Examen : Programmation C et UNIX

Jehan-Antoine Vayssade
19 décembre 2025

Instructions

- **Durée :** 1h30m
- **Note totale / maximale :** 20 points, le total est volontairement supérieur, choisissez !
- **Des points bonus peuvent être cachés !**
- **Les réponses doivent être argumentées**
- **Prenez uniquement un stylo**
- **Pour la rédaction des codes :** en C99, qui compile !

Micro documentation :

```
pthread_create(pthread_t*, const ...attr_t*, void* (*)(void*), void*)
pthread_join(pthread_t, void**);
pthread_mutex_init(pthread_mutex_t*, const ...mutexattr_t*);

FILE *fopen(const char* path, const char* mode);
size_t fread(void* ptr, size_t size, size_t nmemb, FILE*);
size_t fwrite(const void* ptr, size_t size, size_t nmemb, FILE*);
void fclose(FILE* stream);

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

1 Questions de cours (5 points)

Répondez aux questions suivantes de manière concise et claire :

1. **(2 pt)** Expliquez les quatre étapes de la compilation d'un programme C (indice TD 1/2).
2. **(1 pt)** Quelle est la différence entre une variable déclarée `static` dans une fonction et une variable locale classique ?
3. **(2 pt)** Expliquez la difference entre programme, processus et thread.

2 Union et structures (5 points)

Quelle sont les tailles des variables @info.info1, @info.info2 et @info en mémoire ? Que valent les variables @info.info1.ia, @info.info1.ib, @info.info2.ia et @info.info2.ib à la fin de l'exécution du code suivant ? Attention il y a une différence de comportement entre les deux structures.

```
typedef union {
    struct {
        char ia;
        int  ib;
    } info1;
    struct {
        int  ib;
        char ia;
    } info2;
} Info;

Info info;
info.info1.ia = 5;
info.info1.ib = 2;
```

3 Alignement mémoire (7 pt)

Analysez le code, il peut etre utile d'ecrire un AST (Abstract Syntax Tree) pour comprendre une partie du code, ci-dessous et repondez aux questions :

a) **(2 pt)** Quelles sont les valeurs de a, b, c, d, *p et **pp à la fin de l'exécution ?

b) **(1 pt)** Que vaut `sizeof(tab)` et `sizeof(p)` sur une architecture 64 bits ? Justifiez.

c) **(1 pt)** Quelles valeurs seront affichées pour A, B, C, E et F ?

d) **(1 pt)** Pourquoi `*p` et `**pp` affichent-ils la même valeur à la ligne F ?

e) **(2 pt)** Que ce passe-t-il si on écrit `*(tab + 10) = 100;` ? Justifiez.

```
int tab[] = {10, 20, 30, 40, 50};
int *p = tab;
int **pp = &p;
int a = 5000, b = 23, c = 1, d = 2;

*((&b) + 1) = 80;
*((&b) - 1) = 42;

p++;
```

3.1 Fork et processus (5 points)

a) **(2 pt)** Expliquez le concept de **Copy-on-Write (CoW)** dans le contexte de `fork()`.

b) **(1 pt)** Que se passerait-il si on retirait l'appel à `wait(NULL)` ?

c) **(2 pts)** Modifiez le code pour que parent et enfant partagent la variable @compteur.

```
int main(void) {
    int compteur = 5;
    pid_t pid = fork();

    if (pid == 0) {
        compteur += 10;
    } else if (pid > 0) {
        wait(NULL);
        compteur += 20;
    }

    return 0;
}
```

4 Threads et synchronisation (8 points)

Analysez le code suivant et répondez aux questions :

a) **(1 pt)** Dans le contexte des threads, qu'est-ce qu'une **race condition** ?

b) **(2 pt)** Faites en sorte que les deux appels à `retrait` soient exécutés en parallèle.

c) **(1 pt)** Votre code contient-il un scénario d'exécution problématique ?

d) **(2 pts)** Quel pourrait être le solde final dans le ou les différents scénarios ? Justifiez.

e) **(2 pt)** S'il existe un problème, corrigez la fonction `retrait`.

```
long long solde = 1000;

void* retrait(void* arg) {
    long long montant = *(long long*)arg;

    if (solde >= montant)
        solde -= montant;
    else
        printf("Solde insuffisant\n");

    return NULL;
}

int main(void) {
    long long m1 = 800, m2 = 500;
    retrait(&m1), retrait(&m2);
    return 0;
}
```

5 Lecture et écriture d'image PPM (10 points)

Le format PPM P6 est un format d'image non compressé et est structuré ainsi :

— **En-tête (texte ASCII) :** le magic number `"P6"`, suivi de la largeur, la hauteur, et la valeur maximale (généralement 255), séparés par des espaces ou retours à la ligne.

— **Données (binaire) :** les pixels sont stockés en binaire, 3 octets par pixel (R, G, B).

Exemple d'en-tête d'un fichier PPM P6 de 640×480 pixels :

```
P6
640 480
255
<donnees binaires RGB...>
```

On vous donne la structure suivante pour représenter une image :

```
typedef struct {
    unsigned int width;
    unsigned int height;
    unsigned char* data; // rgb
} ImagePPM;
```

a) **(5 pt)** Écrivez la fonction `ImagePPM* read_ppm(const char* chemin)` qui :
   — Ouvre le fichier en mode lecture binaire
   — Lit l'en-tête (magic number, dimensions, valeur max)
   — Alloue la structure `ImagePPM` et le tableau de pixels
   — Lit les données binaires des pixels
   — Retourne `NULL` en cas d'erreur

b) **(4 pt)** Écrivez la fonction `int write_ppm(const char* chemin, ImagePPM* img)` qui :
   — Ouvre le fichier en mode écriture binaire
   — Écrit l'en-tête au format PPM P6
   — Écrit les données binaires des pixels
   — Retourne 0 en cas de succès, -1 en cas d'erreur

c) **(1 pt)** Écrivez la fonction `void free_ppm(ImagePPM* img)` qui libère la mémoire.

Fin du sujet

Relisez attentivement vos réponses avant de rendre votre copie.