TP 2 - PSE

3 séances (MAX. 4) en salle Machine

Gestion des processus
Verrouillage d'un fichier
Communication entre processus (tube)

Objectifs du TP :

1) Se familiariser avec les outils de base permettant de gérer le parallélisme et la communication entre processus sous Linux : utiliser les fonctions ***fork***, ***exec***, ***exit***, ***wait***, ***pipe***, ***lockf***
2) Utiliser ces appels système sous Linux pour écrire des programmes parallèles cohérents et déterministes.

Travail à effectuer :

Ce TP est organisé en 3 parties. La première partie consiste à contrôler l'accès de deux ou plusieurs processus à un fichier en commun, qui joue le rôle d'une ressource commune non partageable. La deuxième partie permet de faire de la programmation parallèle (exécution de plusieurs processus, de la même famille, simultanément) et la troisième partie fait communiquer des processus (de la même famille) via l'utilisation d'un tube. Donc, pour résumer, ce TP consiste à :

1) Etudier le programme ***ajout.c*** (vu en **TD 1**), qui effectue un traitement en exclusion mutuelle, vu en cours, le compiler et le faire exécuter avec ses différentes versions (avec/sans exclusion mutuelle),

2) Etudier un programme parallèle qui recherche les occurrences d'un caractère dans un texte, le tester puis le modifier, sous Linux, en langage C, afin de lui faire effectuer des actions complémentaires avec l'aide de vos chargés de TP et
3) Exécuter et étudier le programme effectuant une communication par tubes (***pipe***).

Description de la première partie du TP

Le programme suivant (***ajout.c***) effectue une boucle de 10 tours permettant d'augmenter de 1 à chaque tour la valeur d'un compteur inscrite dans un fichier nommé **NOMBRE**.
Un programme d'initialisation (***init.c***), à lancer auparavant, permet de donner une valeur initiale à ce compteur.

a) Ecrire un programme ***retrait.c*** qui va diminuer de 1 à chaque tour la valeur du compteur inscrite dans le fichier NOMBRE. ***retrait.c*** peut être facilement déduit du programme ***ajout.c***.
b) Vous devrez lancer en parallèle 2 processus ***ajout*** et ***retrait***, et observer les résultats.
c) L'exécution en parallèle de "***ajout***" et "***retrait***" devrait permettre de retomber sur la valeur initiale. Est-il le cas ?
d) Modifiez les 2 programmes ***ajout.c*** et ***retrait.c*** afin qu'ils puissent, tout en s'exécutant concurremment, fournir un résultat fiable.

```
/*  programme d'ajout sans precaution */
#include <sys/types.h>
#include <errno.h>
#include <sys/file.h>
#include <unistd.h>


#define ID_PROC         Ajout
#define N_Boucles       10
#define ATTENTE         100
#define NB_INST         5


static int num_inst = 0 ;


// fonction de ralentissement permettant un entrelacement des programmes concurrents
void inst_suiv(void) {
    int cpte;
    for(cpte=0; cpte  < ATTENTE*100000; cpte++);
    num_inst= (num_inst<NB_INST)?num_inst+1:1 ; // affichage du n° d'instruction ds la
boucle
    printf(" AJOUT instruction %d \n", num_inst);
```

```
}

void main (int nbarg, char *tbarg[])
{
   int df_v, df_n, nombre, num_boucle, nbo_lus, nbo_ecrits ;
   off_t lret ;

   if ((df_n= open ("NOMBRE",O_RDWR|O_CREAT,0644)) < 0) { perror("ouverture de
NOMBRE"); exit(errno);}


// boucle de modification de la valeur de NOMBRE
   for (num_boucle=0;num_boucle<N_Boucles;num_boucle++)
  {
        inst_suiv();
        if ((lret=lseek(df_n,(off_t)0,SEEK_SET)) < 0) { perror("1er lseek sur NOMBRE");
exit(errno); }
        inst_suiv();
        if ((nbo_lus=read(df_n,&nombre,sizeof(int))) < sizeof(int)) { perror("lecture
NOMBRE"); exit(errno); }
        inst_suiv();
        nombre++;
        inst_suiv();
        if ((lret=lseek(df_n,(off_t)0,SEEK_SET)) < 0)  { perror("2eme lseek sur
NOMBRE"); exit(errno); }
        inst_suiv();
        if ((nbo_ecrits=write(df_n,&nombre,sizeof(int))) < sizeof(int)) {
perror("ecriture NOMBRE"); exit(errno); }
   }


// lecture de la valeur finale
  if ((lret=lseek(df_n,(off_t)0,SEEK_SET)) < 0) { perror("dernier lseek sur NOMBRE");
exit(errno);}
  if ((nbo_lus=read(df_n,&nombre,sizeof(int))) < sizeof(int)) { perror("lecture
NOMBRE"); exit(errno); }
  printf("Valeur finale (PLUS) nombre: %d \n",nombre); close(df_n) ;}
```

Description de la deuxième partie du TP

Le programme parallèle suivant (***rech-car.c***) recherche l'occurrence d'un caractère dans un fichier texte en activant des processus coopérants (un ensemble de processus fils).

a) Vous devrez étudier ce programme, le compiler et l'exécuter.

b) Ensuite vous améliorez son écriture (surtout la partie du **code en gras**) en diminuant sa taille par l'utilisation d'appels système de la famille ***exec*** tout en spécifiant un nom d'un fichier

exécutable et la liste des arguments dont ce dernier a besoin pour s'exécuter avec succès.

```
/*
 Exercice de programmation parallele
 Recherche du nombre d'occurrences d'un caractere dans un texte
 On passe en parametres le nom du fichier, le caractere a rechercher, le nombre de
processus a lancer */
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>

#define     erreur(X)   { perror(X) ; exit (errno) ;}
#define     TBloc       1024
#define     NMaxesclave 5

int main (int nbarg, char *tbarg[])
{ int NBEsclave;
  int  idpro, df , cpt ,cptcar = 0, np[NMaxesclave], etat, NbOccu = 0;
  char Bloc [TBloc -1];

 if (nbarg < 4) { printf("utilisation %s fichier caractere nb_d'esclaves\n", tbarg[0]);
exit(-1); }

 sscanf(tbarg[3],"%d",&NBEsclave);

 if (NBEsclave > NMaxesclave ) { printf("nb esclave trop eleve%d\n", NBEsclave );
exit(-1); }
 if ((df=open(tbarg[1], O_RDONLY))< 0 ) erreur("erreur d'ouverture du fichier en
parametre")

 for (cpt=1; cpt <= NBEsclave ; cpt++)
 {
      if (( idpro= fork ()) < 0)
      erreur("creation de processus")

      if ( idpro == 0)  # processus esclave
      { int nbo ;
         int ordre;
         ordre=cpt;
         cpt=NBEsclave++ ; // arrêt de la boucle pour l'esclave
         while ((nbo=read(df, Bloc, TBloc)) > 0) # lecture du fichier par blocs
         {
              int index, IMax;
              //IMax= (TBloc - nbo)? nbo : TBloc ;
              Imax=nbo ; // nbre max de car a etudier
              for (index=0; index < IMax - 1 ; index++ )
               if ( Bloc[index]  == *tbarg[2] ) cptcar++ ;
              { long cal ; for ( cal=0; cal< 500000 ;cal++);} // boucle de
ralentissement
```

```
            }
               exit ( cptcar ); // renvoie le nb d'occurrences trouvées
          } // fin des esclaves (fi idpro==0)
      } //end for

      // le pere attend la fin des fils
      for (cpt=0; cpt < NBEsclave ; cpt++)
      {
         np[cpt]= wait( &etat );
         NbOccu= NbOccu + (etat >> 8);
         printf(" le processus %d trouve %d %c \n",np[cpt],etat >> 8,*tbarg[2]);
       }
       printf(" le fichier %s contient %d %c \n",tbarg[1],NbOccu,*tbarg[2]);
       exit(0);
}
```

Description de la troisième partie du TP

Le programme ***tube-b.c*** permet de créer un tube de communication entre un processus fils et un processus père. Le processus père écrit dans le tube des *messages* qui sont lus par le fils.

a) Vous étudiez, compilez et exécutez le programme effectuant une communication par tubes.
b) Ensuite vous devrez remplacer la partie du **code en gras** par un appel système de la famille ***exec*** tout en spécifiant un nom d'un fichier exécutable et la liste des arguments dont ce dernier a besoin pour s'exécuter avec succès.

```
/* programme de démonstration de l'appel pipe pour mettre en œuvre la communication
entre processus*/
#include <sys/types.h>
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

#define Erreur(x)       {perror(x); exit(errno);}
#define     mod   5


// récupération d'une chaîne de car en paramètre
int main (int nbarg, char *tbarg[])
{
  int  idpro, tube[2], nbcarlus , attente ;
  char chaine[19], lgmes, lgrec ;

  if(nbarg <= 1) { printf (" utilisation %s <suite de chaines de car> \n", tbarg[0]);
exit(-1) ; }
```

```
 srand(getpid());
 pipe(tube);

if (( idpro= fork ()) < 0) Erreur("creation de processus");

if ( idpro == 0)  { // le fils
      close(tube[1]);
      nbcarlus= read( tube[0], &lgrec, 1) ;
      while ( nbcarlus  > 0)
      {
           nbcarlus= read( tube[0], chaine, lgrec);
           chaine[lgrec] = 0 ;
           printf( "\t\t\t\t\tchaine lue : %s\n",chaine);
           nbcarlus= read( tube[0], &lgrec, 1) ;
     }
     close( tube[0]);
     printf (" \t\t\t\tje suis le recepteur, num  %d et je me termine \n", getpid());
}
else      { // le pere
           int i, np, etat;
           close (tube[0]);
           for (i=1; i< nbarg ; i++)
           {
              lgmes=strlen(tbarg[i]); write( tube[1], &lgmes, 1) ;
              write( tube[1],tbarg[i],strlen(tbarg[i]));
              printf( "chaine emise %d eme message : %s\n",i,tbarg[i]);
              attente = rand() % mod ; sleep (attente);
           }
           close(tube[1]);

           printf (" je suis l'emetteur, num  %d et je me termine \n", getpid());
           np=wait(&etat);
           printf (" Fin du programme\n");
              }
}
```