Examen de INF106

Documents et calculatrices **non autorisés**. Certaines signatures de fonctions sont **fournies en annexe**. Vous pouvez répondre sur le sujet: n'oubliez pas de le rendre **et** d'y inquer votre nom.

I. Questions de cours (7 points)

Une seule réponse possible pour les QCM.

Question I.1 (1 point) – Définissez en une phrase ce qu'est un processus.

Question I.2 (1 point) – Quels sont les bus (ou lignes) utilisés pour communiquer entre processeur(s), mémoire, et unités d'échanges?

a) données, adresses, contrôle
b) arithmethique, données, mémoire
c) données, contrôle, signaux

Question I.3 (1 point) – On considère deux processus A et B. On parle d'un *interblocage* si:

a) A et B accèdent à une ressource en même temps
b) A attend que B libère un sémaphore et B attend que A libère un sémaphore
c) A et B sont dans l'état prêt, mais aucun des deux n'est actif

Question I.4 (1 point) – Lors qu'un processus fait un accès mémoire interdit (*segmentation fault*), le système d'exploitation:

a) notifie l'utilisateur du système avec un message d'erreur
b) termine le processus fautif
c) envoie un signal au processus fautif

Question I.5 (1 point) – La commande "`ln fich-v0.txt fich.txt`":

a) crée un lien symbolique nommé "fich.txt" qui pointe vers le fichier nommé "fich-v0.txt"
b) crée un lien dur ("hard link") nommé "fich.txt" qui pointe vers le même i-node pointé par le fichier "fich-v0.txt"
c) crée un lien symbolique nommé "fich-v0.txt" qui pointe vers le fichier nommé "fich.txt"

Question I.6 (1 point) – Quel appel système UNIX (fonction C) permet de créer un tube anonyme? Quel appel système UNIX (ou commande homonyme) permet de créer un tube nommé?

Question I.7 (1 point) – L'occupation maximale de mémoire des processus en execution dans un système d'exploitation qui dispose de mémoire virtuelle est limitée à:

a) 64 kilooctets
b) la taille de la memoire virtuelle, soit la mémoire physique de la machine (RAM) plus l'éspace disponible sur le disque dur
c) la taille de la mémoire physique de la machine (RAM)

II. Compréhension des services UNIX (7 points)

On considère pour les questions II.1 à II.4 le code suivant :

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

// L7 ********************************************************
int pid = -1;

void compute() { /* ... */ }

int main(int argc, char **argv) {
  int fd;
  pid_t res;

  fd = open("fichier", O_RDWR | O_CREAT, 0644); // L1 *********
  if (fd == -1) {
    perror("open");
    return 1;
  }

  // L5 *******************************************************

  res = fork();
  if (res == -1) {
    perror("fork");
    return 1;
  }
  if (res == 0) {
    dup2(fd, 1);
    execl("/usr/bin/ls", "ls", (char *) NULL); // L2 **********
    fprintf(stderr, "Erreur lors de exec!\n"); // L3 **********
    return 2; // L4 ***************************************
  }
  else {
    printf("Bonjour\n");
    compute(); // calcul complexe
    // L6 ******************************************************
    while (pid == -1){
      int status;
      pid = wait(&status);
      if (pid != -1 && WIFEXITED(status)) {
        if (WEXITSTATUS(status) == 2)
          printf("exit with error\n");
        else
          printf("exit ok.\n");
      }
    }
    return 0;
  }
}
```

Question II.1 (1 point) – Que renvoie la fonction open (ligne **L1**) ? Que se passe-t-il si le fichier « fichier » n'existe pas ?

Question II.2 (1 points) – En s'aidant de la page de manuel de la fonction `dup2`, dont un extrait est donné ci-dessous, expliquez où s'affiche la liste des fichiers du répertoire courant produite par l'exécution de la commande `ls` (ligne **L2**) en supposant que cette exécution fonctionne normalement (`execl` réussi).

```
NOM
dup2 - Dupliquer un descripteur de fichier

SYNOPSIS
#include <unistd.h>
int dup2(int oldfd, int newfd);

DESCRIPTION
dup2() crée une copie du descripteur de fichier oldfd. dup2()
transforme newfd en une copie de oldfd, fermant auparavant
newfd si besoin est.
```

Question II.3 (2 point) – Expliquer ce que fait la fonction `execl` (ligne **L2**) et pourquoi peut-on considérer ligne **L3** que la fonction `execl` a échoué sans tester explicitement le code de retour de celle-ci ? Où s'affiche l'éventuel message d'erreur produit par la ligne **L3**.

Question II.4 (2 points) – On suppose que la fonction *execl* échoue. La ligne **L4** est donc exécutée et le processus enfant renvoie la valeur 2. On souhaite que le processus parent récupère cette valeur et puisse afficher un message d'erreur.

a) Dans un premier temps, expliquez comment le code autour de la ligne **L6** permet au processus père de détecter l'erreur qui s'est produite dans le processus fils. Précisez notamment ce que fait la fonction `wait` et quelle information est transmise au père et comment ?

b) On souhaite, maintenant, améliorer la solution existante. On suppose que le processus parent fait des calculs potentiellement longs dans la fonction `compute` (avant la ligne **L6**). Nous aimerions que le processus père puisse détecter l'erreur signalée par le processus fils tout de suite après la terminaison de ce dernier. Les calculs du processus père doivent être interrompus juste le temps de récupérer la valeur renvoyée par le processus fils et éventuellement afficher un message d'erreur. Indiquez le code nécessaire aux alentours des lignes **L5** et **L7** et expliquez votre code brièvement.

Pour la question II.5 on considère un programme qui produit la sortie suivante :

```
Parent / Adresse de i = 0x7ffe116b2390, valeur de i = 42
Enfant / Adresse de i = 0x7ffe116b2390, valeur de i = 42
Enfant / Adresse de i = 0x7ffe116b2390, valeur de i = 43
Parent / Adresse de i = 0x7ffe116b2390, valeur de i = 42
```

Le code du programme est fourni sur la page suivante.

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

int main(int argc, char **argv) {
  int i = 42;
  pid_t res;

  res = fork();
  if (res == -1) {
    perror("fork");
    return 1;
  }
  if (res == 0) {
    printf("Enfant / Adresse de i = %p, valeur de i = %d\n", &i, i);
    i++;
    printf("Enfant / Adresse de i = %p, valeur de i = %d\n", &i, i);
  }
  else {
    printf("Parent / Adresse de i = %p, valeur de i = %d\n", &i, i);
    // Attente de la fin du processus enfant
    // ...
    printf("Parent / Adresse de i = %p, valeur de i = %d\n", &i, i);
  }
  return 0;
}
```

Question II.5 (1 point) – On constate que bien que l'adresse de la variable `i` soit la même pour les deux processus (`%p` dans `printf` permet d'afficher sous forme hexadécimale la valeur d'un pointeur, i.e. l'adresse vers laquelle il pointe), les deux processus voient des valeurs différentes à cette adresse. En vous aidant des notions vues en cours, expliquez comment deux processus peuvent voir des valeurs différentes à une même adresse.

III. Utilisation de services UNIX (6 points)

Les questions III.1 et III.2 sont indépendantes.

```
#include <fcntl.h>
#include <setjmp.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define NPOINTS 10
#define catch_NO_EXCEPTION 0
#define catch_NULL_POINTER_EXCEPTION SIGSEGV

typedef struct {int x, y;} point_t;
point_t * points[NPOINTS];

jmp_buf context;
int exception;

void handle_exception(int signum){
  signal(signum, SIG_DFL);
  /* L1 ************************************************************/
  // to complete




}

int main (int argc, char * argv[]) {
  for (int i = 0; i < NPOINTS; i++){
    /* L2 bloc try-catch ********************************************/
    // to complete



    switch(exception){
    default:
      /* bloc try */
      modify_point(i);
      break;
    case catch_NULL_POINTER_EXCEPTION:
      /* bloc catch NULL_POINTER_EXCEPTION */
      read_last_value(i);
      break;
    }
  }
```

Code de la question III.1.

Question III.1 (3 points) – Pour la question III.1 on considère le code ci-dessus :

On définit `point_t`, un type structuré décrivant un point et `points`, un tableau de pointeurs vers des structures `point_t`. Une procédure `modify_point` permet de modifier un élément du tableau `points` d'indice `i` passé en paramètre. Nous n'avons pas le code de cette procédure mais on sait qu'elle peut provoquer des déférencements vers des zones de mémoire invalides.

On souhaite implanter un mécanisme d'exception en C. Le texte ci-dessous décrit le fonctionnement attendu. On exécute en premier lieu le bloc try (**`/* bloc try */`**) en rentrant dans un switch et en empruntant l'alternative default. Pour ce faire, `exception` vaut `catch_NO_EXCEPTION` et on exécute `modify_point`.
Si `modify_point` provoque un déréférencement de mémoire, on rattrape l'erreur et on revient au niveau du switch (**`/* L2 bloc try-catch */`**) en positionnant `exception` à `catch_NULL_POINTER_EXCEPTION`. On va cette fois-ci emprunter l'alternative `catch_NULL_POINTER_EXCEPTION`, qui va traiter l'exception.

On utilisera une combinaison de `signal`, `setjmp` et `longjmp` vus dans le cours. Relire les spécifications de ces fonctions, le cas échéant.

Question III.1.a (1 point) – Dans le code ci-dessus, faites en sorte que l'erreur de déférencement de mémoire provoque l'appel à la fonction `handle_exception`.

Question III.1.b (1 point) – Dans le code ci-dessus, faites en sorte que le processus sauvegarde le point de retour en cas d'erreur se trouve juste avant l'exécution au début du switch.

Question III.1.c (1 point) – Dans le code ci-dessus, faites en sorte que l'appel à la fonction `handle_exception` fasse reprendre l'exécution au début du switch en positionnant `exception` sur la nature de l'exception (`catch_NULL_POINTER_EXCEPTION`).

Pour la question III.2 on considère le code ci-dessous :

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
void read_last_value(int i){
  int file;
  int rc;

  file = open("points_file.bin", O_RONLY);
  if (file < 0) {
     perror("cannot read default value for point %d\n", i);
     exit(1);
  }
  /* Question III.2a */
  // to complete

  /* Question III.2.b */
  // to complete

  close(file);
}
```

Question III.2 (3 points) – On sauvegarde régulièrement dans un fichier `"points_file.bin"` notre tableau `points`. Lorsque `modify_point` échoue dans son calcul, on souhaite lire dans le fichier, la dernière valeur de `points[i]` sauvegardée. On va implanter la fonction `read_last_value`.

Question III.2.a (1 point) – On rappelle que `points[i]` est un pointeur vers `point_t`. Tout d'abord, rendre la mémoire d'une allocation éventuellement existante de `points[i]`. Puis, allouer la structure pointée par `points[i]` et vérifier qu'elle est correcte.

Question 2.b (2 points) – Récupérer, dans ce fichier, **uniquement** le `i` ème point pour le stocker dans la structure pointée par `points[i]`. Vérifier que chacune des étapes s'est correctement effectuée sinon terminer le programme en indiquant de deux manières l'origine de l'erreur (sur le terminal et par la terminaison du processus).

Annexe

Voici, pour vous aider, quelques signatures de fonctions qui pourraient être utiles:

```
int fprintf(FILE *stream, const char *format, ...);
int printf(const char *format, ...);
void perror(const char *s);

typedef unsigned long int size_t;
typedef long int ssize_t;
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, void *buf, size_t count);
typedef long int off_t;
off_t lseek(int fd, off_t offset, int whence);

int lockf(int fd, int cmd, off_t len);
int open(const char *pathname, int flags);
int creat(const char *pathname, mode_t mode);

int setjmp(jmp_buf env);
void longjmp(jmp_buf env, int val);

pid_t fork(void);
pid_t wait(int *wstatus);
WEXITSTATUS(wstatus)
WTERMSIG(wstatus)
WIFEXITED(wstatus)
int execl(const char *path, const char *arg, ... /* (char  *) NULL */);

typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
unsigned int alarm(unsigned int seconds);
int kill(pid_t pid, int sig);

void *malloc(size_t size);
void *realloc(void *ptr, size_t size);
void free(void *ptr);
```