Rapport de projet

Administration système et réseaux

Os Travaux pratique

“Grand Prix de Formule 1”

Projet réalisé par
Robin Castermane
Igor Vandervelden
Victor Cotton

Année 2019-2020

Sommaire

Introduction

Dans le cadre de notre cours d'OS travaux pratique, nous devons réaliser une simulation d'un grand prix de Formule 1, depuis les séances d'essais du vendredi jusqu'à la course du dimanche, en passant par les essais du samedi et la séance de qualifications.

3 consignes nous sont données

- La réalisation doit se faire en c sous Linux.
- Il faut utiliser la mémoire partagée comme moyen de communication interprocessus.
- Les sémaphores doivent être utilisés pour synchroniser l'accès à la mémoir partagée.

Analyse du travail

Nous avons découpé le programme en plusieurs parties :

- Les essais
    - Durée 1h30 (P1, P2), que nous avons représentée en 5400 seconde dans le code.
    - Durée 1h00 (P3), que nous avons représentée en 3600 seconde dans le code.

- Les qualifications
    - Chaque qualification se déroule pendant une durée bien précise :
      Q1 : 18 min minutes = 1080 secondes
      Q2 : 15 min minutes = 900 secondes
      Q3 : 12 min minutes = 720 secondes

- La courses
    - Le nombre de tour est calculé selon la taille du circuit.

Pour la mémoire partagée, nous avons fait une structure : *tabVoitures* qui est composé de deux tableaux et deux entier :

- *tableauV* : représente une structure de voiture.
- *courseFinale* : est composé des positions des voitures pour la courses finale.
- *nbVoituresFini* : entier qui indique le nombre de voitures qui ont terminé les essais et les qualifications.
- *nbTourAFaire* : le nombre de tour que les voitures doivent faire pendant la course finale.

Conclusion

Réalisation de l’entièreté du travail demandé :

Nous n’avons pas utilisé de sémaphores dans notre programme. Manque de temps dû à notre organisation, nous n’avons pas réussis à le faire.

Principales difficultés rencontrées :

L’intégration des sémaphores a été une grande source de problèmes, dès que nous essayons de les intégrer de nouveaux problèmes survenaient.

Évolutions futures de l’application :

L'utilisation de sémaphores afin de gérer la concurrence entre processus devrait être faite.

Conclusions personnelles :

Igor

Le fait d’avoir continué le projet à été très satisfaisant malgré les problèmes de sémaphore. J’ai appris beaucoup de chose sur le C ainsi que sur l’organisation du travail en équipe.

Victor

Projet intéressant, avec sujet plaisant à travailler. Ce projet a contribué à ma compréhension et à l’utilisation du langage C dans un premier temps, mais aussi celles des sémaphores et variables partagées dans un second.
Je pense que ce projet respecte le résultat attendu, mais diverses améliorations restent tout de même nécessaires.

Robin

La plus grosse difficulté ici était de repartir sur base d'un projet déjà avancé, le temps de se mettre dans celui-ci n'étant pas évident.
En deuxième lieu, le langage de programmation de type C n'était pas un des plus facile à utiliser. Cependant, nous avons réussi à correctement conclure ce projet même si l'utilisation des sémaphore n'a pas été utilisé. De plus, même si le début du projet nous était déjà fourni, la progression reste fortement visible.
Pour finir, étant avec les mêmes étudiants pour la plupart des projets de groupe, la communication et le travail en équipe fût efficace.

Code

Le code se trouve sur ce lien-ci :
https://github.com/Gri097/ProjetOs2.0/tree/master/ProjetOS/Application

```c
#include "main.h"
#include "utils.c"


// Process pour un tour
//renvoie le temps du tour
double calculTour(struct Voiture *v){
  double tempsPit, tour = 0.0, section;
  if (out() == 1) {
    v->out = 1;
  }
  else{
    for(int i = 0; i < 3; i++){
      section = calculSection();
      if(i == 2){
        tempsPit = stand();
        if (tempsPit > 0) {
          v->pit += 1;
          section += tempsPit;
        }
      }
      tour += section;
      msleep(section);
      v->sections[i] = section;
      v->nbTour += 1;
      v->best[i] = meilleurTemps(section, v->best[i]);
    }
    v->best[3] = meilleurTemps(tour, v->best[3]);
    v->tourActuel= tour;
  }
  return tour;
}

void courseFinale(struct MemoirePartagee *tabVoitures, int i){
  struct Voiture v = tabVoitures->tableauV[i];
  struct Voiture *pointeurV = &v;
  initVarVoiture(pointeurV);


  v.nbTour= 0;

  sleep(1);

  while (v.nbTour < tabVoitures->nbTourAFaire && v.out == 0) { //tabVoitures->nbTourAFaire

    calculTour(pointeurV);
    tabVoitures->tableauV[i] = v;
    msleep(1000);
  }
  tabVoitures->nbVoituresFini += 1;
  exit(0);
}

void essaisEtQualifications(struct MemoirePartagee *tabVoitures, int i, double tempsTotal){
  struct Voiture v;
  struct Voiture *pointeurV = &v;
  if(tempsTotal >= 1080){ // soit essais soit qualif 1
    v = init_voiture(i);
  }
  else{
    v = tabVoitures->tableauV[i];
    initVarVoiture(pointeurV);

  }
  double tempsEnCours = 0;
  //Boucle course
  while (tempsEnCours < tempsTotal && v.out == 0) {
    tempsEnCours += calculTour(pointeurV);
    tabVoitures->tableauV[i] = v;
    if (tempsTotal == 5400 || tempsTotal == 3600) {
 sleep(1.5);
    }
    else{
 sleep(1.5);
    }
  }
  tabVoitures->nbVoituresFini += 1;
  exit(0);
}
```

```c
void afficheTab(struct MemoirePartagee *tabVoitures, int choix){
  struct MemoirePartagee mem = *tabVoitures;
  struct MemoirePartagee *pointeurMem = &mem;
  int finale = 0;
  if(choix == 2){
    triVoiture(pointeurMem->tableauV, 5);
    finale = 1;
  }
  else{
    triVoiture(pointeurMem->tableauV, 3);
  }
  system("clear");
  switch (choix) {
    case 0:
    printf("_____________________________________________________________________________________________\n");
    printf("|                              ESSAIS 1                              |\n");
    break;
    case 3:
    printf("_____________________________________________________________________________________________\n");
    printf("|                              ESSAIS 2                              |\n");
    break;
    case 1:
    printf("_____________________________________________________________________________________________\n");
    printf("|                              ESSAIS 3                              |\n");
    break;
    case 20:
    printf("_____________________________________________________________________________________________\n");
    printf("|                            QUALIFICATIONS 1                        |\n");
    break;
    case 15:
    printf("_____________________________________________________________________________________________\n");
    printf("|                            QUALIFICATIONS 2                        |\n");
    break;
    case 10:
    printf("_____________________________________________________________________________________________\n");
    printf("|                            QUALIFICATIONS 3                        |\n");
    break;
    case 2:

printf("_____________________________________________________________________________________________\n");

    printf("|                               COURSE FINALE                        |\n");
    break;
  }
  if (choix < 4) {
    choix = 20;
  }

  if(finale == 1){ //------------AFFICHE LA FINALE-------------------------

printf("|_____________________________________________________________________________________________|\n");

    printf("| Voiture  | Best Tour   | Tour Actuel     |   Section 1  |   Section 2  |   Section 3  | Pit  | Out  | Nb de tour|\n");
    printf("|----------|-------------|-----------------|--------------|--------------|--------------|------|------|-----------|\n");
    for(int i = 0; i < choix; i++){
        printf("|   %2d     | %.3f\"   |     %.3f\"   |   %.3f\"   |   %.3f\"   |   %.3f\"    |   %d   | %d    |   %2.0f      |\n", mem.tableauV[i].numero,
mem.tableauV[i].best[3], mem.tableauV[i].tourActuel, mem.tableauV[i].sections[0], mem.tableauV[i].sections[1], mem.tableauV[i].sections[2],
mem.tableauV[i].pit, mem.tableauV[i].out, mem.tableauV[i].nbTour);
        printf("|----------|-------------|-----------------|--------------|--------------|--------------|------|------|-----------|\n");
    }
  }
  else{ //----------------------AFFICHE LES QUALS/ESSAIS--------------
 printf("|_____________________________________________________________________________|\n");
    printf("| Voiture  | Best Tour   | Tour Actuel     |   Section 1  |   Section 2  |   Section 3  | Pit  | Out  |\n");
    printf("|----------|-------------|-----------------|--------------|--------------|--------------|------|------|\n");
    for(int i = 0; i < choix; i++){
        printf("|   %2d     | %.3f\"   |    %3.3f\"     |    %2.3f\"  |    %2.3f\"  |    %2.3f\"   |   %d   |   %d   |\n", mem.tableauV[i].numero,mem.tableauV[i].best[3],
mem.tableauV[i].tourActuel, mem.tableauV[i].sections[0],mem.tableauV[i].sections[1], mem.tableauV[i].sections[2], mem.tableauV[i].pit,
mem.tableauV[i].out);
        printf("|----------|-------------|-----------------|--------------|--------------|--------------|------|------|\n");
    }
  }
  printf("_____________________________________________________________________________\n");
  printf("|                                 Best                                      |\n");
  printf("|___________________________________________________________________________|\n");

  printf("|  Best    | Section 1    |\n");
  printf("|----------|--------------|\n");
```

```
  triVoiture(pointeurMem->tableauV, 0);
  printf("|   %2d     |  %.3f\"     |\n", mem.tableauV[0].numero, mem.tableauV[0].best[0]);
  printf("|----------|--------------|\n");
  printf("|  Best    | Section 2    |\n");
  printf("|----------|--------------|\n");
  triVoiture(pointeurMem->tableauV, 1);
  printf("|   %2d     |  %.3f\"     |\n", mem.tableauV[0].numero, mem.tableauV[0].best[1]);
  printf("|----------|--------------|\n");
  printf("|  Best    | Section 3    |\n");
  printf("|----------|--------------|\n");
  triVoiture(pointeurMem->tableauV, 2);
  printf("|   %2d     |  %.3f\"     |\n", mem.tableauV[0].numero, mem.tableauV[0].best[2]);
  printf("|----------|--------------|\n");
}

void triQualifications(struct MemoirePartagee *tabVoitures, int choix){
  triVoiture(tabVoitures->tableauV, 3); //tri de sorte que les voitures qui ont finies restent à leur place dans le tableau
  switch (choix) {
    case 20:
    for (int i = 15; i < 20; i++) {
      tabVoitures->courseFinale[i] = tabVoitures->tableauV[i];
      initVarVoiture(&tabVoitures->tableauV[i]);
      tabVoitures->tableauV[i].best[3] = i + 1000;
    }
    break;
    case 15:
    for (int i = 10; i < 15; i++) {
      tabVoitures->courseFinale[i] = tabVoitures->tableauV[i];
      initVarVoiture(&tabVoitures->tableauV[i]);
      tabVoitures->tableauV[i].best[3] = i + 1000;
    }
    break;
    case 10:
    for (int i = 0; i < 10; i++) {
      tabVoitures->courseFinale[i] = tabVoitures->tableauV[i];
      initVarVoiture(&tabVoitures->tableauV[i]);
      tabVoitures->tableauV[i].best[3] = i + 1000;
    }
    break;
  }

}

void fonctionPere(struct MemoirePartagee *tabVoitures, int choix){
  int calculNbFini;
  if(choix > 4){
    calculNbFini = choix;
  }
  else{
    calculNbFini = 20;
  }
  while(tabVoitures->nbVoituresFini <calculNbFini){
    msleep(500);
    afficheTab(tabVoitures, choix);
  }
  if (choix > 4) {
    triQualifications(tabVoitures, choix);
  }
  docRecap(tabVoitures, choix); //Pour avoir un document récapitulatif du tableau final
}

//Lance le programme
int main(int argc, char *argv[]){
  struct MemoirePartagee *tabVoitures;

  int shmid = shmget(1009, sizeof(tabVoitures), IPC_CREAT | 0666);
  tabVoitures = shmat(shmid, NULL, 0);
  int choix = 0;

  printf("Quelle est la taille du circuit ? (en km (int))");
  int tailleCircuit;
  scanf("%d", &tailleCircuit);

  tabVoitures->nbTourAFaire = calculNbTour(tailleCircuit);
 //printf("test: %d", tabVoitures->nbTourAFaire); //Nombre de tour à faire

  while(choix != 10){
    tabVoitures->nbVoituresFini = 0;

    printf("Que voulez-vous faire? \n");
```

```
printf("0 : pour lancer les essais 1 (P1)  \n");
printf("6 : pour lancer les essais 2 (P2)  \n");
printf("1 : pour lancer les essais 3 (P3)  \n");
printf("2 : pour lancer les qualifs 1 (Q1) \n");
printf("3 : pour lancer les qualifs 2 (Q2) \n");
printf("4 : pour lancer les qualifs 3 (Q3) \n");
printf("5 : pour lancer la course finale   \n");
printf("10 : pour quitter le programme \n");
  scanf("%d", &choix);

  switch(choix){
    case 0 : // P1 -> 1h30
    for(int i = 0; i < 20; i++){
      int pid = fork();
      if (pid < 0) {
        perror("ça forke pas\n");
        return -1;
      }
      else if (pid == 0) {
        srand(getpid());
        essaisEtQualifications(tabVoitures, i, 5400);
      }
    }
    fonctionPere(tabVoitures, 0); // C'est le père qui affiche
    break;
    case 6 : // P2 -> 1h30
    for(int i = 0; i < 20; i++){
      int pid = fork();
      if (pid < 0) {
        perror("ça forke pas\n");
        return -1;
      }
      else if (pid == 0) {
        srand(getpid());
        essaisEtQualifications(tabVoitures, i, 5400);
      }
    }
    fonctionPere(tabVoitures, 3);
    break;
    case 1 : //P3 -> 1h
    for(int i = 0; i < 20; i++){
      int pid = fork();
      if (pid < 0) {
        perror("ça forke pas\n");
        return -1;
      }
      else if (pid == 0) {
        srand(getpid());
        essaisEtQualifications(tabVoitures, i, 3600);
      }
    }
    fonctionPere(tabVoitures, 1);
    break;
    case 2 :
    for(int i = 0; i < 20; i++){
      int pid = fork();
      if (pid < 0) {
        perror("ça forke pas\n");
        return -1;
      }
      else if (pid == 0) {
        srand(getpid());
        essaisEtQualifications(tabVoitures, i, 1080);
      }
    }
    fonctionPere(tabVoitures, 20);
    break;
    case 3 :
    for(int i = 0; i < 15; i++){
      int pid = fork();
      if (pid < 0) {
        perror("ça forke pas\n");
        return -1;
      }
      else if (pid == 0) {
        srand(getpid());
        essaisEtQualifications(tabVoitures, i, 900);
      }
    }
```

```
    fonctionPere(tabVoitures, 15);
    break;
    case 4 :
    for(int i = 0; i < 10; i++){
     int pid = fork();
     if (pid < 0) {
      perror("ça forke pas\n");
      return -1;
     }
     else if (pid == 0) {
      srand(getpid());
      essaisEtQualifications(tabVoitures, i, 720);
     }
    }
    fonctionPere(tabVoitures, 10);
    break;
    case 5 :
    for(int i = 0; i < 20; i++){
     int pid = fork();
     if (pid < 0) {
      perror("ça forke pas\n");
      return -1;
     }
     else if (pid == 0) {
      srand(getpid());
      courseFinale(tabVoitures, i);
     }
    }
    fonctionPere(tabVoitures, 2);
    break;
   }
  }
  return 0;
}
```

```c
#include <stdio.h>
#include <strings.h>
#include <string.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <time.h>
#include <sys/msg.h>
#include <sys/shm.h>
#include <stdbool.h>
#include <printf.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

//Structure d'une voiture. Sera manipulée par les processus fils.
struct Voiture{
  int numero;
  int pit;
  int out;
  double best[4];
  double nbTour;
  double sections[3];
  double tourActuel;
};

//Structure de la mémoire partagée.
struct MemoirePartagee{
  struct Voiture tableauV[20];
  struct Voiture courseFinale[20];
  int nbVoituresFini;
  int nbTourAFaire;
};

// Process pour un tour
//renvoie le temps du tour
double calculTour(struct Voiture *v);

//Fonction qui gère la course Finale
void courseFinale(struct MemoirePartagee *tabVoitures, int i);

//Fonction qui gère les essais et les qualifications
void essaisEtQualifications(struct MemoirePartagee *tabVoitures, int i, double tempsTotal);

//Fonction qui permet d'afficher ce qu'il se passe
void afficheTab(struct MemoirePartagee *tabVoitures, int choix);

//Fonction qui permet de trier le tableau des voitures après chaque qualification
void triQualifications(struct MemoirePartagee *tabVoitures, int choix);

//Fonction qui gère l'affichage, les tris et la création des fichiers
void fonctionPere(struct MemoirePartagee *tabVoitures, int choix);
```

```c
#include "utils.h"

int calculNbTour(int tailleCircuit){
  int res = NB_KM / tailleCircuit;
  return res;
}

int msleep(long tms){
  struct timespec ts;
  int ret;

  if (tms < 0)
  {
    errno = EINVAL;
    return -1;
  }

  ts.tv_sec = tms / 1000;
  ts.tv_nsec = (tms % 1000) * 1000000;

  do {
    ret = nanosleep(&ts, &ts);
  } while (ret && errno == EINTR);

  return ret;
}

void swap(struct Voiture *x, struct Voiture *y){
  struct Voiture temp = *x;
  *x = *y;
  *y = temp;
}

void triVoiture(struct Voiture *voitures, int choix){
  for (int i = 0; i < 19; i++){
    for (int j = 0; j < 19 - i; j++){
      if(choix == 5){
        if (voitures[j].nbTour < voitures[j+1].nbTour){
          swap(&voitures[j], &voitures[j+1]);
        }
      }
      else{
        if (voitures[j].best[choix] > voitures[j+1].best[choix]){
          swap(&voitures[j], &voitures[j+1]);
        }
      }
    }
  }
}

//Renvoie un entier random entre min et min + max
double randomInt(int max, int min){
  return (rand() % max) + min;
}

//Renvoie un temps double au centiÃ¨me
double randomTime(int min, int max){
  double pEntiere = randomInt(min, max);
  double pDecimale = randomInt(999, 0);
  pDecimale /= 1000;
  return pEntiere + pDecimale;
}

double calculSection(){
  return randomTime(15, 35);
}
```

```
double meilleurTemps(double t1, double t2){
  double meilleur;
  if (t1 < t2) {
    meilleur = t1;
  }
  else{
    meilleur = t2;
  }
  return meilleur;
}

double stand(){
  double tempsAdditionnel = 0.0;
  if (randomInt(100,0) >= 97) {
    tempsAdditionnel = randomTime(10, 10);
  }
  return tempsAdditionnel;
}

int out(){
  int resultat = 0;
  if (randomInt(100,0) >= 99) {
    resultat = 1;
  }
  return resultat;
}

void initVarVoiture(struct Voiture *v){
  for(int i = 0; i < 3; i++){
    v->best[i] = 51; //secondes par section
  }
  v->best[3] = 151;
  v->pit = 0;
  v->out = 0;
}

struct Voiture init_voiture(int i){
  int numVoitures[20] = {7,99,5,16,8,20,4,55,10,26,44,77,11,18,23,33,3,27,6,85};
  struct Voiture v;
  struct Voiture *pointeurVoiture = &v;
  v.numero = numVoitures[i];
  initVarVoiture(pointeurVoiture);
  return v;
}

void docRecap(struct MemoirePartagee *tabVoitures, int choix){
    FILE *fichier = NULL;
    switch (choix) {
        case 0:
            fichier = fopen("RecapEssai1.txt", "a");
            break;
 case 3:
            fichier = fopen("RecapEssai2.txt", "a");
            break;
        case 1:
            fichier = fopen("RecapEssai3.txt", "a");
            break;
        case 20:
            fichier = fopen("RecapQualifications1.txt", "a");
            break;
        case 15:
            fichier = fopen("RecapQualifications2.txt", "a");
            break;
        case 10:
```

```
            fichier = fopen("RecapQualifications3.txt", "a");
            break;
        case 2:
            fichier = fopen("RecapCourseFinal.txt", "a");
            break;
    }
    if(fichier != NULL){
        struct MemoirePartagee mem = *tabVoitures;
        struct MemoirePartagee *pointeurMem = &mem;
        int finale = 0;
        if(choix == 2){
          triVoiture(pointeurMem->tableauV, 5);
          finale = 1;
        }
        else{
          triVoiture(pointeurMem->tableauV, 3);
        }
        system("clear");
        fprintf(fichier,"_____\n");
        switch (choix) {
            case 0:
                fprintf(fichier,"|            ESSAIS 1           |\n");
                break;
     case 3:
                fprintf(fichier,"|            ESSAIS 2           |\n");
                break;
            case 1:
                fprintf(fichier,"|            ESSAIS 3           |\n");
                break;
            case 20:
                fprintf(fichier,"|         QUALIFICATIONS 1        |\n");
                break;
            case 15:
                fprintf(fichier,"|         QUALIFICATIONS 2        |\n");
                break;
            case 10:
                fprintf(fichier,"|         QUALIFICATIONS 3        |\n");
                break;
            case 2:
                fprintf(fichier,"|          COURSE FINALE          |\n");
                break;
        }
        if (choix < 4) {
            choix = 20;
        }
        fprintf(fichier,"|_____|\n");
        fprintf(fichier,"| Voiture |   Tour    | Pit | Out |\n");
        fprintf(fichier,"|---------|-----------|-----|-----|\n");
        if(finale == 1){
          for(int i = 0; i < choix; i++){
            fprintf(fichier,"|   %2d    |   %2.0f     |  %d  |  %d  |\n", mem.tableauV[i].numero, (mem.tableauV[i].nbTour),
mem.tableauV[i].pit, mem.tableauV[i].out);
            fprintf(fichier,"---------------------------------|\n");
          }
        }
        else{
          for(int i = 0; i < choix; i++){
            fprintf(fichier,"|   %2d    |   %.3f\"  |  %d |  %d |\n", mem.tableauV[i].numero, mem.tableauV[i].best[3],
mem.tableauV[i].pit, mem.tableauV[i].out);
            fprintf(fichier,"|---------------------------------|\n");
          }
        }
        fprintf(fichier,"|  Best   | Section 1 |\n");
        fprintf(fichier,"|---------|-----------|\n");
```

```
        triVoiture(pointeurMem->tableauV, 0);
        fprintf(fichier,"|   %2d    |   %.3f\" |\n", mem.tableauV[0].numero, mem.tableauV[0].best[0]);
        fprintf(fichier,"|---------|----------|\n");
        fprintf(fichier,"|  Best   | Section 2 |\n");
        fprintf(fichier,"|---------|----------|\n");
        triVoiture(pointeurMem->tableauV, 1);
        fprintf(fichier,"|   %2d    |   %.3f\" |\n", mem.tableauV[0].numero, mem.tableauV[0].best[1]);
        fprintf(fichier,"|---------|----------|\n");
        fprintf(fichier,"|  Best   | Section 3 |\n");
        fprintf(fichier,"|---------|----------|\n");
        triVoiture(pointeurMem->tableauV, 2);
        fprintf(fichier,"|   %2d    |   %.3f\" |\n", mem.tableauV[0].numero, mem.tableauV[0].best[2]);
        fprintf(fichier,"|--------------------|\n");
        fclose(fichier);
    }
    else{
        printf("Ouverture du fichier recap impossible");
    }
}
```

```c
#include "utils.h"

int calculNbTour(int tailleCircuit){
  int res = NB_KM / tailleCircuit;
  return res;
}

int msleep(long tms){
  struct timespec ts;
  int ret;

  if (tms < 0)
  {
    errno = EINVAL;
    return -1;
  }

  ts.tv_sec = tms / 1000;
  ts.tv_nsec = (tms % 1000) * 1000000;

  do {
    ret = nanosleep(&ts, &ts);
  } while (ret && errno == EINTR);

  return ret;
}

void swap(struct Voiture *x, struct Voiture *y){
  struct Voiture temp = *x;
  *x = *y;
  *y = temp;
}

void triVoiture(struct Voiture *voitures, int choix){
  for (int i = 0; i < 19; i++){
    for (int j = 0; j < 19 - i; j++){
      if(choix == 5){
        if (voitures[j].nbTour < voitures[j+1].nbTour){
          swap(&voitures[j], &voitures[j+1]);
        }
      }
      else{
        if (voitures[j].best[choix] > voitures[j+1].best[choix]){
          swap(&voitures[j], &voitures[j+1]);
        }
      }
    }
  }
}

//Renvoie un entier random entre min et min + max
double randomInt(int max, int min){
  return (rand() % max) + min;
}

//Renvoie un temps double au centiÃ¨me
double randomTime(int min, int max){
  double pEntiere = randomInt(min, max);
  double pDecimale = randomInt(999, 0);
  pDecimale /= 1000;
  return pEntiere + pDecimale;
}

double calculSection(){
  return randomTime(15, 35);
}
```

```
double meilleurTemps(double t1, double t2){
  double meilleur;
  if (t1 < t2) {
    meilleur = t1;
  }
  else{
    meilleur = t2;
  }
  return meilleur;
}

double stand(){
  double tempsAdditionnel = 0.0;
  if (randomInt(100,0) >= 97) {
    tempsAdditionnel = randomTime(10, 10);
  }
  return tempsAdditionnel;
}

int out(){
  int resultat = 0;
  if (randomInt(100,0) >= 99) {
    resultat = 1;
  }
  return resultat;
}

void initVarVoiture(struct Voiture *v){
  for(int i = 0; i < 3; i++){
    v->best[i] = 51; //secondes par section
  }
  v->best[3] = 151;
  v->pit = 0;
  v->out = 0;
}

struct Voiture init_voiture(int i){
  int numVoitures[20] = {7,99,5,16,8,20,4,55,10,26,44,77,11,18,23,33,3,27,6,85};
  struct Voiture v;
  struct Voiture *pointeurVoiture = &v;
  v.numero = numVoitures[i];
  initVarVoiture(pointeurVoiture);
  return v;
}

void docRecap(struct MemoirePartagee *tabVoitures, int choix){
    FILE *fichier = NULL;
    switch (choix) {
        case 0:
            fichier = fopen("RecapEssai1.txt", "a");
            break;
 case 3:
            fichier = fopen("RecapEssai2.txt", "a");
            break;
        case 1:
            fichier = fopen("RecapEssai3.txt", "a");
            break;
        case 20:
            fichier = fopen("RecapQualifications1.txt", "a");
            break;
        case 15:
            fichier = fopen("RecapQualifications2.txt", "a");
            break;
        case 10:
```

```
            fichier = fopen("RecapQualifications3.txt", "a");
            break;
        case 2:
            fichier = fopen("RecapCourseFinal.txt", "a");
            break;
    }
    if(fichier != NULL){
        struct MemoirePartagee mem = *tabVoitures;
        struct MemoirePartagee *pointeurMem = &mem;
        int finale = 0;
        if(choix == 2){
          triVoiture(pointeurMem->tableauV, 5);
          finale = 1;
        }
        else{
          triVoiture(pointeurMem->tableauV, 3);
        }
        system("clear");
        fprintf(fichier,"_____\n");
        switch (choix) {
            case 0:
                fprintf(fichier,"|            ESSAIS 1           |\n");
                break;
     case 3:
                fprintf(fichier,"|            ESSAIS 2           |\n");
                break;
            case 1:
                fprintf(fichier,"|            ESSAIS 3           |\n");
                break;
            case 20:
                fprintf(fichier,"|         QUALIFICATIONS 1        |\n");
                break;
            case 15:
                fprintf(fichier,"|         QUALIFICATIONS 2        |\n");
                break;
            case 10:
                fprintf(fichier,"|         QUALIFICATIONS 3        |\n");
                break;
            case 2:
                fprintf(fichier,"|          COURSE FINALE          |\n");
                break;
        }
        if (choix < 4) {
            choix = 20;
        }
        fprintf(fichier,"|_____|\n");
        fprintf(fichier,"| Voiture |   Tour    | Pit | Out |\n");
        fprintf(fichier,"|---------|-----------|-----|-----|\n");
        if(finale == 1){
          for(int i = 0; i < choix; i++){
            fprintf(fichier,"|   %2d    |   %2.0f     |  %d  |  %d  |\n", mem.tableauV[i].numero, (mem.tableauV[i].nbTour),
mem.tableauV[i].pit, mem.tableauV[i].out);
            fprintf(fichier,"---------------------------------|\n");
          }
        }
        else{
          for(int i = 0; i < choix; i++){
            fprintf(fichier,"|   %2d    |   %.3f\"  |  %d |  %d |\n", mem.tableauV[i].numero, mem.tableauV[i].best[3],
mem.tableauV[i].pit, mem.tableauV[i].out);
            fprintf(fichier,"|---------------------------------|\n");
          }
        }
        fprintf(fichier,"|  Best   | Section 1 |\n");
        fprintf(fichier,"|---------|-----------|\n");
```

```
        triVoiture(pointeurMem->tableauV, 0);
        fprintf(fichier,"|   %2d    |   %.3f\" |\n", mem.tableauV[0].numero, mem.tableauV[0].best[0]);
        fprintf(fichier,"|---------|----------|\n");
        fprintf(fichier,"|  Best   | Section 2 |\n");
        fprintf(fichier,"|---------|----------|\n");
        triVoiture(pointeurMem->tableauV, 1);
        fprintf(fichier,"|   %2d    |   %.3f\" |\n", mem.tableauV[0].numero, mem.tableauV[0].best[1]);
        fprintf(fichier,"|---------|----------|\n");
        fprintf(fichier,"|  Best   | Section 3 |\n");
        fprintf(fichier,"|---------|----------|\n");
        triVoiture(pointeurMem->tableauV, 2);
        fprintf(fichier,"|   %2d    |   %.3f\" |\n", mem.tableauV[0].numero, mem.tableauV[0].best[2]);
        fprintf(fichier,"|--------------------|\n");
        fclose(fichier);
    }
    else{
        printf("Ouverture du fichier recap impossible");
    }
}
```