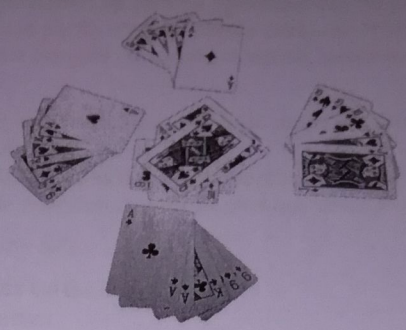
2 deux piles : l'un deck && pile

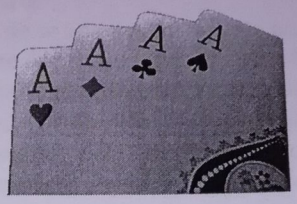
Programmation réseau (Projet)

Communication inter-processus

Le projet à concevoir est la simulation du jeu de cartes "bataille" dont les règles ont été adaptées aux besoins du présent exercice.



Le jeu



Matériel

- de 2 à 4 joueurs
- 52 cartes
  - 4 couleurs (cœur, carreau, trèfle, pique)
  - 13 valeurs (as, roi, dame, valet, 10, 9, 8, 7, 6, 5, 4, 3 et 2)

But du jeu

Le but du jeu est d'avoir le plus de points possible.

Déroulement d'une partie

Une partie est une suite de manches (de 1 à 5).

Déroulement d'une manche

Les 52 cartes sont distribuées aléatoirement aux joueurs.
La manche se décompose en « tours » où chaque joueur met une carte sur la table et celui qui a la carte dont la valeur est la plus élevée les ramasse toutes.
La « force » d'une carte est déterminée par sa valeur et sa couleur. L'as est supérieur au roi qui est supérieur à la dame ...etc. L'ordre de prévalence des couleurs est : cœur, carreau, trèfle et pique. L'as de cœur a une valeur plus grande que l'as de carreau qui lui-même est supérieur à l'as de trèfle ...etc.
Il n'y a donc pas deux cartes qui ont la même force.

Fin de la manche

La manche se termine dès qu'un joueur n'a plus de cartes. Le gagnant est alors celui qui a le plus de points. Chaque joueur reçoit un score qui sa calcule en additionnant la valeur des cartes qu'il a en main au moment où la manche se termine. L'as vaut 1 point, le roi 13 points, la dame 12, le valet 11 et les autres cartes ont leur valeur nominale.

Fin de la partie

La partie se termine soit quand 5 manches ont été jouées, soit quand il n'y a plus qu'un seul joueur.
Le score total est le cumul des scores de toutes les manches.
Le vainqueur de la partie est celui qui a le score total le plus élevé.

Implémentation du jeu

Le projet à réaliser est une application client-serveur qui utilise des sockets TCP/IP pour implémenter le dialogue entre le gestionnaire du jeu (le serveur) et les joueurs (les clients) et une mémoire commune pour partager les informations concernant les joueurs et les scores.

Un joueur (un client) demande l'autorisation de jouer au gestionnaire du jeu (le serveur). Pour ce faire, il communique son nom au serveur. Si la partie n'a pas encore démarré et si le nombre maximum de 4 joueurs n'est pas atteint, le joueur est accepté, sinon il est refusé. Le serveur envoie sa réponse au joueur.

Avant de démarrer une partie le serveur attend que les deux conditions suivantes soient remplies :

1. Il y a au moins deux joueurs.
2. Il s'est écoulé au moins 30 secondes depuis l'acceptation du premier joueur (cette condition permet d'éviter de refuser systématiquement plus que deux joueurs).

Une manche de jeu se déroule comme suit :

- le serveur distribue toutes les cartes aux joueurs
- chaque joueur affiche ses cartes
- déroulement d'un tour
  - le serveur demande une carte
  - chaque joueur choisit une de ses cartes et l'envoie au serveur.
  - quand le serveur a reçu une carte de chaque joueur, il calcule celle dont la valeur est la plus élevée et envoie toutes les cartes du tour au joueur qui a joué la carte gagnante.
- dès qu'un joueur n'a plus de cartes, il le signale au serveur qui envoie un message de fin de manche à tous les joueurs et les joueurs y répondent en envoyant les points qui correspondent aux cartes qu'il a dans son jeu. Le serveur met à jour les scores dans la mémoire partagée.
- à tout moment un joueur peut afficher les scores de tous les joueurs en consultant la mémoire partagée.

Le jeu consiste en au plus 5 manches. Le nombre de manche maximum peut être un argument à passer au serveur.

Le serveur ne gère qu'une seule partie à la fois mais après une partie, il se réinitialise pour être prêt à démarrer la suivante.

Pratiquement, les informations circulent du serveur vers les joueurs et des joueurs vers le serveur via les sockets. Les noms des joueurs et leurs scores sont conservés en mémoire partagée.

Indications de programmation

Il y a 2 programmes à écrire : le serveur et le joueur, ce dernier est le client.

Pour l'implémentation des sockets, le numéro du port est à préciser comme argument du programme. Prévoyez un deuxième argument qui précise si les erreurs sont affichées à la sortie d'erreur ou dans un fichier.

Le serveur ne s'arrête pas.

L'accès à la mémoire partagée doit être sécurisé par l'utilisation de sémaphores en respectant l'algorithme des lecteurs – rédacteurs de « Courtois » qui a été étudié au cours de système d'exploitation.

```
#include "prototypes.h"

typedef int semaphore;                  /* faites preuve d'imagination */

semaphore mutex = 1;                    /* ctrl l'accès à 'rc' */
semaphore bd = 1;                       /* ctrl accès à base de données */
int rc = 0;                             /* nb de procesus lisant */
                                        /* ou voulant lire */

void lecteur(void)
{
   while (TRUE) {                       /* boucle infinie */
      down(&mutex);                     /* obtenir l'accès exclusif à 'rc' */
      rc = rc + 1;                      /* un lecteur de plus */
      if (rc == 1) down(&bd);           /* si c'est le premier lecteur ... */
      up(&mutex);                       /* libérer l'accès exclusif à 'rc' */
      lire_base_de_donnees();           /* lire les données */
      down(&mutex);                     /* obtenir l'accès exclusif à 'rc' */
      rc = rc - 1;                      /* un lecteur de moins */
      if (rc == 0) up(&bd);             /* si c'est le dernier lecteur ... */
      up(&mutex);                       /* libérer l'accès exclusif à 'rc' */
      utiliser_donnees_lues();          /* section non critique */
   }
}

void redacteur(void)
{
   while (TRUE) {                       /* boucle infinie */
      creer_donnees();                  /* section non critique */
      down(&bd);                        /* obtenir l'accès exclusif */
      ecrire_donnees();                 /* mettre à jour les données */
      up(&bd);                          /* libérer l'accès exclusif */
   }
}
```

Les messages d'erreurs affichés grâce à la fonction perror ne sont pas toujours suffisamment explicites pour préciser l'erreur soulevée, prévoyez dans ce travail des messages adéquats.

Quand l'application est terminée, aucun " ipc " ne peut traîner sur le système ! ! ! !

Il faut prévoir le cas où l'un ou l'autre processus serait tué pendant une partie et arrêter la partie s'il ne reste plus qu'un seul joueur.

En cas de « mort » d'un joueur (déconnexion, arrêt demandé par l'utilisateur, ...) durant la période d'inscriptions, il n'est pas inscrit ou il est supprimé si le serveur l'avait déjà inscrit.

En cas de « mort » d'un joueur en cours de partie, il est éliminé et le serveur ne lui envoie plus aucun message mais son nom reste dans la mémoire partagée.

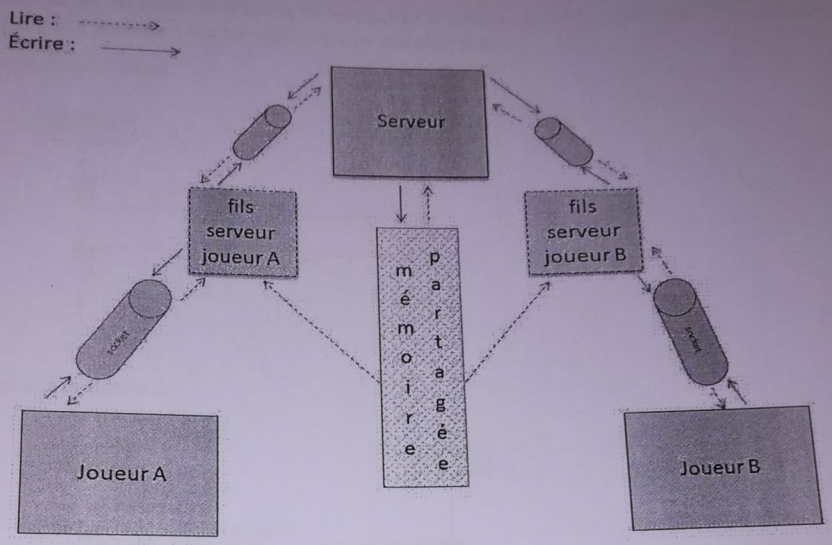
En cas de « mort » du serveur, la partie s'arrête et les clients doivent aussi se terminer.

Deux programmes serveurs ne peuvent pas être exécutés simultanément, ce test doit être prévu lors du lancement du programme.

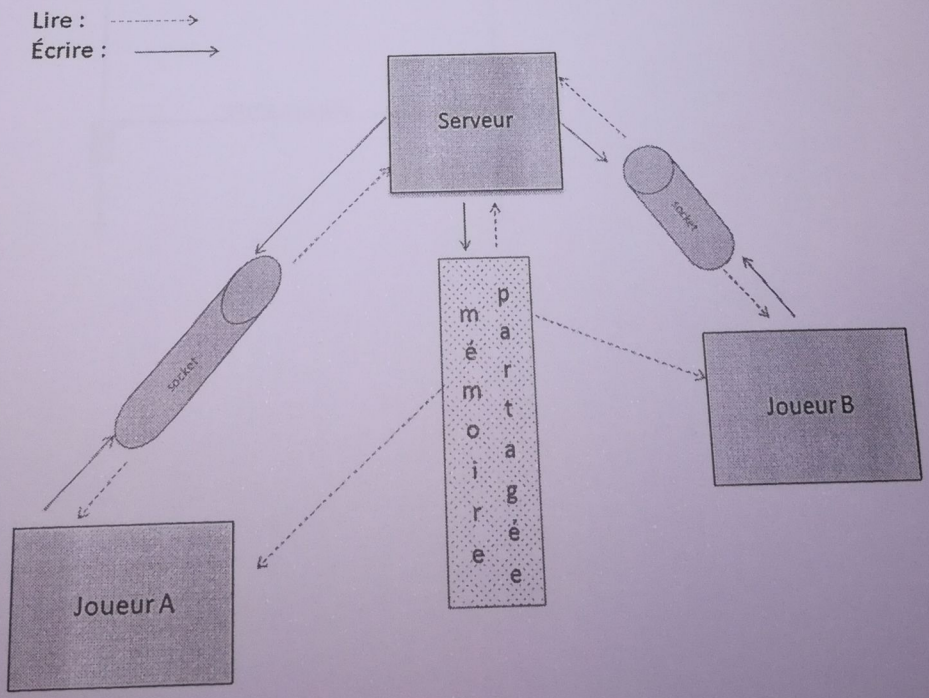
Utilisation des moyens de communications

Les clients et le serveur ne tournent pas nécessairement sur la même machine. La mémoire partagée n'est accessible que par des processus qui tournent sur la même machine que le serveur. Pour ce faire, le serveur doit créer un fils par client qui joue le rôle de client local et accède à la mémoire partagée. Les clients distants communiquent alors avec le fils qui leur est attitré (via sockets) et effectue les affichages et autres entrées/sorties. Dans ce cas , le serveur communique avec ses fils via des pipes.

Les informations qui circulent entre un joueur et le serveur transitent par des "socket" TCP et les informations connues de tous sont conservés dans une "mémoire partagée".



Dans le cadre de ce projet, les joueurs seront sur la même machine que le serveur ce qui permet la simplification suivante :



Programma...

Contraintes pratiques

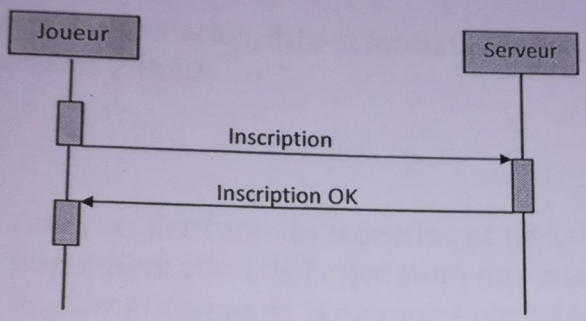
Pour mener à bien ce projet, il est indispensable de travailler de manière modulaire.

Pour compiler et faire l'édition de liens des programmes, un « makefile » doit être écrit.
Chaque fichier doit commencer par les noms, prénoms, nom de login des concepteurs de même qu'un commentaire qui précise ce qu'il contient

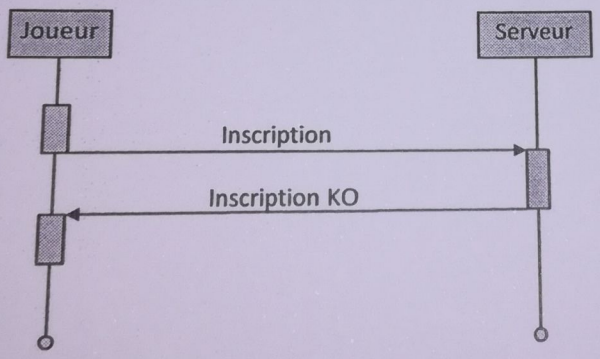
La conception et l'implémentation de cette application constituent l'ultime étape pour les exercices Unix.

Avant de se lancer dans la programmation, il est prudent de commencer par analyser le problème posé. Il est demandé d'établir une liste des messages qui vont être échangés entre le serveur et les joueurs, d'en fixer le format et de concevoir les scénarios. Par exemple, pour l'inscription d'un joueur, le dialogue entre le serveur et le joueur peut se représenter comme suit :

1. Inscription acceptée :



2. Inscription refusée :



3. Annulation de la partie :