**Examen Module Noyau**
**M1 – Master SAR**
**Septembre 2005**

**3 heures - Tout document autorisé**
**Barème donné à titre indicatif**

**P. Sens**

## Exercice 1 : Socket (4 points)

Le serveur présenté dans l'annexe A permet de traiter des requêtes de clients en TCP. Modifiez son code pour qu'il puisse fonctionner simultanément en TCP et UDP.

Le serveur doit pouvoir répondre aussi bien à des clients UDP que TCP. Il doit rester identifier dans les deux cas par le même numéro de port.

*Réponse :*

```
#define PORTSERV 7100
int main(int argc, char *argv[])
{
  struct sockaddr_in sin;  /* Nom de la socket de connexion */
  struct sockaddr_in exp;  /* Nom de la socket du client */
  int sudp ;                 /* Socket de connexion */
  int scom;          /* Socket de communication */

  int fromlen = sizeof (exp);
  int cpt;


  /* creation de la socket TCP */
  if ((sc = socket(AF_INET,SOCK_STREAM,0)) < 0) {
      perror("socket");
      exit(1);
  }

  /* creation de la socket UDP */
  if ((sudp = socket(AF_INET,SOCK_DGRAM,0)) < 0) {
    perror("socket");
    exit(1);
  }

  bzero((char *)&sin,sizeof(sin));
  sin.sin_addr.s_addr = htonl(INADDR_ANY);
  sin.sin_port = htons(PORTSERV);
  sin.sin_family = AF_INET;


  /* nommage UDP */
  if (bind(sudp,(struct sockaddr *)&sin,sizeof(sin)) < 0) {
    perror("bind");
    exit(2);
```

```
  }


  /* nommage TCP */
  if (bind(sc,(struct sockaddr *)&sin,sizeof(sin)) < 0) {
    perror("bind");
    exit(2);
  }

  listen(sc, 5);

  /* Boucle principale */

  for (;;) {

    /* contruire le masque du select */
    fd_set mselect;

    /* Construire le masque du select */
    FD_ZERO(&mselect);
    FD_SET(sc, &mselect); /* socket connexion TCP */
    FD_SET(sudp, &mselect); /* socket UDP */

    if (select(sudp+1, &mselect, NULL, NULL, NULL) == -1) {
      perror("select");
      exit(2);
    }

    /* Un evenement a eu lieu sur sudp ou sc */
    if (FD_ISSET(sudp, &mselect)) {
      /*** UDP ***/
     if (recvfrom(sudp,&cpt,sizeof(cpt),0,(struct sockaddr *)&exp,&fromlen)==-
1){
       perror("recvfrom");
       exit(2);
     }

     cpt+=10;

     /*** Envoyer la reponse ***/
     if (sendto(sudp,&cpt,sizeof(cpt),0,(struct sockaddr *)&exp,fromlen)==-1) {
       perror("sendto");
       exit(2);
   }
    }
    else {
      /*** TPC ***/

      if ( (scom = accept(sc, (struct sockaddr *)&exp, &fromlen))== -1) {
        perror("accept");
        exit(2);
      }

      /*** Lire le message ***/
      if (read(scom,&cpt, sizeof(cpt)) < 0) {
        perror("read");
        exit(1);
      }

      /*** Traitement du message ***/
      cpt+=10;
```

```
      /*** Envoyer la réponse ***/
      if (write(scom, &cpt, sizeof(cpt)) == -1) {
        perror("write");
        exit(2);
      }

      /* Fermer la connexion */
      shutdown(scom,2);
      close(scom);
    }

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

**Exercice 2 : Fichiers - inode (4 points)**

On considère une partition (device) dont la table des inodes sur disque occupe les blocs de 2 à 100 000. Les blocs ont une taille de 512 octets. Chaque inode occupe 128 octets. Les inodes sont rangées de manière consécutive à partir du bloc 2. (La première entrée de la table des inodes occupe les octets 0 à 127 du bloc 2, la seconde entrée les octets 128 à 255 du bloc 2 etc.).

Pour simplifier on considère que le numéro d'inode correspond à l'indice d'entrée dans la table des inodes sur disque.

On considère un fichier d'inode numéro 5 de nom unique « fichier1 ». Ce fichier contient un seul bloc (le bloc numéro 210 000) contenant la chaîne de caractères «examen». Le répertoire courant est associé à l'inode numéro 7. On suppose que les inodes du fichier et du répertoire courant **ne sont pas** en mémoire. Le bloc du fichier n'est également pas en mémoire.

Soit la portion de code suivante :

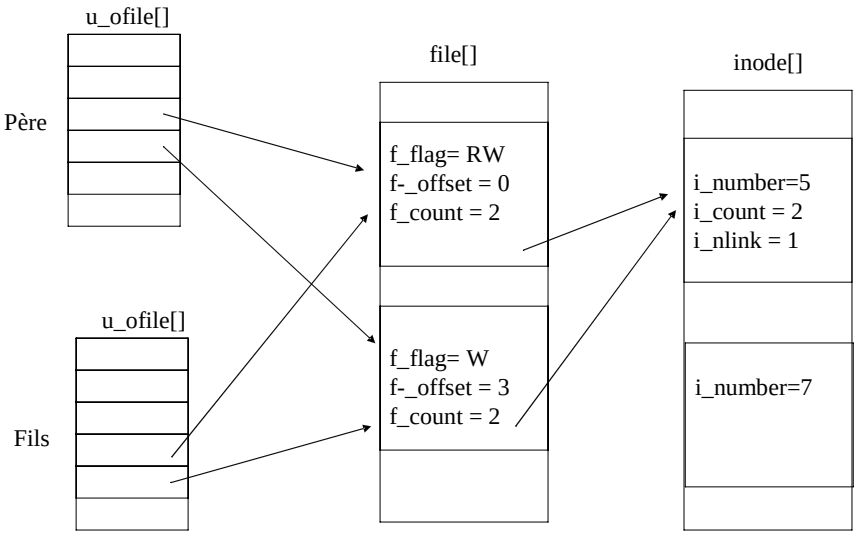
```
1:  int main() {
2:    char buf[10];
3:    int fd1, fd2;
4:    fd1 = open("./fichier1", O_RDWR);
5:    fd2 = open("./fichier1", O_WRONLY);
6:    read(fd2,buf,2);
7:    if (fork() == 0) {
8:       write(fd2, "sep",  3);
9:       exit(1);
10:   }
```

```
11:    wait(NULL);
12:    write(fd1, "05",  2);
13:   close(fd1);
14:    close(fd2);
15:    return 0;
16:  }
```

**1.1** Quel est le contenu du fichier à la fin de ce programme ? (1 point)

*Réponse :*
*« 05pam»*

**1.2** Faites un schéma des structures de données concernant les fichiers après la ligne 8 (en y représentant toutes les informations pertinentes). (1,5 points)




**1.3** Quel est le nombre d'appel à bread (avec les numéro de bloc) et le nombre d'entrées/sorties physiques fait par cette séquence et à quels moments ? (1,5 points)

*Réponse :*
*3 appels à bread :*
** sur le bloc 3 à la ligne 4*
** sur le bloc 210 000 aux lignes 8 et 12.*
*2 E/S physiques :*
** 1 E/S sur bloc 3 ligne 3 pour ramener inode 5 et 7*
** 1 E/S différé du bloc 210 000 ligne 13*

*(éventuellement écriture différée du bloc 3 pour les inodes)*

**Exercice 3 : Programmation dans le noyau (9 points)**

*Remarque : la question 3.3 est complètement indépendante des questions 3.1 et 3.2*

On souhaite implémenter **dans le noyau** les appels système **sys_flock** et **sys_funlock** permettant de verrouiller et déverrouiller un fichier.

Les prototypes des fonctions sont les suivants :

```
int sys_flock(int fd);
```

Permet de verrouiller le fichier ouvert ayant comme descripteur `fd`. Si l'inode correspond à fd n'a été verrouillé par personne l'appel n'est pas bloquant sinon l'appel est bloquant jusqu'à qu'un processus appel funlock

```
int sys_funlock(int fd);
```

Débloque **un** des processus en attente sur l'inode correspond au fichier ouvert de descripteur `fd`.

Ces deux fonctions retournent -1 en cas d'erreur 0 sinon.

Un « flag » IUSERLOCK est ajouté à la liste de états de l'inode (voir annexe B).

**3.1:** Programmez la fonction noyau `sys_flock` en ajoutant si nécessaire des champs dans la structure d'une inode (3 points).

*Réponse :*
*Remarque on suppose que les inodes verrouillées sont toujours en mémoire (étant ouvertes par au moins un processus).*

*Un champs i_lockcount est ajouté dans la structure inode pour compter le nombre de processus en attente de verrou sur l'inode (voir annexe)*

*(la « difficulté » est de réveiller un seul processus )*

*#include <user.h> /* pour la variable « u », zone u du processus courant */*

```
int  sys_flock(int fd) {
     struct file *fp;
     fp = u.u_ofile[fd];

     if (fp == NULL)
          return -1;

     if (!p->f_inode->i_flag &USERLOCK)
          fp->f_inode->i_flag |= IUSERLOCK;   /* Le fichier n'est pas verrouillé */
          return 0 ;
     }

     fp->f_inode->i_lockcount++;
     while (fp->f_inode->i_flag & IUSERLOCK)
          sleep(fp->f_inode);

     p->f_inode->i_lockcount--;
     if (fp->f_inode->i_lockcount > 0)
               fp->f_inode->i_flag &= IUSERLOCK; /* Reverouiller pour le suivant */
     return 0 ;
}
```

**3.2:** Programmez la fonction noyau `sys_funlock` (3 points).

*Réponse :*

```
int  sys_funflock(int fd) {
     struct file *fp;
     fp = u.u_ofile[fd];

     if (fp == NULL)
          return -1;

     if ( fp->f_inode->i_flag & IUSERLOCK) {
          wakeup(fp->f_inode);
          fp->f_inode->i_flag &= ~IUSERLOCK
     }
     return 0;
}
```

**3.3** : On souhaite programmer dans le noyau une fonction sys_blockcount qui permet de

compter le nombre de blocs en mémoire occupé (dans le buffer cache) pris par un fichier. Le propotype de la fonction est le suivant :

```
int sysblockcount(char *name) ;
```

Cette fonction retourne le nombre de buffers cache occupés par le fichier de nom « name ». Pour simplifier on suppose que les fichiers n'ont pas de blocs d'indirections.

Programmez la fonction sysblockcount (3 points).

| *Reponse : à faire …* |
|---|

**Exercice 4 : Etude de codes dans le noyau (3 points)**

On veut analyser l'appel système `dup` qui permet de dupliquer un descripteur de fichier.

Donnez l'algorithme de la fonction dup donné dans l'annexe B.

| *Reponse : à faire …* |
|---|

## ANNEXE A

```
…
#define PORTSERV 7100
int main(int argc, char *argv[])
{
  struct sockaddr_in sin;  /* Nom de la socket de connexion */
  struct sockaddr_in exp;  /* Nom de la socket du client */
  int sc;  /* Socket de connexion */
  int scom;           /* Socket de communication */

  int fromlen = sizeof (exp);
  int cpt;

  /* creation de la socket TCP */
  if ((sc = socket(AF_INET,SOCK_STREAM,0)) < 0) {
      perror("socket");
      exit(1);
  }

  bzero((char *)&sin,sizeof(sin));
  sin.sin_addr.s_addr = htonl(INADDR_ANY);
  sin.sin_port = htons(PORTSERV);
  sin.sin_family = AF_INET;

  /* nommage TCP */
  if (bind(sc,(struct sockaddr *)&sin,sizeof(sin)) < 0) {
    perror("bind");
    exit(2);
  }
  listen(sc, 5);

  /* Boucle principale */
  for (;;) {

      if ( (scom = accept(sc, (struct sockaddr *)&exp,
&fromlen))== -1) {
      perror("accept");
      exit(2);
      }

      /*** Lire le message ***/
      if (read(scom,&cpt, sizeof(cpt)) < 0) {
      perror("read");
      exit(1);
      }

      /*** Traitement du message ***/
      cpt+=10;
```

```
      /*** Envoyer la réponse ***/
      if (write(scom, &cpt, sizeof(cpt)) == -1) {
      perror("write");
      exit(2);
      }

      /* Fermer la connexion */
      shutdown(scom,2);
      close(scom);
  }
  close(sc);
  return 0;
}
```

## ANNEXE B

### Structures de données utiles

```
struct  inode
{
   char    i_flag;
   char    i_count;     /* reference count */
   dev_t   i_dev;       /* device where inode resides */
   ino_t   i_number;    /* i number, 1-to-1 with device address */
   unsigned short  i_mode;
   short   i_nlink;        /* directory entries */
   short   i_uid;          /* owner */
   short   i_gid;          /* group of owner */
   off_t   i_size;         /* size of file */
   union {
      struct {
         daddr_t i_addr[NADDR];  /* if normal file/directory */
         daddr_t i_lastr;        /* last logical block read (for
read-ahead) */
      };
      struct  {
         daddr_t i_rdev;                /* i_addr[0] */
         struct  group   i_group;       /*  multiplexor group fi
le */
      };
   } i_un;
};

/* flags */
#define   ILOCK  01    /* inode is locked */
#define   IUPD   02    /* file has been modified */
#define   IACC   04    /* inode access time to be updated */
```

```
#define   IMOUNT 010   /* inode is mounted on */
#define   IWANT  020   /* some process waiting on lock */
#define   ITEXT  040   /* inode is pure text prototype */
#define   ICHG   0100     /* inode has been changed */
#define   IUSERLOCK    0800     /* inode locked by user */


#define    NOFILE  20             /* max open files per process */


struct  user
{
...
    struct file *u_ofile[NOFILE];
}



/*
 * the dup system call.
 */
int dup(int fdes, int fdes2)
{
        register struct file *fp;
        register i;

        fp = getf(fdes);
        if(fp == NULL)
                return -1;
        i = fdes2;
         if (i<0 || i>=NOFILE) {
               u.u_error = EBADF;
               return -1;
         }


        if (i!=fdes) {
                if (u.u_ofile[i]!=NULL)
                        closef(u.u_ofile[i]);
                u.u_ofile[i] = fp;
                fp->f_count++;
        }
    return i;
}


struct file *getf(int f)
{
        register struct file *fp;

        if(0 <= f && f < NOFILE) {
                fp = u.u_ofile[f];
                if(fp != NULL)
                        return(fp);
        }
        u.u_error = EBADF;
        return(NULL);
}
```

```
/*
 * Internal form of close.
 *
 */
void closef(struct file *fp)
{
...
}
```