MINES Saint-Étienne | Institut Mines-Telecom

Programmation Système

Rapport Projet PSE

Développement d'une Application Multithread Client/Serveur pour la Réservation de Chambres à l'Hôtel California

---

Yasmine SALMOUNI
Yassmina BARA

Juin 2024

Table des matières

1 Exposé du sujet

Notre projet consiste en une application client/serveur développée pour la gestion des réservations de chambres à l'Hôtel California[1]. Notre système permet aux clients de réserver des chambres, de spécifier la durée du séjour et de sélectionner des services additionnels comme le petit déjeuner et le Wi-Fi.

L'application met en place un serveur qui accepte des connexions de clients pour réserver des chambres d'hôtel en utilisant le protocole TCP/IP avec des sockets et une exécution multithread (pour chaque client connecté, un thread distinct est créé) pour gérer plusieurs clients simultanément. Les threads fonctionnent de manière indépendante mais partagent l'accès à une ressource commune qui est la liste des chambres disponibles. Une fois une chambre réservée, un autre thread est lancé pour libérer la chambre après la période de réservation. Cette structure multithread garantit que le serveur peut répondre à plusieurs demandes en parallèle tout en maintenant la mise à jour des données de disponibilité des chambres.

2 Organisation des programmes

2.1 Bibliothèques utilisées

Nous avons utilisé les bibliothèques suivantes pour développer notre application :

- `stdio.h` - Permet les opérations d'entrée/sortie, pour afficher les informations et les messages sur la console du serveur et du client, aidant à suivre le processus de réservation et les erreurs.
- `stdlib.h` - Fournit des fonctions essentielles pour la gestion de la mémoire dynamique, utilisée notamment pour allouer de la mémoire pour les tableaux qui contiennent les informations des clients et des chambres.
- `string.h` - Utilisée pour manipuler des chaînes de caractères, notamment pour construire et traiter les messages échangés entre le client et le serveur.
- `sys/socket.h` - Indispensable pour créer les sockets et gérer la communication réseau entre le serveur et les clients.
- `arpa/inet.h` -facilite la conversion des adresses IP entre leur représentation binaire et leur forme lisible par l'homme, nécessaire pour la configuration des adresses réseau du serveur.
- `unistd.h` - Fournit l'accès aux appels système POSIX comme `close()`, utilisés pour fermer les sockets après la fin de la communication.
- `pthread.h` - Utilisée pour la création et gestion des threads permettant de traiter plusieurs requêtes de clients de manière simultanée et indépendante.
- `time.h` - Nécessaire pour gérer les temporisations et les durées associées aux réservations des chambres, comme la détermination de la durée pendant laquelle une chambre reste réservée.

1. L'Hôtel California fait ici référence à un exemple fictif, inspiré par la célèbre chanson du même nom par les Eagles.

2.2 Le module serveur

— Définition des constantes port du serveur et nombre de chambres (*SERVER_PORT*, *NUM_ROOMS*).
Dans notre simulation, la constante NUM_ROOMS est définie à 40 et SERVER_PORT à 8081.

— Initialisation du mutex pour contrôler l'accès aux ressources partagées (la liste des chambres disponibles).

— Création d'un socket pour écouter les demandes de connexion entrantes (*server_socket*).

— Le serveur commence à écouter sur le port spécifié pour les connexions entrantes : listen(server_socket, 5).
Le numéro 5 indique que le système peut mettre en file d'attente jusqu'à 5 connexions client en attente d'être acceptées.

— Une boucle infinie est initiée pour accepter de manière continue les nouvelles connexions.

— À chaque nouvelle connexion, un nouveau thread est créé pour gérer les interactions avec ce client spécifique. Le socket client est transmis à ce thread pour permettre la communication.

```
while (1) {
    client_address_length = sizeof(client_address);
    int client_socket = accept(server_socket, (struct sockaddr *)&
    client_address, &client_address_length); /* attends connect */
    if (client_socket < 0) {
        perror("Erreur lors de l'acceptation de la connexion");
        continue;
    }

    int *new_sock = malloc(sizeof(int));
    *new_sock = client_socket;
    if (pthread_create(&threads[client_socket], NULL, clientHandler, (
    void *)new_sock) != 0) {
        perror("Erreur lors de la création du thread");
        continue;
    }
}
```

Listing 1 – Gestion des connexions clients et création de threads.

— La fonction *clientHandler* est exécutée par le thread créé pour chaque connexion client. Elle prend en argument un pointeur vers le descripteur le descripteur du socket client (client_socket). Elle gère l'envoi de messages de bienvenue et la collecte du prénom et du nom du client ainsi que de ses préférences, incluant le nombre de chambres, le type de chambre, la durée du séjour, et les services supplémentaires comme le petit déjeuner et le Wi-Fi. La communication entre le serveur et le socket client se fait par le biais d'un buffer. En effet, un buffer est un tableau utilisé pour stocker temporairement des données pendant qu'elles sont transférées d'un endroit à un autre, en l'occurrence, entre le serveur et le client dans le contexte des sockets réseau. Pour transférer des données du serveur vers le socket client, celles-ci sont stockées dans le buffer et ensuite envoyée vers le socket client par le biais de la fonction send().

```
    sprintf(buffer, "Petit déjeuner inclus pour la chambre %d? (1 pour oui, 0 pour non): ", room_number);
        if (send(client_socket, buffer, strlen(buffer), 0) == -1) {
            perror("Erreur lors de l'envoi de la demande de petit déjeuner");
            close(client_socket);
            pthread_mutex_unlock(&mutex);
            pthread_exit(NULL);
        }

```

Listing 2 – Utilisation des buffer pour établire une communication entre le serveur et le socket client. Le serveur demande si le client veut ou pas un petit dejeuner

Pour transférer des données du client vers le serveur, le socket est d'abord initialisé à zéro avec la fonction 'memset()' afin d'éliminer toute donnée résiduelle qui pourrait corrompre la transmission. Ensuite, les informations sont extraites du socket client et stockées dans le buffer à l'aide de la fonction 'recv()'. Les données sont ainsi accessibles au serveur et peuvent être affichées sur sont terminal à l'aide de la fonction printf().

```
    memset(buffer, 0, sizeof(buffer));
        if (recv(client_socket, buffer, sizeof(buffer), 0) <= 0) {
            perror("Erreur lors de la réception du choix pour le petit déjeuner");
            close(client_socket);
            pthread_mutex_unlock(&mutex);
            pthread_exit(NULL);
        }
        has_breakfast = atoi(buffer);
        printf("Petit déjeuner inclus pour la chambre %d: %s\n", room_number, has_breakfast ? "Oui" : "Non");

```

Listing 3 – Utilisation des buffer pour établire une communication entre le serveur et le socket client. Le serveur réceptionne la réponse du client

— Un thread distinct est utilisé pour chaque chambre réservée pour attendre la durée spécifiée par le client et libérer la chambre automatiquement. Ce thread exécute la fonction *releaseRoom*. Elle prends en argument un pointeur vers la structure *ReservationInfo* qui contient des informations sur la réservation du client dont notament la durée de la réservation et le numéro de la chambre. Dans la fonction releaseRoom, une variable i est initialisée à la durée du séjour en jours. Cette durée est ensuite décrémentée chaque minute à l'aide d'une boucle while. Pour simuler l'écoulement d'une minute entre chaque décrémentation, la fonction sleep(1) est utilisée, faisant ainsi attendre le programme pendant une minute avant de passer au tour suivant de la boucle. Le but de cette simulation est de maintenir le verrouillage de la chambre (à l'aide d'un mutex) pendant une période correspondant à la durée du séjour exprimée en jours.

```c
    void *releaseRoom(void *arg) {
    ReservationInfo *info = (ReservationInfo *)arg;
    printf("Chambre %d sera libérée dans %d secondes\n", info->
    room_number, info->duration);
    sleep(1);
    for (int i = info->duration; i > 1; i--) {
        sleep(1);
    }

```

Listing 4 – La Fonction releaseRoom

2.3 Le module client

- Un socket client (*client_socket*) est créé à l'aide de la fonction socket() pour établir une communication réseau.
- L'adresse IP du serveur est configurée sur 127.0.0.1 pour une connexion sur le réseau local, et le numéro de port sur lequel le serveur écoute est spécifié.
- Le client tente de se connecter au serveur à l'adresse spécifiée. En cas d'échec, le programme affiche une erreur et se termine.
- Une fois la connexion établie, le serveur envoie un message de bienvenue que le client reçoit et affiche. La communication entre le socket client et le client se fait par le biais d'un buffer de la même manière que dans le cas du module serveur.
- Le client envoie son prénom et son nom.
- Le client envoie le nombre de chambres souhaitées et attend une réponse du serveur.
- Pour chaque chambre, le client reçoit une liste de chambres disponibles, fait son choix et envoie ce choix au serveur.
- Pour chaque chambre choisie, le client spécifie la durée du séjour et les services supplémentaires (petit déjeuner, Wi-Fi), et reçoit une confirmation pour chaque sélection.

3 Mise en Œuvre d'une Simulation de Réservation

La famille Serpaggi planifie des vacances bien méritées. Pour l'occasion, ils décident de réserver deux chambres dans l'hôtel California : une chambre double pour les parents, choisissant la chambre numéro 1, et une chambre triple pour les enfants, optant pour la chambre numéro 12. Le père, qui est notre client principal, se charge de la réservation en ligne pour s'assurer que tout est prêt pour leur séjour de cinq jours. Lors de la réservation, il s'assure que le petit déjeuner est inclus pour toute la famille. Toutefois, afin de maximiser le temps passé ensemble et de profiter pleinement des activités de vacances, il décide de ne pas opter pour le Wi-Fi dans les chambres des enfants. Ce processus de réservation inclut donc la sélection des chambres, la spécification de la durée du séjour, et la confirmation des services supplémentaires pour assurer le confort de sa famille pendant leur séjour.

La capture d'écran suivante du terminal client montre les étapes détaillées de la réservation.

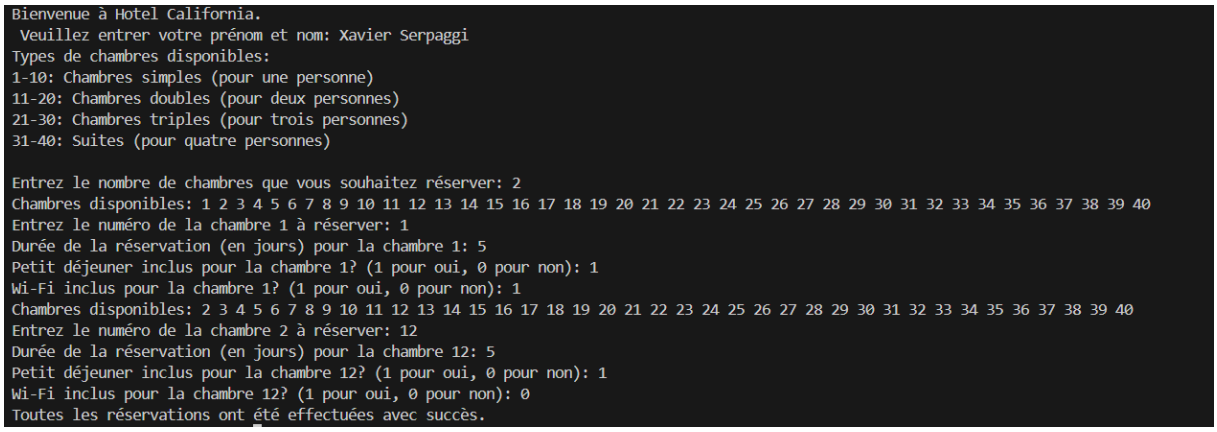

FIGURE 1 – étapes détaillées de la réservation. Terminal serveur.

La capture d'écran suivante du terminal serveur montre les étapes détaillées du traitement des réservations de chambres par le client.

Remarque : Notons que la chambre 1 a été retirée de la liste des chambres disponibles pendant la durée de réservation. Dans notre simulation, un jour de réservation est équivalent à une minute d'attente.

```
Server listening on port 8081
Client: Xavier Serpaggi

Nombre de chambres à reserver: 2
Liste des chambres disponibles envoyée au client.
Chambre 1 réservée
Durée de la réservation pour la chambre 1: 5 jours
Petit déjeuner inclus pour la chambre 1: Oui
Wi-Fi inclus pour la chambre 1: Oui
Chambre 1 sera libérée dans 300 secondes
Liste des chambres disponibles envoyée au client.
Chambre 12 réservée
Durée de la réservation pour la chambre 12: 5 jours
Petit déjeuner inclus pour la chambre 12: Oui
Wi-Fi inclus pour la chambre 12: Non
Chambre 12 sera libérée dans 300 secondes
Chambre 1 libérée
Chambre 12 libérée
```

FIGURE 2 – traitement des réservations de chambres. Terminal client

4 Points à Développer

Nous proposons une mise à jour de notre système de réservation actuel afin d'améliorer la gestion des disponibilités des chambres. Notre système de réservation actuel ne demande pas explicitement les dates de début et de fin de séjour lors de la prise de réservations. Pour remédier à cette situation, nous suggérons d'implémenter une fonctionnalité qui exigera que les dates de début et de fin de séjour soient spécifiées lors de chaque réservation. Cette modification permettra de bloquer la chambre uniquement pour la période spécifique réservée par le client.

Details techniques de l'amélioration

- Modifier le formulaire de réservation pour inclure des champs obligatoires pour les dates de début et de fin.
- Assurer que la base de données enregistre précisément ces intervalles pour éviter les chevauchements et les périodes de non-disponibilité non justifiées.

Table des figures