Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Carthage

Institut National des Sciences Appliquées et de la Technologie

Compte rendu TP 1 : POSIX système temp réel

Réalisé par : Bouhari Fatma et Grami Eya

Spécialité : Ingénierie Réseaux Informatiques et Télécommunications

Groupe: RT 4/2

Année universitaire : 2024-2025

Exercice

Un pont supporte une charge maximale de 15 tonnes. Ce pont est traversé par des camions dont le poids est de 15 tonnes ainsi que par des voitures dont le poids est de 5 tonnes. On vous demande de gérer l'accès au pont de sorte que la charge maximale du pont soit respectée.

Travail demandé:

1. Donnez un programme comportant un moniteur (une fonction d'acquisition et une fonction de libération du pont) qui simule les règles de partage du pont ci-dessus. Votre programme modélise les camions et voitures sous la forme de threads.

2. Quand un véhicule quitte le pont, on souhaite donner la priorité aux camions : lorsqu'une voiture et un camion sont bloqués en attente d'obtenir l'accès au pont, le camion doit être réveillé en premier, sous réserve, bien sûr, que la capacité maximale du pont soit respectée.

Question 1

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

#define MAX_WEIGHT 15
#define TRUCK_WEIGHT 15
#define CAR_WEIGHT 5

pthread_mutex_t mutex;
pthread_cond_t cond;
int bridge_weight = 0;

// Fonction d'acquisition pour entrer sur le pont
void acquire_bridge(int weight) {
    pthread_mutex_lock(&mutex);
```

```
    // Attendre jusqu'à ce que le poids soit acceptable
    while (bridge_weight + weight > MAX_WEIGHT) {
        pthread_cond_wait(&cond, &mutex);
    }

    bridge_weight += weight;
    pthread_mutex_unlock(&mutex);
}

// Fonction de libération pour quitter le pont
void release_bridge(int weight) {
    pthread_mutex_lock(&mutex);
    bridge_weight -= weight;
    pthread_cond_broadcast(&cond); // Réveiller tous les threads en attente
    pthread_mutex_unlock(&mutex);
}

// Fonction pour les threads camions
void* truck(void *arg) {
    acquire_bridge(TRUCK_WEIGHT); // Acquisition du pont pour un camion
    printf("Truck is crossing the bridge.\n");
    sleep(2); // Temps de traversée
    printf("Truck has crossed the bridge.\n");
    release_bridge(TRUCK_WEIGHT); // Libération du pont
    return NULL;
}

// Fonction pour les threads voitures
void* car(void *arg) {
    acquire_bridge(CAR_WEIGHT); // Acquisition du pont pour une voiture
    printf("Car is crossing the bridge.\n");
    sleep(1); // Temps de traversée
    printf("Car has crossed the bridge.\n");
    release_bridge(CAR_WEIGHT); // Libération du pont
    return NULL;
}

int main() {
    pthread_t threads[10];
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&cond, NULL);

    // Création de threads pour 5 voitures et 5 camions
    for (int i = 0; i < 5; i++) {
```

```
        pthread_create(&threads[i], NULL, truck, NULL);
        pthread_create(&threads[i + 5], NULL, car, NULL);
    }

    // Attendre la fin de tous les threads
    for (int i = 0; i < 10; i++) {
        pthread_join(threads[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond);

    return 0;
}
```

Résultats et Observations

Sortie du programme

```
Truck is crossing the bridge.
Truck has crossed the bridge.
Car is crossing the bridge.
Car is crossing the bridge.
Car is crossing the bridge.
Car has crossed the bridge.
Car has crossed the bridge.
Car has crossed the bridge.
Car is crossing the bridge.
Car is crossing the bridge.
Car has crossed the bridge.
Car has crossed the bridge.
Truck is crossing the bridge.
Truck has crossed the bridge.
Truck is crossing the bridge.
Truck has crossed the bridge.
Truck is crossing the bridge.
Truck has crossed the bridge.
Truck is crossing the bridge.
Truck has crossed the bridge.
```

Explication :

- Les camions et les voitures traversent le pont en alternance, sans priorité stricte pour les camions.
- La sortie montre que chaque véhicule attend son tour si la capacité maximale est atteinte.
- Des camions peuvent traverser en premier, suivis de voitures, et inversement.
- La limite de poids de 15 tonnes est respectée : un camion traverse seul, tandis que plusieurs voitures peuvent traverser ensemble si leur poids total ne dépasse pas la limite.
- Une voiture ou un camion ne peut accéder au pont que lorsque le poids actuel sur le pont le permet, assurant ainsi que le pont n'est jamais surchargé.

Question 2 :

Modifications de code :

1.Variable waiting_trucks :

- Cette variable garde une trace du nombre de camions en attente. Elle est incrémentée lorsqu'un camion tente d'entrer sur le pont et est bloqué en attente. Elle est décrémentée lorsque le camion est réveillé et commence à traverser le pont.

2.Condition de priorité dans acquire_bridge :

- La condition dans la boucle while vérifie deux éléments :
    - Le poids total sur le pont ne doit pas dépasser la limite maximale (bridge_weight + v->weight > MAX_WEIGHT).
    - Si le véhicule est une voiture (type "Car") et qu'il y a des camions en attente (waiting_trucks > 0), la voiture doit attendre, même si la capacité du pont le permet.
- Cette condition garantit que les camions en attente ont toujours **la priorité.**

3.Décrémentation de waiting_trucks :

- Lorsqu'un camion accède au pont après avoir été réveillé, la variable waiting_trucks est décrémentée, indiquant qu'il y a un camion de moins en attente.

code :

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>

#define MAX_WEIGHT 15
#define TRUCK_WEIGHT 15
#define CAR_WEIGHT 5
#define NUM_VEHICLES 10

int bridge_weight = 0;
int waiting_trucks = 0;
pthread_mutex_t lock;
pthread_cond_t truck_cond;
pthread_cond_t car_cond;

typedef struct {
    int weight;
    char type[6];
    int id;
} Vehicle;

void acquire_bridge(Vehicle *v) {
    pthread_mutex_lock(&lock);
    if (strcmp(v->type, "Truck") == 0) {
        while ((bridge_weight + v->weight) > MAX_WEIGHT) {
            waiting_trucks++;
            pthread_cond_wait(&truck_cond, &lock);
            waiting_trucks--;
        }
    } else {
        while ((bridge_weight + v->weight) > MAX_WEIGHT || waiting_trucks > 0) {
            pthread_cond_wait(&car_cond, &lock);
        }
    }
    bridge_weight += v->weight;
    printf("%s %d entering bridge. Current weight: %d\n", v->type, v->id, bridge_weight);
    pthread_mutex_unlock(&lock);
}

void release_bridge(Vehicle *v) {
    pthread_mutex_lock(&lock);
    bridge_weight -= v->weight;
    printf("%s %d leaving bridge. Current weight: %d\n", v->type, v->id, bridge_weight);
    if (waiting_trucks > 0 && (bridge_weight + TRUCK_WEIGHT) <= MAX_WEIGHT) {
        pthread_cond_signal(&truck_cond);
    } else {
        pthread_cond_broadcast(&car_cond);
    }
    pthread_mutex_unlock(&lock);
}
```

```
void *vehicle(void *arg) {
    Vehicle *v = (Vehicle *)arg;
    acquire_bridge(v);
    sleep(2); // Vehicle is crossing the bridge.
    release_bridge(v);
    return NULL;
}

int main() {
    pthread_t threads[NUM_VEHICLES];
    Vehicle vehicles[NUM_VEHICLES];
    pthread_mutex_init(&lock, NULL);
    pthread_cond_init(&truck_cond, NULL);
    pthread_cond_init(&car_cond, NULL);

    // Initialize vehicles, mutex, and condition variable...
    for (int i = 0; i < NUM_VEHICLES; ++i) {
        vehicles[i].id = i;
        if (i % 2 == 0) {
            strcpy(vehicles[i].type, "Truck");
            vehicles[i].weight = TRUCK_WEIGHT;
        } else {
            strcpy(vehicles[i].type, "Car");
            vehicles[i].weight = CAR_WEIGHT;
        }
        if (pthread_create(&threads[i], NULL, vehicle, (void *)&vehicles[i])) {
            fprintf(stderr, "Error creating thread\n");
            return 1;
        }
    }

    for (int i = 0; i < NUM_VEHICLES; ++i) {
        pthread_join(threads[i], NULL);
    }

    pthread_cond_destroy(&truck_cond);
    pthread_cond_destroy(&car_cond);
    pthread_mutex_destroy(&lock);

    return 0;
}
```

Résultats et Observations :

Sortie du programme

```
Truck 0 entering bridge. Current weight: 15
Truck 0 leaving bridge. Current weight: 0
Truck 2 entering bridge. Current weight: 15
Truck 2 leaving bridge. Current weight: 0
Truck 4 entering bridge. Current weight: 15
Truck 4 leaving bridge. Current weight: 0
Truck 6 entering bridge. Current weight: 15
Truck 6 leaving bridge. Current weight: 0
Truck 8 entering bridge. Current weight: 15
Truck 8 leaving bridge. Current weight: 0
Car 1 entering bridge. Current weight: 5
Car 7 entering bridge. Current weight: 10
Car 5 entering bridge. Current weight: 15
Car 5 leaving bridge. Current weight: 10
Car 9 entering bridge. Current weight: 15
Car 7 leaving bridge. Current weight: 10
Car 3 entering bridge. Current weight: 15
Car 1 leaving bridge. Current weight: 10
Car 9 leaving bridge. Current weight: 5
Car 3 leaving bridge. Current weight: 0
```

Explication :

- **Gestion des priorités des véhicules :** Les camions (15 tonnes) ont priorité pour accéder au pont, et les voitures (5 tonnes) ne peuvent y entrer que si aucun camion n'attend.
- **Respect de la charge maximale :** Le pont ne permet qu'une charge totale de 15 tonnes à la fois, gérée via un verrou (pthread_mutex_t) et des conditions (pthread_cond_t).
- **Synchronisation concurrente :** Les threads représentant les véhicules utilisent des mécanismes de verrouillage et des conditions pour garantir un accès sûr et ordonné au pont.