le cnam

2025 - 2026

Dossier d'analyse

APPLICATION CLIENT-SERVEUR

Banumathey BALENDRAN
Loucia LAIZE
LP CHEF DE PROJET - DEVELLOPEMENT SECURITE EXPLOITATION

Sommaire

Introduction

Développement d'une première version d'une application client-serveur pour la gestion de réservations de places de spectacles. L'application permet à un client de consulter le nombre de places disponibles pour différents spectacles et d'effectuer des réservations. Cette première version implémente des fonctionnalités de base en utilisant des tubes anonymes comme mécanisme de communication inter-processus.

Architecture de l'application

Notre application est composée d'un seul programme contenant à la fois le code client et serveur.

Le programme marche avec une communication parent-enfant. Le processus serveur (père) et le processus client (fils) communiquent et interagissent via des tubes anonymes.

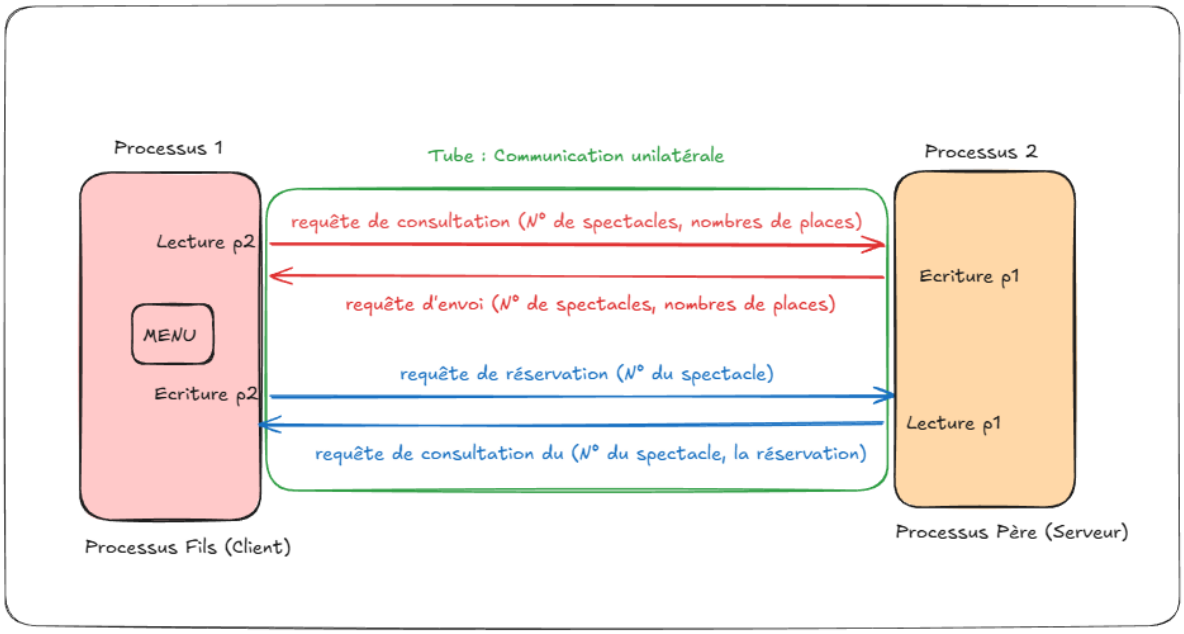

FIGURE 1: INTERCOMMUNICATION ENTRE LE CLIENT ET LE SERVEUR

Le client effectue d'abord une consultation, puis une réservation.

Cette architecture repose sur le modèle de Communication Inter-Processus (IPC) par filiation[1]. Le programme se duplique sur le modèle du Processus Père, qui joue le rôle du Serveur, responsable de gérer le flux et le statut des données (le nombre de places) et de traiter les requêtes.

Le Processus Fils hérite des descripteurs du père, et agit comme le Client, responsable de l'interface utilisateur, du menu pour la consultation comme le montre le schéma, et de l'envoi/réception des messages.

[1] (avec la fonction primitive fork()).

Plus précisément, la communication est assurée par deux tubes anonymes (pipes), permettant un flux de données bidirectionnel entre ces deux processus, ce qui permet la synchronisation des échanges.

Le schéma suivant illustre cette communication :

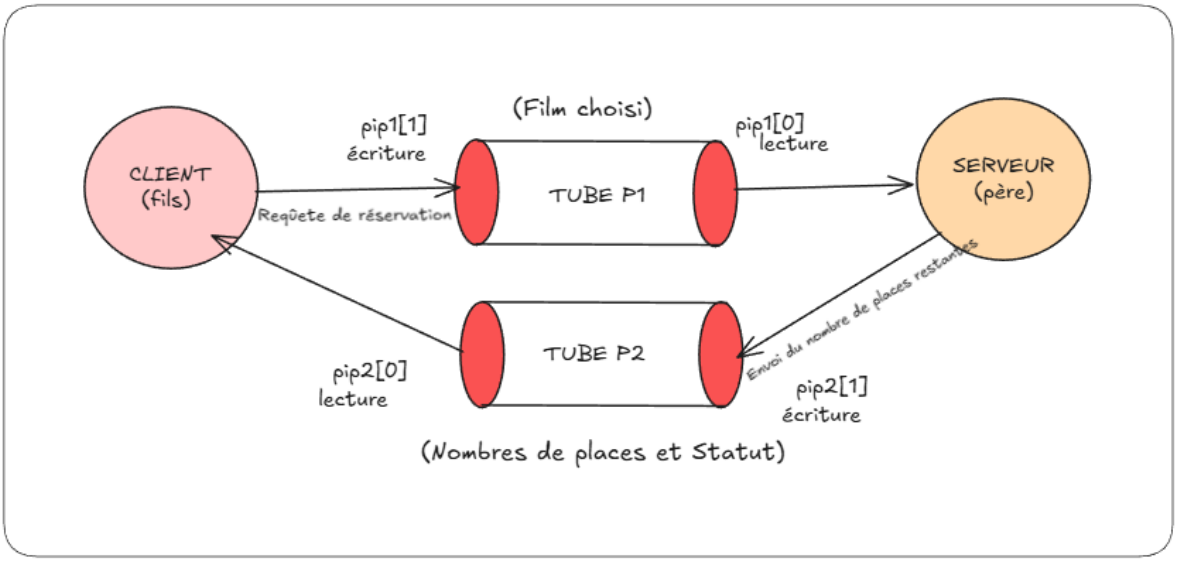

FIGURE 2: COMMUNICATION ENTRE LES PROCESSUS PERE ET FILS POUR GERER LA RESERVATION DU FILM ET AFFICHER LE NOMBRE DE PLACES RESTANTS

Données

1. Table et structure

Le Serveur utilise une structure de code pour gérer l'index des spectacles. Cela simplifie le traitement du côté développement puisque la gestion se fait avec des entiers et pas des caractères. Dans cette première version, cette structure est un tableau (ou une liste) où chaque entrée est liée à un identifiant de spectacle (ID). La structure de la table des spectacles repose sur :

- Identifiant du spectacle (ID) : un entier positif (index d'accès)
- Nombre de places restantes : un entier positif (donnée)

  Ainsi, pour un spectacle donné la structure contiendra N entrées, soit une paire avec le nom du spectacle, la chaine de caractère, et un entier désignant le nombre de places restantes.

Le Serveur (Père) utilise ce tableau. L'ID Spectacle (1, 2, 3) est lié à l'index du tableau. Le tableau stocke bien la quantité de places disponibles. Et dès la réception de la requête du client, le serveur accède et modifie l'élément du tableau pour gérer les places restantes.

2. Données échangées

Les données sont échangées en séquence bilatéralement via les deux tubes. Les messages sont des suites d'octets, des entiers et des chaînes de caractères.

Par soucis de simplification (pour gérer plus facilement les requête avec les entiers), nous décidons d'afficher l'ID de la réponse à la requête du client. Les données échangées sont des entiers positifs (sauf le message de statut).

Opération	Donnée	Type	Rôle
Requête Client serveur	Numéro/ ID du spectacle	Entier	Demande de consultation
Réponse Serveur client	Nombres de places restants	Entier	Réponse à la consultation
Requête Client Serveur	Numéro du spectacle puis Nombres de places restants	Entier	Demande de réservation
Réponse Serveur client	Places restantes	Entier	Réponse de réservation
Réponse Serveur client	Statut de réservation	Chaîne de caractère	Réponse de réservation

Outils de communications

Pour faire communiquer les 2 processus dans le programme nous avons donc utilisées les 2 tubes anonymes reliés par leur filiation père fils, avec leurs descripteurs de lecture et d'écriture. Le serveur et le client seront 1 seul programme exécuté en ligne de comme un seul et même programme contenant 2 processus.

Également, la communication est permise par les primitives systèmes Unix spécifiques aux tubes anonymes. Le canal de communication se fait avec la primitive pipe () qui permet de créer les tubes. Elle renvoie les deux descripteurs de fichier (lecture et écriture) pour chaque tube.

La distinction entre les processus et l'attribution des rôles se fait avec la primitive fork(). Enfin, la primitive close() ferme les échanges entre les tubes, ce qui assure que le flux de données est bien unidirectionnel sur chaque tube (par exemple, le Client ferme l'extrémité de lecture du tube de requête). Finalement les principales primitives utilisées pour échanger les données sont les primitives write() (pour envoyer) et read() (pour recevoir) sont utilisées. Le serveur attend la terminaison du client avec la primitive wait() pour bloquer son exécution jusqu'à sa terminaison, ce qui permet une exécution synchronisée et ordonnée.

La communication est ainsi bidirectionnelle et synchronisée entre les tubes via les différentes primitives.

Algorithme

L'algorithme du programme suit une séquence en trois étapes :

- **Étape 1 :** Initialisation du processus père, puis amorçage de la création des deux tubes et de la création du processus fils via sa duplication
- **Étape 2 :** Configuration du Serveur (processus père) et du Client (processus fils) : Chaque processus ferme les descripteurs inutilisés pour établir les canaux de communication.
    - Début de la communication inter-processus et des échanges du fils au père : le Client initialise le dialogue :
        - Le Client demande une consultation (lit les films, envoie ID du film choisi) et le Serveur répond (envoi les places restantes de ce film).
        - Le Client demande une réservation (envoie ID du film et le nombre de places à réserver) et le Serveur traite la demande et répond (envoie la réponse selon le statut).
- **Étape 3 :** Terminaison du processus père - le Serveur attend la fin de l'exécution du Client avant de terminer à son tour

Conclusion

Telle que conçu et pensé, la structure de notre code possède des limites.

D'un point de vue fonctionnel, en 1er lieu, le client ne peut pas choisir entre une consultation et une réservation. Le client ne peut faire qu'une opération, et de fait, le code ne peut pas permettre de faire plusieurs requêtes successives. Enfin, des contrôles sur les entrées utilisateur devraient être implémentées.

Suivant cela, d'un point de vue technique, cela soulève d'autres contraintes. En effet, la communication est basée sur la structure du code, son architecture. Changer l'ordre nuit aux échanges ainsi le code n'est pas forcément modulable notamment au niveau du menu de consultation. Ainsi, la difficulté première résiderait dans le fait que le serveur devrait répondre à plusieurs requêtes en temps réel de la part de client. Ça veut dire qu'il devrait supporter la charge de requêtes, donc un risque de surcharge. De plus, l'introduction de plusieurs clients nécessiterait donc de revoir cette architecture afin de gérer plusieurs clients simultanés, ce qui poserait d'autres contraintes. A cela, s'ajoute l'absence de persistance des données sont perdues à l'arrêt du programme ce qui pourrait être une amélioration à adresser.

Également, le programme tel qu'il est au niveau du serveur ne distingue les types de requêtes et adresse ces dernières par ordre, sans distinction réelle, avec la première donnée est pour la consultation et les deux suivantes pour la réservation, ce qui devrait être restructuré.

Enfin, nous pensons que notre programme pourrait implémenter des files pour gérer les différentes requêtes, les temps de réservations, et éviter et faire en sorte que 2 clients ne réservent pas en même temps.

Notre 1ère version rencontre donc plusieurs limitations et notre programme devra évoluer au niveau fonctionnel, technique et structurel, pour répondre à ces considérations.