Renquet Arnaud

Djurakholov Bekhzod

Jasoigne Thibauld

Rapport de projet système d'exploitation TP

Année 2011-2012

Sommaire :

1. introduction.

Avant de commencer,

Comme dans tous aéroports et pour tout ce qui concerne la météo dans l'aviation tant civile que militaire, la sécurité météo est très importante.

Comment faire pour savoir si un avion peut ou ne peut pas décoller d'un aéroport si on ne connait pas la météo dont les modèles sont analysés par les météorologues de « Belgocontrol »(en Belgique).

On en vient donc au projet.

Ce projet a pour but de simuler le transfert d'information sur demande d'un pilote d'avion qui demanderait des renseignements sur les conditions de vol sur la ligne qui lui est assignée.

Ce type de programme est typique des programme utilisé dans les aéroports actuel mais est automatisé.

2. Analyse et plan du projet.

Ce projet est réparti en 3 grandes parties.

I. le serveur
II. le client
III. la gestion météo.

I. Le serveur.

Le serveur est une des parties les plus importantes du projet et du programme car c'est la partie qui sert de liant entre le client et le fichier ATIS.
Le serveur va recevoir une demande de la part du client via un fifo .
Le serveur devra aller chercher l'information sur le fichier ATIS pour autant que celui-ci ne soit pas locké car entrain d'être mis à jour.
Une fois que le serveur aura l'information météo voulue, il la renverra vers le client qui à demandé l'information.

II. Le client.

Le programme client est un programme qui demande l'information au serveur via un fifo.
Si le client ne reçoit pas toute l'information, il envoie au serveur un « ack ko » pour que le serveur renvoie l'information, sinon, il envoie un « ack ok » pour dire qu'il à bien reçu l'information.

III. La gestion météo.

La gestion météo se compose de deux sous partie :

- Le fichier ATIS qui est un fichier que l'on crée et où on importe l'information relative à la météo et où le serveur ira chercher l'information concernant la météo.
- Le gestionnaire météo qui s'occupe de mettre à jour le fichier ATIS et quand il met à jour le fichier ATIS, il le locke pour que le serveur ne puisse pas y accéder pendant que les dernières informations sont rajoutée au fichier ATIS. Une fois cela fini, il le délocke afin que le serveur puisse répondre aux attentes des pilotes qui demandent les données météo.

3. conclusion.

On a rencontré quelque difficultés d'organisation car il se fait que l'on à pratiquement jamais su se rencontrer tout les 3 en même temps sauf que depuis on a tous SKYPE ce qui permet quand même de pouvoir se parler et parler du projet en « visu » via pc.

Point de vue délai, on a su faire le projet malgré les absences des uns et des autres.

Pour la partie technique, ayant eu quelque déboire avec centos(pour ma part), on l'a implémenté sur un linux donc unix, sur une version ubuntu 11.10.

On a pu cependant, malgré tout les soucis d'organisation, pu arriver au bout et finir ce projet dans les temps impartis.

Ce projet nous à permis de mieux comprendre ce qu'est unix dans le sens comment on fait pour traiter un file system, comment gérer des demandes , comment gérer un serveur,etc…

4. annexe.

Client.c

```
#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <ctype.h>

#include <errno.h>

#include <unistd.h>

#include <string.h>
```

```
#include <stdlib.h>

#include "client-serveur.h"

int main(int argc, char *argv[]) {
 requete_t requete;
 reponse_t reponse;
 int i;
 int len;
 int fdW, fdR;
 int nbRead, nbWrite;

 if (argc != 2) {
  printf("Usage = client nomDeFamilleCherche\n");
  exit(EXIT_FAILURE);
 }

 /* Préparation du champ pointAccesClient de la requête */
 sprintf(requete.pointAccesClient, "./serveur-client.%d", getpid());

 /* Préparation du champ contenu : On y recopie argv[1] en mettant en */
 /* majuscules                                          */
 len = strlen(argv[1]);
 for (i=0 ; i < len ; i++) {
```

```
  requete.contenu[i] = toupper(argv[1][i]);
}
requete.contenu[len] = '\0';

/* Création du tube nommé du client */
if (mkfifo(requete.pointAccesClient, S_IRUSR|S_IWUSR) < 0) {
  if (errno != EEXIST) {
    perror("mkfifo(tube nommé client");
    exit(EXIT_FAILURE);
  }
  else {
    printf("%s existe deja : on suppose que c'est un pipe nomme\n",
              requete.pointAccesClient );
    printf("et qu'on peut continuer le programme sans probleme\n");
    puts("");
  }
}

/* Envoi de la requête vers le serveur */
fdW = open(NOMPOINTACCESSERV, O_WRONLY);
if (fdW == -1) {
  perror("open(NOMPOINTACCESSERV)");
  exit(-1);
}
```

```
nbWrite = write(fdW, &requete, sizeof(requete));
if (nbWrite < sizeof(requete)) {
        perror("pb ecriture sur pipe nomme");
}

/* On n'aura pas d'autres messages à envoyer au serveur. On peut donc    */
/* fermer le tube                                                        */
close(fdW);

/* On ouvre seulement maintenant le tube pour écouter la réponse, sinon  */
/* on serait bloqué jusqu'à ce que le serveur ouvre le pipe en écriture  */
/* pour envoyer sa réponse. Si on ouvrait le tube avant d'envoyer la     */
/* requête, le serveur ne recevrait jamais la requête et donc n'ouvrirait*/
/* jamais le tube ==> On serait bloqué indéfiniment.                     */
fdR = open(requete.pointAccesClient, O_RDONLY);
if (fdR == -1) {
  perror("open(tube nommé)");
  exit(-1);
}

/* On lit la réponse */
nbRead = read(fdR, &reponse, sizeof(reponse));
if (nbRead != sizeof(reponse)) {
  printf("Communication avec le serveur probablement rompue\n");
```

```
  exit(EXIT_FAILURE);
 }

 /* On affiche la réponse */
 printf("Reponse du serveur = \"%s\"\n", reponse.contenu);

 /* NB : Dans cette application, on utilise pas le point d'accès privé */
 /* vers le serveur                                                    */

 /* On ferme le tube côté réception */
 close(fdR);

 /* On détruit le tube nommé du client (puisque ce tube nommé ne servira */
 /* plus jamais.                                                         */
 if (unlink(requete.pointAccesClient) < 0) {
  perror("unlink");
  exit(EXIT_FAILURE);
 }

 return EXIT_SUCCESS;
}
```

Client-serveur.c

```
/************************************************/
/* Déclarations communes au client et au serveur */
```

```
/***********************************************/

#define NOMPOINTACCESSERV "FiFo_client-serveur"

/* Structure des messages utilisés dans le sens client-->serveur */
/* NB : les 2 chaînes de caractères se terminent par '\0'       */
typedef struct{
  char pointAccesClient[128];
  char contenu[256];
} requete_t;

/* Structure des messages utilisés dans le sens serveur-->client */
/* NB : les 2 chaînes de caractères se terminent par '\0'       */
typedef struct{
  char pointAccesPrive[128];
  char contenu[256];
} reponse_t;
```

Serveur-atis.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <ctype.h>
#include <errno.h>
```

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

#include "client-serveur.h"

#define NBENFANTS 3

int dansPool[2];
int pereEnfant[NBENFANTS][2];

void traiterRequete(requete_t *requete, FILE *f){
 reponse_t reponse;
 char ligne[256];
 int fdW;
 int nbWrite;

 /* Calcul reponse->pointAccesPrive */
 strcpy(reponse.pointAccesPrive, "");
        /* Le point d'accès privé n'est pas utilisé dans cette appli */

 /* Calcul reponse.contenu. */
 while (feof(f) == 0) {
  if (fgets(ligne, sizeof(ligne), f) != NULL) {
```

```
   if (strncmp(ligne, requete->contenu, strlen(requete->contenu)) == 0){
          break;
   }
 }
}
if (feof(f) == 0) {
 strcpy(reponse.contenu, ligne);
 reponse.contenu[strlen(ligne)-1] = '\0'; /* Supprime le '\n' final */
}
else {
 strcpy(reponse.contenu,"Désolé, je ne le connais pas");
}
/* On ne pas de fclose(f), comme on le faisait lors de la question Q3 */
/* En revanche, on se remet au début du fichier pour que la prochaine */
/* recherche de ce fils se déroule correctement (NB : on n'a pas      */
/* besoin de faire clearerr(f) car fseek efface l'indicateur de fin   */
/* de fichier pour le flux si l'appel se déroule bien)                */
if (fseek(f, 0, SEEK_SET) < 0) {
 perror("fseek");
 exit(EXIT_FAILURE);
}

/* Affichage */
printf("L'enfant %d du serveur a recu \"%s\" et repond \"%s\"\n",
```

```
            getpid(),
            requete->contenu,
            reponse.contenu );

 /* Connexion au point d'accès client et réponse */
 fdW = open(requete->pointAccesClient, O_WRONLY);
 if (fdW == -1) {
  perror("open(pointAccessClient)");
  exit(EXIT_FAILURE);
 }
 nbWrite = write(fdW, &reponse, sizeof(reponse));
 if (nbWrite < sizeof(reponse)) {
  perror("pb ecriture sur le fifo");
 }

 /* Dans cette application, le client ne renvoie pas de requête ultérieure*/
 /* nécessitant une réponse ==> On peut fermer ce tube                    */
 close(fdW);

}

void attenteRequete(int monNum){
 requete_t requete;
 FILE *f;
```

```
  /* Ouverture du fichier de M. et Mme */
  f = fopen("atis.txt", "r");
  if (f == NULL) {
    perror("open(atis.txt)");
    exit(EXIT_FAILURE);
  }

  /* Attente de requête */
  while(1){
    printf("L'enfant %d (monNum=%d) se met dans le pool\n", getpid(), monNum);

    /* On écrit qu'on est disponible pour recevoir une nouvelle requete */
    write(dansPool[1], &monNum, sizeof(monNum));

    /* On attend une requête */
    read(pereEnfant[monNum][0], &requete, sizeof(requete));
    printf("L'enfant %d a ete sorti du pool pour traiter une requete\n", getpid());

    /* On la traite */
    traiterRequete(&requete,f);
  }
}
```

```
int main() {
 requete_t requete;
 int fdR;
 int nbRead;
 int i;
 int enfant;

 /* Création du tube nommé du serveur */
 if (mkfifo(NOMPOINTACCESSERV, S_IRUSR|S_IWUSR) < 0) {
  if (errno != EEXIST) {
    perror("mkfifo(fifo client");
    exit(EXIT_FAILURE);
  }
  else {
    printf("%s existe deja : on suppose que c'est un fifo\n",
              NOMPOINTACCESSERV );
    printf(" et qu'on peut continuer le programme sans probleme\n");
    puts("");
  }
 }

 /* Création du pool de fils                                    */
 /* Le pool est représenté par un tube sur lequel le père lit les       */
 /* numéros des fils qui se considèrent prêts à traiter des requêtes     */
```

```
pipe(dansPool);
for (i=0 ; i<NBENFANTS ; i++){
  pipe(pereEnfant[i]);
}
for (i=0 ; i<NBENFANTS ; i++){
  if (fork() == 0) {
    attenteRequete(i);
  }
}

/* Ouverture de ce tube nommé (equivalent a une ouverture de connexion).
   NB : On l'ouvre en RDWR et pas en RDONLY,
   car (cf. Blaess) : si nous ouvrons le client en lecture seule, à chaque
   fois que le client se termine, la lecture du tube nomme va echouer
   car le noyau detecte une fermeture de fichier. En demandant un tube
   en lecture et en ecriture, nous evitons ce genre de situation, car il
   reste toujours au moins un descripteur ouvert sur la sortie
*/
fdR = open(NOMPOINTACCESSERV, O_RDWR);
if (fdR == -1) {
  perror("open(fifo)");
  exit(EXIT_FAILURE);
}
/* Attente de requêtes */
```

```
 do{
   nbRead = read(fdR, &requete, sizeof(requete));
   if (nbRead != sizeof(requete)) {
     perror("read sur fifo NOK");
     return EXIT_FAILURE;
   }
   /* On choisit, dans le pool, l'enfant en mesure de traiter cette requête */
   /* S'il n'y en a pas, on en attend un                                    */
   /* Noter qu'une fois que le read est fini, l'enfant n'est plus dans le   */
   /* pool !                                                                */
   read(dansPool[0], &enfant, sizeof(int));

   /* On envoie la requête a l'enfant */
   write(pereEnfant[enfant][1], &requete, sizeof(requete));
 }while(1);
}
```

Gestionnaire météo.

```
/*
```

Gestionnaire Météo:

!note: possiblement trop de #include par rapport à ce qui est nécessaire

vérifier le chemin des fichiers (différent Windows/Linux)

vérifier system(" clear ") pour effacer l'écran

utilse 2 getchar pour pause écran (car le buffer garde le premier ENTER et je ne sais plus comment éviter cela)

```
!programme: affiche menu 3 options

1: vérifie si ATIS existe ou sinon le crée et y écrit un string

2: écrit un second string dans atis si read_lock n'existe pas sinon ne fait rien !! mode a+ pour écrire à la

suite du premier string (si w+ on écrase le fichier)

3: quitter

*/


#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <errno.h>

#include <unistd.h>

#include <string.h>

#include <ctype.h>

#include <fcntl.h>


FILE * fpA;

FILE * fpW;

FILE * fpR;

char * fileA = "atis.txt";

char * fileR = "read_lock.txt";

char * fileW = "write_lock.txt";
```

```
void menu(){

system("clear");

printf("Options: \n");

printf("1: Vérifier existence ATIS\n");

printf("2: Modifier ATIS\n");

printf("x: Quitter\n");

printf("Choix: ");

}


void verifATIS() {

fpA = fopen(fileA,"r");

if(fpA == 0) { //atis pas trouvé => création et écriture

fpA = fopen(fileA, "w");

fprintf(fpA,"%s","EBLG 1803 00000KT 0600 FG OVC008 BKN040 PROB40 2024 0300 DZ FG OVC002
BKN040");

fclose(fpA);

printf("Le fichier ATIS a été créé\n");

getchar();

}

else {

printf("Fichier ATIS trouvé");

getchar();

}

}
```

```
void ecrireATIS() {

fpR = fopen(fileR, "r");

if(fpR != 0) { //read_lock existe => atis est lu par serveur => retour menu

printf("Le fichier ATIS est en cours de lecture\n");

getchar();

}

else {

fpW = fopen(fileW, "w"); //atis est libre => ecriture nouvelle ligne

fclose(fpW);

fpA = fopen(fileA, "a+"); //a+ = rajoute à la suite >< w+ = écraser fichier

fprintf(fpA,"%s","EBBR 0615 20015KT 8000 RA SCT010 OVC015 TEMPO 0608 5000 RA BKN005 BECMG
0810 9999 NSW BKN025");

fclose(fpA);

remove(fileW);

printf("Le fichier ATIS a été modifié");

getchar();

}

}



int main() {

char choix;

menu();

do{

choix=getchar();
```

```
switch(choix){
case '1': {
verifATIS();
getchar();
menu();
}
break;
case '2': {
ecrireATIS();
getchar();
menu();
}
break;
}
}
while(choix != 'x');
return 0;
}
```