Examen Module Noyau
Filière Système et Réseaux
Janvier 2003

3 heures - Tout document autorisé
Barème donné à titre indicatif

P. Sens

Exercice 1 : Fichiers (5 points)

1.1 Comment faire en sorte qu'un même fichier possède deux noms différents dans l'arborescence des fichiers ? (0,5 point)

Réponse : il suffit de créer un lien hard sur le fichier

On considère un fichier d'inode 367 sur la partition (device) numéro 1. Ce fichier possède deux noms : "fichier1" dans le répertoire courant et "fichier2" dans le répertoire "/tmp".

Soit la portion de code suivante :

```
1:  int main() {
2:    char buf[10];
3:    int fd1 , fd2;
4:    fd1 = open("./fichier1", O_RDONLY);
5:    if (fork() == 0) {
6:       fd2 = open("/tmp/fichier2", O_RDWR);
7:       write(fd2, "1234",  5);
8:       close(fd1); close(fd2);
9:       exit(1);
10:   }
11:   wait(NULL);
12:   read(fd1, buf, 2);
13:   buf[2] = 0;
14:   printf("%s", buf);
15:   close(fd1);
16:   return 0;
17: }
```

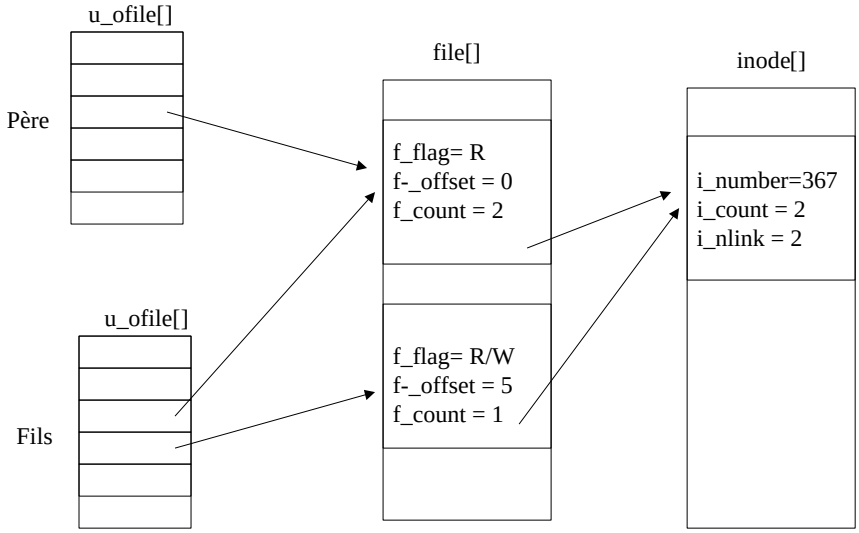
1.2 Quel est l'affichage fait par ce programme ? (0,5 point)

Réponse : 12

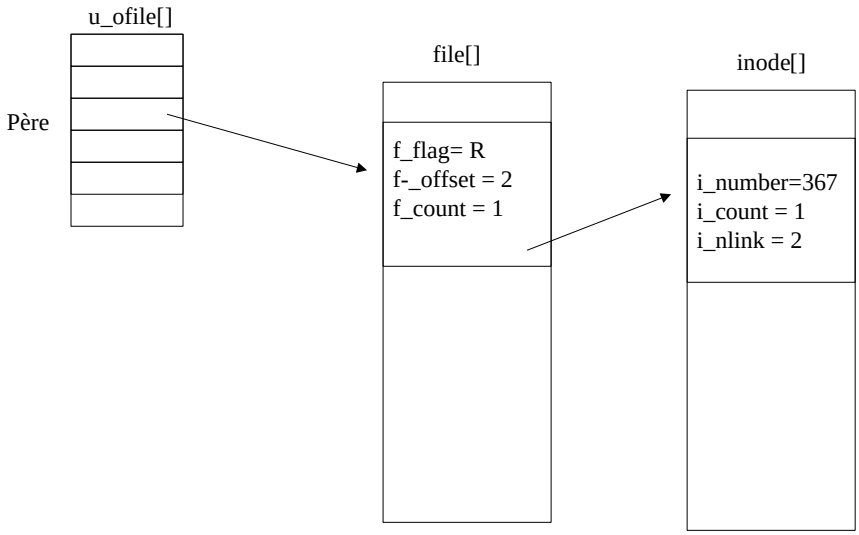
1.3. Faites un schéma des structures de données concernant les fichiers après la ligne 7

et après la ligne 12 (en y représentant toutes les informations pertinentes). (2 points)

Réponse :



Après la ligne 12 :



1.4 Quelle est la différence entre les champs i_count et i_nlink de la structure inode ? (0,5 point)

Réponse : i_count est le nombre d'ouvertures ayant eu lieu sur le fichier
i_nlink est le nombre de lien hard (nom) désignant le fichier

1.5 On considère qu'avant cette portion de code, aucune inode et aucun bloc de fichier n'est chargé en mémoire. Quel est le nombre d'entrée-sortie fait par cette séquence et à quels moments ? (1,5 points)

Réponse :
ligne 4 : chargement de l'inode du répertoire courant + lecture du répertoire courant + chargement de l'inode de fichier 1 = 3 E/S
ligne 6 : chargement de l'inode de la racine + lecture répertoire / + chargement inode répertoire tmp + lecture tmp = 4 E/S (inode de fichier 2 et déjà en mémoire)
ligne 7: lecture du bloc + ecriture différée = 1 E/S
ligne 8 : écriture du bloc = 1 E/S
ligne 12 : bloc déjà en mémoire = 0 E/S
Total : 7 E/S

- **Exercice 2 : Code noyau (10 points)**

On souhaite analyser le code de la fonction interne itrunc présenté en annexe.

2.1 Donnez l'algorithme de la fonction itrunc (4 points)

Réponse :
itrunc

```
En entrée la structure inode du fichier que l'on veut « tronquer »

Vérifier que le fichier est  standard

Pour toutes les entrées de blocs de l'inode
   Si l'entrée n'est pas vide
      Si il s'agit d'un bloc direct
         Libérer le bloc
      Sinon
         Si il s'agit d'un bloc d'indirection direct
            Liberer tous les blocs désignés dans le bloc (appel à
tloop)
         Si il s'agit d'un bloc d'indirection double
            Liberer tous les blocs correspondants(appel à tloop)

         Si il s'agit d'un bloc d'indirection triple
            Liberer tous les blocs correspondants (appel à tloop)
Positionner à 0 la taille du fichier
Marqué l'inode et le fichier modifiés
```

```
tloop
Fonction récursive qui libère les blocs contenus dans les blocs
d'indirection
En entrée : le bloc d'indirection, le niveau d'indirection

Pour tous les entrées du bloc :
   Si le bloc n'a pas été lu
      Lire le bloc et le placer dans le buffer cache (bread)
      Si erreur lors de l'E/S, libérer le buffer
   Sinon
      Si l'entrée est vide passer à la suivante
      Si l'entrée désigne un bloc d'indirection
         Libérer le bufffer du bloc initial
         Appel recursif à tloop pour le bloc désigné par l'entrée
      Sinon
         Liberer le bloc (free) /* bloc de données */

Liberer le buffer du bloc d'indirection
Liberer le bloc d'indirection
```

2.2 Par quelle fonction interne et à quel moment la fonction itrunc est elle appelée ? (1,5 points)

Réponse :
itrunc est appelée par iput lorsque le champs i_nlink passe à 0

2.3 On souhaite implémenter *dans le noyau* l'appel système unlink qui détruit un nom dans le système de fichiers (la commande rm fait un appel direct à unlink). Programmez cet appel système dont le prototype est le suivant. (2,5 points)

```
void unlink(char *nom_fichier);
```

Réponse :

```
Void unlink(char *nom_fichier) {
    struct inode *ip ;
    ip = namei(uchar, 2);
    if (ip == NULL)
        return;
    if (ip->i_nlink != 0 )
        ip->i_nlink --;
    iput(ip);
}
```

2.4 On veut maintenant réaliser une fonction interne undelete qui à partir d'un numéro d'inode retrouve (si elle existe encore) l'inode d'un fichier effacé par erreur et lui attribue un nouveau nom.
Le prototype de undelete est :

```
void undelete(int inode_number, char *new_name);
```

où inode_number est le numéro de l'inode que l'on veut récupérer et new_name et le nouveau nom associé au fichier.

Donnez le principe de fonctionnement de la fonction undelete (on ne demande pas le code C détaillé!) (2 points)

Réponse :

```
Recherche de l'inode inode_number (iget)
Si l'inode n'existe plus
   Rechercher parmi les inodes libres en mémoire une qui
correspond à inode_number
   Si pas d'inode trouvée
      Rechercher parmi toutes les inodes libres sur disque
      Si non trouvée retourner récuperation impossible
      Sinon ramener l'inode en mémoire
Augmenter le i_nlink de l'inode

Pour tous les blocs designés par l'inode
   Recherche les blocs dans le buffer cache
   Si non trouvé
      rechercher les blocs parmi les blocs libre sur disque
   Si non trouvé passer au suivant
   Retirer le bloc de la liste des blocs libres

Ajouter une nouvelle entrée <inode_mumber, newname> dans le
répertoire courant
```

Exercice 3 : Question de cours (5 points)

Soit un noyau *préemptif*. On considère trois threads posix T1, T2 et T3 avec le scénario

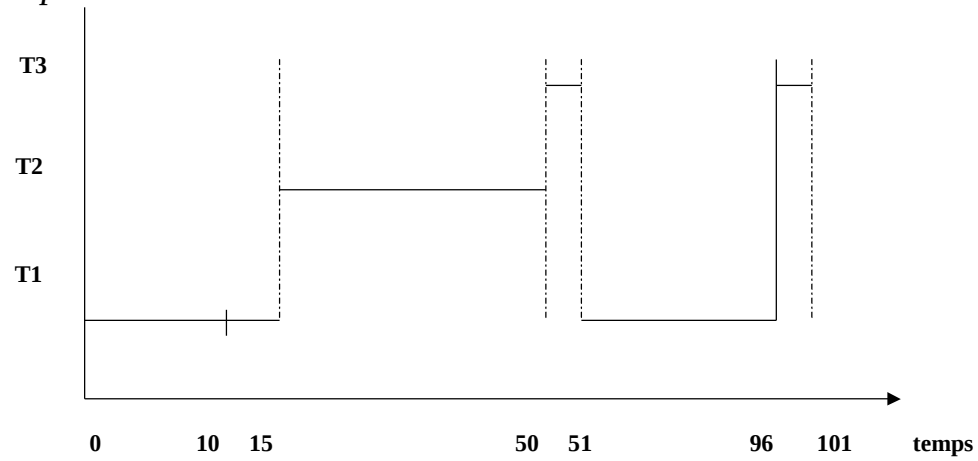
suivant :

- T1 est de classe SCHED_OTHER. Elle est lancée à t = 0. Lors d'un appel système, T1 verrouille à t=10 la variable X du noyau. La variable X est manipulée par T1 pendant 50 unités de temps.
- T2 est de classe SCHED_RR, elle devient prête à t = 15. Elle doit s'exécuter pendant 100 unités de temps avant de se bloquer à nouveau.
- T3 est de classe SCHED_FIFO. Elle devient prête à t = 50. Après 1 unité de temps, T3 a besoin de la variable X du noyau. Elle la manipule pendant 5 unités avant de se rendormir.

On suppose que les threads de classe SCHED_FIFO sont plus prioritaires que ceux de classe SCHED_RR qui sont eux-même plus prioritaires que ceux de classe SCHED_OTHER.

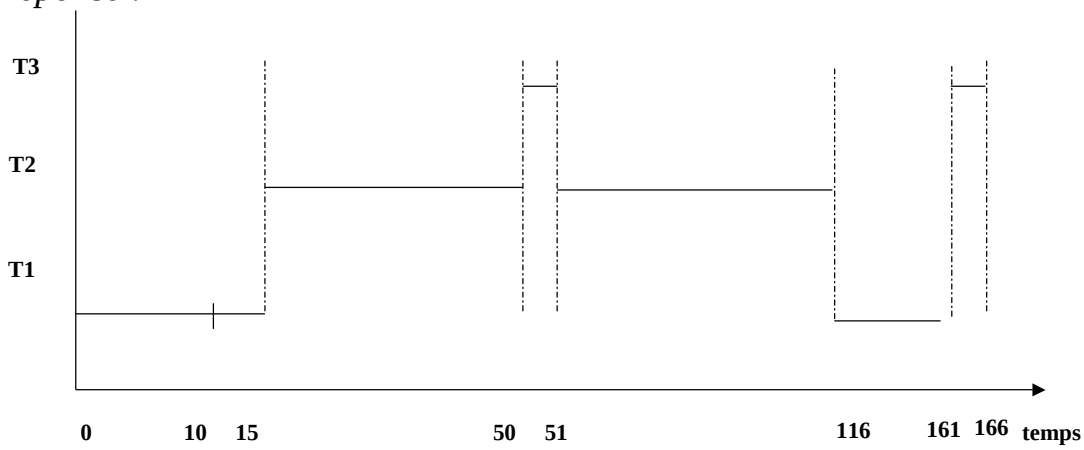
3.1 A quel moment T3 a fini son calcul (se rendort) si on considère qu'il y a un mécanisme d'héritage de priorité. Faites un diagramme temporel représentant les threads actifs. (1,5 points)

Réponse :



3.2 Reprendre la question 3.1 dans un système sans héritage de priorité. (1,5 points)

Réponse :



3.3 Pourquoi la commutation entre deux threads d'un même processus est plus rapide que la commutation entre deux processus ? (1 point)

Réponse :
Dans un même processus, 2 threads partagent le même espace d'adressage. La commutation n'exige donc pas de changement de table des pages et donc pas de flush de la table interne de traduction d'adresses (TLB)

3.4 Brièvement, quelle est la différence entre un vfork et un fork ? Pourquoi dans les Unix récents le vfork est-il désuet ? (1 point)

Réponse :
Contrairement au fork, un vfork ne duplique pas les pages du père (données et pile). Il y a dans ce cas un vrai partage entre le père et le fils. Pour éviter tout conflit, le père reste bloquer tant que le processus fils n'a pas appelé exec ou exit.

Le vfork est actuellement désuet à cause du copie-sur-ecriture utilisé dans les unix récent afin d'éviter les recopies.

ANNEXE

```
/*
 * Free all the disk blocks associated
 * with the specified inode structure.
 * The blocks of the file are removed
 * in reverse order. This FILO
 * algorithm will tend to maintain
 * a contiguous free list much longer
 * than FIFO.
 */
itrunc(ip)
register struct inode *ip;
{
        register i;
        dev_t dev;
        daddr_t bn;

        i = ip->i_mode & IFMT;
        if (i!=IFREG)
                return;
        dev = ip->i_dev;
        for(i=NADDR-1; i>=0; i--) {
                bn = ip->i_un.i_addr[i];
                if(bn == (daddr_t)0)
                        continue;
                ip->i_un.i_addr[i] = (daddr_t)0;
                switch(i) {

                default:
                        free(dev, bn);
                        break;

                case NADDR-3:
                        tloop(dev, bn, 0, 0);
                        break;

                case NADDR-2:
                        tloop(dev, bn, 1, 0);
                        break;

                case NADDR-1:
                        tloop(dev, bn, 1, 1);
                }
        }
        ip->i_size = 0;
        ip->i_flag |= ICHG|IUPD;
}
```

```
tloop(dev, bn, f1, f2)
dev_t dev;
daddr_t bn;
{
        register i;
        register struct buf *bp;
        register daddr_t *bap;
        daddr_t nb;

        bp = NULL;
        for(i=NINDIR-1; i>=0; i--) {
                if(bp == NULL) {
                        bp = bread(dev, bn);
                        if (bp->b_flags & B_ERROR) {
                                brelse(bp);
                                return;
                        }
                        bap = bp->b_un.b_daddr;
                }
                nb = bap[i];
                if(nb == (daddr_t)0)
                        continue;
                if(f1) {
                        brelse(bp);
                        bp = NULL;
                        tloop(dev, nb, f2, 0);
                } else
                        free(dev, nb);
        }
        if(bp != NULL)
                brelse(bp);
        free(dev, bn);
}

/*
 * place the specified disk block
 * back on the free list of the
 * specified device.
 */
free(dev, bno)
dev_t dev;
daddr_t bno;
{
....
}
```

```
/*
 * Convert a pathname into a pointer to
 * an inode. Note that the inode is locked.
 *
 * func = function called to get next char of name
 *      &uchar if name is in user space
 *      &schar if name is in system space
 * flag = 0 if name is sought
 *      1 if name is to be created
 *      2 if name is to be deleted
 */
struct inode *
namei(func, flag)
int (*func)();
{
....
}
```

Rappel de structures de données utiles

```
#define NINDIR  (BSIZE/sizeof(daddr_t))

/*
 * The I node is the focus of all
 * file activity in unix. There is a unique
 * inode allocated for each active file,
 * each current directory, each mounted-on
 * file, text file, and the root. An inode is 'named'
 * by its dev/inumber pair. (iget/iget.c)
 * Data, from mode on, is read in
 * from permanent inode on volume.
 */

#define  NADDR 13
#define  NINDEX 15

struct group {
   short g_state;
   char  g_index;
   char  g_rot;
   struct group *g_group;
   struct inode *g_inode;
   struct file  *g_file;
   short g_rotmask;
   short g_datq;
   struct chan *g_chans[NINDEX];
};
```

```
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] */
         structgroup  i_group;  /*  multiplexor group file */
      };
   } i_un;
};


extern struct inode inode[];/* The inode table itself */
struct inode *mpxip;     /* mpx virtual inode */

/* flags */
#define  ILOCK 01    /* inode is locked */
#define  IUPD  02    /* file has been modified */
#define  IACC  04    /* inode access time to be updated */
#define  IMOUNT010   /* 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 */

/* modes */
#define  IFMT  0170000     /* type of file */
#define     IFDIR 0040000  /* directory */
#define     IFCHR 0020000  /* character special */
#define     IFBLK 0060000  /* block special */
#define     IFREG 0100000  /* regular */
#define     IFMPC 0030000  /* multiplexed char special */
#define     IFMPB 0070000  /* multiplexed block special */
#define  ISUID 04000    /* set user id on execution */
#define  ISGID 02000    /* set group id on execution */
#define ISVTX   01000    /* save swapped text even after use */
#define  IREAD 0400     /* read, write, execute permissions */
#define  IWRITE0200
```

```
#define  IEXEC 0100


struct buf
{
   int b_flags;     /* see defines below */
   struct buf *b_forw;    /* headed by d_tab of conf.c */
   struct buf *b_back;    /*  "  */
   struct buf *av_forw;      /* position on free list, */
   struct buf *av_back;      /*     if not BUSY*/
   dev_t  b_dev;       /* major+minor device name */
   unsigned b_bcount;    /* transfer count */
   union {
       caddr_t b_addr;       /* low order core address */
       int   *b_words;    /* words for clearing */
       struct filsys *b_filsys;/* superblocks */
       struct dinode *b_dino;  /* ilist */
       daddr_t *b_daddr;     /* indirect block */
   } b_un;
   daddr_t   b_blkno;     /* block # on device */
   char  b_xmem;         /* high order core address */
   char  b_error;     /* returned after I/O */
   unsigned int b_resid; /* words not transferred after error */
};
```