Examen : C et UNIX

Dr. Jehan-Antoine Vayssade

12 juin 2025

Instructions

— **Durée :** 2 heures
— **Note totale / maximale :** 20 points
— **Des points bonus peuvent être cachés !**
— **Les réponses doivent être argumentées**
— **Prenez uniquement un stylo**
— **Attention aux pièges**

Compréhension du langage C

Question (1 point 5 min)

Quelle sera la sortie du code suivant ? Expliquer la différence fondamental.

```
if (~0) // opérateur tild
    printf("Affichage␣1\n");

if (-0) // opérateur minus
    printf("Affichage␣2\n");
```

Question (2 point 5 min)

Identifiez les zones mémoires utilisées par les variables suivantes. Expliquez leurs emplacements et durées de vie à l'aide d'un schéma.

```
static int counter = 0;

int main() {
    char buffer[10];
    double *value = (double*) malloc(sizeof(double));
}
```

Question (2 poin≈ 8 min)

Quelle est la taille des variables @u, @a et @b en mémoire ? Que peut-on dire des valeurs de @a et @b à la fin de lexécution ?

```
union Info {
    float a;
    int b;
};

Info u;
u.a = 5;
u.b = 2;
```

Question (3 points ≈ 15 min)

Écrivez une fonction en C qui itère sur une chaîne de caractères et remplace toutes les lettres majuscules (A à Z) par leur équivalent en minuscules (a à z). Modifiez la chaîne en place en utilisant uniquement le pointeur fourni. Par exemple, la chaîne "BonJOUR" devient "bonjour". Quelle propriété doit être garantie sur la chaîne "bonjour" ? Le code doit être réalisé sans condition (if, switch, terner, ...) et sans fonction standard (strlen, strcpy, strdup, ...). (Indice : 'A' = 65, 'a' = 97). On pourra également considérer que la chaîne de caractères ne contient que des lettres.

```
void to_lowercase(char *string) {
    // TODO
}
```

Question (4 points ≈ 20 min)

Analysez le code suivant, qui implémente un système de journalisation de messages par accumulation. Expliquez étape par étape son comportement. Identifiez les éventuelles erreurs et, le cas échéant, proposez les corrections appropriées. Indiquez également les risques potentiels dans un contexte multi-thread.

```c
#include <stdlib.h>
#include <string.h>

typedef struct {
    char *msg;
    int id;
} LogEntry;

void add_log(LogEntry **log, char *message) {
    LogEntry *entry = (LogEntry*)malloc(sizeof(LogEntry));
    entry->msg = (char*)malloc(strlen(message) + 1);
    strcpy(entry->msg, message);
    entry->id = 1;
    *log = entry; // Remplace l'entrée
}

void free_log(LogEntry *log) {
    free(log);
}

int main() {
    LogEntry *log = NULL;
    add_log(&log, "Erreur␣système");
    add_log(&log, "Connexion␣perdue");
    free_log(log);
    return 0;
}
```

Synchronisation des processus

Documentation

```c
#include <pthread.h>
#include <semaphore.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *
   attr, void *(*start_routine)(void*), void *arg);
int pthread_cancel(pthread_t thread);
int pthread_join(pthread_t thread, void **value_ptr);
void pthread_exit(void *value_ptr);

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);

int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_destroy(sem_t *sem);
int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);
```

Question (3 points ≈ 15 min)

Pour améliorer la réactivité du drone, plusieurs capteurs sont utilisés simultanément pour collecter des données. Chaque capteur transmet ses données à un thread dédié via une mémoire partagée. Vous devez modifier/corriger/simplifier la fonction `collect_data` pour qu'elle puisse être exécutée de manière concurrente par plusieurs threads, sans conflit d'accès aux ressources partagées. Idem pour la fonction `read_data` qui est utilisée pour récupéré une information collectée. De plus on considérera une mise en attente des processus s'il n'y a pas de place pour écrire ou pas d'élément à lire.

```c
// Ressource partagée
#define size 100
ProcessedSensorData *shared_buffer[size];
int tail_index = 0;
int head_index = 0;

void collect_data(ProcessedSensorData *data) {
    shared_buffer[tail_index];
    tail_index = (tail_index+1) % size;
}

ProcessedSensorData* read_data() {
    ProcessedSensorData *tmp = shared_buffer[head_index++];
    shared_buffer[head_index++] = data;
    tail_index = (head_index+1) % size;
    return tmp;
}
```

Question (5 points ≈ 30 min)

On cherche a crée un système de bot téléphonique anti-arnaque, nommé GrannyPrankster. Il répond aux appels de télémarketing pour piéger les escrocs en imitant une grand-mère naïve. Le programme actuel ne permet de traité qu'un seul apelle et suit un flux séquentiel : (1) conversion de la parole en texte (STT), (2) génération de réponses via un modèle de langage (LLM), (3) synthèse vocale (TTS) pour répondre avec une voix de grand-mère. Le système utilise un processeur multicoeur (4 coeurs minimum) et un module audio asynchrone. Proposez un code permettant de réduire la latence et ainsi permettre de traiter plusieurs appels simultanément. `Si vous manquez de temps, proposez un schéma de synchronisation détaillé (2 pts)`..

Code de la question à 5 points

```c
typedef struct {
    float *audio_input;
    int sample_rate, call_duration_ms;
} CallAudio;

typedef struct {
    CallAudio *raw_call;
    char *transcribed_text;
    char *granny_response;
    float *synthesized_response;
    int call_id;
} ProcessedCallData;

int main() {
    while (1) {
        ProcessedCallData *data = init_call_data();
        // Capture de l'audio brut de l'appel
        capture_call_audio(data);
        // Conversion de la parole en texte (STT)
        speech_to_text(data);
        // Génération d'une réponse via un LLM
        generate_granny_response(data);
        // Synthèse vocale de la réponse (TTS)
        text_to_speech(data);
        // Envoie de la réponse
        send_call_audio(data);
        // Libération des ressources
        free_call_data(data);
    }
    return 0;
}
```