# Examen de Système d'exploitation (OS 302)

28 mai 2019

Enseignants : O. Aktouf, J. Marconot, K. Ndiaye

Documents de CM, TD et TP autorisés

## Partie machine

**(durée estimée : 1 heure)**

**Consignes générales**

Cet examen sur machine comporte 2 exercices.

Dans votre répertoire privé (celui dans lequel vous êtes positionné juste après votre connexion) vous devez créer un répertoire de nom **`os302VOTRENOM`** (« VOTRENOM » doit être remplacé par votre nom de famille en majuscules, sans accent, sans tiret, sans espace,…).

Dans le répertoire **`os302VOTRENOM`**, créez un répertoire **`EXO1`** et un autre **`EXO2`**.

Le(s) source(s) de l'exercice 1 doivent se trouver dans le répertoire `EXO1` et ceux de l'exercice 2 doivent être dans le répertoire `EXO2`.

**Exercice 1 : création de processus (8 points)**

L'objectif de cet exercice est d'effectuer la différence de 2 matrices carrées 2×2. Le programme principal permet la création d'un processus fils par opération élémentaire. Le processus père se synchronise alors sur la terminaison de tous les processus fils afin de s'assurer que tous les calculs sont bien finis et récupérer les résultats individuels.

Les fonctions nécessaires à cet exercice sont décrites ci-dessous. Certaines sont fournies, d'autres sont à compléter.

**Fichier func1.h**

```
#ifndef DM
// DM : dimensions matrice carrée
#define DM 2
// NF : nombre de processus fils
#define NF DM*DM

int **mat1, **mat2, **matdiff;
// mat1 et mat2 sont les 2 matrices lues au clavier. La matrice matdiff
correspond à la différence des 2 matrices mat1 et mat2.

/*** fonctions à compléter     ***/

void lecture_matrice(void);
//lecture des éléments des 2 matrices mat1 et mat2

int difference(int v1, int v2);
// différence de 2 entiers

void creation(pid_t tab[NF]);
// création de 4 processus fils dont chacun calculera une différence
// élémentaire

void resultat(pid_t tab[NF], int **matdiff) ;
// synchronisation du père avec ses 4 fils et récupération des résultats des
// calculs effectués par les fils

/*** fonctions fournies        ***/

int **allocate_matrice(int i, int j);
// allocation de la zone mémoire pour la création des matrices

void affiche_matrice(int **mat, int i, int j);
// affichage des éléments d'une matrice

void free_matrice(int** mat, int i);
// libération de la zone mémoire occupée par une matrice

void free_matrices(void);
//libération de la zone mémoire occupée par les 3 matrices de cet exercices
#endif
```

**Fichier func1.c**

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <wait.h>
#include "func1.h"

void lecture_matrice(void)
{ // Lit 2 matrices d'éléments entiers et de dimensions 2 par 2 – A compléter
printf("Lecture matrices \n");
}

int difference(int v1, int v2)
{ //renvoie la différence des 2 entiers transmis en arguments – A compléter
printf("Calcul différence élémentaire\n");

}

void creation(pid_t tab[NF])
{/*
Permet de créer 4 processus fils dont les PID sont stockés dans le tableau tab.
Chaque processus fils effectue une différence élémentaire :
-     Le fils 1 calcule la différence de mat1[0][0] et mat2[0][0]
-     Le fils 2 calcule la différence de mat1[0][1] et mat2[0][1]
-     Le fils 3 calcule la différence de mat1[1][0] et mat2[1][0]
-     Le fils 4 calcule la différence de mat1[1][1] et mat2[1][1]
Chaque processus fils transmet le résultat de son calcul au processus père au
moyen de la fonction exit(). A compléter */

printf("Création fils \n");
}

void resultat(pid_t tab[NF], int **matdiff)
{/*
Permet au processus père de se synchroniser sur la terminaison de ses 4 fils
dont les PID sont transmis dans le paramètre tab[NF]. Le père récupère le
résultat de chaque fils et construit la matrice résultat dans le paramètre
matdiff. A compléter. */

printf("résultats fils \n");
}

int **allocate_matrice( int i, int j)
{
  int **mat;
  int k;

  mat = malloc(i*sizeof(*mat));
  for (k=0; k<i; k++){
    mat[k] = malloc(j*sizeof(*mat[k]));
  }
  return mat;
}

void affiche_matrice(int** mat, int i, int j)
{int k, l;
```

```
  for (k=0; k<i; k++){
    for (l=0; l<j; l++){
      printf("%d \t", mat[k][l]);
    }
    printf("\n");
  }
}

void free_matrice(int** mat, int i)
{int k;

  for (k=0; k<i; k++)
  {
    free(mat[k]);
  }
  free(mat);
}

void free_matrices(){
  free_matrice(mat1,DM);
  free_matrice(mat2,DM);
  free_matrice(matdiff,DM);
}
```

**Fichier exo1.c**

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <wait.h>
#include "func1.h"

/** ne doit pas être modifié **/

int main (void)
{
  pid_t mes_fils[NF];
  int status;

  mat1=allocate_matrice(DM,DM);
  mat2=allocate_matrice(DM,DM);
  matdiff=allocate_matrice(DM,DM);

  lecture_matrice();
  creation(mes_fils);
  resultat(mes_fils, matdiff);
  affiche_matrice(matdiff,DM,DM);

return 0;
}
```

**Fichier Makefile**

```
all: exo1 clean

exo1: func1.o main.o
	gcc -o exo1 func1.o main.o

func1.o: func1.c func1.h
	gcc -o func1.o -c func1.c -g -Wall

main.o: exo1.c func1.h
	gcc -o main.o -c exo1.c -g -Wall

clean:
	rm -rf *.o
```

**Exercice 2 : communication par tubes ordinaires (4 points)**

Ecrie un programme en C qui permet à un processus père de communiquer avec un processus fils via un tube ordinaire. Le père crée un tube ordinaire puis un processus fils. Le fils envoie un message au père via le tube, il utilise le tube en écriture seulement. Le père attend le message du fils, il utilise le tube en lecture. Après la transmission du message le père et le fils se terminent. On fait l'hypothèse que les messages manipulés ne dépassent pas 100 caractères.

**Fichier func2.h**

```
#ifndef DNI
#define DNI

int p[2] ;
char message[100];
int nb_char;

/* fonctions à compléter */
int creation_tube(void) ;
// Création du tube par le processus père. La fonction retourne le code
// d'erreur de la création de tube.

int creation_fils(void) ;
// Création du processus fils par le processus père.

int ecriture_message (int des_edr, char *message) ;
//Ecriture du message du fils dans le tube.

int lecture_message (int des_lect, char *message) ;
//Lecture du message par le père dans le tube.

/* fonction fournie */
int lecture_chaine(char * chaine) ;
/*  lecture d'une chaîne de caractères au clavier. Cette chaîne représente le
message à transmettre. */

#endif
```

**Fichier func2.c**

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include "func2.h"

int creation_tube(void)
{
// crée un tube sans nom en utilisant le tableau  p.

printf("Création tube \n");

return 0; // - 1 en cas d'échec - A compléter
}
```

```
int creation_fils(void)
{ /*Création du processus fils par le processus père. La fonction renvoie le
code de retour de l'appel de création de processus. */

printf("Création fils \n");

return 0; // -1 en cas d'échec - A compléter
}

int ecriture_message (int des_edr, char *message)
{/* Ecriture du message du fils dans le tube. La fonction prend en entrée un
descripteur en écriture du tube des_edr et le message qui doit être écrit. La
fonction retourne le nombre de caractère écrits dans le tube. */
return -1; // à adapter
}

int lecture_message (int des_lect, char *message)

{/* Lecture du message par le père dans le tube, le message est affiché dans la
console. La fonction prend en entrée un descripteur en lecture du tube des_lect
et retourne le nombre de caractères lus. */

return -1; // à adapter
}

/* fonction fournie */
int lecture_chaine(char chaine[100])
{/* lecture d'une chaîne de caractères au clavier. Cette chaîne représente le
message à transmettre.
La fonction place la chaîne lue dans le paramètre chaine et renvoie le nombre de
caractères contenus dans la chaîne. */

printf("Veuillez saisir une chaîne de moins de 100 caractères : \n") ;
fgets(chaine, 100, stdin);
printf("chaine lue %s", chaine);
return (strlen(chaine)-1);
}
```

**Fichier exo2.c**

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include "func2.h"

/** ne doit pas être modifié **/

int main (void){
   nb_char = lecture_chaine(message);
   printf("nombre de caractères lus %d \n", nb_char);
   creation_tube();
   creation_fils();

   return 0;
}
```

**Fichier Makefile**

```
all: exo2 clean

exo2: func2.o main.o
	gcc -o exo2 func2.o main.o

func2.o: func2.c func2.h
	gcc -o func2.o -c func2.c -g -Wall

main.o: exo2.c func2.h
	gcc -o main.o -c exo2.c -g -Wall

clean:
	rm -rf *.o
```