2019-2020

Haute Ecole Economique et Technique

Projet OS

Gaël Dieuzeide
Brian Luk
Gaëtan Görtz
Florian Degives

Table des matières

Introduction :

Il nous est demandé de programmer un programme paramétré, en C sous Linux, simulant une course entière de Formule 1, depuis les séances d'essai jusqu'à la course finale.

Ce programme doit utiliser la mémoire partagée comme moyen de communication inter-processus et utiliser des sémaphores pour synchroniser l'accès à cette mémoire.

Il y a 20 voitures engagées dans le grand prix, comportant 20 numéros différents

La course est divisée en plusieurs parties : 3 séances d'essai libre (Px), 3 séances de qualification (Qx) : les deux premières séances éliminent chaque fois les 5 dernières voitures pour la course suivante et leur donne leur place respective (de 20 à 16 et ensuite de 15 à 11), conservant donc les 10 premières pour participer à la course finale des essais.

Le principe est le suivant, lors de notre simulation, nous devons, lors de chaque séance d'essais :

- Relever les temps dans chaque secteur à chaque passage de chaque voiture
- Savoir si une voiture est au stand (P), ou si elle est OUT (si out, on conserve son meilleur temps)
- Toujours savoir qui a le meilleur temps dans chacun des secteurs
- Faire un classement des voitures en fonction de leur tour le plus rapide et le conserver

Ensuite, lors de chaque séance de qualification, nous devons :

- Relever aussi les temps dans les 3 secteurs à chaque passage de chaque voiture
- Toujours savoir qui a le meilleur temps dans chaque secteur
- Savoir si une voiture est au stand ou OUT (si out, conserver son meilleur temps)
- Faire un classement en fonction du tour complet le plus rapide
- A la fin de Q1, 15 voitures qualifiées, 5 éliminées, qui auront les places de 20 à 16
- A la fin de Q2, 10 voitures qualifiées, 5 éliminées qui auront les places de 15 à 11
- A la fin de Q3, les voitures seront placées de 10 à 1 en fonction de leur classement

Lors de la course :

- Classement dynamique
- Toujours relever les temps de chaque voiture dans les 3 secteurs
- Savoir qui a le tour le plus rapide
- Savoir si la voiture est au stand, ou si elle est out. Ici les voitures OUT seront directement mises en bas de classement
- Conserver le classement final et le tour le plus rapide

Remarque : Le stand (S) se trouve toujours dans le secteur 3

Analyse :

Le corps contient 7 fonctions.

1. ***Ramdom*** permet de générer un nombre aléatoire compris entre les deux paramètres de la fonction. Ce code est utilisé pour déterminer les temps des secteurs mais aussi le caractère aléatoire de l'accès aux stands ou des crashs des voitures.
2. ***Secteur*** permet de générer un temps aléatoire pour un secteur. Cette méthode comprend un compteur de secteur afin que chaque temps de secteur calculé soit unique.
3. ***Pit*** et ***Out*** sont des méthodes qui font appels à Ramdom pour déterminer si une voiture doit rentrer aux stands ou si elle est out.
4. ***Min*** est une méthode qui nous retourne le plus petit nombre entre les deux paramètres reçus. Cette méthode sera surtout utilisée pour déterminer les meilleurs tours et secteurs.
5. ***Simulation*** est une grosse fonction qui contient elle-même plusieurs sous-fonctions. On commence par créer et attacher la mémoire partagée qui permettra la communication des différentes parties du programme avec les voitures stockées. Après cela, on initialise les sémaphores et ses fonctions utiles comme wait, post ou lecture. La lecture correspond au moment où on accède à la section critique pour aller lire les informations en vue de les afficher ou les exporter vers un fichier. Nous trouvons ensuite une fonction de tri (sort()) inspirée d'un modèle internet et modifié pour convenir à nos besoins. Une fonction d'affichage (affichage()) qui va afficher à l'écran tous les x temps l'état actuel des différents concurrent.
   La fonction Simulation termine par la création des processus fils qui vont utiliser la mémoire partagée pour transmettre les informations sur les voitures et le détachement de la mémoire partagée.
   Une petite boucle for permet de gérer le taux de rafraîchissement de l'affichage, nous ne sommes plus parvenus à faire afficher à la fin de chaque tour.

6. ***ExportTXT*** sert, comme son nom l'indique, à exporter les tableaux finaux des scores des voitures dans des fichiers séparés en fonction de l'étape à laquelle on se trouve.

7. ***Main*** va gérer le lancement des différentes étapes de la courses. Elle va aussi utiliser d'autres variables de bases nécessaires au reste du code et utiliser les fonctions en les instanciant avec les variables. Le main se termine donc sur l'affichage final des résultats de la course.

Conclusion :

_____Ce projet était assez compliqué à réaliser, car au fur et à mesure de l'avancement de celui-ci, nous devions de plus en plus nous documenter pour premièrement comprendre, et ensuite intégrer ça correctement dans le code.

Il y a eu aussi des problèmes de gestion du temps, car plus le projet avançait, plus celui-ci demandait de temps, que ça soit au niveau de la compréhension, ou de l'application en terme de code. Nous nous sommes vite retrouvé avec du retard, ce qui a demandé à chacun de travailler un peu plus de son côté.

Ce travail a aussi amené une notion de responsabilité à chacun, dans le sens où celui-ci nous demandait de fournir différentes améliorations/avancements/corrections dans le code pour que le travail se déroule correctement.

En vue d'une amélioration du code, nous pourrions par exemple régler les problèmes de bestLap et bestSector qui dès qu'une voiture est out prennent la valeur de 0, ce qui n'est pas très réaliste. De plus au niveau de l'affichage, la boucle ne rafraichit pas toujours au bon moment ce qui crée des erreurs dans l'affichage du temps total qui ne tient compte que de un ou deux secteurs et non de l'entièreté du tour.

Nous tenons quand même à signaler que d'autres équipes nous ont aidées pour la gestion des sémaphores.

Conclusion personnelles :

Brian :

Malgré les difficultés rencontrées et un retard conséquent sur l'évolution du projet, nous avons finalement pu finir le code à temps. Nous sommes restés bloqués pendant des semaines sur l'affichage dynamique du tableau qui affichait les meilleurs temps des secteurs. Nous avions eu également des problèmes sur comment définir les meilleurs temps des différentes voitures.

Néanmoins, je pense que nous avions réalisé les consignes principales du projet et nous sommes fier de présenter un travail fini et fonctionnel.

Gaël :

Je suis plutôt satisfait du résultat final. Le projet me semblait insurmontable vu ce que j'avais réussi à faire à ma première deuxième mais avec de la motivation et la présence du groupe à tous les TP on a su rendre quelque chose qui correspond plus ou moins au travail demandé. Le fait de devoir beaucoup se documenter et avancer une seule fois par semaine pendant deux heures nous a un peu ralenti. Mais le groupe s'est entraidé et chacun y a mis du sien pour bien avancer. Il manque quelques détails mais le gros y est, et c'était mon objectif personnel en démarrant.

Gaëtan :

Après un départ des plus difficiles, nous nous sommes lancés dans ce projet et avons réussi à avoir un code à peu près fonctionnel. Le fait de devoir travailler avec un plus gros groupe que l'année passée a été plus compliqué pour moi car je ne connaissais pas bien mes partenaires mais petit à petit on a appris à se connaître et à exploiter les forces de chacun. A la fin de l'année, notre code ne faisait qu'une petite partie du résultat attendu mais grâce à l'aide de certains groupe et de beaucoup d'heure de recherche, nous avons pu résoudre certains problèmes. Bien que le code ne soit pas complet, je suis relativement content de ce que nous avons pu achever.

Florian :

Début assez difficile, nous avons longtemps "stagné" sur quelques problèmes. En y ajoutant la compréhension de la matière et la documentation, tout ceci nous a vite donné du retard. Donc nous avons dû travailler d'une manière plus rigoureuse pour arriver à terminer ce projet car les deux heures de TP ne nous donnaient clairement pas assez de temps pour se documenter et avancer sur le code en même temps.

Sources :

http://jean-luc.massat.perso.luminy.univ-amu.fr/ens/docs/thread-sem.html

https://sites.uclouvain.be/SystInfo/notes/Theorie/html/Threads/coordination.html

http://www.montefiore.ulg.ac.be/services/verif/cours/sp/html/mem-part2/shm.html

https://www.geeksforgeeks.org/wait-system-call-c/

https://www.developpez.net/forums/d1433633/c-cpp/c/debuter/fonction-tri-croissant-tri-decroissant/

https://stackoverflow.com/questions/327893/how-to-write-a-compare-function-for-qsort-from-stdlib

https://www.geeksforgeeks.org/comparator-function-of-qsort-in-c/

https://stackoverflow.com/questions/13372688/sorting-members-of-structure-array

https://stackoverflow.com/questions/13372688/sorting-members-of-structure-array

https://www.geeksforgeeks.org/structure-sorting-in-c/

Annexe :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <string.h>
#include <semaphore.h>
#include <sys/sem.h>


//Variables globales pour les temps
int sectorCompter; //Pour générer des temps aléatoires à chaque secteur en fonction du nombre d'appel à la fonction
int maxTime = 50000; //Temps max pour un secteur
int minTime = 40000; //Temps min pour un secteur
int minPit = 20000; //Temps min en pit
int maxPit = 25000; //Temps max en pit


//Définition d'une structure voiture
struct voiture{
   int ID; //ID du pilote
   double S1; //Temps S1
   double S2; //Temps S2
   double S3; //Temps S3
   double tour; //Temps tour = somme des temps
   double bestLap; //Temps meilleur tour
   int isOut; //Statut de out ou non (0=non, 1=oui)
   int estPit; //Nombre de passage en pit
   double totalTime; //Temps total de la course de cette voiture
}voiture[20];


//Cette méthode permet de générer un nombre aléatoire
//Utilisée pour le temps d'un secteur, les pits et crashs
int ramdom(int a, int b){
   srand(time(NULL) * (getpid()) * sectorCompter); //Changer le temps de la fonction actuel pour éviter que les temps se ressemblent
```

```
   return (rand()%(b-a)+a);
}


//Cette méthode permet de générer un temps de secteur
//Utilisée pour faire le temps des différents secteurs
double secteur(){
   if(!sectorCompter){ //Initialiser le compteur de secteur
      sectorCompter = 0;
   }
   sectorCompter++;
   usleep(30000); //Léger temps d'attente pour ralentir l'affichage
   return ((double)ramdom(minTime, maxTime)/(double)1000);
}


//Cette méthode permet de générer un int pour mettre la voiture en pit ou non
int pit(){
   if(ramdom(0,20) == 1){
   return 1;
   }
   else{
      return 0;
   }
}


//Cette méthode permet de gérer le out ou non de la voiture
int out(){
   if(ramdom(0,500) == 1){
      return 1; //Voiture sera out
   }
   else{
      return 0; //Voiture non out
   }
}


//Cette méthode permet de savoir quel est le temps le plus petit
double min(double a, double b){
      if (a > b) return b;
      if (a < b) return a;
      return a;
}
```

```
//Lance la simulation de la course
//nbreVoiture = nombre de voiture qui doivent tourner
//raceTime = définit le temps max que peut durer une course
//voitures = le tableau contenant les voitures
void Simulation(int nbreVoiture,double raceTime,struct voiture pilotes[20]){
  int shmId;
  int semId;
  struct sembuf operation;
  int nbLecteur;
  struct voiture *circuit; //Structure qui stocke les voitures actuellement en piste
  shmId = shmget(666, 20*sizeof(struct voiture), IPC_CREAT|0666);
  if(shmId == -1){ //Erreur lors de la création de la mémoire partagée
     printf("Erreur shmId = -1 ");
     exit(1);
  }
  circuit = shmat(shmId,0,0);
  if(circuit == (struct voiture*)-1){ //Erreur lors de l'accès à la mémoire partagée
     printf("Erreur shmat = -1");
     exit(1);
  }
  void initSem(){ //Initialise le sémaphore
     key_t seKey;
     semId = semget(semCle, 2, IPC_CREAT | 0666);
     if( semKey < 0 )
     {
        printf("Erreur semid\n");
        exit(0);
     }
     semctl(semId, 0, SETVAL, 1);
     semctl(semId, 1, SETVAL, 1);
  }
  void wait(int i){ //Met le thread en attente
     operation.sem_num = i;
     operation.sem_op = -1;
     operation.sem_flg = SEM_UNDO;
     semop(semId, &operation, 1);
  }
  void post(int i){ //Enlève le thread de la liste d'attente
     operation.sem_num = i;
     operation.sem_op = 1;
     operation.sem_flg = SEM_UNDO;
     semop(semId, &operation, 1);
  }
```

```
void commencerLecture(){ //Début section critique
    wait(0);
    nbLecteur++;
    if (nbLecteur == 1) {
        wait(1);
    }
    post(0);
}
void arreterLecture(){ //Fin section critique
    wait(0);
    nbLecteur--;
    if (nbLecteur == 0) {
        post(1);
    }
    post(0);
}
initSem();


//Cette méthode permet de sort les voitures prise sur internet et modifie pour notre projet
void sort(){
    struct voiture tempo; //Structure temporaire pour stocker les voitures en cours de tri
    commencerLecture(); //Section critique début
    memcpy(pilotes,circuit,nbreVoiture*sizeof(struct voiture));
    arreterLecture(); //Section critique fin
    for(int a = 0; a < nbreVoiture ; a++){
        for(int b = 0; b < nbreVoiture - 1 ; b++){
            if(pilotes[b].bestLap > pilotes[b+1].bestLap){
                tempo = pilotes[b+1];
                pilotes[b+1] = pilotes[b];
                pilotes[b] = tempo;
            }
        }//Fin for b
    }//Fin for a
}//Fin sort


//Affichage à l'écran des différents temps quand appelle
void affichage(){
    sort();
    double bestS1 = 99.0;
    double bestS2 = 99.0;
    double bestS3 = 99.0;
    double bestLap = 999.0;
    int j;
   system("clear"); //Nettoyer l'affichage entre chaque écran
```

```
//Affichage écran
 printf("|N°\t|S1\t|S2\t|S3\t|Tour\t\t|Best Tour\t|PIT\t|OUT\t|\n");
 printf("\n");
 for(j = 0; j < nbreVoiture ; j++){
    printf("|%d\t",pilotes[j].ID); //Affiche l'id du pilote
    if (pilotes[j].S1 == 0){ //Affiche le temps S1
       printf("|NULL\t");
    }
    else{
       printf("|%.3f\t",pilotes[j].S1);
    }
    if (pilotes[j].S2 == 0){ //Affiche le temps S2
       printf("|NULL\t");
    }
    else{
       printf("|%.3f\t",pilotes[j].S2);
    }
    if (pilotes[j].S3 == 0){ //Affiche le temps S3
       printf("|NULL\t");
    }
    else{
       printf("|%.3f\t",pilotes[j].S3);
    }
    if (pilotes[j].tour < 100.000){ //Affiche le temps du tour
       printf("|%.3f\t\t",pilotes[j].tour);
    }
    else{
       printf("|%.3f\t",pilotes[j].tour);
    }
    if (pilotes[j].bestLap < 100.000){ //Affiche le meilleur temps
       printf("|%.3f\t\t",pilotes[j].bestLap);
    }
    else{
       printf("|%.3f\t",pilotes[j].bestLap);
    }
    if(pilotes[j].estPit != 0){ //Affiche le nombre de pit du pilote
       printf("|%d\t",pilotes[j].estPit);
    }
    else{
       printf("|0\t");
    }
    if(pilotes[j].isOut == 1){ //Affiche si le pilote est out
       printf("|X\t|\n");
    }
    else{
       printf("|\t|\n");
```

```
        }
      }//Fin affichage
      //Définir les meilleurs temps
      //TO-DO Regler le probleme des best temps = 0
      for(j = 0; j<  nbreVoiture ; j++){
        if(bestS1 > pilotes[j].S1){
          bestS1 = pilotes[j].S1;
        }
        if(bestS2 > pilotes[j].S2){
          bestS2 = pilotes[j].S2;
        }
        if(bestS3 > pilotes[j].S3){
          bestS3 = pilotes[j].S3;
        }
        if(bestLap > pilotes[j].bestLap){
          bestLap = pilotes[j].bestLap;
        }
      }
      //Afficher les meilleurs temps
      printf("Best S1 : %.3f \n",bestS1);
      printf("Best S2 : %.3f \n",bestS2);
      printf("Best S3 : %.3f \n",bestS3);
      printf("Best tour : %.3f \n",bestLap);
  }

  //Faire tourner les voitures
  for(int i=0;i < nbreVoiture;i++){
    if(fork() == 0){ //Creation processus fils
      circuit = shmat(shmId,0,0);
      if(circuit == (struct voiture*)-1){ //Erreur lors de l'accès à la mémoire partagée
        printf("Erreur shmat = -1");
        exit(1);
      }
      circuit[i].totalTime = 0;
      circuit[i].ID = pilotes[i].ID;
      circuit[i].bestLap = 999;
      circuit[i].isOut = 0;
      circuit[i].estPit = 0;
      while(circuit[i].isOut == 0 && circuit[i].totalTime < raceTime){
        circuit[i].tour = 0;
        if(out() == 1){ //Si la voiture se crash on met les secteurs à 0 et on termine le
processus
          circuit[i].isOut = 1;
          circuit[i].S1 = 0;
          circuit[i].S2 = 0;
          circuit[i].S3 = 0;
```

```
                exit(0);
            }else{
              //Calcul temps S1 et mise en mémoire
                wait(1);
                circuit[i].S1 = secteur();
                circuit[i].totalTime += circuit[i].S1;
                circuit[i].tour += circuit[i].S1;
                post(1);
                //Calcul temps S2 et mise en mémoire
                wait(1);
                circuit[i].S2 = secteur();
                circuit[i].totalTime += circuit[i].S2;
                circuit[i].tour += circuit[i].S2;
                post(1);
                if(pit() == 1){ //Calcul pit
                    circuit[i].estPit += 1;
                    usleep(50000);
                    wait(1);
                    circuit[i].S3 = ((double)ramdom(minPit,maxPit)/(double)1000) + secteur();
//Ajout du temps dans pit
                    post(1);
                }else{
                    wait(1);
                    circuit[i].S3 = secteur();
                    usleep(50000);
                    post(1);
                }
                //Calcul temps S3 et mise en mémoire
                wait(1);
                circuit[i].tour += circuit[i].S3;
                circuit[i].totalTime += circuit[i].S3;
                circuit[i].bestLap = min(circuit[i].bestLap,circuit[i].tour);
                post(1);
                usleep(50000);
            }
        }
        exit(0);
    }
  }
  //Gère le rafraîchissement de l'affichage
  for(int compteur = 0; compteur< ((int)raceTime/130*3); compteur++){
    affichage();
    usleep(90000);
  }

  if(shmdt(circuit) == -1) { //Erreur lors de la libération de la mémoire partagée
```

```
      printf("Erreur shmdt");
      exit(1);
   }
}

//Créer les fichiers pour sauvegarder les résultats
int exportTXT(int step, int nbrVoiture,struct voiture pilotes[20]){
   FILE* fichier = NULL;
   //Pour définir les noms des fichiers
   switch (step){
      case 1 :
         fichier = fopen("Essai1.txt", "w+");
         //printf("Ecriture essai 1");
          break;
      case 2 :
         fichier = fopen("Essai2.txt", "w+");
         //printf("Ecriture essai 2");
          break;
      case 3 :
         fichier = fopen("Essai3.txt", "w+");
         //printf("Ecriture essai 3");
          break;
      case 4 :
         fichier = fopen("Qualif1.txt", "w+");
         //printf("Ecriture qualif 1");
          break;
      case 5 :
         fichier = fopen("Qualif2.txt", "w+");
         //printf("Ecriture qualif 2");
          break;
      case 6 :
         fichier = fopen("Qualif3.txt", "w+");
         //printf("Ecriture qualif 3");
          break;
      case 7 :
         fichier = fopen("Course.txt", "w+");
         //printf("Ecriture course");
          break;
   }
   if (fichier != NULL){
     //Début de l'écriture
     fprintf(fichier,"|N°\t|S1\t\t|S2\t\t|S3\t\t|Tour\t\t|Best\t\t|PIT\t|OUT\t|\n");
     fprintf(fichier,"\n");
     for(int j = 0; j < nbrVoiture ; j++){
        fprintf(fichier,"|%d\t",pilotes[j].ID); //Imprime le N°
        if (pilotes[j].S1 == 0){ //Imprime le temps S1
```

```
            fprintf(fichier,"|NULL\t");
        }
        else{
           fprintf(fichier,"|%.3f\t",pilotes[j].S1);
        }
        if (pilotes[j].S2 == 0){ //Imprime le temps S2
            fprintf(fichier,"|NULL\t");
        }
        else{
           fprintf(fichier,"|%.3f\t",pilotes[j].S2);
        }
        if (pilotes[j].S3 == 0){ //Imprime le temps S3
            fprintf(fichier,"|NULL\t");
        }
        else{
           fprintf(fichier,"|%.3f\t",pilotes[j].S3);
        }
        if (pilotes[j].tour == 0){ //Imprime le temps du tour
            fprintf(fichier,"|NULL\t\t");
        }
        else if(pilotes[j].tour<100.000){
            fprintf(fichier,"|%.3f\t\t",pilotes[j].tour);
        }
        else{
           fprintf(fichier,"|%.3f\t",pilotes[j].tour);
        }
        if (pilotes[j].bestLap < 100.000){ //Imprime le meilleur temps
            fprintf(fichier,"|%.3f\t\t",pilotes[j].bestLap);
        }
        else{
           fprintf(fichier,"|%.3f\t",pilotes[j].bestLap);
        }
        if(pilotes[j].estPit != 0){ //Imprime le nombre de pit du pilote
            fprintf(fichier,"|%d\t",pilotes[j].estPit);
        }
        else{
           fprintf(fichier,"|0\t");
        }
        if(pilotes[j].isOut == 1){ //Imprime si le pilote est out
            fprintf(fichier,"|X\t|\n");
        }
        else{
        fprintf(fichier,"|\t|\n");
        }
    }//Fin ecriture
fclose(fichier);
```

```
  }
  return 0;
}



//Main
int main (void){
  int id[20] = { 7, 99, 5, 16, 8, 20, 4, 55, 10, 26, 44, 77, 11, 18, 23, 33, 3, 27, 63,88}; //Id des
pilotes
  int lapTime = 130; //Temps moyen pour un tour afin de déterminer le temps de la course
  int totalLap = 45; //Nombres de tour
  struct voiture tri[20];
  for(int i = 0; i < 20 ; i++){ //Attribuer les id au tableau de voiture
    tri[i].ID = id[i];
  }
  //Essai 1
    //printf("\t\nEssai 1");
     sleep(1);
     Simulation(20,5400,tri);
     exportTXT(1,20,tri);
     sleep(2);
     system("clear");
  //Essai 2
    //printf("\t\nEssai2");
     sleep(1);
     Simulation(20,5400,tri);
     exportTXT(2,20,tri);
     sleep(2);
     system("clear");
  //Essai 3
    //printf("\t\nEssai 3");
     sleep(1);
     Simulation(20,3600,tri);
     exportTXT(3,20,tri);
     sleep(2);
     system("clear");
  //Qualif 1 toutes les voitures
    //printf("\t\nQualif 1");
     sleep(1);
     Simulation(20,1080,tri);
     exportTXT(4,20,tri);
     sleep(2);
     system("clear");
  //Qualif 2 les 15 premières voitures
```

```
        //printf("\t\nQualif 2");
        sleep(1);
        Simulation(15,900,tri);
        exportTXT(5,15,tri);
        sleep(2);
        system("clear");
    //Qualif 3 les 10 premières voitures
        //printf("\t\nQualif 3");
        sleep(1);
        Simulation(10,720,tri);
        exportTXT(6,10,tri);
        sleep(2);
        system("clear");
    //Course
        //printf("\t\nCourse");
        sleep(1);
        Simulation(20,(lapTime*totalLap),tri);
        exportTXT(7,20,tri);
        sleep(2);
        system("clear");
}
```