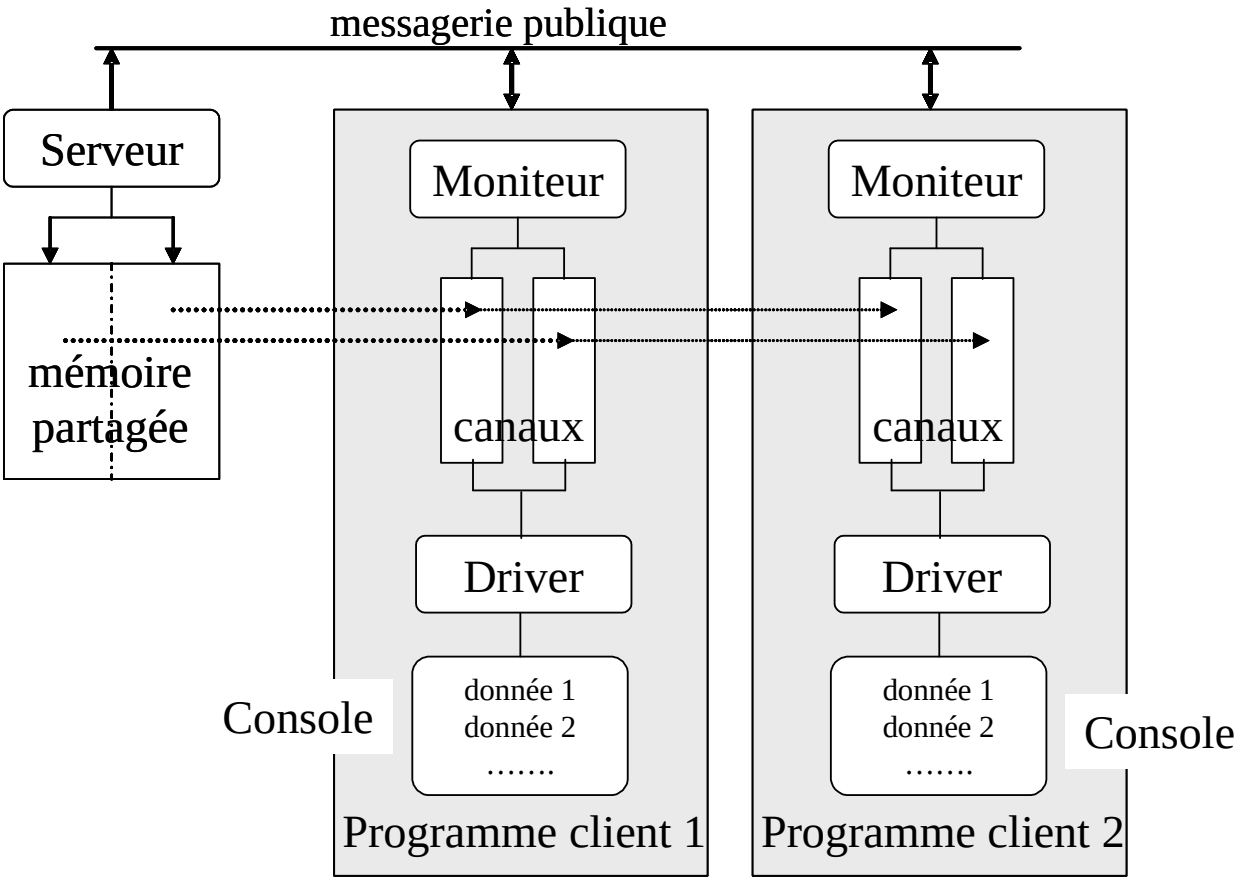
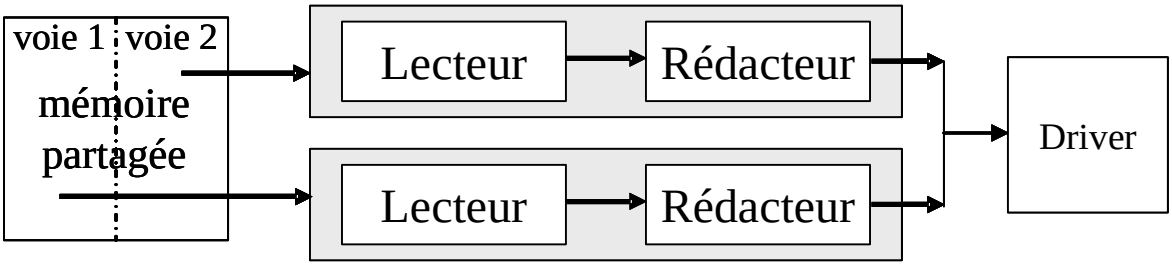
Projet Client - Serveur

I) Le sujet

Un serveur génère des données toutes les secondes. Ces données sont issues aléatoirement de l'une des *NVOIES* d'entrée. Le paramètre *NVOIES* est fixé à *2* dans le projet.
Il s'agit d'écrire un programme *client* qui affiche en temps réel ces données numériques. Le code source (en C) du serveur est fourni.
Le travail demandé est l'écriture du client.

II) Synoptique





Détail des canaux

III) Description du serveur

L'ensemble des programmes sources du serveur ainsi que quelques autres fichiers se trouvent sur le git : . Un fichier README vous indiquera plus précisément le contenu des fichiers ou répertoires.

Pour utiliser le serveur, recopier le fichier serveur.zip et décompressez le. La commande *make* compile tout le serveur. Le lancement du serveur se fait par la commande SV en indiquant le nombre de secondes pendant lesquelles vous souhaitez que le serveur soit actif : *SV <nb_s>*

Pour écrire le client, vous disposez des fichiers d'en-tête CL_include et CL_def.h dans le fichier (zippé) client.zip. Ces fichiers d'entête contiennent la liste des bibliothèques, les structures et les constantes utilisées.

III.1) Protocole de dialogue :

Le serveur discute avec les clients de trois façons distinctes :

- Le client demande sa connexion au serveur par une messagerie publique créée par le serveur et connue de tous les utilisateurs par l'intermédiaire d'un nom de fichier unique (clé d'accès).
- Le serveur fournit la clé d'accès aux données par la messagerie publique.
- Le serveur informe chaque client de l'arrivée d'une donnée par des signaux.
- Le serveur enregistre les données dans une mémoire partagée (TAMPONS) accessible à tous les clients.

III.2) Définition d'une session (pour un client)

- **Connexion :** (fichier SV_connect.c, fonction ReceptionClients)
  1. Le client s'identifie en émettant un message de type CONNECT contenant son PID.
  2. Le serveur répond en émettant un message de type PID contenant
  - soit la clé d'accès à la mémoire partagée,
  - soit une chaîne de caractères vide si le serveur est saturé.
  3. Le serveur attend un acquittement du client par un message de type ACK contenant le PID du client.

- **Déconnexion :**
  4. Le client envoie un message de type DECONNECT contenant son PID.
  5. Le serveur supprime le client de la liste.

- **Transfert de données :** (fichier SV_data.c, fonction SignalDonnee)
  6. Le serveur informe les clients de l'arrivée d'une donnée dans les TAMPONS par l'émission à tous les clients connectés d'un signal *SIGUSR1* pour la voie1 et *SIGUSR2* pour la voie2.
  7. L'accès à la mémoire partagée se fait par exclusion mutuelle (géré avec sémaphore créé par le serveur.

III.3) Format d'une donnée

Les deux tampons sont des buffers circulaires de nombres entiers de taille (*0..BUF_SZ – 1*).

IV) Travail demandé

- Etudier le programme du serveur

- Vous devez écrire l'application CLIENT. Cette application comporte :
  - un processus MONITEUR qui assure la connexion et la synchronisation des entrées/sorties des données,
  - un processus LECTEUR par voie qui lit les données dans les tampons,
  - un processus REDACTEUR par voie qui transmet les données dans un ordre défini au DRIVER de l'écran.

- Contraintes :
  - A chaque voie est associé un couple LECTEUR – REDACTEUR.
  - Les processus LECTEUR et REDACTEUR sont créés par le processus MONITEUR du client.
  - L'affichage est géré par un processus (DRIVER) qui communique avec un seul des canaux. Quand un rédacteur a reçu 5 données du lecteur, il demande l'accès au DRIVER. Il transmet les 5 données, puis libère la ressource DRIVER.
  - Le programme doit être modulaire : les fonctions LECTEUR, MONITEUR, REDACTEUR, DRIVER doivent être dans des fichiers distincts.
  - Avant de lire dans les tampons, le LECTEUR doit s'assurer que le serveur n'est pas en train d'y écrire.

- Organisation du travail
  - Mettre en place la communication avec le serveur pour que le client se fasse connaître et se connecte au serveur.
  - Réfléchir au mécanisme de synchronisation et aux moyens de communications entre processus.
  - Ecrire le MONITEUR, le LECTEUR et le REDACTEUR. Les données seront affichées dans l'ordre d'arrivée. On réfléchira d'abord à l'algorithme de chacune des fonctions.
  - Affichage séquencé : écriture du DRIVER d'écran. Il pourra être un programme C avec un main().

Bien évidemment, on programmera de façon propre ! (gestion des erreurs, commentaires, …)

Projet Client – Serveur / Fiche de suivi

Binôme :

Compte Linux :

Séance 1 Répertoire :

Travail effectué :

Séance 2 Répertoire :

Travail effectué :

Séance 3 Répertoire :

Travail effectué :

Séance 4 Répertoire :

Travail effectué :