Université de Technologie de Compiègne

Devoir 2

Pascal Quach, Korantin Toczé

UV: *Maîtrise des systèmes informatiques (SR01)*
Date de rendu: *30 Décembre 2019*
Semestre: *A19*

Exercice 1 : Arbre généalogique de processus

Partie 1.

- Le `PID` du `shell` est 2000.
- Le `PID` du processus correspondant au parent est 2400.
- La numérotation des processus est séquentielle (incrémentation par 1).
- Un processus de `PID` *p* ne peut être exécuté qu'après la fin de l'exécution du processus de `PID` *p-1*.

Donner l'arbre généalogique des processus générés par chaque programme.

Programme 1.

```c
#include<unistd.h>
int main() {
    (fork() || fork()) && (fork() || fork());
}
```

Listing 1: Programme 1

Réponse. Le **ET** logique (`&&`) n'évalue pas la deuxième opérande si la première est fausse. Le **OU** logique (`||`) n'évalue pas la deuxième opérande si la première est vraie. La fonction `fork()` renvoie 0, donc faux, au processus fils et une valeur non nulle, donc vrai, au processus parent. La figure en annexe est utile pour observer le déroulement en détail du programme. L'arbre généalogique des processus est donc le suivant :

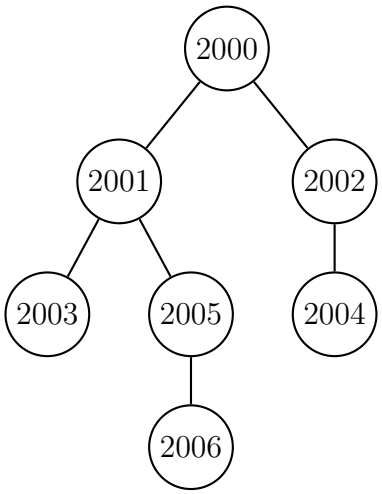


Figure 1: Arbre généalogique du programme 1

Programme 2.

```c
#include <unistd.h>
#include <stdlib.h>
int main() {
    int i = 0;
    fork();
    while (i < 4) {
        if (getpid() % 2 == 0) {
            fork();
        }
        i++;
    }
}
```

Listing 2: Programme 2

Réponse. La variable de la boucle `while`, `i` est présente dans le processus fils. L'arbre de déroulement du programme 2 est disponible en annexe. L'arbre généalogique des processus est donc le suivant :

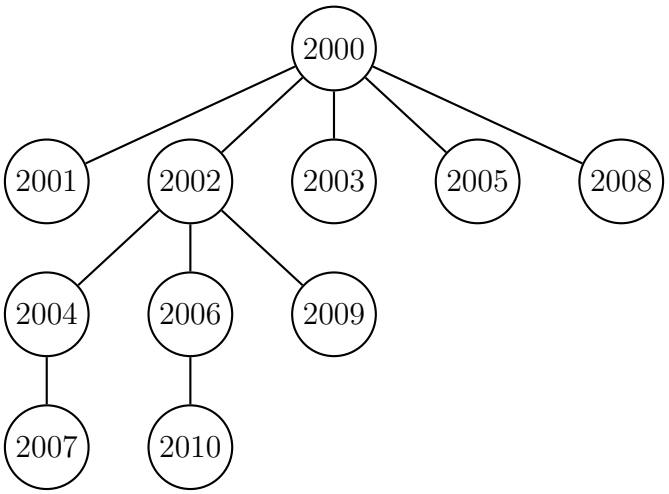
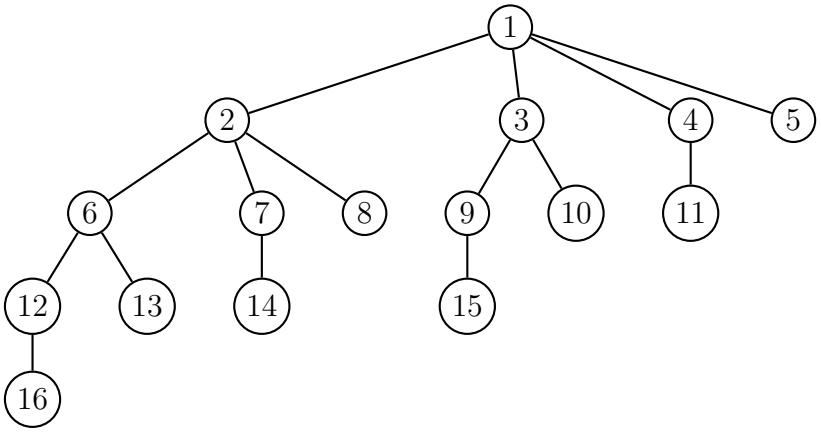


Figure 3: Arbre généalogique du programme 2

Partie 2.

En utilisant la fonction `fork()`, proposer un programme `C` permettant de générer cet arbre de processus. Pour chaque processus, afficher son `PID` et celui de son père.



Réponse. On réalise une boucle itérative pour exécuter `fork()` successivement.

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
    for (size_t i = 1; i < 5; i++) {
        int n = fork();
        if (n > 0) {
            printf("Je suis le parent.\n Le PID de mon père
               est : %d.\n Le PID de mon fils est : %d.\n",
               getppid(), getpid());
        }
        else {
            printf("Je suis le fils.\n Le PID de mon père
               est : %d.\n Le PID de mon fils est : %d.\n",
               getppid(), getpid());
        }
    }
    return EXIT_SUCCESS;
}
```

Listing 3: Partie 2 - Programme en `C`

Exercice 2 : Gestionnaire d'applications

On va programmer un gestionnaire d'applications personnalisé (`Application-Manager`). La liste des applications à lancer est stockée dans le fichier `list_appli.txt`. Vous disposez de quelques exemples d'applications (`power_manager.c`, `network_manager.c`, `get_time.c`).

Question 1. Écrire un programme `ApplicationManager.c` qui doit:

- Créer un ensemble de processus fils chacun est responsable à l'exécution d'une application.
- Lors de l'arrêt d'une application, informer l'utilisateur en lui affichant le nom de l'application terminée.
- S'arrêter après avoir fermer toutes les applications en cours d'exécution lors de la réception d'un ordre de mise en veille de la part de `power_manager` (signal `SIGUSR1`).

NB : Lorsque `ApplicationManager` reçoit un signal `SIGUSR1` de la part d'un autre processus, il ne ferme pas les applications.

Réponse.

Question 2. Modifier le programme `power_manager.c` pour envoyer le signal `SIGUSR1` à l'`ApplicationManager` lorsque l'utilisateur tape 1 dans le fichier `mise_en_veille.txt`.

Réponse.

Exercice 3

L'objectif de cet exercice est de paralléliser le calcul de la somme ou le produit de deux matrices. La somme et le produit doivent se faire en parallèle, donc une approche dans laquelle le processus père attend la fin du traitement réalisé par un fils afin d'en créer un autre ne sera pas acceptée. L'écriture et la lecture des fichiers doivent se faire en utilisant `fread()` et `fwrite()`.

Question 1.

Créez un programme `Somme.c`. Lors de l'implémentation de ce programme, supposez que les matrices sont déjà saisies dans les fichiers.

- Il reçoit en paramètres dans la fonction `main` :
  - deux chemins vers deux fichiers binaires contenant chacun une matrice carrée de la même taille
  - le nombre de lignes N d'une des matrices.
- Dans le programme `Somme.c`, on commence par lire les deux matrices.
- Ensuite, on crée N processus fils. Chaque processus fils i :
  - calcule la somme des $i$ème lignes des deux matrices;
  - communique le résultat au processus père en utilisant les pipes.
- Une fois que le processus père récupère la somme de chaque ligne, il affiche la matrice résultante.

Réponse. On commence par une lecture des fichiers passés en arguments du main, puis on les stocke dans un tableau. On crée ensuite n tableaux pour n pipes, un par processus. Ensuite, on crée un processus par ligne qui calcule la somme des éléments, puis l'on écrit dans le pipe de ce processus. Enfin, on lit tous les résultats et on les affiche.

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <wait.h>

int main(int argc, char const *argv[])
{
  FILE* f;
  f=fopen(argv[1],"rb");
  int n=atof(argv[3]);//recuperation du n
  int** tableaux,i,*ligne,N;
  int *matrice,*matrice2,j;

  matrice=malloc(n*n*sizeof(int));
  matrice2=malloc(n*n*sizeof(int));
  //lecture des matrices
  if (f!=NULL){
```

```
18      fread(matrice,sizeof(int),n*n,f);
    }else{
20      printf("ouverture fichier 1 impossible\n");
        return 0;
22  }
    fclose(f);
24


26  f=fopen(argv[2],"rb");
    if (f!=NULL){
28      fread(matrice2,sizeof(int),n*n,f);
    }else{
30      printf("ouverture fichier 1 impossible\n");
        return 0;
32  }
    fclose(f);
34 //affichage de l'opération
    for(i=0;i<n;i++){
36      for(j=0;j<n;j++){
            printf("%3d ",matrice[i*n+j]);
38      }
        if(i==n/2) printf(" + ");
40      else printf("   ");
        for(j=0;j<n;j++){
42          printf("%3d ",matrice2[i*n+j]);
        }
44      printf("\n");
    }
46 //génération des pipes et des tableaux associés
    tableaux=malloc(n* sizeof(int*));
48  for (i=0;i<n;i++){
        tableaux[i]=malloc(2*sizeof(int));
50      pipe(tableaux[i]);

52  }
 //génération des processus
54  for(i=0;i<n;i++){
        N = fork();
56      if (N==0){//processus fils
            printf("Calcul de la ligne numero %d\n",i);
58          ligne=malloc(n*sizeof(int));
            for(j=0;j<n;j++){//calcul d'une ligne du résultat
60              ligne[j]=matrice[i*n+j]+matrice2[i*n+j];
            }
62          write(tableaux[i][1],ligne,n*sizeof(int));//ecriture
               dans le pipe associé au processus
            free(ligne);
64          exit(0);
```

```
    }

  }
  //attente des processus fils
  while(wait(NULL)>0);
  //affichage du résultat
  printf("Resultat:\n");
  ligne=malloc(n*sizeof(int));
  for(i=0;i<n;i++){
    read(tableaux[i][0],ligne,n*sizeof(int));//lecture du
       pipe
    for(j=0;j<n;j++){
        printf("%3d ",ligne[j]);//affichage
    }
    printf("\n");
  }
  free(ligne);
  free(matrice);
  free(matrice2);
  for (i=0;i<n;i++){
    free(tableaux[i]);

  }
  free(tableaux);
  return 0;
}
```

Listing 4: Programme Somme.c

Question 2.

De même, créez un programme `Produit.c` qui reçoit les mêmes paramètres que le programme `Somme.c` mais qui effectue le produit de deux matrices.

- Dans le programme `Produit.c`, vous devez créer N processus fils, tout comme le programme `Somme.c`.
- Cependant, chaque processus fils va calculer le produit de la $i$ème ligne de la première matrice avec toutes les colonnes de la deuxième matrice.
- Ensuite, il communique le résultat au processus père.
- Une fois que le processus père récupère la somme de chaque ligne, il affiche la matrice résultante.

Réponse. Seul le calcul est différent du programme précédent, on ajoute une boucle pour parcourir toutes les colonnes de la première matrice et toutes les lignes de la seconde à chaque itération.

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <wait.h>

int main(int argc, char const *argv[])
{

  FILE* f;
  f=fopen(argv[1],"rb");
  int n=atof(argv[3]);
  int** tableaux,i,k;
  int *ligne,N;
  int *matrice,*matrice2,j;

  //creation des matrices
  matrice=malloc(n*n*sizeof(int));
  matrice2=malloc(n*n*sizeof(int));
  //lecture des matrices
  if (f!=NULL){
    fread(matrice,sizeof(int),n*n,f);
  }else{
    printf("ouverture fichier 1 impossible\n");
    return 0;
  }
  fclose(f);


  f=fopen(argv[2],"rb");
  if (f!=NULL){
    fread(matrice2,sizeof(int),n*n,f);
```

```
  }else{
    printf("ouverture fichier 1 impossible\n");
    return 0;
  }
  fclose(f);

  //creation des tableaux et des pipes associés
  tableaux=malloc(n* sizeof(int*));

  for (i=0;i<n;i++){
    tableaux[i]=malloc(2*sizeof(int));
    pipe(tableaux[i]);

  }

  //affichage de l'opération
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
        printf("%3d ",matrice[i*n+j]);
    }
    if(i==n/2) printf(" * ");
    else printf("   ");
    for(j=0;j<n;j++){
        printf("%3d ",matrice2[i*n+j]);
    }
    printf("\n");
  }

//calcul du résultat
  for(i=0;i<n;i++){
    N = fork();
    if (N==0){
        printf("Calcul de la ligne numero %d\n",i);
        ligne=malloc(n*sizeof(int));
        for(j=0;j<n;j++){
          for ( k = 0; k < n; k++) {
            ligne[j]+=matrice[i*n+k]*matrice2[k*n+j];//calcul
               du résultat
          }

        }
        write(tableaux[i][1],ligne,n*sizeof(int));//ecriture
           dans le pipe
        free(ligne);
        exit(0);
    }

  }
```

```
  //attente des processus fils
79 while(wait(NULL)>0);

81 //affichage des résultats
  printf("Resultat:\n");
83 ligne=malloc(n*sizeof(int));
  for(i=0;i<n;i++){
85   read(tableaux[i][0],ligne,n*sizeof(int));//lecture du
       pipe
     for(j=0;j<n;j++){
87       printf("%3d ",ligne[j]);//affichage
     }
89   printf("\n");
  }
91 free(ligne);
  free(matrice);
93 free(matrice2);
  for (i=0;i<n;i++){
95   free(tableaux[i]);

97 }
  free(tableaux);
99 return 0;
}
```

Listing 5: Programme Produit.c

Question 3.

Créez un programme `ManipMatrice.c` qui permet de :
- saisir des matrices ou de les générer aléatoirement,
- de les stocker dans des fichiers binaires,
- et de faire appel en utilisant `execv()`, aux programmes `Somme.c` ou `Produit.c` selon le choix de l'utilisateur.

Réponse. Dans cette question, on va demander à l'utilisateur d'entrer deux matrices (stockées dans les fichiers "matrice1" et "matrice2"). On supposera les deux programmes déja compilés dans le même dossier sous le nom "Somme" et "Produit". On génere alors les matrices aléatoirement ou selon les entrées utilisateurs, puis on appelle les programmes précédents en utilisant execv.

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

int main(){
  int choix,n,i,chiffre,j,k;
  char CHAINE[25];
  strcpy(CHAINE,"matrice1");
  srand(time(NULL));
  FILE * f;

  do{
    printf("Combien de lignes\n");
    scanf(" %d",&n)==0;
  }while(n<0);

  for(k=0;k<2;k++){//saisie des deux matrices
  do{
      printf("Tapez 1 pour saisir une matrice, 2 pour la
         générer aléatoirement\n");
      scanf(" %d",&choix);
      switch(choix){
        case 1:

          f=fopen(CHAINE,"wb");

          if(f==NULL){
            printf("Erreur lors de la creation du fichier
               %d\n",k);
            return 0;
          }
          for(i=0;i<n;i++){
            for (j = 0; j < n; j++) {
              scanf(" %d",&chiffre);
              fwrite(&chiffre,sizeof(int),1,f);//saisie de
                 tous les chiffre de la matrice
            }
          }
          fclose(f);
          break;

        case 2:
          f=fopen(CHAINE,"wb");
          if(f==NULL){
            printf("Erreur lors de la creation du fichier
```

```
                %d\n",k);
            return 0;
          }
          for(i=0;i<n;i++){
            for (j = 0; j < n; j++) {
              chiffre=rand()%200;//generation aleatoire de
                 tous les chiffres de la matrice
              fwrite(&chiffre,sizeof(int),1,f);
            }
          }
          fclose(f);
          break;


      }
    }while(choix<1||choix>2);//verification des choix
      strcpy(CHAINE,"matrice2");//pour creer le fichier
         matrice 2

}
//mise en place des arguments
char** tabarg,buffer[25];
tabarg=malloc(5*sizeof(char*));
for ( i = 0; i < 5; i++) {
  tabarg[i]=malloc(25*sizeof(char));
}
strcpy(tabarg[1],"matrice1");
strcpy(tabarg[2],"matrice2");
sprintf(buffer,"%d",n);
strcpy(tabarg[3],buffer);
tabarg[4]=NULL;

do{
printf("Que voulez vous
   faire?\n1:Addition\n2:Multiplication\n");
scanf(" %d",&choix);

switch(choix){
  case 2:
    printf("Appel de Produit\n" );
    strcpy(tabarg[0],"./Produit");
    execv("./Produit",tabarg);

    break;
  case 1:
    printf("Appel de Somme\n" );
    strcpy(tabarg[0],"./Somme");
    execv("./Somme",tabarg);
```

```
      break;
90
92    }
  }while(choix<1||choix>2);
94  for ( i = 0; i < 4; i++) {
    free(tabarg[i]);
96  }
  free(tabarg);
98  return 0;
  }
```

Listing 6: Programme ManipMatrice.c

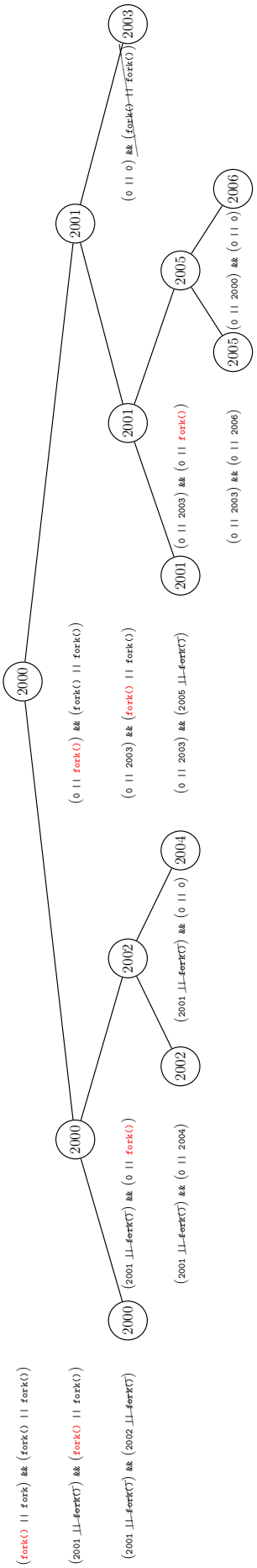


Figure 2: Arbre du déroulement des processus du programme 1

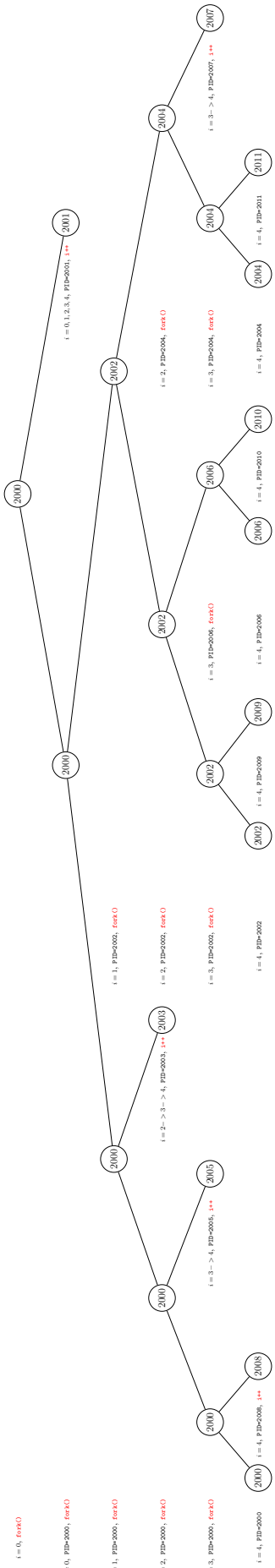


Figure 4: Arbre du déroulement des processus du programme 2