {UNDEFINED, DOVER, CALAIS, DUNKERQUE} boat_d;

typedef struct Boat_t
{
  pid_t        pid;
  int         index;
  boat_p       position;
  boat_d       direction;
  int         state_changed;
  MessageQueue  mq1;
  MessageQueue  mq2;
} Boat;

typedef struct Dock_t
{
  int index;
  int boat_index;
} Dock;

#endif /* COMMON_H */
```

5.3 Gestion.h

```c
#ifndef GESTION_H
#define GESTION_H

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>

//#include "Ressources.h"
#include "Common.h"

char* getProp(const char *fileName, const char *propName);
void init_ressources(int nb_boats);

Semaphore mutex_boat;

```

```
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 sigaction act;
25   char* msg = malloc(sizeof(msg));
26   act.sa_handler = handler;
27
28   srand(getpid());
29
30     /*if(sigaction(SIGUSR1, &act, 0) == -1)
31     {
32         perror("sigaction error");
33         exit(errno);
34     }
35
36     if(sigaction(SIGUSR2, &act, 0) == -1)
37     {
38         perror("sigaction error");
39         exit(errno);
40     }*/
41
42   // Initialisation des ressources
43   init_ressources(&mutex_sync, &mutex_boat, &shm_boat, index);
44
45   // Placement de l'état par défaut du bateau
46   boat.pid = getpid();
47   boat.index = index;
48   boat.position = SEA;
49   boat.direction = UNDEFINED;
50   boat.state_changed = 0;
51
```

```
52  wait_sem(mutex_boat);
53  memcpy(shm_boat.pShm + (index * sizeof(Boat)), &boat, sizeof(Boat));
54  signal_sem(mutex_boat);
55
56  while (!stop)
57  {
58    // Lecture de l'état du bateau
59    wait_sem(mutex_boat);
60    memcpy(&boat, shm_boat.pShm + (index * sizeof(Boat)), sizeof(Boat));
61    signal_sem(mutex_boat);
62
63    char* port_name;
64
65    switch(boat.position)
66    {
67      case SEA:
68        // Premier voyage
69        if (boat.direction == UNDEFINED)
70          boat.direction = rand() % 3 + 1;
71        // Les bateaux viennent d'un port
72        else
73        {
74          if (boat.direction == CALAIS || boat.direction == DUNKERQUE)
75            boat.direction = DOVER;
76          else
77            boat.direction =  rand() % (3 - 2 + 1) + 2;
78        }
79
80        port_name = ports_name[boat.direction - 1];
81
82        // Simulation de la traversée
83        int duration = rand() % (30 - 15 + 1) + 15;
84        sprintf(msg, "Traversée vers %s (%d secondes)", port_name, duration);
85        print_boat(index, msg);
86        sleep(duration);
87
88        boat.state_changed = 1;
89        boat.position = ENTERS_PORT;
90        break;
91
92      case ENTERS_PORT:
93      {
94        // Récupération des ressources du port
95        open_port_ressources(&sem_port, &mutex_dep, &mutex_arr, &shm_dep, &
    shm_arr, port_name);
96
97        wait_sem(mutex_arr);
98        memcpy(&cpt_arr, shm_arr.pShm, sizeof(int));
99        cpt_arr++;
100       memcpy(shm_arr.pShm, &cpt_arr, sizeof(int));
101       signal_sem(mutex_arr);
102       signal_sem(sem_port);
103
104       sprintf(msg, "Devant l'entrée de %s", port_name);
105       print_boat(index, msg);
106
107       //pause();
108       wait_sem(mutex_sync);
109
110       sprintf(msg, "Entree dans le port de %s", port_name);
111       print_boat(index, msg);
112       boat.position = DOCK;
113       break;
```

```
114         }
115         case DOCK:
116         {
117           int nb_docks = (strcmp(port_name, "Douvre") == 0) ? 3 : 2;
118           sprintf(msg, "Debut Embarquement");
119           print_boat(index, msg);
120 
121           // Création ressouces du quai
122           open_dock_ressources(&mutex_dock, &shm_dock, port_name, nb_docks);
123 
124           // Recherche de l'id du quai
125           int i;
126           int dock_index;
127           int found = 0;
128           Dock dock;
129           wait_sem(mutex_dock);
130           for (i = 0; i < nb_docks && !found; i++)
131           {
132             memcpy(&dock, shm_dock.pShm + (i * sizeof(Dock)), sizeof(Dock));
133             printf("Tour %d - Dock %d - Bateau : %d\n",i ,dock.index, dock.
    boat_index);
134             if (dock.boat_index == index)
135             {
136               dock_index = dock.index;
137               printf("Index trouvé : %d\n", dock_index);
138               found = 1;
139             }
140           }
141           signal_sem(mutex_dock);
142 
143           // Création de la sémaphore correspondante
144           sem_dock.oflag = O_RDWR;
145             sem_dock.mode  = 0644;
146             sem_dock.value = 0;
147             sprintf(sem_dock.semname, "%s%s%d", SEM_DOCK, port_name, dock_index
    );
148 
149           sprintf(msg, "Sem : %s", sem_dock.semname);
150           print_boat(index, msg);
151           open_sem(&sem_dock);
152 
153           // Debloque le quai
154           signal_sem(sem_dock);
155 
156           // TODO Reception d'un signal autorisant la sortie du port
157           pause();
158 
159           sprintf(msg, "Fin Embarquement");
160           print_boat(index, msg);
161           boat.position = LEAVES_PORT;
162           break;
163         }
164         case LEAVES_PORT:
165 
166           wait_sem(mutex_dep);
167           memcpy(&cpt_dep, shm_dep.pShm, sizeof(int));
168           cpt_dep++;
169           memcpy(shm_dep.pShm, &cpt_dep, sizeof(int));
170           signal_sem(mutex_dep);
171           signal_sem(sem_port);
172 
173           sprintf(msg, "Devant la sortie de %s", port_name);
174           print_boat(index, msg);
```

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

```
236     sprintf(mutex_dep->semname,"%s%s", MUTEX_DEP, port_name);
237
238     // MUTEX_ARR
239     mutex_arr->oflag = O_RDWR;
240     mutex_arr->mode  = 0644;
241     mutex_arr->value = 1;
242     sprintf(mutex_arr->semname,"%s%s", MUTEX_ARR, port_name);
243
244     // SHM_DEP
245     shm_dep->sizeofShm = sizeof(int);
246     shm_dep->mode = O_RDWR;
247     sprintf(shm_dep->shmName,"%s%s", SHM_DEP, port_name);
248
249     // SHM_ARR
250     shm_arr->sizeofShm = sizeof(int);
251     shm_arr->mode = O_RDWR;
252     sprintf(shm_arr->shmName,"%s%s", SHM_ARR, port_name);
253
254     open_sem(sem_port);
255     open_sem(mutex_dep);
256     open_sem(mutex_arr);
257
258     open_shm(shm_dep);
259     mapping_shm(shm_dep, sizeof(int));
260
261     open_shm(shm_arr);
262     mapping_shm(shm_arr, sizeof(int));
263 }
264
265 void open_dock_ressources(Semaphore* mutex_dock, Shm* shm_dock, char*
        port_name, int nb_docks)
266 {
267     mutex_dock->oflag = O_RDWR;
268     mutex_dock->mode  = 0644;
269     mutex_dock->value = 1;
270     sprintf(mutex_dock->semname,"%s%s", MUTEX_DOCK, port_name);
271
272     open_sem(mutex_dock);
273
274     // SHM_DOCK
275     shm_dock->sizeofShm = sizeof(Dock) * nb_docks;
276     shm_dock->mode = O_RDWR;
277     sprintf(shm_dock->shmName,"%s%s", SHM_DOCK, port_name);
278
279     open_shm(shm_dock);
280     mapping_shm(shm_dock, sizeof(Dock) * nb_docks);
281 }
282
283 void print_boat(int index, char* msg)
284 {
285     char* color[] = {"\x1B[31m", "\x1B[32m", "\x1B[33m", "\x1B[34m", "\x1B[35m"
          , "\x1B[36m"};
286     char* reset = "\033[0m";
287
288     printf("Bateau %d> %s%s%s\n", index, color[index], msg, reset);
289 }
290
291 void handler(int sig)
292 {
293     printf("Signal recu %d\n", sig);
294 }
```

5.7 Port.h

```
#ifndef PORT_H
#define PORT_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <string.h>

#include "Common.h"
#include "Ressources.h"

void create_processes(int nb_docks, char* port_name);
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);
Boat get_actual_boat(boat_p position, char* port, int nb_boats, Shm shm_boat)
    ;
char* getProp(const char *fileName, const char *propName);

#endif /* PORT_H */
```

5.8 Port.c

```
#include "Port.h"

int main(int argc, char** argv)
{
  Semaphore sem_port;
  Semaphore mutex_boat;
  Semaphore mutex_dep;
  Semaphore mutex_dock;
  Semaphore mutex_arr;
  Semaphore mutex_sync;

  Shm shm_dep;
  Shm shm_arr;
  Shm shm_boat;
  Shm shm_dock;

  Boat boat;
  char* port_name = argv[1];
  int cpt_arr = 0;
  int cpt_dep = 0;
  int stop = 0;
  int nb_boats = atoi(getProp(PROP_FILE, "nb_boats"));
  int nb_docks = 0;
    int i;

    sscanf(argv[2], "%d", &nb_docks);

  // Création des processus fils
  create_processes(nb_docks, port_name);

  // Initialisation des ressources
  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
    );
```

```
33
34    // Mise a 0 des compteurs
35    wait_sem(mutex_dep);
36    memcpy(shm_dep.pShm, &cpt_dep, sizeof(int));
37    signal_sem(mutex_dep);
38
39    wait_sem(mutex_arr);
40    memcpy(shm_arr.pShm, &cpt_arr, sizeof(int));
41    signal_sem(mutex_arr);
42
43    while (!stop)
44    {
45      // En attente de bateau
46      printf("Port %s > En attente de bateau\n", port_name);
47      wait_sem(sem_port);
48
49      // Compteur de depart
50      wait_sem(mutex_dep);
51      memcpy(&cpt_dep, shm_dep.pShm, sizeof(int));
52      if (cpt_dep > 0)
53      {
54        // Recherche du bateau
55        wait_sem(mutex_boat);
56        boat = get_actual_boat(LEAVES_PORT, port_name, nb_boats, shm_boat);
57        signal_sem(mutex_boat);
58
59        printf("Port %s > Bateau %d sort\n", port_name, boat.pid);
60
61        // Décrémente le compteur
62        cpt_dep--;
63        memcpy(shm_dep.pShm, &cpt_dep, sizeof(int));
64
65        signal_sem(mutex_dep);
66
67        // Envoie d'un signal au bateau
68        //kill(boat.pid, SIGUSR2);
69        mutex_sync.oflag = O_RDWR;
70        mutex_sync.mode  = 0644;
71        mutex_sync.value = 1;
72        sprintf(mutex_sync.semname,"%s%d", MUTEX_SYNC, boat.index);
73
74        open_sem(&mutex_sync);
75        sleep(1);
76        signal_sem(mutex_sync);
77        close_sem(mutex_sync);
78      }
79      else
80      {
81        signal_sem(mutex_dep);
82
83        // Recherche du bateau
84        wait_sem(mutex_boat);
85        boat = get_actual_boat(ENTERS_PORT, port_name, nb_boats, shm_boat);
86        signal_sem(mutex_boat);
87
88        printf("Port %s > Réservation pour le bateau %d\n", port_name, boat.
      index);
89
90        // TODO Reservation du quai
91        int found = 0;
92        wait_sem(mutex_dock);
93        for (i = 0; i < nb_docks && !found; i++)
94        {
```

```
95          Dock tmpDock;
96          memcpy(&tmpDock, shm_dock.pShm + (i * sizeof(Dock)), sizeof(Dock));
97          printf("Port %s > Bateau - %d Quai %d - %d\n", port_name, boat.index,
        tmpDock.index, tmpDock.boat_index);
98          // Recherche du premier quai disponible
99          if (tmpDock.boat_index == -1)
100         {
101           tmpDock.boat_index = boat.index;
102           memcpy(shm_dock.pShm + (i * sizeof(Dock)), &tmpDock, sizeof(Dock));
103           found = 1;
104         }
105       }
106       signal_sem(mutex_dock);
107
108       // Compteur d'arrivée
109       wait_sem(mutex_arr);
110       memcpy(&cpt_arr, shm_arr.pShm, sizeof(int));
111       if (cpt_arr > 0)
112       {
113         printf("Port %s > Bateau %d entre\n", port_name, boat.pid);
114
115         // Décrémente le compteur
116         cpt_arr--;
117         memcpy(shm_arr.pShm, &cpt_arr, sizeof(int));
118
119         signal_sem(mutex_arr);
120
121         // Envoie d'un signal au bateau
122         //kill(boat.pid, SIGUSR1);
123         mutex_sync.oflag = O_RDWR;
124         mutex_sync.mode  = 0644;
125         mutex_sync.value = 1;
126         sprintf(mutex_sync.semname,"%s%d", MUTEX_SYNC, boat.index);
127
128         open_sem(&mutex_sync);
129         sleep(1);
130         signal_sem(mutex_sync);
131         close_sem(mutex_sync);
132       }
133       else
134         signal_sem(mutex_arr);
135     }
136   }
137
138   return EXIT_SUCCESS;
139 }
140
141 void create_processes(int nb_docks, char* port_name)
142 {
143   pid_t child_pid;
144   int i;
145
146   // Création des quais
147   for (i = 0; i < nb_docks; i++)
148   {
149     if ((child_pid = fork()) < 0)
150     {
151       perror("fork failure");
152       exit(1);
153     }
154
155     if (child_pid == 0)
156     {
```

```
157         char* p = malloc(sizeof(p));
158         char* d = malloc(sizeof(d));
159         sprintf(p, "%d", i);
160         sprintf(d, "%d", nb_docks);
161         execl("Dock", "DOCK", port_name, p, d, NULL);
162       }
163     }
164
165     // Création de GenVehicle
166
167     if ((child_pid = fork()) < 0)
168     {
169       perror("fork failure");
170       exit(1);
171     }
172
173     if (child_pid == 0)
174     {
175       execl("GenVehicle", "GENVEHICLE", port_name, NULL);
176     }
177 }
178
179 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)
180 {
181     // MUTEX_BATEAU
182     mutex_boat->oflag = O_RDWR;
183       mutex_boat->mode  = 0644;
184       mutex_boat->value = 1;
185       strcpy(mutex_boat->semname, MUTEX_BOAT);
186
187     // SEM_PORT
188     sem_port->oflag = (O_CREAT | O_RDWR);
189       sem_port->mode  = 0644;
190       sem_port->value = 0;
191       sprintf(sem_port->semname,"%s%s", SEM_PORT, port_name);
192
193     // MUTEX_DEP
194     mutex_dep->oflag = (O_CREAT | O_RDWR);
195       mutex_dep->mode  = 0644;
196       mutex_dep->value = 1;
197       sprintf(mutex_dep->semname,"%s%s", MUTEX_DEP, port_name);
198
199     // MUTEX_DOCK
200     mutex_dock->oflag = (O_CREAT | O_RDWR);
201       mutex_dock->mode  = 0644;
202       mutex_dock->value = 1;
203       sprintf(mutex_dock->semname,"%s%s", MUTEX_DOCK, port_name);
204
205     // MUTEX_ARR
206     mutex_arr->oflag = (O_CREAT | O_RDWR);
207       mutex_arr->mode  = 0644;
208       mutex_arr->value = 1;
209       sprintf(mutex_arr->semname,"%s%s", MUTEX_ARR, port_name);
210
211     // SHM_DEP
212     shm_dep->sizeofShm = sizeof(int);
213     shm_dep->mode = O_CREAT | O_RDWR;
214     sprintf(shm_dep->shmName,"%s%s", SHM_DEP, port_name);
215
216     // SHM_ARR
```

```
217   shm_arr->sizeofShm = sizeof(int);
218   shm_arr->mode = O_CREAT | O_RDWR;
219   sprintf(shm_arr->shmName,"%s%s", SHM_ARR, port_name);
220
221   // SHM_DOCK
222   shm_dock->sizeofShm = sizeof(Dock) * nb_docks;
223   shm_dock->mode = O_CREAT | O_RDWR;
224   sprintf(shm_dock->shmName,"%s%s", SHM_DOCK, port_name);
225
226   shm_boat->sizeofShm = sizeof(Boat) * nb_boats;
227   shm_boat->mode = O_RDWR;
228   strcpy(shm_boat->shmName, SHM_BOAT);
229
230   sem_unlink(sem_port->semname);
231   sem_unlink(mutex_dep->semname);
232   sem_unlink(mutex_dock->semname);
233   sem_unlink(mutex_arr->semname);
234   sem_unlink(mutex_dock->semname);
235
236   open_sem(sem_port);
237   open_sem(mutex_dep);
238   open_sem(mutex_dock);
239   open_sem(mutex_arr);
240   open_sem(mutex_boat);
241   open_sem(mutex_dock);
242
243   open_shm(shm_boat);
244   mapping_shm(shm_boat, sizeof(Boat) * nb_boats);
245
246   open_shm(shm_dock);
247   mapping_shm(shm_dock, sizeof(Dock) * nb_docks);
248
249   open_shm(shm_dep);
250   mapping_shm(shm_dep, sizeof(int));
251
252   open_shm(shm_arr);
253   mapping_shm(shm_arr, sizeof(int));
254 }
255
256 Boat get_actual_boat(boat_p position, char* port, int nb_boats, Shm shm_boat)
257 {
258   // Parcours de la shm pour trouver le bateau concerné
259   int i;
260   int found;
261   char* ports_name[] = {"Douvre", "Calais", "Dunkerque"};
262   Boat *tmp = malloc(sizeof(Boat));
263   boat_d direction;
264
265   // Recherche le nom du port pour l'enum
266   for (i = 0, found = 0; i < 3 && !found; i++)
267   {
268     if (strcmp(port, ports_name[i]) == 0)
269     {
270       found = 1;
271       direction = i + 1;
272     }
273   }
274
275   // Recherche le bateau aux portes du port
276   for (i = 0, found = 0; i < nb_boats && !found; i++)
277   {
278     memcpy(tmp, shm_boat.pShm + (i * sizeof(Boat)), sizeof(Boat));
```

```
279     printf("Recherche: Bateau : %d - Port %d - Position %d\n", tmp->index,
        tmp->direction, tmp->position);
280     if (tmp->position == position && tmp->direction == direction)
281       found = 1;
282   }
283
284   return *tmp;
285 }
286
287 char* getProp(const char *fileName, const char *propName)
288 {
289   FILE*    file = NULL;
290   char*    token = NULL;
291   char  line[128];
292   char  sep[2] = "=";
293   int    i;
294   int    loginFound = 0;
295
296   if ((file = fopen(fileName, "r")) == NULL)
297   {
298     perror("Opening file\n");
299     exit(errno);
300   }
301   else
302   {
303         while (fgets(line, sizeof line, file) != NULL)
304         {
305         token = strtok(line, sep);
306       i = 0;
307
308         while(token != NULL)
309         {
310         if (i == 0)
311         {
312           if (strcmp(token, propName) == 0)
313             loginFound++;
314         }
315         else if (i != 0 && loginFound == 1)
316         {
317           char *password = malloc(sizeof(char *) * 30);
318           strcpy(password, token);
319           fclose(file);
320           return password;
321         }
322             token = strtok(NULL, sep);
323         i++;
324         }
325       }
326   }
327
328   fclose(file);
329   return NULL;
330 }
```

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
9  #include "Ressources.h"
10 #include "Common.h"
11
12 void init_ressources(Semaphore* sem_dock, Semaphore* mutex_dock, Shm*
       shm_dock, char* port_name, int dock_index, int nb_docks);
13
14 #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    Shm shm_dock;
8    int dock_index;
9    int nb_docks;
10   int stop = 0;
11   char* port_name = argv[1];
12
13   sscanf(argv[2], "%d", &dock_index);
14   sscanf(argv[3], "%d", &nb_docks);
15
16   init_ressources(&sem_dock, &mutex_dock, &shm_dock, port_name, dock_index,
       nb_docks);
17
18   // Mise a zero des info de la Shm
19   Dock dock;
20   dock.index = dock_index;
21   dock.boat_index = -1;
22   wait_sem(mutex_dock);
23   memcpy(shm_dock.pShm + (dock_index * sizeof(Dock)), &dock, sizeof(Dock));
24   signal_sem(mutex_dock);
25
26   while (!stop)
27   {
28     // Attente d'un bateau
29     printf("Quai %s %d > Attente d'un bateau\n", port_name, dock_index);
30     wait_sem(sem_dock);
31
32     wait_sem(mutex_dock);
33     memcpy(&dock, shm_dock.pShm + (dock_index * sizeof(Dock)), sizeof(Dock));
34     signal_sem(mutex_dock);
35
36     printf("Quai %s %d > Bateau %d a quai \n", port_name, dock_index, dock.
       boat_index);
37
38   }
39   return 0;
40 }
41
42 void init_ressources(Semaphore* sem_dock, Semaphore* mutex_dock, Shm*
       shm_dock, char* port_name, int dock_index, int nb_docks)
43 {
44   // SEM_DOCK
45   sem_dock->oflag = (O_CREAT | O_RDWR);
46     sem_dock->mode  = 0644;
47     sem_dock->value = 0;
```

```
48      sprintf(sem_dock->semname,"%s%s%d", SEM_DOCK, port_name, dock_index);
49
50    // MUTEX_DOCK
51    mutex_dock->oflag = O_RDWR;
52      mutex_dock->mode  = 0644;
53      mutex_dock->value = 1;
54      sprintf(mutex_dock->semname,"%s%s", MUTEX_DOCK, port_name);
55
56    // SHM_DOCK
57    shm_dock->sizeofShm = sizeof(Dock) * nb_docks;
58    shm_dock->mode = O_RDWR;
59    sprintf(shm_dock->shmName,"%s%s", SHM_DOCK, port_name);
60
61    sem_unlink(sem_dock->semname);
62
63    open_sem(sem_dock);
64    open_sem(mutex_dock);
65
66    open_shm(shm_dock);
67    mapping_shm(shm_dock, sizeof(Dock) * nb_docks);
68  }
```

5.11 GenVehicle.h

```
1  #ifndef GENVEHICLE_H
2  #define GENVEHICLE_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #endif /* GENVEHICLE_H */
```

5.12 GenVehicle.c

```
1  #include "GenVehicle.h"
2
3  int main(int argc, char** argv)
4  {
5    //printf("NOM DU PORT : %s\n", argv[1]);
6    return 0;
7  }
```

6 Conclusion