Examen Module Noyau
Filière Système et Réseaux
Septembre 2003

3 heures - Tout document autorisé
Barème donné à titre indicatif

P. Sens

Exercice 2 : Fichiers (5 points)

On considère que seul l'inode du répertoire courant est en mémoire et qu'aucun bloc de fichier n'est chargé en mémoire.

Le contenue du fichier ../fichier1 est « acbc ».
Soit la portion de code suivante :

```
1:  int main() {
2:    char buf[10];
3:    int fd1 , fd2;
4:    fd1 = open("../fichier1", O_RDONLY);
5:    read(fd1, buf, 2);
6:    if (fork() == 0) {
7:       fd2 = open("../fichier1", O_RDWR);
8:       write(fd2, "1234",  5);
9:       close(fd1); close(fd2);
10:      exit(1);
11:   }
12:   wait(NULL);
13:   read(fd1, buf, 2);
14:   buf[2] = 0;
15:   printf("%s", buf);
16:   close(fd1);
17:   return 0;
18: }
```

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

1.3. Faites un schéma des structures de données concernant les fichiers juste après la ligne 8 et après la ligne 14 (en y représentant toutes les informations pertinentes). (2 points)

1.4 Quel est le contenu du fichier ../fichier1 à la fin d'exécution de ce programme ? (1

point)

1.5 Quel est le nombre d'entrées/sorties fait par cette séquence et à quels moments ? (1,5 points)

Exercice 2 : Programmation thread (5 points)

Soit le programme suivant :

```
....
pthread_mutex_t m1 = PTHREAD_MUTEX_INITIALIZER; /* Pour la
condition c1 */
pthread_mutex_t m2 = PTHREAD_MUTEX_INITIALIZER; /* Pour la
condition c2 */

pthread_cond_t c1 = PTHREAD_COND_INITIALIZER;
pthread_cond_t c2 = PTHREAD_COND_INITIALIZER;

char cmd[MAXBUF];
int active;

/* Code du thread workers */

void *worker(void *a) {
  char local_cmd[MAXBUF];
  while (1) {

    pthread_mutex_unlock(&m1);
    pthread_cond_wait(&c1, &m1);

     /* Copie locale du message généré par le main */
    memcpy(cmd, local_cmd, MAXBUF) ;

    pthread_mutex_unlock(&m2);
    pthread_cond_signal(&c2);

    /* Traitement du message */
    Traitement(local_cmd)

  }
  pthread_exit(0);
}

int main() {
  int i;
  pthread_t t;

  setbuf(stdout, NULL);

  /* Creation du thread worker */
   if (pthread_create(&t, NULL , worker, NULL) != 0 ) {
      perror("pthread_create");
```

```
      exit(1);
    }

  /* Production de messages pour le worker */
  while (1){

    produire(cmd); // Fonction qui remplit le buffer cmd

    pthread_mutex_unlock(&m1);
    pthread_cond_signal(&c1);


    pthread_mutex_unlock(&m2);
    pthread_cond_wait(&c2, &m2) ;

  }
}
```

2.1 Quelle est l'utilité des variables conditions c1 et c2 ? (1 point)

2.2 Pour améliorer les performances de traitement, on souhaite définir N workers. Pour chaque donnée produite par le « main », on veut réveiller le worker le plus prioritaire. Modifier le programme en conséquence. (3 points)

2.3 Dans leur traitement les workers ne font aucune entrée/sortie, est-ce que le fait d'avoir plusieurs threads workers est interessant ? Justifier brièvement votre réponse (1 point)

Exercice 3 : Code noyau (10 points)

Dans l'unix du TD, on souhaite analyser le code de la commande `check` présentée en annexe.

3.1 Donnez l'algorithme de la fonction `pass1` en indiquant l'algorithme de fonction `chk` et `duped`. (4 points)

3.2 Donnez l'algorithme de la fonction `check`? (2 points)

3.3 A la fin de l'algorithme, quelle est la signification des valeurs contenues dans les champs `ndirect`, `ninder` et `niinder` ? (1,5 points)

3.4 Quelle l'utilité de la variable `bmap` ? (1 points)

3.5 A la suite de quelle circonstance un bloc peut être alloué à plusieurs fichiers différents ? (1,5 points)

ANNEXE

```
#define  NI 16
#define  NB 10

struct   filsys   sblock;
struct   dinode   itab[INOPB*NI];
char  *bmap;

ino_t ino;

ino_t nrfile;
ino_t ndfile;
ino_t nbfile;
ino_t ncfile;

daddr_t  ndirect;
daddr_t  nindir;
daddr_t  niindir;
daddr_t  niiindir;
daddr_t  nfree;
daddr_t  ndup;

int   nerror;


check()
{
   register i, j;
   ino_t mino;
   daddr_t d;
   long n;

   nrfile = 0;
   ndfile = 0;
   ncfile = 0;
   nbfile = 0;

   ndirect = 0;
   nindir = 0;
   niindir = 0;
   niiindir = 0;

   ndup = 0;
   bread((daddr_t)1, (char *)&sblock, sizeof(sblock));
   mino = (sblock.s_isize-2) * INOPB;
   ino = 0;
   n = (sblock.s_fsize - sblock.s_isize + BITS-1) / BITS;
```

```
    if (n != (unsigned)n) {
        printf("Check fsize and isize: %ld, %u\n",
            sblock.s_fsize, sblock.s_isize);
    }
    bmap = malloc((unsigned)n);

    for(i=0; i<(unsigned)n; i++)
        bmap[i] = 0;
    for(i=2;; i+=NI) {
        if(ino >= mino)
            break;
        bread((daddr_t)i, (char *)itab, sizeof(itab));
        for(j=0; j<INOPB*NI; j++) {
            if(ino >= mino)
                break;
            ino++;
            pass1(&itab[j]);
        }
    }
}

pass1(ip)
register struct dinode *ip;
{
    daddr_t ind1[NINDIR];
    daddr_t ind2[NINDIR];
    daddr_t ind3[NINDIR];
    register i, j;
    int k, l;

    i = ip->di_mode & IFMT;
    if(i == 0) {
        sblock.s_tinode++;
        return;
    }
    if(i == IFCHR) {
        ncfile++;
        return;
    }
    if(i == IFBLK) {
        nbfile++;
        return;
    }
    if(i == IFDIR)
        ndfile++; else
    if(i == IFREG)
        nrfile++;
    else {
        printf("bad mode %u\n", ino);
        return;
    }

    for(i=0; i<NADDR; i++) {
```

```
        if(ip->di_addr[i] == 0)
            continue;
        if(i < NADDR-3) {
            ndirect++;
            chk(ip->di_addr[i], "data (small)");
            continue;
        }
        nindir++;
        if (chk(ip->di_addr[i], "1st indirect"))
                continue;
        bread(ip->di_addr[i], (char *)ind1, BSIZE);
        for(j=0; j<NINDIR; j++) {
            if(ind1[j] == 0)
                continue;
            if(i == NADDR-3) {
                ndirect++;
                chk(ind1[j], "data (large)");
                continue;
            }
            niindir++;
            if(chk(ind1[j], "2nd indirect"))
                continue;
            bread(ind1[j], (char *)ind2, BSIZE);
            for(k=0; k<NINDIR; k++) {
                if(ind2[k] == 0)
                    continue;
                if(i == NADDR-2) {
                    ndirect++;
                    chk(ind2[k], "data (huge)");
                    continue;
                }
                niiindir++;
                if(chk(ind2[k], "3rd indirect"))
                    continue;
                bread(ind2[k], (char *)ind3, BSIZE);
                for(l=0; l<NINDIR; l++)
                    if(ind3[l]) {
                        ndirect++;
                        chk(ind3[l], "data (garg)");
                    }
            }
        }
    }
}

chk(bno, s)
daddr_t bno;
char *s;
{
    register n;

    if (bno<sblock.s_isize || bno>=sblock.s_fsize) {
        printf("%ld bad; inode=%u, class=%s\n", bno, ino, s);
```

```
      return(1);
   }
   if(duped(bno)) {
      printf("%ld dup; inode=%u, class=%s\n", bno, ino, s);
      ndup++;
   }
   return(0);
}

duped(bno)
daddr_t bno;
{
   daddr_t d;
   register m, n;

   d = bno - sblock.s_isize;
   m = 1 << (d%BITS);
   n = (d/BITS);
   if(bmap[n] & m)
      return(1);
   bmap[n] |= m;
   return(0);
}
```

Rappel de structures de données utiles

```
/*
 * Inode structure as it appears on
 * a disk block.
 */
struct dinode
{
   unsigned short  di_mode;        /* mode and type of file */
   short  di_nlink;       /* number of links to file */
   short  di_uid;         /* owner's user id */
   short  di_gid;         /* owner's group id */
   off_t  di_size;        /* number of bytes in file */
   char   di_addr[40]; /* disk block addresses */
   time_t di_atime;    /* time last accessed */
   time_t di_mtime;    /* time last modified */
   time_t di_ctime;    /* time created */
};
#define   INOPB 8   /* 8 inodes per block */

/* 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 */
```

```
/*
 * Structure of the super-block
 */
struct filsys {
   unsigned short s_isize;  /* size in blocks of i-list */
   daddr_t  s_fsize;     /* size in blocks of entire volume */
   short    s_nfree;     /* number of addresses in s_free */
   daddr_t  s_free[NICFREE];/* free block list */
   short    s_ninode;    /* number of i-nodes in s_inode */
   ino_t    s_inode[NICINOD];/* free i-node list */
   char     s_flock;     /* lock during free list manipulation */
   char     s_ilock;     /* lock during i-list manipulation */
   char     s_fmod;      /* super block modified flag */
   char     s_ronly;     /* mounted read-only flag */
   time_t   s_time;      /* last super block update */
   /* remainder not maintained by this version of the system */
   daddr_t  s_tfree;     /* total free blocks*/
   ino_t    s_tinode;    /* total free inodes */
   short    s_m;         /* interleave factor */
   short    s_n;         /* " " */
   char     s_fname[6];  /* file system name */
   char     s_fpack[6];  /* file system pack name */
};
```