Programmation système UNIX

Exemples du cours

Joëlle Luter – Juillet 2015

Liste des exemples

/* 1 - Affichage a l'ecran du contenu d'un fichier dont le nom est passe en parametre */ 3
/* 2 - Lancement de ls et stockage de son resultat dans le fichier toto. (ls>toto) */ 4
/* 3 - Affichage par le processus de ses: PID, PPID, GID, EGID, UID, EUID */.....5
/* 4 - Affichage de l'environnement d'un processus */ 6
/* 5 - Creation de processus fils */.....7
/* 6 - Creation de processus fils -Adoption du fils par init*/.....8
/* 7 - Remplacement du code du processus : exec() */.....9
/* 8 - Communication entre process de meme famille par tube */.....10
/* 9 - ps | wc -l */.....12
/* 10 - Tubes nommés */.....13
/* 11 - Gestion des signaux - Posix */.....15
/* 12 - Gestion des signaux - ANSI C */.....16
/* 13 - Gestion des signaux - temporisation */.....17
/* 14 - Emission d'un signal */.....18
/* 15 - Exemples de tirage de cle */.....19
/* 16 – IPC : Memoire partagee - exemple_shm1.c */.....20
/* 17 - IPC: Files de messages - exemple_msg1.c */.....23
/* 18 – IPC : Memoire partagee + semaphore - exemple_sem1.c */ 25
/* 19 – IPC : Memoire partagee + semaphore - exemple_product.c */.....31
/* 20 – IPC : Memoire partagee + semaphore - exemple_conso.c */.....34
/* 21 - Creation d'un thread Posix*/.....37
/* 22 - Problème de concurrence entre threads */.....38
/* 23 - Exclusion mutuelle de 2 threads */.....40
/* 24 - Cohorte : ouverture barriere d un parking */ 42
/* 25 - Variable conditionnelle : gestion d'alarme */ 44
/* 26 - Threads et fichiers: lecture concurrente avec read()*/.....46
/* 27 - Threads et fichiers: lecture concurrente avec pread()*/.....48

/* 1 - Affichage a l'ecran du contenu d'un fichier dont le nom est passe en parametre */

```c
/* Affichage sur la sortie standard du contenu d'un fichier dont le nom est
passe en parametre */

#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char** argv){
  int fd, nb;
  char buf[512];

/* controle du nombre d'arguments passes sur la ligne de commande */
  if (argc!=2) {
    write(2, "Nb arguments incorrect\n",30);
    exit(1);
  }

/* ouverture du fichier en lecture seule */
  if ((fd=open(argv[1], O_RDONLY))==-1) {
    perror("Ouverture");
    exit(1);
  }

/* boucle de lecture-affichage du contenu du fichier */
  while((nb=read(fd, buf, sizeof(buf)))>0) {
      write(1,buf,nb);
      }

/* fermeture du fichier */
  close(fd);

  return(0);
}
```

/* 2 - Lancement de ls et stockage de son resultat dans le fichier toto. (ls>toto) */

```
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main(void){
  int fd;

  /* Ouverture fichier toto */

  if ((fd=open("toto", O_RDWR|O_CREAT|O_TRUNC,0644))==-1) {
    perror("Ouverture toto");
    exit(1);
  }

  /* Fermeture de la sortie standard */
  close(1);

  /* Redirection de toto sur la sortie standard */

  if (dup(fd) <0) {
    perror("Duplication");
    exit(1);
  }

  /* Fermeture du descripteur fd */

  close(fd);

  /* Recouvrement du process courant par le code de ls */

  execlp("ls", "ls", NULL);

  perror("Lancement ls");

  return(1);
}
```

/* 3 - Affichage par le processus de ses: PID, PPID, GID, EGID, UID, EUID */

```
#include <unistd.h>
#include <stdio.h>

int main(){

  printf("Mon identifiant (PID) est %d\n", getpid());
  printf("L'identifiant de mon pere(PPID) est %d\n", getppid());
  printf("Mon utilisateur reel (UID) est %d\n", getuid());
  printf("Mon utilisateur effectif (EUID) est %d\n", geteuid());
  printf("Mon groupe reel (GID) est %d\n", getgid());
  printf("Mon groupe effectif (EGID) est %d\n", getegid());

  return(0);
}

/* Resultat obtenu

   Mon identifiant (PID) est 2306
   L'identifiant de mon pere(PPID) est 2078
   Mon utilisateur reel (UID) est 501
   Mon utilisateur effectif (EUID) est 501
   Mon groupe reel (GID) est 501
   Mon groupe effectif (EGID) est 501
*/
```

/* 4 - Affichage de l'environnement d'un processus */

```
#include <stdio.h>

int main(int argc, char** argv, char** envp){
  int i=0;
  for (i=0;envp[i]!=NULL; i++)
    printf("%s\n",envp[i]);
  printf("\n%d variables d'environnement initialisées\n",i);
  return(0);
}
```

/* 5 - Creation de processus fils */

```
#include <stdio.h>
#include <unistd.h>  /* pour fork() */
#include <stdlib.h>  /* pour exit() */
#include <sys/wait.h> /* pour wait */
#include <signal.h>  /* pour sleep */

int main(void){
  int pid1=0, pid2=0;
  int status;  /* pour code retour wait()*/

  if ((pid1=fork()) == -1){
    perror("Creation fils 1");
    exit(1);
  }

  if (pid1 == 0) {
    /* Code du fils 1 */
    printf("\nJe suis le fils 1 de PID  %d, mon pere est %d\n", getpid(),
getppid());
    printf("\nFils 1 vous dit Au revoir!!!\n");
    exit(0);
  }

  /* Code du pere */

  if ((pid2=fork()) == -1){
    perror("Creation fils 2");
    exit(1);
  }

 if (pid2 == 0) {
    /* Code du fils 2 */
    printf("\nJe suis le fils 2 de PID  %d, mon pere est %d\n", getpid(),
getppid());
    printf("\nFils 2 vous dit Au revoir!!!\n");
    exit(0);
  }

/* Code du pere */

 printf("Je suis le pere de PID %d et j'ai 2 fils %d et %d \n", getpid(), pid1,
pid2);

 wait(&status);
 wait(&status);


  return(0);
}
```

/* 6 - Creation de processus fils -Adoption du fils par init*/

```
#include <stdio.h>
#include <unistd.h>  /* pour fork() */
#include <stdlib.h>  /* pour exit() */
#include <sys/wait.h> /* pour wait */
#include <signal.h>  /* pour sleep */

int main(void){
  int pid1=0, pid2=0;

  if ((pid1=fork()) == -1){
    perror("Creation fils 1");
    exit(1);
  }

  if (pid1 == 0) {
    /* Code du fils 1 */
    printf("\nJe suis le fils 1 de PID  %d, mon pere est %d\n", getpid(),
getppid());
    sleep(2);
    printf("\nFils1 : Mon pere est %d \n", getppid());
    printf("\nFils 1 vous dit Au revoir!!!\n");
    exit(0);
  }

  /* Code du pere */

  if ((pid2=fork()) == -1){
    perror("Creation fils 2");
    exit(1);
  }

 if (pid2 == 0) {
    /* Code du fils 2 */
    printf("\nJe suis le fils 2 de PID  %d, mon pere est %d\n", getpid(),
getppid());
    sleep(3);
    printf("\nFils2 : Mon pere est %d \n", getppid());
    printf("\nFils 2 vous dit Au revoir!!!\n");
    exit(0);
  }

/* Code du pere */

 printf("Je suis le pere de PID %d et j'ai 2 fils %d et %d \n", getpid(), pid1,
pid2);
 sleep(1);
 printf("\nPere vous dit Au revoir!!!\n");



  return(0);
}
```

/* 7 - Remplacement du code du processus : exec() */

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

int main(int argc, char** argv) {

  if (argc != 3){
    printf("Nb arguments invalide \n");
    exit(1);
  }

  switch(argv[1][0]) {
  case 'c':
    execl("/usr/bin/gzip", "gzip", argv[2], NULL);
    perror("lancement compress");
    exit(1);

  case 'd':
    execl("/usr/bin/gunzip", "gunzip", argv[2], NULL);
    perror("lancement uncompress");
    exit(1);

  default:
    printf("argument invalide \n");
    exit(1);
  }

}
```

```
/* 8 - Communication entre process de meme famille par tube */
/* pere : cree un tube, cree 2 fils */
/* fils1 : "client"  inscrit une commande dans le tube*/
/* fils 1: "vendeur" : recupere la commande du client et l'affiche
*/



#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>


int main(void) {

  int p=0, tube[2], s, nb;
  char buf[512];

  /* creation du tube */
  if (pipe(tube) == -1) {
    perror("Creation pipe impossible");
    exit(1);
  }

  /* creation du premier processus fils (client) */
  if ((p=fork())==-1){
    perror("fork");
    exit(1);
  }

  if (p==0) {
    /* fermeture du descripteur en lecture sur le tube */
    close(tube[0]);

    /* ecriture de la commande dans le tube */
    if (write(tube[1], "chocolat", strlen("chocolat"))==-1){
      perror("ecriture tube");
      exit(1);
    }
    close(tube[1]) ;
    exit(0);

  } /* fin du code du fils 1 */

  /* creation du second processus fils (vendeur) */
  if ((p=fork())==-1){
    perror("fork");
    exit(1);
  }

  if (p==0) {
    /* code du processus fils 2 */

    /* fermeture du descripteur en ecriture sur le tube */
    close(tube[1]);

    /* lecture de la commande dans le tube */
```

```
    if ((nb=read(tube[0], buf, sizeof(buf)))==-1){
      perror("ecriture tube");
      exit(1);
    }

    /* affichage sur la sortie standard*/
    write(1, buf, nb);


    exit(0);
  } /* fin du code du fils 2 */

  /* le processus pere ferme les descripteurs qu'il n'utilise pas et attend la
terminaison des fils */
  close(tube[0]);
  close(tube[1]);

  wait(&s);
  wait(&s);


  return 0;
}
```

/* 9 - ps | wc -l */

```
/* lancement de la commande ps | wc -l : le processus pere cree le tube, cree un
fils pour ps et un autre pour wc et attend leur terminaison*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(void) {
  int p=0, tube[2], s;

  /* creation du tube */
  if (pipe(tube) == -1) {
    perror("Creation pipe impossible");
    exit(1);
  }

  /* creation du premier processus fils */
  if ((p=fork())==-1){
    perror("fork");
    exit(1);
  }

  if (p==0) {
    /* code du processus fils 1 : va executer la commande ps */
    close(1);       /* fermeture sortie std */
    dup(tube[1]);   /* redirection sortie std sur desc. en ecriture du tube */
    close(tube[0]);  /* fermeture du descripteur inutilise */
    close (tube[1]); /* inutile de garder 2 descripteurs ouverts pour le
                        meme fichier */
    execlp("ps", "ps", NULL);
    perror("ps");
    exit(0);
  } /* fin du code du fils 1 */

  /* creation du second processus fils */
  if ((p=fork())==-1){
    perror("fork");
    exit(1);
  }

  if (p==0) {
    /* code du procesuss fils 2 : va executer la commande wc -l */
    close(0);
    dup(tube[0]);
    close(tube[0]);
    close (tube[1]);
    execlp("wc", "wc","-l", NULL);
    perror("wc");
    exit(0);
  } /* fin du code du fils 2 */

  /* le processus pere ferme les descripteurs qu'il n'utilise pas*/
  close(tube[0]);
  close(tube[1]);
  /* le processus pere attend la terminaison des fils */
  wait(&s);
  wait(&s);
  return 0;
}
```

/* 10 - Tubes nommés */

/* programme appele avec "vendeur" ou le texte de la commande en parametre*/
/* "vendeur" :le process cree un tube appele "CARNET", l'ouvre en lecture, recupere la commande du client et l'affiche */
/* commande : le process ouvre le tube nomme "CARNET" en ecriture et y inscrit la commande */

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>  /* pour mkfifo */

int main(int argc, char **argv) {

  int fd;
  char commande[512];
  FILE *in;

  if (argc <2) {
    printf("Nombre d'arguments invalide \n");
    exit(1);
  }

  if (strcmp(argv[1], "vendeur") == 0) {  /* code du vendeur */

    if (mkfifo("CARNET", 0600)==-1) {   /* creation du tube */
      perror("vendeur: creation CARNET");
      exit(1);
    }

    if ((fd=open("CARNET", O_RDONLY))==-1) { /* ouverture du tube en lecture */
      perror("vendeur: ouverture CARNET");
      exit(1);
    }

    if ((in=fdopen(fd, "r"))==NULL) { /* ouverture d'un flux associe a fd */
      perror("vendeur: ouverture flux CARNET");
      exit(1);
    }

    fscanf(in, "%s", commande);
    printf("Recu commande : %s\n", commande);

    fclose(in);
    unlink("CARNET");
  }

  else {   /* code du client */

    if ((fd=open("CARNET", O_WRONLY))==-1) {
      perror("client: ouverture CARNET");
      exit(1);
    }

    if (write(fd, argv[1], strlen(argv[1]))!= strlen(argv[1])) {
      perror("client: ecriture CARNET");
      exit(1);
```

```
    }
    close(fd);
  }
  return 0;
}


/* Exemple de lancement:

$./a.out vendeur &
$./a.out chocolat
Recu commande : chocolat

*/
```

```
/* 11 - Gestion des signaux - Posix */
 /* Controle d'avancement d'une application:
Le programme principal effectue un calcul
- Blocage de CTRL-C
- Reception de SIGUSR1 (16 sous Solaris) : Affichage du resultat
intermediaire et suite du calcul
- Reception de SIGUSR2 (17 sous Solaris) : Affichage du resultat
et fin du programme
*/
#include <stdio.h>
#include <limits.h>  /* pour ULONG_MAX */
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

unsigned long somme=0;

/* fonction de traitement des signaux SIGUSR1 et SIGUSR2 */
void espion(int numSig) {
  printf("Somme = %lu\n", somme);
  switch(numSig){
  case SIGUSR1:
    break;
  case SIGUSR2:
    exit(0);
  }
}

int main(void) {

  sigset_t masque;
  struct sigaction act;

  /* Application du masque de blocage des signaux : on interdit au processus de
recevoir CTRL-C (SIGINT). Tous les autres signaux sont autorisés */

  sigemptyset(&masque);
  sigaddset(&masque, SIGINT);
  sigprocmask(SIG_SETMASK, &masque, NULL);

/* Installation du handler espion() pour SIGUSR1 et SIGUSR2. On interdit la
reception de tout  signal durant l'execution de la fonction espion() */

  act.sa_handler=espion;
  act.sa_flags=0;
  sigfillset(&(act.sa_mask));
  sigaction(SIGUSR1, &act, NULL);
  sigaction(SIGUSR2, &act, NULL);

/* Traitement */

  printf("Je suis le processus numero: %d\n", getpid());
  while(somme<ULONG_MAX) {
    somme++;
  }
  printf("Calcul termine: %lu", somme);
  return(0);
}
```

```
/* 12 - Gestion des signaux - ANSI C */

/* Controle d'avancement d'une application:
Le programme principal effectue un calcul
- Ignorance de CTRL-C
- Reception de SIGUSR1 (16 sous Solaris) : Affichage du resultat
intermediaire et suite du calcul
- Reception de SIGUSR2 (17 sous Solaris) : Affichage du resultat
et fin du programme
*/

#include <stdio.h>
#include <limits.h>  /* pour ULONG_MAX */
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

unsigned long somme=0;

/* fonction de traitement des signaux SIGUSR1 et SIGUSR2 */
void espion(int numSig) {
  printf("Somme = %lu\n", somme);
  switch(numSig){
  case SIGUSR1:
    signal(SIGUSR1, espion); /* On rearme le signal */
    break;
  case SIGUSR2:
    exit(0);
  }
}

int main(void) {

  /* on interdit au processus de recevoir CTRL-C (SIGINT). */
  signal(SIGINT, SIG_IGN);

/* Installation du handler espion() pour SIGUSR1 et SIGUSR2. */

  signal(SIGUSR1, espion);
  signal(SIGUSR2, espion);

/* Traitement */

  printf("Je suis le processus numero: %d\n", getpid());
  while(somme<ULONG_MAX) {
    somme++;
  }
  printf("Calcul termine: %lu", somme);

  return(0);
}
```

```
/* 13 - Gestion des signaux  - temporisation */

/*  Mise en place d'une temporisation - Utilisation d'un saut non
local
Le programme principal attend 5 seconde la saisie d'une valeur
Quand le delai est depasse, il affiche un message et se termine

- Envoi du signal SIGALRM au bout de 5 secondes
- La fonction de gestion du signal renvoie au programme principal

*/

#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <setjmp.h>

sigjmp_buf contexteAlarme; /* buffer de sauvegarde du contexte d'execution */

/* fonction de traitement du signal SIGALRM */
void gestionAlarme(int numSig) {
/* branchement a l'instruction sauvegardee dans le buffer contexteAlarme : appel
a sigsetjmp()  */
  siglongjmp( contexteAlarme, 1);
}

int main(void) {

  int i;

/* Installation du handler gestionAlarme  pour SIGALRM */

  signal(SIGALRM,gestionAlarme);

/* Traitement */

  printf("Vous avez 5 secondes pour entrer un nombre entier\n");

/* Memorisation du contexte d'execution dans le buffer contexteAlarme */

  if (sigsetjmp(contexteAlarme, 1) == 0) { /* sigsetjmp() renvoie 0 : 1er appel
*/
    alarm(5);     /* Mise en place du timer */
    scanf("%d", &i);
    alarm(0);  /* saisie effectuee, on annule le timer */
    printf("OK: %d saisi\n",i);
  }
  else {
/* sigsetjmp() a renvoyé 1 : branchement depuis la fonction gestionAlarme */
    printf("Trop tard!!!\n");
  }

  return(0);
}
```

/* 14 - Emission d'un signal */

/* Un processus crée un fils et se met en attente.
Le fils envoie le signal de terminaison à son père
et attend qu'il ait terminé son exécution.*/

```
#include <unistd.h>
#include <signal.h>

int main(void) {

int pere;
if (fork() == 0) {
   /* processus fils */
   pere = getppid();
   kill(pere, SIGTERM);
   while(kill(pere, 0)==0) {
      printf("pere non termine\n");
      sleep(1);
   }
   printf("mon nouveau pere %d\n",
                              getppid());
   }
else {
   /* processus pere */
   sleep(10);
  }
return 0;
}
```

/* 15 - Exemples de tirage de cle */

```
#include <sys/ipc.h>
#include <stdio.h>

int main(void) {

  printf("%d\n", ftok("exemple_pipe.c", 'A'));   /* 1090916642 */
  printf("%d\n", ftok("exemple_pipe.c", 'G'));   /* 1191579938 */
  printf("%d\n", ftok("exemple_fork1.c", 'A'));  /* 1090916631 */
  printf("%d\n", ftok("exemple_fork1.c", 'G'));  /* 1191579927 */
  printf("%d\n", ftok("exemple_pipe.c", 1));     /* 17174818  */
  printf("%d\n", ftok("exemple_fork1.c", 1));    /* 17174807 */

  return 0;
}
```

/* Exemples de tirage de cle */
/* Limites de ftok */

```
/* Le fichier lien_fork1 est un lien hard sur exemple_fork1.c (meme numero d'i-
node)
La cle tiree est la meme */

#include <sys/ipc.h>
#include <stdio.h>

int main(void) {

  printf("%d\n", ftok("exemple_fork1.c", 'A'));  /* 1090916631 */
  printf("%d\n", ftok("lien_fork1", 'A'));       /* 1090916631 */


  return 0;
}
```

```
/* 16 – IPC : Memoire partagee - exemple_shm1.c */

/* 2 programmes:
- exemple_shm1.c: cree un segment de memoire partagee,
                  y inscrit un message
                  se met en sommeil en attente de reception d'une
reponse
                  affiche la reponse
- exemple_shm1b.c: ouvre le segment de memoire
                   affiche le message recu
                   ecrit la reponse

Attention : exemple_shm1 doit etre lance en premier. Pas de
gestion de la synchronisation.

*/

#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>


int main(void) {

  int cle = 1234, id;
  char *adresse;  /* adresse du segment de memoire partagee */
  char msg[80]="Quand venez vous diner?";

/* Creation d'un segment de mmoire partagee de 80 octets */

  if ((id = shmget(cle, 80, 0600|IPC_CREAT))==-1) {
    perror("Creation memoire partagee");
    exit(1);
  }

/* Attachement de la zone de memoire a l'espace memoire du processus */

  if ((int)(adresse = shmat(id, NULL, 0)) == -1) {
    perror("Attachement memoire partagee");
    shmctl(id, IPC_RMID, NULL);  /* Suppression du segment */
    exit(1);
  }

  /* Ecriture du message */

  strcpy(adresse, msg);

/* Attente de modification du segment de memoire */

  while (strncmp(msg, adresse, 80)==0)
    sleep(1);

/* Affichage de la reponse */

  printf("Ils viennent diner %s\n", adresse);
```

```
/* Detachement du segment de memoire de l'espace du processus */

  shmdt(adresse);

/* Suppression du segment */

  shmctl(id, IPC_RMID, NULL);

  return 0;
}
```

```
/*      IPC : Memoire partagee - exemple_shm1b.c */
/* 2 programmes:
- exemple_shm1.c: cree un segment de memoire partagee,
                  y inscrit un message
                  se met en sommeil en attente de reception d'une
reponse
                  affiche la reponse
- exemple_shm1b.c: ouvre le segment de memoire
                   affiche le message recu
                   ecrit la reponse

Attention : exemple_shm1 doit etre lance en premier. Pas de
gestion de la synchronisation.
*/

#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

int main(void) {

  int cle = 1234, id;
  char *adresse;  /* adresse du segment de memoire partagee */

  /* Ouverture du segment de mmoire partagee */

  if ((id = shmget(cle, 80, 0))==-1) {
    perror("Ouverture memoire partagee");
    exit(1);
  }

  /* Attachement de la zone de memoire a l'espace memoire du processus */

  if ((int)(adresse = shmat(id, NULL, 0)) == -1) {
    perror("Attachement memoire partagee");
    exit(1);
  }

  /* Affichage du message recu */

  printf("Message recu: %s\n", adresse);

  /* Ecriture de la reponse */

  strcpy(adresse,"Ce soir");

  sleep(1);

/* Detachement du segment de memoire de l'espace du processus */

  shmdt(adresse);

  return 0;
}
```

```
/* 17 - IPC: Files de messages - exemple_msg1.c */
/* 2 programmes:
   - exemple_msg1.c: cree ou ouvre une file de messages,
                     y inscrit 3 messages
                     se termine
   - exemple_msg1b.c: cree ou ouvre une file de messages
                      affiche les 3 messages lus
                 supprime la file
                      se termine
la file de message peut etre cree par l'un des deux processus, La
lecture est par defaut bloquante
*/

#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>


int main(void) {

  int cle , id, i;

  struct msg {
    long type;
    char data[80];
  }message;

 char *mes[3]={"message 1", "message 2","message 3"};

/* Obtention de la cle */

  cle = ftok("exemple_msg1.c", 'A');

/* Creation d'une file de messages */

  if ((id = msgget(cle, 0600|IPC_CREAT))==-1) {
    perror("Creation file de messages");
    exit(1);
  }

  /* Ecriture de 3 messages */

  for (i=0;i<3;i++) {
    message.type = i+1;
    strcpy(message.data, mes[i]);
    if (msgsnd(id, &message, 80, 0)==-1){
      perror("Ecriture message");
      exit(1);
    }
  }

  return 0;
}
```

```
/*      IPC : Files de messages - exemple_msg1b.c */
/* 2 programmes:
   - exemple_msg1.c: cree ou ouvre une file de messages,
                     y inscrit 3 messages
                     se termine
   - exemple_msg1b.c: cree ou ouvre une file de messages
                      affiche les 3 messages lus
                 supprime la file
                      se termine

 la file de message peut etre cree par l'un des deux processus, La
lecture est par defaut bloquante
*/

#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>


int main(void) {

  int cle , id, i;

  struct msg {
    long type;
    char data[80];
  }message;

/* Obtention de la cle */

  cle = ftok("exemple_msg1.c", 'A');

/* Creation d'une file de messages */

  if ((id = msgget(cle, 0600|IPC_CREAT))==-1) {
    perror("Creation file de messages");
    exit(1);
  }

  /* Ecriture de 3 messages */

  for (i=0;i<3;i++) {
    if (msgrcv(id, &message, 80, 0, 0)==-1){
      perror("Ecriture message");
      exit(1);
    }
    printf("%s\n", message.data);
  }

  /* Suppression de la file de messages */
  msgctl(id, IPC_RMID, NULL);
  return 0;
}
```

```
/* 18 – IPC : Memoire partagee + semaphore - exemple_sem1.c */

/* 2 programmes:
- exemple_sem1.c: cree ou ouvre le segment de memoire partagee,
                  cree ou ouvre le semaphore,
                  prend le semaphore
                  inscrit un message
                  libere le semaphore invitation
                  se met en sommeil en attente de reception d'une
reponse
                  affiche la reponse
- exemple_sem1b.c:  cree ou ouvre le segment de memoire partagee,
                    cree ou ouvre le semaphore,
                    ouvre le segment de memoire
              se met en sommeil en attente de reception d'une
invitation
                    affiche le message recu
                    ecrit la reponse
                    libere le semaphore reponse
*/

#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/sem.h>
#include <errno.h>

int main(void) {

  int cle = 1234, cleSem = 4567, id, idSem;
  char *adresse;  /* adresse du segment de memoire partagee */
  char msg[80]="Quand venez vous diner?";

  union semun  /* pour semctl() */
  {
    int val;
    struct semid_ds *buf;
    unsigned short int *array;
    struct seminfo *__buf;
  }arg;

  struct sembuf semBuf;  /* pour semop() */
  unsigned short int tab[2];

  /* Creation d'un segment de memoire partagee de 80 octets */

  if ((id = shmget(cle, 80, 0600|IPC_CREAT))==-1) {
    perror("Creation memoire partagee");
    exit(1);
  }

  /* Attachement de la zone de memoire a l'espace memoire du processus */

  if ((int)(adresse = shmat(id, NULL, 0)) == -1) {
    perror("Attachement memoire partagee");
```

```
    shmctl(id, IPC_RMID, NULL);  /* Suppression du segment */
    exit(1);
  }

  /* Creation d'un ensemble de semaphores */

  if ((idSem = semget(cleSem, 2, IPC_CREAT|IPC_EXCL|0600)) != -1) {

    /* Creation de l'ensemble, on initialise les semaphores a 0*/
    tab[0]=0;  /* invitation */
    tab[1]=0;  /* reponse */
    arg.array=tab;
    if (semctl(idSem, 0, SETALL, arg)==-1) {
      perror("exemple_sem1:init semaphore");
      semctl(idSem, 0, IPC_RMID);
      shmctl(id, IPC_RMID, NULL);
      exit(1);
    }
  }
  else {
    /* erreur: soit l'ensemble existe deja, dans ce cas, on l'ouvre sans
initialisation
               soit erreur reelle*/

    if (errno == EEXIST) {   /* l'ensemble est deja cree */
      if ((idSem = semget(cleSem, 2, IPC_CREAT)) == -1) {
      perror("Creation semaphore 1");
      shmctl(id, IPC_RMID, NULL);
      exit(1);
      }
    }
    else {  /* erreur reelle */
      perror("Creation semaphore 2");
      shmctl(id, IPC_RMID, NULL);
      exit(1);
    }
  }

  /* Ecriture du message */

  strcpy(adresse, msg);

  /* Liberation du semaphore invitation */
  semBuf.sem_num=0;
  semBuf.sem_op=1;
  semBuf.sem_flg=0;

  if (semop(idSem, &semBuf, 1)==-1) {
    perror("Liberation semaphore invitation");
    semctl(idSem, 0, IPC_RMID);
    shmctl(id, IPC_RMID, NULL);
    exit(1);
  }

/* Attente de modification du segment de memoire */

  semBuf.sem_num=1;
  semBuf.sem_op=-1;
  semBuf.sem_flg=0;
  if (semop(idSem, &semBuf, 1)==-1) {
    perror("Attente reponse");
    semctl(idSem, 0, IPC_RMID);
```

```
    shmctl(id, IPC_RMID, NULL);
    exit(1);
  }

  /* Affichage de la reponse */

  printf("Ils viennent diner %s\n", adresse);

  /* Detachement du segment de memoire de l'espace du processus */

  shmdt(adresse);

  /* Suppression du segment */

  semctl(idSem, 0, IPC_RMID);
  shmctl(id, IPC_RMID, NULL);

  return 0;
}
```

```
/*      IPC : Memoire partagee + semaphore - exemple_sem1b.c */

/* 2 programmes:
- exemple_sem1.c: cree ou ouvre le segment de memoire partagee,
                  cree ou ouvre le semaphore,
                  prend le semaphore
                  inscrit un message
                  libere le semaphore invitation
                  se met en sommeil en attente de reception d'une
reponse
                  affiche la reponse
- exemple_sem1b.c:  cree ou ouvre le segment de memoire partagee,
                    cree ou ouvre le semaphore,
                    ouvre le segment de memoire
              se met en sommeil en attente de reception d'une
invitation
                    affiche le message recu
                    ecrit la reponse
                    libere le semaphore reponse
*/

#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/sem.h>
#include <errno.h>

int main(void) {

  int cle = 1234, cleSem = 4567, id, idSem;
  char *adresse;  /* adresse du segment de memoire partagee */

  union semun {  /* pour semctl() */
    int val;                    /* value for SETVAL */
    struct semid_ds *buf;       /* buffer for IPC_STAT, IPC_SET */
    unsigned short int *array;  /* array for GETALL, SETALL */
    struct seminfo *__buf;      /* buffer for IPC_INFO */
  }arg;

  struct sembuf semBuf;  /* pour semop() */
  unsigned short int tab[2];

  /* Ouverture du segment de memoire partagee */

  if ((id = shmget(cle, 80, 0600|IPC_CREAT))==-1) {
    perror("Ouverture memoire partagee");
    exit(1);
  }

  /* Attachement de la zone de memoire a l'espace memoire du processus */

  if ((int)(adresse = shmat(id, NULL, 0)) == -1) {
    perror("Attachement memoire partagee");
    shmctl(id, IPC_RMID, NULL);
    exit(1);
```

```
  }

  /* Creation d'un ensemble de semaphores */

  if ((idSem = semget(cleSem, 2, IPC_CREAT|IPC_EXCL|0600)) != -1) {

    /* Creation de l'ensemble, on initialise les semaphores a 0*/

    tab[0]=0;  /* invitation */
    tab[1]=0;  /* reponse */
    arg.array=tab;
    if (semctl(idSem, 0, SETALL, arg)==-1) {
      perror("exemple_sem1:init semaphore");
      semctl(idSem, 0, IPC_RMID);
      shmctl(id, IPC_RMID, NULL);
      exit(1);
    }
  }
  else {
    /* erreur: soit l'ensemble existe deja, dans ce cas, on l'ouvre sans
initialisation
       soit erreur reelle*/

    if (errno == EEXIST) {   /* l'ensemble est deja cree */
      if ((idSem = semget(cleSem, 2, IPC_CREAT)) == -1) {
      perror("Obtention semaphore 1");
      shmctl(id, IPC_RMID, NULL);
      exit(1);
      }
    }
    else {  /* erreur reelle */
      perror("Creation semaphore 2");
      shmctl(id, IPC_RMID, NULL);
      exit(1);
    }
  }

  /* Attente de modification du segment de memoire */

  semBuf.sem_num=0;
  semBuf.sem_op=-1;
  semBuf.sem_flg=0;
  if (semop(idSem, &semBuf, 1)==-1) {
    perror("Attente invitation");
    semctl(idSem, 0, IPC_RMID);
    shmctl(id, IPC_RMID, NULL);
    exit(1);
  }

  /* Affichage du message recu */

  printf("Message recu: %s\n", adresse);

  /* Ecriture de la reponse */

  strcpy(adresse,"Ce soir");

  /* Liberation du semaphore reponse */
  semBuf.sem_num=1;
  semBuf.sem_op=1;
  semBuf.sem_flg=0;
```

```
  if (semop(idSem, &semBuf, 1)==-1) {
    perror("Liberation semaphore invitation");
    semctl(idSem, 0, IPC_RMID);
    shmctl(id, IPC_RMID, NULL);
    exit(1);
  }

  /* Detachement du segment de memoire de l'espace du processus */

  shmdt(adresse);

  sleep(2);

  /* Suppression des IPC */
  semctl(idSem, 0, IPC_RMID);
  shmctl(id, IPC_RMID, NULL);

  return 0;
}
```

/* 19 – IPC : Memoire partagee + semaphore - exemple_product.c */

/* 2 programmes:
- exemple_product.c:
ecrit des nombres dans un tableau d'entiers (implante en memoire partagee), gere en buffer tournant de taille maximale 50
se bloque quand il n'y a plus de place disponible

- exemple_conso.c:
lit des nombres dans le tableau en memoire partagee et les affiche
se bloque quand il n'y a plus rien a lire.

Ces 2 programmes tournent en boucle infinie. Les signaux CTRL-C (SIGINT) et SIGUSR1 permettent de terminer leur execution en supprimant les IPC, CTRL-C (SIGINT). A cet effet, on met les identifiant des IPC en variable globale
***/**

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/sem.h>
#include <errno.h>
#include <signal.h>

#define MAX 50    /* taille du buffer */
typedef int tablo[MAX];

/* Variables globales */
int shmId, semId;

/* Traitement des signaux SIGUSR1 et SIGINT */
void fin(int numSig) {
  semctl(semId, 0, IPC_RMID);
  shmctl(shmId, IPC_RMID, NULL);
  exit(0);
}

/* Operation P (-1) sur un semaphore */
int P(int id, int num) {
  struct sembuf semBuf;  /* pour semop() */

  semBuf.sem_num=num;
  semBuf.sem_op=-1;
  semBuf.sem_flg=0;

  if (semop(id, &semBuf, 1)==-1) {
    perror("Producteur: Prise semaphore");
    semctl(semId, 0, IPC_RMID);
    shmctl(shmId, IPC_RMID, NULL);
    return -1;
  }
return 0;
}
```

```
/* Operation V (+1) sur un semaphore */
int V(int id, int num) {
  struct sembuf semBuf;  /* pour semop() */

  semBuf.sem_num=num;
  semBuf.sem_op=1;
  semBuf.sem_flg=0;

  if (semop(id, &semBuf, 1)==-1) {
    perror("Producteur: Liberation semaphore");
    semctl(semId, 0, IPC_RMID);
    shmctl(shmId, IPC_RMID, NULL);
    return -1;
  }
return 0;
}

int main(void) {

  key_t cleShm , cleSem;
  int indice=0, numMsg=0;
  tablo *adresse;  /* adresse du segment de memoire partagee */
  union semun  /* pour semctl() */
  {
    int val;
    struct semid_ds *buf;
    unsigned short int *array;
    struct seminfo *__buf;
  }arg;
  unsigned short int tabSem[2];

  /* Tirage des cles */

  cleSem = ftok("/home/LUTER/PROSYST/exemple_product.c", 'A');
  cleShm = ftok("/home/LUTER/PROSYST/exemple_product.c", 'B');

  /* Creation d'un segment de memoire partagee  */

  if ((shmId = shmget(cleShm, sizeof(adresse), 0600|IPC_CREAT))==-1) {
    perror("Creation memoire partagee");
    exit(1);
  }

  /* Attachement de la zone de memoire a l'espace memoire du processus */

  if ((int)( adresse = (tablo*)shmat(shmId, NULL, 0) ) == -1) {
    perror("Attachement memoire partagee");
    shmctl(shmId, IPC_RMID, NULL);  /* Suppression du segment */
    exit(1);
  }

  /* Creation d'un ensemble de semaphores */

  if ((semId = semget(cleSem, 2, IPC_CREAT|IPC_EXCL|0600)) != -1) {

    /* Creation de l'ensemble, on initialise les semaphores */
    tabSem[0]=0;  /* nb ecrit */
    tabSem[1]=MAX;  /* nb dispo */
    arg.array=tabSem;
    if (semctl(semId, 0, SETALL, arg)==-1) {
      perror("exemple_sem1:init semaphore");
      semctl(semId, 0, IPC_RMID);
```

```
      shmctl(shmId, IPC_RMID, NULL);
      exit(1);
    }
  }
  else {
    /* erreur: soit l'ensemble existe deja, dans ce cas, on l'ouvre sans
initialisation
       soit erreur reelle*/

    if (errno == EEXIST) {   /* l'ensemble est deja cree */
      if ((semId = semget(cleSem, 2, IPC_CREAT|0600)) == -1) {
      perror("Creation semaphore 1");
      shmctl(semId, IPC_RMID, NULL);
      exit(1);
      }
    }
    else {  /* erreur reelle */
      perror("Creation semaphore 2");
      shmctl(shmId, IPC_RMID, NULL);
      exit(1);
    }
  }

  signal(SIGUSR1, fin);
  signal(SIGINT, fin);

  /* Boucle d'ecriture */

  while(1) {
    sleep(1);

      /* Decrementation du semaphore nb dispo */
    if (P(semId, 1)==-1)
      exit(1);

    /* modification du segment de memoire */
    numMsg++;
    indice%=MAX;
    (*adresse)[indice]=numMsg;
    printf("Producteur: ecriture message n° %d\n",numMsg);
    indice++;

    /* Incrementation du semaphore nb ecrits */
    if (V(semId, 0)==-1)
      exit(1);
  }

  return 0;
}
```

```
/* 20 – IPC : Memoire partagee + semaphore - exemple_conso.c */

/* 2 programmes:
   - exemple_product.c:
         ecrit des nombres dans un tableau d'entiers (implante en
memoire partagee), gere en buffer tournant de taille maximale 50
       se bloque quand il n'y a plus de place disponible

   - exemple_conso.c:
         lit des nombres dans le tableau en memoire partagee et
les affiche
       se bloque quand il n'y a plus rien a lire.

Ces 2 programmes tournent en boucle infinie. Les signaux CTRL-C
(SIGINT) et  SIGUSR1 permettent de terminer leur execution en
supprimant les IPC, CTRL-C (SIGINT). A cet effet, on met les
identifiant des IPC en variable globale
*/

#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/sem.h>
#include <errno.h>
#include <signal.h>

#define MAX 50    /* taille du buffer */
typedef int tablo[MAX];

/* Variables globales */
int shmId, semId;

/* Traitement des signaux SIGUSR1 et SIGINT */
void fin(int numSig) {
  semctl(semId, 0, IPC_RMID);
  shmctl(shmId, IPC_RMID, NULL);
  exit(0);
}

/* Operation P (-1) sur un semaphore */
int P(int id, int num) {
  struct sembuf semBuf;  /* pour semop() */

  semBuf.sem_num=num;
  semBuf.sem_op=-1;
  semBuf.sem_flg=0;

  if (semop(id, &semBuf, 1)==-1) {
    perror("Producteur: Prise semaphore");
    semctl(semId, 0, IPC_RMID);
    shmctl(shmId, IPC_RMID, NULL);
    return -1;
  }
return 0;
}
```

```
/* Operation V (+1) sur un semaphore */
void V(int id, int num) {
  struct sembuf semBuf;  /* pour semop() */

  semBuf.sem_num=num;
  semBuf.sem_op=1;
  semBuf.sem_flg=0;

  if (semop(id, &semBuf, 1)==-1) {
    perror("Producteur: Liberation semaphore");
    semctl(semId, 0, IPC_RMID);
    shmctl(shmId, IPC_RMID, NULL);
    return -1;
  }
return 0;
}

int main(void) {

  key_t cleShm , cleSem;
  int indice=0;
  tablo *adresse;  /* adresse du segment de memoire partagee */
  union semun  /* pour semctl() */
  {
    int val;
    struct semid_ds *buf;
    unsigned short int *array;
    struct seminfo *__buf;
  }arg;
  unsigned short int tabSem[2];

  /* Tirage des cles */

  cleSem = ftok("/home/LUTER/PROSYST/exemple_product.c", 'A');
  cleShm = ftok("/home/LUTER/PROSYST/exemple_product.c", 'B');

  /* Creation d'un segment de memoire partagee */

  if ((shmId = shmget(cleShm, sizeof(adresse), 0600|IPC_CREAT))==-1) {
    perror("Creation memoire partagee");
    exit(1);
  }

  /* Attachement de la zone de memoire a l'espace memoire du processus */

  if ((int)( adresse = (tablo*)shmat(shmId, NULL, 0) ) == -1) {
    perror("Attachement memoire partagee");
    shmctl(shmId, IPC_RMID, NULL);  /* Suppression du segment */
    exit(1);
  }

  /* Creation d'un ensemble de semaphores */

  if ((semId = semget(cleSem, 2, IPC_CREAT|IPC_EXCL|0600)) != -1) {

    /* Creation de l'ensemble, on initialise les semaphores */
    tabSem[0]=0;  /* nb ecrit */
    tabSem[1]=MAX;  /* nb dispo */
    arg.array=tabSem;
    if (semctl(semId, 0, SETALL, arg)==-1) {
      perror("exemple_sem1:init semaphore");
```

```
      semctl(semId, 0, IPC_RMID);


      shmctl(shmId, IPC_RMID, NULL);
      exit(1);
    }
  }
  else {
    /* erreur: soit l'ensemble existe deja, dans ce cas, on l'ouvre sans
initialisation
       soit erreur reelle*/

    if (errno == EEXIST) {   /* l'ensemble est deja cree */
      if ((semId = semget(cleSem, 2, IPC_CREAT|0600)) == -1) {
      perror("Creation semaphore 1");
      shmctl(semId, IPC_RMID, NULL);
      exit(1);
      }
    }
    else {  /* erreur reelle */
      perror("Creation semaphore 2");
      shmctl(shmId, IPC_RMID, NULL);
      exit(1);
    }
  }

  signal(SIGUSR1, fin);
  signal(SIGINT, fin);

  /* Boucle de lecture */

  while(1) {
  sleep(1);

    /* Decrementation du semaphore nb ecrits */
    if (P(semId,0 )==-1)
      exit(1);

    /* Affichage du message lu */
    indice%=MAX;
    printf("Consommateur: Lecture message n° %d\n",(*adresse)[indice]);
    indice++;

    /* Incrementation du semaphore nb dispo */
    if (V(semId, 1)==-1)
      exit(1);
  }

  return 0;
}
```

/* 21 - Creation d'un thread Posix*/

```
/* gcc -Wall -lpthread -lposix4 -o thread1 ex_thread1.c */

#define _REENTRANT
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

void *foncThread1(void *arg){      /* affiche son parametre */
  int val = (int)arg;
  printf("argument thread 1 :%d:\n", val);
  pthread_exit(NULL);
}

void *foncThread2(void *arg){      /* affiche son parametre */
  char *ch=(char *)arg;
  printf("argument thread 2 :%s:\n", ch);
  pthread_exit(NULL);
}

void *foncThread3(void *arg){      /* retourne un entier */
  int i=10; ;
  pthread_exit((void *)i);
}

int main(void){

  pthread_t idThread1, idThread2,  idThread3; /* identifiant des threads */
  int val=5;
  char *chaine="bonjour";
  void *retour;

  /* creation thread 1*/
  if (pthread_create(&idThread1, NULL, foncThread1, (void*)val)!=0){
    printf("erreur creation thread1\n");
    exit(1);
  }

  /* creation thread 2*/
  if (pthread_create(&idThread2, NULL, foncThread2, (void*)chaine)!=0){
    printf("erreur creation thread2\n");
    exit(1);
  }

  /* creation thread 3*/
  if (pthread_create(&idThread3, NULL, foncThread3, NULL)!=0){
    printf("erreur creation thread3\n");
    exit(1);
  }

  /* attente de la terminaison des threads */
  pthread_join(idThread1, NULL);
  pthread_join(idThread2, NULL);
  pthread_join(idThread3, &retour);

  /* affichage de la valeur retournee par le thread 3*/
  printf("valeur de retour thread 3 :%d:\n", (int)retour);

  return 0;
}
```

```
/* 22 - Problème de concurrence entre threads */
/* Execution concurrente de 2 threads */
/* gcc -Wall -lpthread -lposix4 -o thread2 ex_thread2.c */

#define _REENTRANT
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

unsigned int _cpt;

void *foncThread1(void *arg){
  unsigned int savcpt;
  while (_cpt<5){
    savcpt=_cpt;
    printf("Thread1: _cpt=%u - ", _cpt);
    _cpt++;
    printf(" %u+1=%u\n",savcpt, _cpt);
  }
}

void *foncThread2(void *arg){
  unsigned int savcpt;
  while (_cpt<5){
    savcpt=_cpt;
    printf("Thread2: _cpt=%u - ", _cpt);
    _cpt++;
    printf(" %u+1=%u\n",savcpt, _cpt);
  }
}

int main(void){

  pthread_t idThread1, idThread2; /* identifiant des threads */

  _cpt=0;

  /* creation thread 1*/
  if (pthread_create(&idThread1, NULL, foncThread1, NULL)!=0){
    printf("erreur creation thread1\n");
    exit(1);
  }

  /* creation thread 2*/
  if (pthread_create(&idThread2, NULL, foncThread2, NULL)!=0){
    printf("erreur creation thread2\n");
    exit(1);
  }

  /* attente de la terminaison des threads */
  pthread_join(idThread1, NULL);
  pthread_join(idThread2, NULL);

  printf("thread main :%u:\n", _cpt);

  return 0;
}
```

```
/* ex_thread2.c */
/* Execution concurrente de 2 threads */

/* Résultat de l'exécution */

Thread1: _cpt=0 -  0+1=1
Thread1: _cpt=1 -  1+1=2
Thread1: _cpt=2 -  2+1=3
Thread1: _cpt=3 -  3+1=4
Thread1: _cpt=4 -  4+1=5
Thread2: _cpt=0 -  0+1=6     -----> PROBLEME!!!!!
thread main :6:
```

/* 23 - Exclusion mutuelle de 2 threads */
/* gcc -Wall -lpthread -lposix4 -o thread3 ex_thread3.c */

```
#define _REENTRANT
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

unsigned int _cpt;
pthread_mutex_t _m = PTHREAD_MUTEX_INITIALIZER; /* declaration du mutex */

void *foncThread1(void *arg){
  unsigned int savcpt;
  while (_cpt<5){
    pthread_mutex_lock(&_m);
    savcpt=_cpt;
    printf("Thread1: _cpt=%u - ", _cpt);
    _cpt++;
    printf(" %u+1=%u\n",savcpt, _cpt);
    pthread_mutex_unlock(&_m);
  }
}

void *foncThread2(void *arg){
  unsigned int savcpt;
  while (_cpt<5){
    pthread_mutex_lock(&_m);
    savcpt=_cpt;
    printf("Thread2: _cpt=%u - ", _cpt);
    _cpt++;
    printf(" %u+1=%u\n",savcpt, _cpt);
    pthread_mutex_unlock(&_m);
  }
}

int main(void){
    pthread_t idThread1, idThread2; /* identifiant des threads */

  _cpt=0;

  /* creation thread 1*/
  if (pthread_create(&idThread1, NULL, foncThread1, NULL)!=0){
    printf("erreur creation thread1\n");
    exit(1);
  }

  /* creation thread 2*/
  if (pthread_create(&idThread2, NULL, foncThread2, NULL)!=0){
    printf("erreur creation thread2\n");
    exit(1);
  }

  /* attente de la terminaison des threads */
  pthread_join(idThread1, NULL);
  pthread_join(idThread2, NULL);

  printf("thread main :%u:\n", _cpt);
  return 0;
}
```

```
/* ex_thread3.c */
/* Exclusion mutuelle de 2 threads */

/* Résultat de l'exécution */

Thread1: _cpt=0 -  0+1=1
Thread1: _cpt=1 -  1+1=2
Thread1: _cpt=2 -  2+1=3
Thread1: _cpt=3 -  3+1=4
Thread1: _cpt=4 -  4+1=5
Thread2: _cpt=5 -  5+1=6   --------> OK
thread main :6:
```

```
/* 24 - Cohorte : ouverture barriere d un parking */
/* cf support cours Programmation Concurrente */
/* gcc -Wall -lpthread -o thread8 ex_thread8.c */


#define _REENTRANT
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>

void detectionEntree(); /* fonction bloquante tant qu aucune voiture ne se
presente */
void detectionSortie(); /* fonction bloquante tant qu aucune voiture ne se
presente */

void ouvertureEntree(); /* fonction actionnant l ouverture d une barriere */
void ouvertureSortie(); /* fonction actionnant l ouverture d une barriere */

sem_t _nbLibr; /* declaration du semaphore nb places libres  */


void *entree(void *arg){

  while(1){
    /* Attente detection voiture */
    detectionEntree();

    /*P(nbLibres) : bloquant si pas de place */
    if (sem_wait(&_nbLibr)==-1){
      perror("Entree:erreur wait:");
      pthread_exit(NULL);
    }
    /* ouverture de la barriere */
    ouvertureEntree();

  } /*fin while */

}

void *sortie(void *arg){

  while(1){
   /* Attente detection voiture */
    detectionSortie();

    /*V(nbLibres) : incremente le nombre de places libres */
    if (sem_post(&_nbLibr)==-1){
      perror("Sortie:erreur post:");
      pthread_exit(NULL);
    }
    /* ouverture de la barriere */
    ouvertureSortie();

  }/*fin while */
}
```

```
int main(void){

  pthread_t idThreadE, idThreadS; /* identifiant des threads */

  /* initialisation semaphore au nombre de places du parking */
  if (sem_init(&_nbLibr, 0, 50)==-1){
    perror("main:erreur init nbLibr:");
    exit(1);
  }

  /* creation thread entree */
  if (pthread_create(&idThreadE, NULL, entree, NULL)!=0){
    printf("erreur creation thread entrees\n");
    exit(1);
  }

  /* creation thread sortie */
  if (pthread_create(&idThreadS, NULL, sortie, NULL)!=0){
    printf("erreur creation thread sorties\n");
    exit(1);
  }

  /* attente de la terminaison des threads */
  pthread_join(idThreadE, NULL);
  pthread_join(idThreadS, NULL);

/* suppression du semaphores */
  sem_destroy(&_nbLibr);

  return 0;
}}
```

/* 25 - Variable conditionnelle : gestion d'alarme */
/*(d'apres Ch. Blaess)*/
/* gcc -Wall -lpthread -lposix4 -o thread5 ex_thread5.c */

```
#define _REENTRANT
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

pthread_cond_t _condAlarm = PTHREAD_COND_INITIALIZER; /* declaration de la
                                                        variable conditionnelle */
pthread_mutex_t _m = PTHREAD_MUTEX_INITIALIZER;  /* declaration du mutex */

int alerte = 0;

void *temperature(void *arg){
  int t = 20, s;
  while(1){
    t+=(rand_r(&s)%9) - 5;
    printf("temperature = %d\n", t);
    if((t<16)||(t>24)){
      pthread_mutex_lock(&_m);
      alerte = 1;
      pthread_cond_signal(&_condAlarm); // Signal
      pthread_mutex_unlock(&_m);
    }
    else
      alerte = 0;
    sleep(1);
  } /* fin while*/
}

void *alarme(void *arg){
  while(1){
    pthread_mutex_lock(&_m);
    while (!alerte)
       pthread_cond_wait(&_condAlarm, &_m); // Attente
    pthread_mutex_unlock(&_m);
    printf("alarme\n");
    sleep(1);
  } /* fin while*/
}

int main(void){

  pthread_t idThreadT, idThreadA; /* identifiant des threads */

  /* creation thread 1*/
  if (pthread_create(&idThreadT, NULL, temperature, NULL)!=0){
    printf("erreur creation thread1\n");
    exit(1);
  }

  /* creation thread 2*/
  if (pthread_create(&idThreadA, NULL, alarme, NULL)!=0){
    printf("erreur creation thread2\n");
    exit(1);
  }
```

```
  /* attente de la terminaison des threads */
  pthread_join(idThreadT, NULL);
  pthread_join(idThreadA, NULL);

  printf("fin thread main\n");

  return 0;
}
```

```
/* 26 - Threads et fichiers: lecture concurrente avec read()*/
/* gcc -Wall -lpthread -lposix4 -o thread6 ex_thread6.c */

#define _REENTRANT

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int _fd;

void *fonction1(void *arg){
  int i;
  char buf[50];

 i=0;
  while((i<50)&&(read(_fd, &buf[i], 1)>0)){
    i++;
  }
  buf[i]='\0';
  printf("thread1 :%s:", buf);

  pthread_exit(NULL);

}

void *fonction2(void *arg){
  int i;
  char buf[50];

  i=0;
  while((i<50)&&(read(_fd, &buf[i], 1)>0)){
    i++;
  }
  buf[i]='\0';
  printf("thread2 :%s:", buf);


  pthread_exit(NULL);
}

int main(void){

  pthread_t idThread1, idThread2; /* identifiant des threads */

  if ((_fd=open("toto.txt", O_RDONLY))==-1){
    perror("main ouverture fichier:");
    pthread_exit(NULL);
  }

  /* creation thread 1*/
  if (pthread_create(&idThread1, NULL, fonction1, NULL)!=0){
    printf("erreur creation thread1\n");
    exit(1);
  }
```

```
  /* creation thread 2*/
  if (pthread_create(&idThread2, NULL, fonction2, NULL)!=0){
    printf("erreur creation thread2\n");
    exit(1);
  }

  /* attente de la terminaison des threads */
  pthread_join(idThread1, NULL);
  pthread_join(idThread2, NULL);

  close(_fd);

  printf("fin thread main\n");

  return 0;
}
```

Fichier d'origine:

```
abcdefghijklmnopqrstuvwxyz
```

Résultat

```
thread1 :abcdefghijklmnopqrstuvwxyz:
thread2 :ilmnopqrstuvwxyz:
fin thread main
```

```
/* 27 - Threads et fichiers: lecture concurrente avec pread()*/
/* gcc -Wall -lpthread -lposix4 -o thread7 ex_thread7.c */

#define _REENTRANT

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int _fd;

void *fonction1(void *arg){
  int i;
  char buf[50];

 i=0;
  while((i<50)&&(pread(_fd, &buf[i], 1, i)>0)){
    i++;
  }
  buf[i]='\0';
  printf("thread1 :%s:", buf);

  pthread_exit(NULL);

}

void *fonction2(void *arg){
  int i;
  char buf[50];

  i=0;
  while((i<50)&&(pread(_fd, &buf[i], 1, i)>0)){
    i++;
  }
  buf[i]='\0';
  printf("thread2 :%s:", buf);


  pthread_exit(NULL);
}

int main(void){

  pthread_t idThread1, idThread2; /* identifiant des threads */

  if ((_fd=open("toto.txt", O_RDONLY))==-1){
    perror("main ouverture fichier:");
    pthread_exit(NULL);
  }

  /* creation thread 1*/
  if (pthread_create(&idThread1, NULL, fonction1, NULL)!=0){
    printf("erreur creation thread1\n");
    exit(1);
  }
```

```
  /* creation thread 2*/
  if (pthread_create(&idThread2, NULL, fonction2, NULL)!=0){
    printf("erreur creation thread2\n");
    exit(1);
  }

  /* attente de la terminaison des threads */
  pthread_join(idThread1, NULL);
  pthread_join(idThread2, NULL);

  close(_fd);

  printf("fin thread main\n");

  return 0;
}
```

Fichier d'origine:

```
abcdefghijklmnopqrstuvwxyz
```

Résultat

```
thread1 :abcdefghijklmnopqrstuvwxyz:
thread2 : abcdefghijklmnopqrstuvwxyz:
fin thread main
```