Systèmes et réseaux

Cours 1

Jean-Jacques.Levy@inria.fr
Ecole polytechnique

tel: 39-63-56-89
http://pauillac.inria.fr/~levy/

http://www.poly.polytechnique.fr/poly/...
.../~www/poly/maj2-syst-96/

Intervenants

- TD1
  1. Damien Doligez (INRIA-Rocquencourt–ENS)
  2. Robert Ehrlich (INRIA-Rocquencourt)
- TD2
  1. François Rouaix (INRIA-Rocquencourt)
  2. Frédéric Ruget (Chorus Systèmes)

Plan

Systèmes d'exploitation

Réseaux

Système de fichiers en Unix

Le *shell* Unix

Le langage C. Typage faible. C++. Ansi C

Manipulations des chaines de caractères et des pointeurs en C

Environnement de programmation adb, dbx, gdb

Exercices

Systèmes d'exploitation (*operating systems*)

- Applications
  (emacs, vi, cc, gdb, sort, etc)

- Librairies systèmes (entrées/sorties),
  niveau 3 ou plus de `man` en Unix

- Appels système (`read, write`),
  niveau 2 de `man` en Unix

- Noyau unix (`/vmunix`),
  exécution en mode superviseur

Le système est l'ensemble des librairies, des appels-système et du noyau.

Systèmes (historique)

- traitement par lots (*batch*)
- multiprogrammation (60-65)
- temps partagé (1963, SDS 930, Butler Lampson, UCB; 1967, GE 645 Multics, Corbato – Organick, MIT)
- temps partagé standard (1969, PDP 7–11, Unix, Ken Thompson, Dennis Ritchie, Bell labs; 1979, Vax/750, Bill Joy, Ozalp Babaoglu, UCB)

```
TIT     .MULTICS SYSTEM : AN EXAMINATION OF ITS STRUCTURE
ZAUT    .ORGANICK, ELLIOTT I.
DAT     .1972
EDI     .MIT PRESS,CAMBRIDGE, MA
```

Autres systèmes:

- Alto, Dorado, Macintosh, Taos (Xerox PARC)
- CTSS, ITS (MIT)
- Tops10/20, Sail (DEC, SU)
- SunOS, System V, Posix (Sun, AT&T)
- Mach (CMU), OSF ($\neq$ Sun, AT&T)
- VMS, Windows NT . . . Sprite, Opal, Plan 9, Topaz

Réseaux

Fonctions

- Réseaux locaux: distribution des fichiers, interaction.

- Interconnexion: courier, transferts de fichiers, distribution de logiciel et d'information (Web).

Technologie

- Réseaux locaux (1975, Ethernet, Metcalfe, Xerox PARC, 3-10Mbds; Token ring 10Mbds, CMU/ITC; 1985 FDDI 100Mbds, INRIA; 1990 1Gbds ATM, Autonet de DEC/SRC)

- Interconnexion (ARPAnet 1970; `uucp` 1980; Internet 1985)

Système de fichiers Unix

- Arborescent
- Liens en dûr, et liens symboliques
- `/vmunix` le noyau du système
- `/etc/init` le programme qui s'exécute au `boot`.
  - Il lance l'interpréteur de commandes `/bin/sh` sur le fichier `/etc/rc` (en multi-utilisateur) et ainsi initialise les partitions du système de fichiers, le réseau, nettoie les fichiers temporaires de `/tmp`, lance les démons, la comptabilité et les *spooler*s.
  - Puis il lance pour tous les terminaux marqués dans `/etc/ttys` un programme d'écoute correspondant. En général, ce programme est `/etc/getty` qui initialise la ligne, fixe un *process group*; ses actions décrites dans `/etc/gettytab` sont paramétrées par le type de terminal. Pour les lignes marquées interactives, on envoie la banière de login, et on lit le nom de l'utilisateur (`knaff`).
  - On exécute alors le programme `/bin/login knaff`. Si ce dernier réussit (bon mot de passe qui figure dans `/etc/passwd`), un *shell* interpréteur de commandes démarre. Il y a plusieurs *shell* possibles. La correspondance entre l'utilisateur et le *shell* est indiquée dans `/etc/passwd`.

Système de fichiers Unix (exemples)

1. `/bin`, `/usr/bin`, `/usr/local/bin`, `/usr/ucb` binaires des commandes
2. `/lib`, `/usr/lib`. `/usr/local/lib` librairies
3. `/tmp` fichiers temporaires
4. `/usr/users/cie6/knaff` *home directory* (la correspondance se trouve dans `/etc/passwd`)
5. `/etc` fichiers annexes système
6. `/dev` les périphériques
7. `/man` les manuels

L'interpréteur de commandes (`/bin/sh`)

- faire `man sh`, `man csh`, `man tcsh`, `man bash`, `man perl`
- faire `learn` sur `poly`
- entrée/sortie standards
- redirection de celles-ci
- composition *pipe*s
- jokers `*`, `?`
- petit langage de contrôle
- variables d'environnement
- *script-shells*, paramètres `$0`, `$1`, `$2`, ...
- fonctions `` `xxx` ``
- *quotes* `'a*b'`, `"a*$1"`

```
TIT     .The UNIX programming environment
SLA     .Brian W. Kernighan, Rob Pike. - Englewood Cliffs, NJ
        .: Prentice-Hall, 1984. - X-357 p. ; 24 cm. - (Prentice-
        .Hall software series).
```

L'interpréteur de commandes (exemples)

- who | wc -l
- hist | spline | psplot | lpr
- (cd directory1; tar cf - fichier1) | (cd directory2; tar xf -)
- (date; who) > foto1
- (date; who) >> foto1
- mail -s URGENT levy <lettre_de_protestation
- ls a*.c, echo *
- emacs notre éditeur de texte favorit (Richard Stallman, FSF)

Le langage C – Dennis Ritchie (Bell labs 1972)

- BCPL, B, C, typage faible, portabilité, ne pas gaspiller un seul octet
- fonctions et procedures non imbriquées (pour efficacité)
- manipulation agréable des pointeurs
- bon interface avec Unix (tout Unix est écrit en C)

```
TIT     .BCPL : THE LANGUAGE AND ITS COMPILER
AUT     .Richards, Martin. / Whitby-Strevens, Colin.
DAT     .1979
EDI     .CAMBRIDGE UNIVERSITY PRESS,CAMBRIDGE, MA

TIT     .The C programming language
SLA     .Brian W. Kernighan, Dennis M. Ritchie. - Englewood Cliffs,
        .NJ : Prentice Hall, 1978. - X-228 p. ; 24 cm. - (Prentice
        .Hall software series).
```

Exemple 1 de programme C

```
const N = 100;
var a: array [1..N] of integer;

procedure Insertion;

    var i, j, v: integer;

    begin
    for i := 2 to N do
        begin
        v := a[i]; j := i;
        while a[j-1] > v do
            begin a[j] := a[j-1]; j:= j-1 end;
        a[j] := v;
        end;
    end;
```

```
#define N     100
int a[100];

Insertion()
{
    int i, j, v;

    for (i = 1; i < N; ++i) {
        v = a[i]; j = i;
        while (a[j-1] > v) {
            a[j] = a[j-1]; j = j-1;
        }
        a[j] = v;
    }
}
```

Exemple 1 (suite)

```
Insertion()
{
    register int i, j, v;

    for (i = 1; i < N; ++i) {
        v = a[i];
        for (j = i; (j > 0) && (a[j-1] > v); --j)
            a[j] = a[j-1];
        a[j] = v;
    }
}
```

Exemple 1 (suite)

```
Insertion()
{
    int v, *p, *q;

    for (p = &a[1]; p < &a[N]; ++p)
        v = *p;
        for (q = p; (p > &a[0]) && (*(q-1) > v); --q)
            *q = *(q-1);
        *q = v;
    }
}

Insertion()
{
    register int v, *p, *q;

    for (p = a + 1; p < a + N; ++p)
        v = *p;
        for (q = p; (p > a) && (*(q-1) > v); --q)
            *q = *(q-1);
        *q = v;
    }
}
```

Example 1 – Programme principal

```
#include <stdio.h>
#define N     100
int a[N];

Insertion (int a[]; int n)
{
...
}

main()
{
    int x, n;

    printf ("Entrez les nombres a trier (-1 pour terminer)\n");
    n = 0;
    for (;;) {
        scanf ("%d\n", &x);
        if ((x == -1) || (n >= N))
            break;
        a[n++] = x;
    }
    Insertion (a, n);
    for (i = 0; i < n; ++i)
        printf ("a[%2d] = %d\n", i, a[i]);
}
```

Remarques sur C (1)

- Appel par référence == Appel par valeur sur l'adresse

```
Exchange (int *p, int *q)
{
    int r;

    r = *p; *p = *q; *q = r;
}

main ()
{
    int x, y;
    Exchange (&x, &y);
}
```

- En C, on a l'égalité suivante:

  `&a[i]` = `a + i`

  Donc, si on a un tableau `int a[1000]`, une procédure `f` de signature `f(int *p)` peut s'écrire `f(&a[0])` ou `f(a)`. On peut aussi déclarer la signature sous la forme `f(int p[])` avec un tableau ouvert.

- En C, les chaines de caractères sont des tableaux de caractères finis par le caractère null (0 en ASCII). Donc

```
char nom[] = "levy";
```

```
nom[0] == 'l'                nom[1] == 'e'
nom[2] == 'v'                nom[3] == 'y'
nom[4] == 0 == '\0'
```

Remarques sur C (2)

- Toute expression suivie de ";" (et précédée éventuellement d'une étiquette) devient une instruction. Comme `x = 1` est une expression retournant la valeur de son membre droit `1`, on écrit:

```
if (x == 3)
    x = 1;
else
    x = 0;
}
```

- Une instruction composée est une suite d'éventuelles déclarations et d'instructions encadrée par { et }.

- Les enregistrements "structures":

```
struct point {
    int x;
    int y;
};

struct point p, q;

... p.x = q.x - 2; ...

struct complex {
    float re;
    float im;
} z1, z2;

struct complex z3;

... z.re = 3.14; ...
```

Remarques sur C (3)

- Structures récursives:

```
struct tnode {
    char *word;
    int count;
    struct tnode *left;
    struct tnode *right;
}

struct t {
    ...
    struct s *p;
};

struct s {
    ...
    struct t *q;
}
```

- Pointeurs vers des structures:

```
struct tnode *p;

...(*p).left...
...p->left...
```

- Pointeurs vers des fonctions:
  `int *f();` est une fonction retournant un pointeur vers des `int`
  `int (*pf)();` est un pointeur vers une fonction

Remarques sur C (4)

- Unions (type somme ou variant):

```
union complexe {
   struct cartesien {
       float: re;
       float: im;
   };
   struct polaire {
       float: rho;
       float: theta;
   };
} x, y;

...x.cartesien.re...
...y.polaire.rho...
```

- Typedefs (nouvelle structure de données)

```
typedef unsigned long   u_long;

u_long x, y;
```

- Cœrcion “cast”

```
float x = 3.14;
int i;
i = (int) x;

short x;
char *p;
p = (char *) &x;
```

Remarques sur C (5)

- **sizeof**, **malloc**

```
#include <stdlib.h>

/* talloc: make a node */
struct tnode *talloc (void)
{
    return (struct tnode *) malloc (sizeof(struct tnode));
}
```

Example de programme C (1)

- Entrer le programme sous *emacs* et le sauver dans `prog1.c`

```
#include <stdio.h>

main() /* count digits, white space, others */
{
    int c, i , nwhite, nother, ndigit[10];

    nwhite = nother = 0;
    for (i = 0; i < 10; ++i)
        ndigit[i] = 0;
    while ((c = getchar()) != EOF) {
        switch (c) {
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
            ++ ndigit [c - '0'];
            break;
        case ' ':
        case '\n':
        case '\t':
            nwhite++;
            break;
        default:
            nother++;
            break;
        }
    }
    printf("digits =");
    for (i = 0; i < 10; i++)
        printf(", white space = %d, other = %d\n",
            nwhite, nother);
}
```

Example de programme C (2)

- Commande de compilation et exécution

```
% cc prog1.c
% a.out
```

- Mise au point. On peut se servir de `adb`, `dbx` ou `gdb`. Ce dernier est fortement conseillé sous *emacs*, si on a des bonnes macros pour interfacer la souris.

  Mettre dans le fichier `.emacs` de son homedir, la ligne
  `(load "/usr/users/labos/profs/levy/emacs/x-mouse-s.elc")`

  Compiler `prog1.c` avec l'option `-g`
  `% cc -g prog1.c`

  Sous *emacs*, taper la commande `<ESC-X gdb>` sur le fichier binaire `a.out`. Dans le binaire, le nom du fichier source et des indications sur les variables et numéros de lignes ont été rajoutées par l'option `-g` de `cc`. LIRE LE MANUEL GDB.

  On peut par exemple aller dans le source en pointant avec la souris (bouton de gauche), et mettre un point d'arrêt avec la commande CTL-X-<SPACE> met un point d'arrêt sur cette ligne.

  En allant dans le *buffer* gdb, on peut taper
  `(gdb) run`, pour démarrer
  `(gdb) p nwhite` pour imprimer nwhite
  `(gdb) cont ...`

Quelques exemples de programmes de librairie

En `/usr/local/src/src.pmax/ultrix-3.1-pmax/lib/libc/gen`, le programme de conversion de chaines ASCII en entier.

```
% more atoi.c
/* @(#)atoi.c   4.3 (Berkeley) 81/02/28 */
atoi(p)
register char *p;
{
        register int n;
        register int f;

        n = 0;
        f = 0;
        for(;;p++) {
                switch(*p) {
                case ' ':
                case '\t':
                        continue;
                case '-':
                        f++;
                case '+':
                        p++;
                }
                break;
        }
        while(*p >= '0' && *p <= '9')
                n = n*10 + *p++ - '0';
        return(f? -n: n);
}
```

Regarder `lib/libc/mips/gen/atof.c`!!

Quelques exemples de programmes de librairie

En `lib/libc/mips/gen/strncmp.c`

```
% more strncmp.c
/* $Header: strncmp.c,v 1.1 87/02/16 11:17:00 dce Exp $ */

/*
 * Compare strings (at most n bytes):  s1>s2: >0  s1==s2: 0  s1<s2: <0
 */

strncmp(s1, s2, n)
register char *s1, *s2;
register n;
{

        while (--n >= 0 && *s1 == *s2++)
                if (*s1++ == '\0')
                        return(0);
        return(n<0 ? 0 : *s1 - *--s2);
}
```

Regarder `lib/libc/mips/gen/atof.c`!!

Quelques exemples de programmes de librairie

En `lib/libc/mips/gen/strcpy.c`

```
% more strncpy.c
/* $Header: strncpy.c,v 1.1 87/02/16 11:17:02 dce Exp $ */

/*
 * Copy s2 to s1, truncating or null-padding to always copy n bytes
 * return s1
 */

char *
strncpy(s1, s2, n)
register char *s1, *s2;
{
        register i;
        register char *os1;

        os1 = s1;
        for (i = 0; i < n; i++)
                if ((*s1++ = *s2++) == '\0') {
                        while (++i < n)
                                *s1++ = '\0';
                        return(os1);
                }
        return(os1);
}
```

Exercices en TD et à la maison

- Débutants en C: prendre le poly du TC et faire les premiers exemples de programme en C (tri bulle, tri insertion, etc). N'utiliser que scanf et printf pour les entrées/sorties.

- Connaisseurs de C: vérifier vos connaissances sur les chaines de caractères (en prévision du mini-shell). Donc faire:

  1. `echo aa bb cc` (lire donc argv et argc et imprimer)

  2. `cat aa bb` (lire argv et argc et lire les fichier correspondants)
     `cat` (sans arguments)
     `aa bb`
     `EOF`

  3. `cat1` (lire l'entrée standard et imprimer les fichiers correspondants)
     `aa bb cc`
     `EOF`

  4. `cat1 aa bb` (que fait-il?)...

  5. `sort` (trier les lignes de stdin par ordre lexicographique)

Pour lire, on ne fait qu'avec les primitives du poly de TC `getchar()`, `getc()`, `putchar`, `putc`,`printf`) et on utilise pour le moment que les *file pointers* (pas de *fd* SVP). On expliquera la logique des redirections que dans la lecon sur le système de fichier (TD2).

/usr/include/sys/ino.h?

```
/*
 * 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 */
/*
 * the 40 address bytes:
 *      39 used; 13 addresses
 *      of 3 bytes each.
 */
```

Systèmes et réseaux

Cours 2

Jean-Jacques.Levy@inria.fr
Ecole polytechnique

tel: 39-63-56-89
http://pauillac.inria.fr/∼levy/

http://www.poly.polytechnique.fr/poly/...
.../∼www/poly/maj2-syst-96/

Plan

- Le *shell* Unix
- Le langage C (structures – rappel, préprocesseur, Ansi C)
- Librairies systèmes
- Entrées – sorties
- Disques
- Montages, fichiers *device*
- Structure des fichiers en V7
- Structure des fichiers en BSD
- Exercices

Le *shell* Unix

- paramètres $1, $2

```
kpr% more flatex
#!/bin/sh
if test -r $1
then
  rm -rf $1.tex
fi
Ftex <$1 >$1.tex
/usr/local/bin/latex $1.tex
/bin/rm -rf $1.tex
```

- pipes dans des shell-scripts

```
kpr% more interdir
#! /bin/sh

# Packages operation (JJL--nov 89)
# d1 := d2 & d3
# copy in d1 files common between d2 and d3.

d1=`cd $1; pwd`

diff -s -r $2 $3     |\
    grep 'Files .* are identical$'     |\
    awk '{print $2}'| sed -e "s,^$2/,,"     |\
    (cd $2; cpio -pdma $d1)
```

Le *shell* Unix (suite)

- exemple d'utilisation de `for`

```
#! /bin/sh

DEST="ma@src.dec.com georges"
# DEST=levy

set -xv

# ------------------------------------------
# # le texte
#
for i in standard.tex standard.bib
do
   mail -s "$i" ${DEST} <$i
done

# ------------------------------------------
pour envoyer le postscript

aptex -Psec -of standard.ps standard.dvi
/bin/rm -f standard.ps.Z
compress standard.ps
uuencode standard.ps.Z standard.ps.Z > standard.ps.Z.uu
/bin/rm -f x??
split -1500 standard.ps.Z.uu
ll x??
mail -s "TODO PostScript" ${DEST} << ---LA-FIN---

# To get the PostScript
# Let xaa, xab, xac, xad ... be the parts. Then catenate without
# blank lines in any x??, decode, uncompress and print as follows.
```

```
cat x?? >standard.ps.Z.uu
uudecode standard.ps.Z.uu
uncompress standard.ps.Z
lpr standard.ps
rm x??

---LA-FIN---

for i in x??
do
   mail -s "standard.ps-$i" ${DEST} <$i
done
```

Le *shell* Unix (suite . . .)

- exemple d'utilisation de <<

```
# -------------------------------------------
# Pour envoyer les dessins

tar cf standard.tex-dessins.tar dessins
/bin/rm -f  standard.tex-dessins.tar.Z
compress standard.tex-dessins.tar
uuencode standard.tex-dessins.tar.Z standard.tex-dessins.tar.Z > standard.tex-dessins.tar.Z.uu
/bin/rm -f x??
split -1500  standard.tex-dessins.tar.Z.uu
ll x??

mail -s "TODO drawings" ${DEST} << ---LA-FIN---

# To get the drawings
# Let xaa, xab, xac, xad ... be the parts. Then catenate without
# blank lines in any x??, decode, uncompress and print as follows.

cat x?? >standard.ps.Z.uu
uudecode standard.ps.Z.uu
uncompress standard.ps.Z
lpr standard.ps
rm x??

---LA-FIN---

for i in x??
do
  mail -s "standard.tex-dessins-$i" ${DEST} <$i
done

/bin/rm -f *.uu
```

Le langage C (structures – rappel, préprocesseur, Ansi C)

- revoir structures du cours 1

- préprocesseur `/lib/cpp`. Ne pas oublier de mettre des parenthèses pour les précédences. Ne pas faire d'effet de bord dans un paramètre dupliqué. Pour voir les effect du préprocesseur, faire `cc -E toto.c`.

  ```
  #define  Max(x, y)       ((x) < (y) ? (y) : (x))
  ```

- utiliser le type `enum` autant que possible

  ```
  enum boolean {NO, YES}
  ```

  au lieu de

  ```
  #define NO            0
  #define YES           1
  ```

- Ansi C. Autrefois en C, les procédures étaient mal définies comme

  ```
  int f (x, y)
      int x;
      int y;
  {
  ...
  }
  ```

au lieu de

```
int f (int x, int y)
{
...
}
```

Comme le compilateur est 1 passe, les prototypes peuvent maintenant s'écrire

```
    int i, j;
    int f (int, int);
    struct toto x;
```

Librairies systèmes

- les appels système sont générés à partir des fonctions du niveau 2 du manuel. Exemple: brk, sbrk, read, write, open, lseek.
- des librairies sont fournies pour avoir des interfaces plus commodes. Ce sont les fonctions du niveau 3 du manuel: malloc, fopen, fread, fwrite, fseek.
- ces librairies sont contenues dans un fichier `archive` dit librairie comme `/lib/libc.a`. Il y a d'autres librairies comme la librairie mathématique `/lib/libm.a`, la librairie X-windows `/usr/lib/libX11.a`.
- la commande `cc` prend comme arguments des sources `xxx.c`, mais aussi des binaires relogeables `yyy.o` résultats d'une compilation précédente ou des librairies `/lib/libm.a`. Pour obtenir `yyy.o`, faire `cc -c yyy.c`. Pour ne pas taper le nom de la librairie, on fait simplement `cc xxx.c -lm`.

Entrées – sorties

- `open, read, write, close, seek` marchent avec un petit entier *file descriptor*. L'entrée standard a, par convention, 0 pour *file descriptor*; la sortie standard 1; la sortie erreur standard 2.

- `fopen, fread, fwrite, ...` fonctionnent avec des *file pointers*. Tout est décrit dans `man fread` ou dans `/usr/include/stdio`.

```
extern struct _iobuf {
    int     _cnt;               /* characters left */
    char    *_ptr;              /* next character position */
    char    *_base;             /* location of buffer */
    int     _bufsiz;            /* buffer size */
    short   _flag;              /* mode of file access */
    char    _file;              /* file descriptor */
} _iob[_N_STATIC_IOBS];

#define stdin     (&_iob[0])
#define stdout    (&_iob[1])
#define stderr    (&_iob[2])

#define getc(p)   (--(p)->_cnt>=0? *(p)->_ptr++&0377:_filbuf(p))
#define getchar() getc(stdin)
#define putc(x,p) \
    (--(p)->_cnt>=0? \
        ((int)(*(unsigned char *)(p)->_ptr++=(x))):\
        _flsbuf((unsigned char)(x),p))
#define putchar(x)    putc(x,stdout)

#define _IOFBF      00000
```

```
#define _IOREAD     00001
#define _IOWRT      00002
#define _IONBF      00004
#define _IOMYBUF    00010
#define _IOEOF      00020
#define _IOERR      00040
#define _IOSTRG     00100
#define _IOLBF      00200
#define _IORW       00400
#define _IOAPPEND   01000
```

Exemples typiques

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

main()
{
    int c;

    while ((c = getchar()) != EOF)
        putchar (tolower(c));
    return 0;
}
```

```
#include <stdio.h>

/* cat: concatenate files */
main (int argc, char *argv[])
{
    FILE *fp;
    void filecopy (FILE *, FILE *);
    if (argc == 1)
        filecopy (stdin, stdout)
    else
        while (--argc > ))
            if ((fp = fopen (*++argv, "r")) == NULL) {
                printf ("cat: can’t open %s\n", *argv);
                return 1;
        } else {
                filecopy (fp, stdout);
                fclose (fp);
        }
    return 0;
```

```
}

void filecopy (FILE *ifp, FILE *ofp)
{
    int c;
    while ((c = getc (ifp)) != EOF)
        putc (c, ofp);
}
```

Exemple (suite)

```
#include <stdio.h>

/* cat: concatenate files */
main (int argc, char *argv[])
{
    FILE *fp;
    void filecopy (FILE *, FILE *);
    char *prog = argv[0];

    if (argc == 1)
        filecopy (stdin, stdout)
    else
        while (--argc > ))
            if ((fp = fopen (*++argv, "r")) == NULL) {
                fprintf (stderr, "%s: can't open %s\n",
                   prog, *argv);
                exit (1);
        } else {
                filecopy (fp, stdout);
                fclose (fp);
        }
    if (ferror (stdout)) {
        fprint (stderr, "%s: error writing stdout\n", prog);
        exit (2);
    }
    exit (0);
}

void filecopy (FILE *ifp, FILE *ofp)
{
    int c;
    while ((c = getc (ifp)) != EOF)
        putc (c, ofp);
}
```

Disques

- Cylindres, plateaux, secteurs
- 3600 t/s (16 ms/t) à 7200 t/s (8 ms/t)
- 800 à 2000 cylindres (12-15 ms de déplacement du bras)
- 18 à 100 secteurs par piste
- 1 à 8 plateaux

Un bloc est adressé comme un tableau à 3 dimensions $< c, p, s >$. Les groupes contiennent en général 16 cylindres. (En mode SCSI, le triplet n'est plus accessible et on donne seulement un numéro de bloc logique; $< c, p, s >$ est presque partout une fonction lexicographique croissante de ce numéro logique).

```
TIT     .Computer architecture : a quantitative approach
SLA     .David A. Patterson, John L. Hennessy ; with a contribution
        .by David Goldberg. - San Mateo, CA : Morgan Kaufmann, 1990.
        .- XXVIII-594 p. + annexes ; 25 cm.
```

Montages, partitions, *devices*

- `/dev` contient les périphériques. `ls -l` donnent

```
crw-------  1 root      56,   0 Nov 21  1990 /dev/rrz0a
crw-------  1 root      56,   1 Sep 23 18:34 /dev/rrz0b
crw-------  1 root      56,   2 Nov 21  1990 /dev/rrz0c
brw-------  1 root      21,   0 Sep 23 18:18 /dev/rz0a
brw-------  1 root      21,   1 Nov 21  1990 /dev/rz0b
brw-------  1 root      21,   2 Nov 21  1990 /dev/rz0c
```

  Les 2 chiffres devant la date sont les nombres majeur et mineur de ce périphérique. Le 1er indexe une table des *drivers* qui gèrent ce *device*, le 2ème est un sous numéro pour distinguer ce périphérique parmi ceux qui sont gérés par un même *driver*.

- Une *device* peut être sous forme caractères ou par bloc. Dans le 2ème cas, toute lecture physique se fera avec “bufferisation”.

- Les noms de fichiers sont associés à des *devices* par montage. Faire `/etc/mount` ou regarder la table `/etc/fstab`. Certains noms peuvent être montés sur des *devices* distants par NFS (cf. plus tard).

Système de fichier V7

- noms $\neq$ nœuds internes
- sur le disque, nœud interne *inode*

```
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 */
/*
 * the 40 address bytes:
 *      39 used; 13 addresses
 *      of 3 bytes each.
 */
```

- les blocs d'information *data blocks*
- dans un directory, les blocs data contiennent les entrées suivantes (paires <*inode*, noms>)

```
#define DIRSIZ  14
struct  direct
{
        ino_t   d_ino;
        char    d_name[DIRSIZ];
};
```

Système de fichier V7 (suite)

```
TIT     .The Design of the UNIX operating system
SLA     .Maurice J. Bach. - Englewood Cliffs, NJ :
        .Prentice-Hall, 1986. - XIV-471 p. ; 23 cm.
```

Accès au fichier:

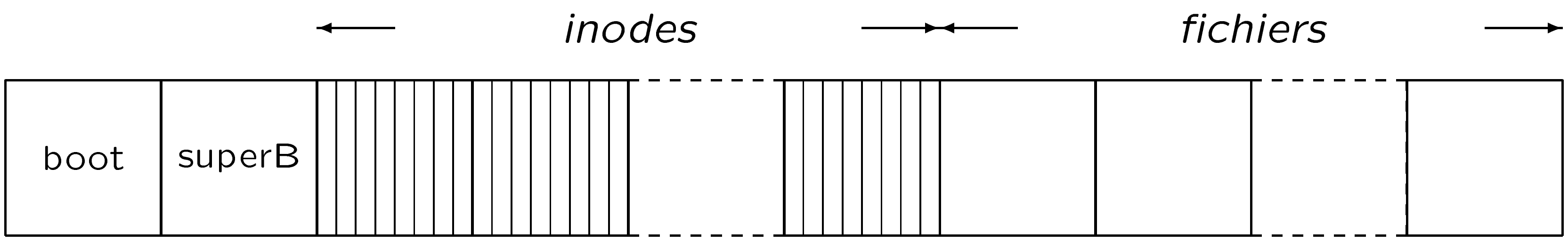
- les blocs data contiennent 512 octets
- accès direct pour petits fichiers ($10 \times 512 = 5k$)
- accès doublement indirect (70 kO)
- accès doublement indirect (8 MO)
- accès triplement indirect (1 GO)

L'indirection n'est pas génante, à cause du cache disque (cf. plus tard).

Une partition:

- un bloc de *boot* (code pour charger `/unix`)
- un bloc descriptif *superblock*
- la zone *inodes*, 8 par bloc
- la zone donnée

Système de fichier V7 (suite)



Système de fichier BSD (4.0)

- blocs données sont loin des inodes (4 MO inodes pour 140 MO données). Faire la commande `df -i`.
- blocs très petits $\Longrightarrow$ déplacements bras disque
- on perd tout si on perd le *superblock*

En Unix 4.0BSD, la taille des blocs est de 1024. D'où

- le débit a doublé (*throughput*)
- moins de besoins de blocs d'indirection

Il reste le problème de l'éparpillement des blocs de données. Les débits varient de 175 kO/s à 30 kO/s.

L'éparpillement provient de la liste libre des blocs de donnée, unique pour toute la partition. Au début, la liste et toute la partition sont fixées par la commande `mkfs`. On précise l'entrelacement optimal pour laisser le temps de rotation nécessaire entre 2 accès moyens à des blocs de donnée successifs.

Remarque: beaucoup des commandes Unix prennent les arguments en ordre canonique (souvent alphabétique) à cause de la convention * des shells. Or créer des fichiers et les supprimer dans ces conditions avec une stratégie LIFO aboutit à l'éparpillement!

Système de fichier BSD (4.3)

- blocs de 8k
- groupes de cylindres (avec une copie du *superblock*, une zone *inode*, et une zone donnée). (Typiquement 16 cylindres / groupe).
- le *superblock* contient un bitmap des blocs libres, ce qui permet d'avoir une politique autre que LIFO pour l'allocation.
- fragments (1k) pour éviter des gachis de l'ordre de 40% avec blocs de 4k. Les fragments d'un bloc doivent être consécutifs.
- localité
- fiabilité (le superblock n'est jamais au même endroit) dans le groupe de cylindre

Tout est décrit dans `/usr/include/ufs/fs.h` avec les macros correspondantes pour donner les adresses disques.

Système de fichier BSD (4.3) suite

Par exemple:

```
#define itod(fs, x) \
        ((daddr_t)(cgimin(fs, itog(fs, x))) + \
        (blkstofrags((fs), (((x) % (fs)->fs_ipg) / INOPB(fs))))))

#define itog(fs, x)       ((x) / (fs)->fs_ipg)

#define cgbase(fs, c)    ((daddr_t)((fs)->fs_fpg * (c)))
#define cgstart(fs, c) \
    (cgbase(fs, c) + (fs)->fs_cgoffset * ((c) & ~((fs)->fs_cgmask)))
#define cgimin(fs, c)    (cgstart(fs, c) + (fs)->fs_iblkno)    /* inode blk */

#define blkstofrags(fs, blks)    /* calculates (blks * fs->fs_frag) */ \
    ((blks) << (fs)->fs_fragshift)

#define    INOPB(fs)    ((fs)->fs_inopb)
```

$$\begin{aligned}\texttt{itod}(x) &= \texttt{cgimin}(x/1920) + \texttt{blkstofrags}((x\%1920)/64)\\ &= \texttt{cgstart}(x/1920) + 32 + \texttt{frag} << 3\\ &= (4320 \times \texttt{cg}) + 24 \times (\texttt{cg\&0xf}) + 32 + \texttt{frag} \times 8\end{aligned}$$

Partition disque (`/root` sur `/dev/rrz1a` Dec Station 3100)

Adresse 0: 8k de boot,
Adresse 8k: superblock de 8k

```
 fs_link = 0x0,                /* linked list of file systems */
 fs_rlink = 0x0,               /*     used for incore super blocks */
 fs_sblkno = 16,               /* addr of super-block in filesys */
 fs_cblkno = 24,               /* offset of cyl-block in filesys */
 fs_iblkno = 32,               /* offset of inode-blocks in filesys */
 fs_dblkno = 272,              /* offset of first data after cg */
 fs_cgoffset = 24,             /* cylinder group offset in cylinder */
 fs_cgmask = -16,              /* used to calc mod fs_ntrak */
 fs_time = 696960299,          /* last time written */
 fs_size = 16384,              /* number of blocks in fs */
 fs_dsize = 15343,             /* number of data blocks in fs */
 fs_ncg = 4,                   /* number of cylinder groups */
 fs_bsize = 8192,              /* size of basic blocks in fs */
 fs_fsize = 1024,              /* size of frag blocks in fs */
 fs_frag = 8,                  /* number of frags in a block in fs */
/* these are configuration parameters */
 fs_minfree = 10,              /* minimum percentage of free blocks */
 fs_rotdelay = 0,              /* num of ms for optimal next block */
 fs_rps = 60,                  /* disk revolutions per second */
/* these fields can be computed from the others */
 fs_bmask = -8192,             /* ``blkoff'' calc of blk offsets */
 fs_fmask = -1024,             /* ``fragoff'' calc of frag offsets */
 fs_bshift = 13,               /* ``lblkno'' calc of logical blkno */
 fs_fshift = 10,               /* ``numfrags'' calc number of frags */
/* these are configuration parameters */
 fs_maxcontig = 1,             /* max number of contiguous blks */
 fs_maxbpg = 256,              /* max number of blks per cyl group */
/* these fields can be computed from the others */
 fs_fragshift = 3,             /* block to frag shift */
```

```
 fs_fsbtodb = 1,              /* fsbtodb and dbtofsb shift constant */
 fs_sbsize = 8192,            /* actual size of super block */
 fs_csmask = -512,            /* csum block offset */
 fs_csshift = 9,              /* csum block number */
 fs_nindir = 2048,            /* value of NINDIR */
 fs_inopb = 64,               /* value of INOPB */
 fs_nspf = 2,                 /* value of NSPF */
 fs_optim = 0,                /* optimization preference, see below */
 fs_sparecon = {0, 0, 0, 0, 0},  /* reserved for future constants */
/* sizes determined by number of cylinder groups and their sizes */
 fs_csaddr = 272,             /* blk addr of cyl grp summary area */
 fs_cssize = 1024,            /* size of cyl grp summary area */
 fs_cgsize = 2048,            /* cylinder group size */
/* these fields should be derived from the hardware */
 fs_ntrak = 15,               /* tracks per cylinder */
 fs_nsect = 36,               /* sectors per track */
 fs_spc = 540,                /* sectors per cylinder */
/* this comes from the disk driver partitioning */
 fs_ncyl = 61,                /* cylinders in file system */
/* these fields can be computed from the others */
 fs_cpg = 16,                 /* cylinders per group */
 fs_ipg = 1920,               /* inodes per group */
 fs_fpg = 4320,               /* blocks per group * fs_frag */
/* this data must be re-computed after crashes */
 fs_cstotal = {cs_ndir = 84,
 cs_nbfree = 710,
 cs_nifree = 7146,
 cs_nffree = 122},
/* these fields are cleared at mount time */
 fs_fmod = 0 '\000',          /* super block modified flag */
 fs_clean = 1 '\001',         /* file system is clean flag */
 fs_ronly = 0 '\000',         /* mounted read-only flag */
 fs_flags = 0 '\000',         /* currently unused flags */
#define    fs_cleantimer    fs_flags/* Clean byte timeout factor */
 fs_fsmnt = {"/\000nt\000ufs_mount",/* name mounted on */
 '\000' <repeats 498 times>},
/* these fields retain the current block allocation info */
```

```
fs_cgrotor = 3,              /* last cg searched */
fs_csp = {0xc3644000,        /* list of fs_cs info buffers */
0x0 <repeats 31 times>},
fs_cpc = 4,                  /* cyl per cycle in postbl */
fs_postbl = {{0, 5, 3, 1, 8, 6, 4, 2}, /* head of blocks for each rotation */
{34, 41, 39, 37, 35, 42, 40, 38},
{70, 68, 75, 73, 71, 69, 76, 74},
{106, 104 102, 109, 107, 105, 103, 110},
{-1, -1, -1, -1, -1, -1, -1, -1} <repeats 28 times>},
fs_magic = 72020,            /* magic number */
fs_rotbl = ...
```

Groupe de cylindres (même exemple, ...)

```
cg_link = 0x0,                    /* linked list of cyl groups */
cg_rlink = 0x0,                   /*     used for incore cyl groups */
cg_time = 696960289,              /* time last written */
cg_cgx = 0,                       /* we are the cgx'th cylinder group */
cg_ncyl = 16,                     /* number of cyl's this cg */
cg_niblk = 1920,                  /* number of inode blocks this cg */
cg_ndblk = 4320,                  /* number of data blocks this cg */
cg_cs = {cs_ndir = 28,            /* cylinder summary information */
cs_nbfree = 119,
cs_nifree = 1862,
cs_nffree = 10},
cg_rotor = 2400,                  /* position of last used block */
cg_frotor = 2688,                 /* position of last used frag */
cg_irotor = 54,                   /* position of last used inode */
cg_frsum = {0, 0,  0, 1, 0, 1 0},  /* counts of available frags */
cg_btot = {0, 0, 0, 0, 0, 0, 0, 8, 17, 31, 16 32, 0, 15, 0
<repeats 17 times>},  /* block totals per cylinder */
cg_b = {{0, 0, 0, 0, 0, 0, 0, 0},  /* positions of free blocks */
{0, 0, 0, 0, 0, 0, 0, 0},
{0 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0},
{1, 1, 1, 1, 1, 1, 1, 1},
{5, 1, 1, 2, 3, 1, 2, 2},
{6, 4, 3, 4, 3, 4, 3, 4},
{5, 2, 0, 2, 3, 2, 1, 1},
{7, 3, 4, 4, 3, 3, 4, 4},
{0, 0, 0, 0, 0, 0, 0, 0},
{4, 2, 2, 2, 1, 0, 2, 2},
{0, 0, 0, 0, 0, 0, 0, 0} <repeats 17 times>},
cg_iused = {"\377\377\377\377\377\377\177\000p",  /* used inode map */
'\000' <repeats 247 times>},
cg_magic = 590421,               /* magic number */
cg_free = {"\000"}}              /* free block map */
```

Exemples de *inode*

Tout est décrit dans `/usr/include/ufs/ufs_inode.h`, `/usr/include/sys/gnode_common.h`.

```
     2 drwxr-xr-x 15 root         1024 Feb  1 17:04 /

di_gcom = {gn_gc =
           {gc_mode = 0x41ed,            /* 0: mode and type of file */
            gc_nlink = 15,               /* 2: number of links to file */
            gc_uid = 0,                  /* 4: owner's user id */
            gc_gid = 0,                  /* 6: owner's group id */
            gc_size = {val = {1024, 0}}, /* 8: number of bytes in file */
            gc_atime = {tv_sec = 669396747, tv_usec = 90158},
            gc_mtime = {tv_sec = 696960289, tv_usec = 1198},
            gc_ctime = {tv_sec = 696960289, tv_usec = 1198}},
            dg_db = {288,
                       0 <repeats 11 times>},
            dg_ib = {0, 0, 0},
            dg_blocks = 2,/* 104: blocks actually held */

     3 drwxr-xr-x  2 root         8192 Jan 13  1989 /lost+found

di_gcom = {gn_gc =
           {gc_mode = 0x41ed,            /* 0: mode and type of file */
            gc_nlink = 2,                /* 2: number of links to file */
            gc_uid = 0,                  /* 4: owner's user id */
            gc_gid = 0,                  /* 6: owner's group id */
            gc_size = {val = {8192, 0}}, /* 8: number of bytes in file */
            gc_atime = {tv_sec = 621008968, tv_usec = 686698},
            gc_mtime = {tv_sec = 600669697, tv_usec = 288278},
            gc_ctime = {tv_sec = 600669697, tv_usec = 288278}},
           dg_db = {280,
                      0 <repeats 11 times>},
           dg_ib = {0, 0, 0},
```

```
        dg_blocks = 16,/* 104: blocks actually held */

   4 -rwxr-xr-x  1 root      2113004 Mar  7  1990 /vmunix

di_gcom = {gn_gc = {gc_mode = 0x81ed,
                    gc_nlink = 1,
                    gc_uid = 0,
                    gc_gid = 0,
                    gc_size = {val = {2113004, 0}},
          gc_atime = {tv_sec = 696619351, tv_usec = 290190},
          gc_mtime = {tv_sec = 636832739, tv_usec = 389864},
          gc_ctime = {tv_sec = 660850797, tv_usec = 662236}},
          dg_db = {296,               /* 40: disk block addresses */
                   304,
                   312,
                   320,
                   328,
                   336,
                   344,
                   352,
                   360,
                   368,
                   376,
                   384},
          dg_ib = {392, 0, 0},        /* 88: indirect blocks */
          dg_blocks = 4144,           /* 104: blocks actually held */
```

Directories en BSD

Un directory est une liste d'entrées du format non fixe suivant:

```
#define MAXNAMLEN       255

struct gen_dir {
      u_long  gd_ino;                 /* not meaningful for all sfs types */
      u_short gd_reclen;              /* how large this structure really is */
      u_short gd_namelen;             /* how long the named component is */
      char    gd_name[MAXNAMLEN + 1]; /* what the name is */
        };
```

Il y a des ordres spéciaux en BSD pour lire les directories `readdir`, ...

```
TIT     .The Design and implementation of the 4.3BSD UNIX operating
        .system
SLA     .Samuel J. Leffler, Marshall Kirk McKusick, Michael J.
        .Karels, John S. Quarterman. - Reprint. with corrections. -
        .Reading, MA ; Don Mills, Ontario ; Wokingham, GB :
        .Addison-Wesley, 1989. - XXII-471 p. ; 24 cm.
```

Exercices en TD et à la maison

- Forts: la partition

  sur

  est ouverte en lecture à tous pour au moins une semaine. Lire les superblocs du fichier spécial

  `/dev/`

  et imprimer son état d'allocation. Si vous avez le temps, lire le contenu d'un fichier (sans passer par l'ordre `read` normal du système de fichier. Indication: se servir des fichiers *include* décrivant les formats d'un système de fichier (superbloc, groupe de cylindres, inode).

- Moyens: voir avec les Chefs de Travaux Pratiques.

- Débutants:

  1. `cat` et exercices du chapitre 1.
  2. Lire un fichier et l'imprimer en transformant les minuscules en majuscules.

Systèmes et réseaux

Cours 3

Jean-Jacques.Levy@inria.fr
Ecole polytechnique

tel: 39-63-56-89
http://pauillac.inria.fr/∼levy/

http://www.poly.polytechnique.fr/poly/...
.../∼www/poly/maj2-syst-96/

Plan

Revoir <<, `cpp`, *file descriptors*

Protection des fichiers (rappel)

Variables globales / locales. Table des symboles

Initialisations, Variables *register*

Compilation séparée – Format des binaires

Erreurs dans un appel système

Processus

Espaces d'adressage

Créations de processus

Exercices

Protection des fichiers (rappel)

- La protection est rangée dans les *inodes*

```
#include <sys/inode.h>

di_mode = 0x41ed == 040755                 /* mode de "/" */
                 == IFDIR
                  | IREAD | IWRITE | IEXEC
                  |(IREAD | IEXEC) >> 3
                  |(IREAD | IEXEC) >> 6

di_mode = 0x41ed == 0100755               /* mode de "/vmunix" */
                 == IFDIR
                  | IREAD | IWRITE | IEXEC
                  |(IREAD | IEXEC) >> 3
                  |(IREAD | IEXEC) >> 6
```

- les bits de mode et de protection sont définis dans **/usr/include/sys/inode.h**.

```
#define IFPORT          0010000         /* port (named pipe) */
#define IFCHR           0020000         /* character special */
#define IFDIR           0040000         /* directory */
#define IFBLK           0060000         /* block special */
#define IFREG           0100000         /* regular */
#define IFLNK           0120000         /* symbolic link */
#define IFSOCK          0140000         /* socket */
#define IFPIPE          0160000         /* pipe */

#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 IWRITE          0200
#define IEXEC           0100
```

Protection des fichiers (rappel)

- `chmod` permet de modifier les accès d'un fichier.

```
% ls -ld Mail
drwxr--r--  4 levy         512 Oct 30 15:48 Mail
% chmod go-r Mail
% ls -ld Mail
drwx------  4 levy         512 Oct 30 15:48 Mail
```

- `chmod` permet de modifier les accès d'un fichier.

- les modes sont pour le propriétaire (*user*), le groupe, et les autres. Peu précis mais dense. Les modes principaux sont `read`, `write`, `execute`. Pour un *directory*, `execute` veut dire qu'on peut faire `cd`.

- le mode `suid` *set user id* permet à un fichier exécutable de prendre provisoirement les droits du propriétaire. C'est utile pour pouvoir écrire un serveur (courrier, forum).

- le mode `sticky` n'est pratiquement plus utilisé. C'est une optimisation pour ne pas lire trop souvent un fichier exécutable.

Variables statiques – Noms externes, Table des symboles

- Par défaut, les variables globales sont "externes". Sinon, il faut leur mettre le préfixe `static`. Les variables externes sont connues des autres modules quand on fait `cc -c xxx.c`, `cc -c yyy.c` et `cc xxx.o yyy.o`. La table des symboles *namelist* externes (et internes) d'un binaire est obtenue par la commande `nm -g` sur Vax.

- Les variables `static` peuvent être aussi locales. Ce sont alors des variables rémanentes, dont la valeur reste inchangée quand on sort du bloc correspondant et on y rerentre.

- Exemple d'implantation `xxx.c`

```
static int sp = 0;
static stack[1000];
void push (float x)  {stack[sp++] = x;}
float pop (void)     {return stack[--sp];}
```

- Exemple d'utilisation `yyy.c`.

```
extern void push (float);
extern float pop (void);
```

  On peut aussi le mettre dans un "fichier d'interface" `xxx.h` et faire

```
#include    "xxx.h"
```

Initialisation des variables

- Les variables locales et globales peuvent être initialisées. Pour les globales, c'est au moment du chargement du binaire en mémoire que ces variables sont initialisées. Pour les locales, c'est à chaque entrée dans le bloc (sauf pour les `static`).

- Les tableaux et structures globales peuvent être initialisées. Dans le cas des tableaux, on peut déclarer le tableau ouvert et laisser `cc` la déterminer en fonction du nombre d'éléments initialisés.

- Attention, il y a 2 formats d'initialisation (le vieux et le moderne qui respecte plus les structures). Utiliser toujours le nouveau format.

- Exemples

```
int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

char prof[] = "levy";
char prof[] = {'l', 'e ', 'v', 'y', '\0'}'
char *profs[] = {"levy", "ehrlich", "rouaix",
                 "ruget", "doligez",
                 NULL};
```

- Variables *register*. Pour signaler à `cc` que la variable est très souvent utilisée. Ne pas trop utiliser.

Compilation séparée – Format des binaires

- On a vu que le compilateur C `cc` appelait l'éditeur de lien `/bin/ld` pour permettre la compilation séparée de plusieurs modules.

```
% cc -c xxx.c
% cc -c yyy.c
% cc xxx.o yyy.o
% nm -gn a.out | more
```

- Les fichiers `.o` sont des binaires relogeables. Il y a toute l'information pour mettre le binaire à toute adresse.

- Les fichiers `a.out` sont des binaires absolus. Sur ds5000:

```
00400140 T __start ...
```

  et sur Vax:

```
00000000 T start
```

- `cc -r` permet de faire un fichier `.o` à partir de 2 fichiers `.o`

Format des binaires (suite)

- Dans un fichier binaire Unix, il y a 3 zones: `text` le programme, `data` les données initialisées, `bss` les données non initialisées.

- `size a.out` permet de connaître la taille de ces trois zones.

```
% size /vmunix
text    data    bss     dec     hex
1150864 122384  486672  1759920 1adab0
```

- Quand le programme s'exécute, une 4ème zone est celle de la *pile* dans laquelle on met les variables locales (au fur et à mesure de leur utilisation), les *stack frames* pour passer les arguments aux procédures (cf cours de compilation).

- En début d'exécution, la zone `bss` est mise à zéro. Ne pas trop utiliser ce fait.

- `limit` en Cshell permet de connaître les tailles maximales autorisées pour ces valeurs.

```
% limits
datasize        65536 kbytes
stacksize       512 kbytes
```

- le type d'un binaire se vérifie par les 2 premiers octets (*magic number*)

Makefile, sccs, rcs

- `make` donne un ensemble de règles pour reconstruire les modules d'un programme. (Attention: tabulations $\neq$ espaces!) Un tri topologique sur les dates permet de reconstruire le minimum.

```
DESTDIR= /usr/local/bin

all: xxx zzz

xxx: xxx.o yyy.o
	cc -O -o xxx xxx.o yyy.o

xxx.o: xxx.c
	cc -c xxx.c

yyy.o: yyy.c
	cc -c yyy.c

zzz: zzz.c
	cc -O -o zzz zzz.c

install: all
	cp xxx zzz ${DESTDIR}

clean:
	rm -f xxx zzz xxx.o yyy.o
```

- `sccs`, `rcs`, `cvs` permettent de gérer des numéros de versions. Très important pour les gros programmes.

Erreurs dans un appel système

- `errno` est une variable qui a une valeur après tout appel-système. En Unix, l'appel système retourne une valeur anormale (-1 ou NULL par exemple), et `errno` permet d'avoir un diagnostic plus précis. Ses valeurs possibles sont définies dans `/usr/include/errno.h`

- `perror (char *s);` permet d'écrire un message standard (en anglais) avec l'argument `s` suivi de `:`

- dans un gros système , il est bon de regrouper les messages d'erreur. Sinon, il n'y a pas d'uniformité.

Processus

- Un programme en exécution est appelé un *processus*. A tout moment un même utilisateur peut avoir plusieurs processus:

```
% ps
  PID TT STAT  TIME COMMAND
21908 p0 S     0:01 -tcsh (tcsh)
21918 p0 S     1:23 emacs cours.ftex
22670 p0 R     0:00 ps
21920 p1 I     0:02 -tcsh (tcsh)
```

- Unix est multi-utilisateur, et controle les processus de plusieurs utilisateurs.

```
% ps aux
USER       PID %CPU %MEM   SZ  RSS TT STAT   TIME COMMAND
figer    27300  2.4  0.4 1074  981 ?  S N   31:35 StelBot -n MagOt_ -u arthus
figer     2611  2.3  0.5 1166 1072 ?  S N   38:21 StelBot
nobody   14461  2.0  0.1  407  149 ?  S      0:00 /usr/local/w3/bin/httpd
charaya  14421  1.3  0.1  548  223 r4 S      0:00 -tcsh (tcsh)
charaya  14413  0.9  0.5 2139 1120 ?  S      0:00 xterm -ls
charaya  14430  0.9  0.0  129   52 r4 S      0:00 rlogin sil
root     14433  0.4  0.0   82   36 ra S      0:00 rlogind
potters  14395  0.2  0.1  477  204 r2 S      0:00 elm
potters  14378  0.2  0.5 2139 1123 ?  S      0:00 xterm -ls
potters  14387  0.2  0.1  548  221 r2 I      0:00 -tcsh (tcsh)
root      9675  0.1  0.1  374  108 ?  S      4:11 /usr/local/w3/bin/httpd
volkov   14078  0.1  0.5 2137 1120 ?  S      0:00 xterm -ls
pelikan  10471  0.1  0.4 1440  798 ?  S      0:19 xmailtool
brunet\oe  12396  0.1  1.0 3161 2107 ?  S      0:00 xterm -ls -geometry 85x25+0-0
root      9481  0.1  0.0  168   96 ?  S     17:11 /usr/et\c{c}ypserv
root      9692  0.1  0.0    6    4 ?  S      4:43 /et\c{c}update
```

Processus – Espaces d'adressage

- Un système en temps partagé doit donner l'illusion
  à tout processus qu'il peut disposer de tout l'environnement.

- Unix donne du temps d'exécution à chaque processus avec un algorithme d'ordonnancement *scheduling*

```
TIT     .COMPUTER AND JOB-SHOP SCHEDULING THEORY
AUT     .Coffman, Edward G.. Ed.
EDI     .WILEY-INTERSCIENCE,NEW YORK
DAT     .1976
```

- protection mémoire et *mémoire virtuelle*. Chaque processus fonctionne dans son monde. Fondamental pour que le système soit sûr. Contrexemples: Macintosh, Alto, Dorado, Windows 95.

- accès social aux ressources, et donc synchronisation (*ipc – inter process communication*). Unix est traditionnellement faible sur ce point, mais s'est amélioré dans ses variantes récentes (Mach).

- les processus Unix sont lourds (à commuter).

Créations de processus

- `fork()` duplique le processus courant, et crée un processus fils par division cellulaire. `fork()` répond 0 pour le processus fils et le numéro (unique) du processus pour le processus père. C'est la seule manière de créer des processus Unix. Elle est lourde, mais permet de passer toute l'information au fils (comme en biologie). Tout l'espace `text`, `data`, `bss`, `stack` est copié.

```
for (i = 0; i < 15; i++)
    if (fork() == 0) {
        printf ("child proc %d = %x\n", i, getpid());
        exit (1);
    }
/* suite du pe ‘re */
r = wait(&status);
printf ("r = %x, status =%\n", r,  status);
```

- `exec` permet de remplacer l'image mémoire d'un processus par celle déduite à partir d'un fichier `a.out`.

```
if (fork() == 0) {
    execlp ("xxx", "xxx", "a", "b", (char *) 0);
    fprintf (stderr, "xxx not executed\n");
    exit (2);
    }
```

- En BSD, on a trouvé insupportable de faire `fork` puis `exec`. D'où le nouvel appel système `vfork`.

Duplication de *file descriptors*

- Si on écrit un *shell*, on veut lancer des nouveaux programmes avec `exec` et avec l'entrée/sortie standard prédéfinie (à cause de $<$ ou $>$). Or les *file descriptors* traversent `exec`. D'où

```
fd1 = open ("fichier1", 1);
fd2 = open ("fichier2", 2);
fd3 = open ("fichier3", 2);
if ((pid = fork()) == 0) {
    close (0); dup (fd1);
    close (1); dup (fd2);
    close (2); dup (fd3);
    execlp ("prog", ...);
    exit (1);
}
```

- `dup()` duplique le *file descriptor* sur le premier disponible. On peut aussi utiliser `dup2()`.

- Réciproquement, on veut pouvoir remettre au terminal les *file descriptors* standards avec

```
fd1 = open ("/dev/tty", 1);
fd2 = open ("/dev/tty", 2);
fd3 = open ("/dev/tty", 2);
if ((pid = fork()) == 0) {
    close (0); dup (fd1);
    close (1); dup (fd2);
    close (2); dup (fd3);
    execlp ("prog", ...);
    exit (1);
}
```

Duplication de *file descriptors* (suite)

- `FILE *fdopen(int fd, char *mode)`
  permet d'avoir un *file pointer* en fonction d'un *file descriptor*.

- `int fileno(FILE *fp)`
  donne le *file descriptor* associé au *file pointer*.

Exercices en TD et à la maison

- Débutants:

  `ls` et `ls -R`. On devra parcourir la structure renvoyée par `stat()`. (Faire attention aux liens).

- Chevronnés:

  `grep` parallèle, ie faire une commande `pgrep [-N] string [file]` où N est le nombre de processus forkés et *file* un fichier contenant des noms de fichiers (un par ligne) qu'on obtient en faisant par exemple `find . -type f -print`

  La sémantique de `pgrep [-N] string [file]` est:

  – soient f_1 f_2 ... f_n les n lignes de fichier d'entrée. On met dans N seaux équi-répartis les f_i.

  – on forke `grep string` sur chacun des seaux. Remarque: on peut prendre le `grep` du système ou écrire un `grep` perso trivial sans KMP ou Boyer-Moore avec bêtement `strcmp` et la méthode quadratique en *string* et *input*.

  – 2 options sont possibles pour `pgrep`:

    1. on liste toutes les lignes des fichiers contenant string,

    2. on arrête le tout dès qu'on a trouvé une ligne. Utiliser `kill(p,SIGKILL)` pour tuer p.

Reste à bien comprendre comment débugger en gdb avec plusieurs processus....

Systèmes et réseaux

Cours 4

Jean-Jacques.Levy@inria.fr
Ecole polytechnique

tel: 39-63-56-89
http://pauillac.inria.fr/∼levy/

http://www.poly.polytechnique.fr/poly/...
.../∼www/poly/maj2-syst-96/

Plan

Goto's non locaux en C

Concurrence et synchronisation

Mémoire partagée

Groupes de processus

Signaux

Exercices

Goto's non locaux en C

- `setjmp`, `longjmp` En C, il n'y a pas de procédures emboîtées. Donc pour faire un saut global, on sauve les registres et le pointeur de pile par `setjmp`, et on peut restituer cet environnement par `longjmp`. Attention, les variables globales ne sont pas restituées. Par convention, `setjmp` retourne 0, et `longjmp` la valeur de son 2nd argument ($\neq 0$). Exemple:

```
#include <setjmp.h>

jmp_buf env;
int i = 0;
main ()
{
    void exit();

    if(setjmp(env) != 0) {
        (void) printf("2eme retour de setjmp:   i=%d\n", i);
        exit(0);
    }
    (void) printf("1er retour de setjmp: i=%d\n", i);
    i = 1;
    g();
    /*NOTREACHED*/
}

g()
{
    longjmp(env, 1);
    /*NOTREACHED*/
}
```

donne `i=0` et `i=1`.

Processus et synchronisation

Séquentiel $\rightarrow$ Concurrent
est une étape conceptuelle importante.

Le monde de l'informatique est concurrent:

- machines multi-processeurs,
- drivers pour des périphériques lents,
- les utilisateurs sont concurrents,
- systèmes distribué ont de multiple clients,
- réduire le temps d'exécution

Dans la programmation système, on a la concurrence pour:

- l'accès concurrent à une ressource commune,
- la gestion des événements asynchrones,
- la détection de fin des processus.

Exemple: Pile concurrente

Comment rendre Pop et Push concurrents?

```
Pile *head = NULL;

Pile *Pop()
{
    Pile *topElement;

    while (head == NULL)
        Wait_nonEmpty ();
    topElement = head;
    head = head -> next;
}

void Push (Pile *newElement)
{
    newElement -> next = head;
    head = newElement;
    Signal_nonEmpty ();
}
```

Il faut d'abord que l'accès à Pop et à Push soit exclusif, et écrire `Wait_nonEmpty` et `Signal_nonEmpty`.

Section critique (ou exclusion mutuelle) est un classique de la programmation concurrente.

Concurrence et mémoire partagée

- Exclusion mutuelle
- Producteur, Consommateur
- Lecteurs et écrivains
- Registres atomiques

```
TIT     .Principles of concurrent programming
SLA     .M. Ben-Ari. - Englewood Cliffs, NJ ; London ; New
        .Delhi : Prentice-Hall, 1982. - XV-172 p. ; 22 cm.
```

Section critique -1-

```
turn = 1;

p1()
{
    for (;;) {
        while (turn == 2)
            ;
        Crit1();
        turn = 2;
        Reste1();
    }
}

p2()
{
    for (;;) {
        while (turn == 1)
            ;
        Crit2();
        turn = 1;
        Reste2();
    }
}
```

Ok, mais manque de souplesse.

Section critique -2-

```
c1 = c2 = 1;
```

```
p1()
{
    for (;;) {
        while (c2 == 0)
            ;
        c1 = 0;
        Crit1();
        c1 = 1;
        Reste1();
    }
}
```

```
p2()
{
    for (;;) {
        while (c1 == 0)
            ;
        c2 = 0;
        Crit2();
        c2 = 1;
        Reste2();
    }
}
```

Erreur car:

	c1	c2
	1	1
p1 teste c2	1	1
p2 teste c1	1	1
p1 positionne c1	0	1
p2 positionne c2	0	0
p1 fait Crit1()	0	0
p2 fait Crit2()	0	0

Section critique -3-

```
c1 = c2 = 1;

p1()                                p2()
{                                   {
    for (;;) {                          for (;;) {
        c1 = 0;                             c2 = 0;
        while (c2 == 0)                     while (c1 == 0)
            ;                                   ;
        Crit1();                            Crit2();
        c1 = 1;                             c2 = 1;
        Reste1();                           Reste2();
    }                                   }
}                                   }
```

Deadlock car:

```
                                     c1   c2
                                      1    1
p1 positionne c1                      0    1
p2 positionne c2                      0    0
p1 teste c1                           0    0
p2 teste c2                           0    0
...
```

Algorithme de Dekker

```
c1 = c2 = 1;
turn = 1;
```

```
p1()
{
    for (;;) {
        c1 = 0;
        while (c2 == 0)
            if (turn == 2) {
                c1 = 1;
                while (turn == 2)
                    ;
                c1 = 0;
    }
        Crit1();
        turn = 2;
        c1 = 1;
        Reste1();
    }
}
```

```
p2()
{
    for (;;) {
        c2 = 0;
        while (c1 == 0)
            if (turn == 1) {
                c1 = 1;
                while (turn == 1)
                    ;
                c2 = 0;
    }
        Crit2();
        turn = 1;
        c2 = 1;
        Reste2();
    }
}
```

L'algorithme de Dekker pour n processeurs.

```
enum {OUT, WANTING, IN} c[n];

s = 1;

pi()
{
    for (;;) {
        do {
           c[i] = WANTING;
           while (turn != i)
                if (c[turn] == OUT)
                    turn = i;
           c[i] = IN;
           ok = TRUE;
           for (j = 0; j < n; j++)
                if (j != i)
                    ok = ok & (c[j] != IN);
        } while (!ok);
        Crit1();
        c[i] = OUT;
        Reste1();
    }
}
```

L'algorithme de Peterson est plus simple que Dekker, quoique moins intuitif.

Algorithme de Peterson (IPL Juin 81)

```
c1 = c2 = 1;
turn = 1;
```

```
p1()
{
  for (;;) {
    c1 = 0;
    turn = 1;
    while (c2 == 0 && turn == 1)
        ;
    Crit1();
    c1 = 1;
    Reste1();
  }
}
```

```
p2()
{
   for (;;) {
     c2 = 0;
     turn = 2;
     while (c1 == 0 && turn == 2)
         ;
     Crit2();
     c2 = 1;
     Reste2();
   }
}
```

Sémaphores

- l'algorithme de Dekker n'a pas de sens si on dispose de l'instruction machine indivisible `TEST_AND_SET`.

- s est un booléen $s \in \{0, 1\}$. Les opérations $P(s)$, $V(s)$ sont indivisibles. `Dijkstra 65`.

```
s = 1;

p1()                                    p2()
{                                       {
    for (;;) {                              for (;;) {
        P(s);                                   P(s);
        Crit1();                                Crit2();
        V(s);                                   V(s);
        Reste1();                               Reste2();
    }                                       }
}                                       }
```

- Si $s = 1$, $P(s)$ positionne s à 0. Si $s = 0$, $P(s)$ se bloque.

- $V(s)$ met s à 1 et réveille un processus bloqué sur s.

Les sémaphores ont une lourde implémentation en Unix que personne n'utilise.

```
TIT     .The Design of the UNIX operating system
SLA     .Maurice J. Bach. - Englewood Cliffs, NJ :
        .Prentice-Hall, 1986. - XIV-471 p. ; 23 cm.
```

Filiation des processus

- `fork` établit une filiation entre processus. Tous les processus sont créés à partir de `init` (processus 1), cf cours 1.

- `exit` termine un processus. Unix regarde si le processus à un père actif. Sinon `init` devient son père (processus 1). Si oui, le fils passe à l'état zombie (avec uniquement un descripteur temporaire dans la table des processus, pour qu'on puisse récupérer sa mémoire, ses fichiers ouverts, etc) et disparait complètement quand le père fait `wait`. (`init` appelle toujours `wait` dès qu'un de ses fils termine.)

- le shell crée implicitement cette filiation en exécutant les commandes comme fille de lui-même. Les variables d'environnement sont passées en dernier argument de `exec`, et tout processus peut y accéder par `getenv()`

```
char *getenv (char *name);
```

Groupes de processus

- Session correspond à un login (ou remote login)
- Dans une session, groupe de processus *foreground* et *background*.
- Control-C interrompt les processus *foreground*
- une déconnexion envoie un signal à toute une session (en fait au leader de la session).
- `/dev/tty` est le terminal de la session.

Signaux

- Les signaux sont des interruptions logicielles. Ils permettent la communication entre processus, le traitement de phénomènes externes (donc asynchrones), ou les erreurs d'un processus. La communication entre processus se fait par les fonctions `kill` et `signal`.

```
kill (pid, signum);
signal (signum, function);
```

- `kill` envoie le signal `signum` au processus `pid`. Les nombres `signum` sont décrits dans `signal.h` et valent moins que 32. Si `pid` < 0, le signal est envoyé au "process group". Si `pid` $= -1$ et l'envoyeur est le super-utilisateur, envoyé à tout le monde.

- `signal` permet de dire l'action à entreprendre (*handler* du signal) sur la réception d'un tel signal. Valeurs conventionnelles: `SIG_IGN` pour ignorer, `SIG_DFL` pour action par défaut, ...

- Les évènements asynchrones externes sont par exemple la réception d'un caractère C-C (interruption), C-Z (suspension), C-_ (quit) sur un terminal, la déconnexion d'un terminal, la fin de délai d'une alarme posée par un processus.

Signaux (bis)

- Le signal peut arriver à tout moment, et donc au cours de la modification d'une structure de donnée (éventuellement de la bibliothèque C — par exemple, la liste libre de `malloc`). Il faut donc faire attention. On peut toujours positionner un *flag*, et le tester plus tard. De manière générale, les signaux ne feront pas de mal pour les procédures *réentrantes*. Très peu le cas en Unix, par exemple tout *stdio* n'est pas réentrant.

```
#include <signal.h>
#include <setjmp.h>
jmp_buf sjbuf;

main()
{
    int onintr(void);
    if (signal (SIGINT, SIG_IGN) != SIG_IGN)
        signal (SIGINT, onintr);
    setjmp (sjbuf);

    for (;;) {
        /* boucle d'interpretation */
        ...
    }

onintr()
{
    signal (SIGINT, onintr);
    printf ("\nInterrupt\n");
    longjmp (sjbuf, 0);
}
```

Signaux (ter)

- `alarm(n)` génère un signal `SIGALRM` dans les `n` secondes.

```
signal (SIGALRM, onalarm);
alarm (sec);
...

onalarm ()
{
    kill (pid, SIGKILL);
}
```

- Un *shell* prend les signaux pour les commandes qui ne sont pas en *background*. Et de manière général, il est malsain que 2 programmes père et fils qui ont tous les 2 un *toplevel* prennent un même signal. Ils vont tous les 2 revenir à leur boucle de lecture et se partager ce que tape l'utilisateur!

```
if (fork() == 0)
    exec (...);
signal (SIGINT, SIG_IGN);
wait (&status);
signal (SIGINT, onintr);
```

- Après le traitement d'un signal, on redémarre exactement à l'endroit de l'arrêt (par exemple si on positionne un *flag*. Si on est dans une lecture au terminal, on veut que le signal soit pris en compte tout de suite. Pour cela, un tel `read` s'arrête et fait l'erreur `EINTR`.

Exercices en TD et à la maison

- Débutants (cf dernier cours):

  `grep` parallèle, ie faire une commande `pgrep [-N] string [file]` où N est le nombre de processus forkés et *file* un fichier contenant des noms de fichiers (un par ligne) qu'on obtient en faisant par exemple `find . -type f -print`

  La sémantique de `pgrep [-N] string [file]` est:

  – soient $f_1\ f_2 \ldots f_n$ les n lignes de fichier d'entrée. On met dans N seaux équi-répartis les f_i.

  – on forke `grep string` sur chacun des seaux. Remarque: on peut prendre le `grep` du système ou écrire un `grep` perso trivial sans KMP ou Boyer-Moore avec bêtement `strcmp` et la méthode quadratique en *string* et *input*.

  – on liste toutes les lignes des fichiers contenant string,

- Chevronnés:

  Faire un minishell comprenant: & ; $<$ $>$ et l'appel des commandes standards (exec). On s'efforcera de traiter les signaux SIGINT et SIGSTOP. Cet exercice reservira plus tard pour le remote-shell.

Exercices en TD et à la maison -2-

- Experts:

  Faire une paquet de processus légers en utilisant `setjmp` et `longjmp` pour commuter. L'ordonnancement peut être sans préemption ou avec des signaux alarme. Lancer un Dekker ou Peterson.

Systèmes et réseaux

Cours 5

`Jean-Jacques.Levy@inria.fr`

Ecole polytechnique

tel: 39-63-56-89

`http://pauillac.inria.fr/~levy/`

`http://www.poly.polytechnique.fr/poly/...`

`.../~www/poly/maj2-syst-96/`

Plan

initialisations, *Typedef* en C, `lint`

variables d'environnement en shell

Les *devices* très spéciaux: mem, kmem, null

Le processeur MIPS R2000

Mémoire virtuelle sur MIPS R2000

Pages: stratégies de remplacement

Changements d'adressages

Processus résidents ou *swappés*

Allocation des pages

Structures `proc` et `user`

Mode superviseur sur MIPS R2000

Appels système et commutation de contexte

Exercices.

Les *devices* très spéciaux: mem, kmem, null

- `/dev/mem`, `/dev/kmem` représentent la mémoire physique et la mémoire avec l'adressage du noyau. Exemple (dans `ps.c`):

```
        kmem = open("/dev/kmem"; 0);
        if (kmem < 0) {
            perror(kmemf);
            exit(1);
        }
...
kget(dest, loc, size)
        char *dest;
        register unsigned loc;
        register unsigned size;
{
        lseek(kmem, (off_t)loc, 0);
        if (read(kmem, dest, size) != size)
            printf("error reading kmem at %x\n", loc);

}
```

- `/dev/null` == la poubelle

```
% mf "\mode:=$MODE; mag:=$MAG; scrollmode; input $NAME" </dev/null
```

Une architecture standard (von Neumann)

- la mémoire contient instructions et données
- les calculs se font à l'intérieur de registres de l'unité arithmétique et logique ULA (*ALU* en anglais)
- le traffic entre mémoire et registres se fait par des instructions `load` et `store`
- le compteur ordinal représente l'adresse mémoire de l'instruction en cours d'exécution (`pc` *program counter*)
- un code condition CC donne le résultat de chaque instruction
- des branchements (conditionnels en fonction du CC) font des ruptures de séquence dans l'exécution des intructions
- un des registres sert de pointeur vers une pile qui sert au contrôle des appels de procédures ou fonctions
- un mot d'état SR (*status register*) donne des informations sur le mode d'exécution (système ou utilisateur)

Une architecture standard (von Neumann) – bis

- chaque instruction a un grand nombre de modes d'adressage possibles (immédiat, indirect à travers un registre, indirect avec déplacement court, indirect avec déplacement long, idem avec post incrémentation, ...).

Un exemple d'architecture: le processeur R2000/R3000 des Dec Stations 3100 ou 5000.

```
TIT     .mips, RISC Architecture
SLA     .Gerry Kane - MIPS Computer Systems, Inc.
        .Prentice Hall, 1987. -
```

- temps d'une tâche $= C \times T \times I$ où
  $C =$ nombre de cycles par instruction,
  $T =$ temps par cycle (vitesse de l'horloge),
  $I =$ nombre d'intructions dans la tâche.

- Le temps mis par une intruction dépend donc de C et T.

- Le R2000 est une machine RISC (*Reduced Instruction Set Computer*). On veut donc $C = 1$. Pour y arriver:

  1. pipeline des instructions
  2. architecture des `load` et `store`
  3. `load` retardés
  4. branchements retardés

- les seules opérations ALU ne peuvent être que des 2 formes:

```
rd = rs op rt
rd = rs op CONSTANTE(16 bits)
```

où `rs, rt, rd` sont 32 registres. Toute instruction tient sur 32 bits; 6 bits sont réservés pour désigner l'opération (code-op).

- les seules instructions `load` et `store` font communiquer la mémoire et les registres. (On a toujours $\mathtt{r0} = 0$)
- les branchements et sauts sont de la forme

```
beq    rs, rt, OFFSET(16 bits)
jump   CONSTANTE(26 bits)
```

  où pour les branchements, l'adresse est obtenue par $\mathtt{pc} + \text{OFFSET} \times 4$, et pour les sauts par $(\mathtt{pc}\&0\text{xf}0000000) + (\text{CONSTANTE} \times 4)$.

- Il y a 32 registres. Un exemple (obtenu par `cc -S ps.c`):

```
# 238                 argc--, argv++;
   lw   $15, 72($sp)
   addu $24, $15, -1
   sw   $24, 72($sp)
   lw   $25, 76($sp)
   addu $8, $25, 4
   sw   $8, 76($sp)

# 275                 if ((tptr = index(mytty,'y')) != 0)
   la   $4, mytty
   li   $5, 121
   jal  index
   sw   $2, tptr
   lw   $10, tptr
   beq  $10, 0, $44
```

R2000/R3000

- Pipeline: 1 instruction de retard sur `load` et branchements ou sauts ...

- Ces instructions sont implantées en mémoire comme suit

```
# 238                 argc--, argv++;
0x40020c <main+44>: lw $t7,72($sp)
0x400210 <main+48>: lw $t9,76($sp)
0x400214 <main+52>: addiu $t8,$t7,-1
0x400218 <main+56>: addiu $t0,$t9,4
0x40021c <main+60>: sw $t0,76($sp)
0x400220 <main+64>: sw $t8,72($sp)
0x400224 <main+68>: lw $t1,72($sp)

# 275                 if ((tptr = index(mytty,'y')) != 0)
0x40037c <main+412>: lui $a0,4096
0x400380 <main+416>: addiu $a0,$a0,32272
0x400384 <main+420>: jal 0x405bf0 <index>
0x400388 <main+424>: li $a1,121
0x40038c <main+428>: sw $v0,-30568($gp)
0x400390 <main+432>: lw $t4,-30568($gp)
0x400394 <main+436>: nop
0x400398 <main+440>: beq $t4,$zero,0x4003b0 <main+464>
```

qu'on obtient sous `gdb` en tapant

```
(gdb) info line 238
(gdb) x/i
```

Espaces d'adressage sur ds5000

- `sys/mips/mips/vmparam.h` décrit les paramètres de la mémoire virtuelle dépendant de la ds5000

- pour l'utilisateur

```
kpr% nm -gn a.out
00400170 T __start
004001b4 T moncontrol
...
100006c0 D nl
10000b04 D _iob
10000b60 D _ctype__
100010a4 D sys_errlist
...
10010768 B rpc_createerr
10010780 B end
```

- pour le système Unix

```
kpr% nm -gn /vmunix
8000007c A kstackflag
80030000 T eprol
80030000 T start
...
80150f00 B end
c0000000 A Sysbase
c2a00000 A usrpt
c363c000 A cabase
c3640000 A kmembase
c3640000 A calimit
c4640000 A kmemlimit
c4640000 A VA_forkutl
...
ffffc000 A u
```

Adressage – bis

- On remarque d'abor que la table des pages est du matériel dans le Vax, mais, sur Mips, dans la mémoire normale de la mémoire du système `usrpt` *User Page table*. Sur Vax, comme les page sont petites (512 octets), la table des pages est à 2 niveaux. Il y a aussi des TLBs pour accélerer la traduction.

- Le système est résident en mémoire, 1.5 MO. (C'est trop) Il ne se sert pas du MMU (zone `kseg0`).

- Les processus utilisateurs ont une partie en mémoire physique et une autre sur disque. Tout processus a une partie dans l'espace mémoire réservé aux utilisateurs (0 à 0x7fffffff) et une autre dans le système (table des pages, structures `proc` et `u` dans la zone `kseg2`).

- Un processus utilisateur a sa zone `text` en `0x400000`, sa zone donnée en `10000000`, sa pile décroissant à partir de `0x7fffffff`. Au bout de la pile, il y a une page de zone rouge pour tester les débordements.

Protection des pages – Partage

- Les pages de `text` sont protégées en écriture.

- Les pages de données et de pile sont modifiables. Sur la première modification, on positionne le bit `dirty` pour cette page, sur l'exception du TLB.

- Toutes les autres pages (autres processus, système) sont inaccessibles.

- On peut partager des pages entre processus, en les faisant correspondre à la même adresse physique. Il faut toutefois faire bien attention que les instructions ou données, stockées dans la mémoire physique, peuvent contenir des adresses virtuelles qui devront être valables dans les 2 espaces d'adressage. Ce partage est le régime normal pour la partie `text` qui peut être partagé (table des pages incluse).

Adressage – ter

La structure pour les *page table entries* dépend de l'*endianité* du Mips. *Big endian* met le bit le plus significatif en tête, *little endian* le moins significatif d'abord. On remarque aussi qu'on peut désigner des champs de bits dans les structures de C.

```
struct pte
{
#ifdef MIPSEB
unsigned int pg_pfnum:20,  /* HW: core page frame number or 0 */
             pg_n:1,       /* HW: non-cacheable bit */
             pg_m:1,       /* HW: modified (dirty) bit */
             pg_v:1,       /* HW: valid bit */
             pg_g:1,       /* HW: ignore pid bit */
             :4,
             pg_swapm:1,   /* SW: page must be forced to swap */
             pg_fod:1,     /* SW: is fill on demand (=0) */
             pg_prot:2;    /* SW: access control */
#endif MIPSEB
#ifdef MIPSEL
unsigned int pg_prot:2,    /* SW: access control */
             pg_fod:1,     /* SW: is fill on demand (=0) */
             pg_swapm:1,   /* SW: page must be forced to swap */
             :4,
             pg_g:1,       /* HW: ignore pid bit */
             pg_v:1,       /* HW: valid bit */
             pg_m:1,       /* HW: modified (dirty) bit */
             pg_n:1,       /* HW: non-cacheable bit */
             pg_pfnum:20;  /* HW: core page frame number or 0 */
#endif MIPSEL
};
```

Swapping – Pagination

- le *swap* copie complètement un processus en mémoire ou le remet complètement sur disque. La partie disque qui correspond au disque de swap n'est pas un système de fichier, mais une partition disque où on écrit directement (en mode `raw`). Les zones de swap sont alloués par la structure `dmap`, où l'adresse donne la taille.

- la pagination permet d'avoir partiellement un processus en mémoire. Par example, au chargement d'un fichier lors d'`exec`, le chargement peut se faire à la demande (page par page). Le principe de la pagination est que les processus ont une certaine localité dans les références mémoire, et ne se servent pas de toute leur mémoire de manière aléatoire. Plus, l'espace de travail (*working set*) est petit, meilleure sera le comportement paginé.

- Unix-BSD utilise les 2 techniques. Le *swapper*, processus 0, indissociable du noyau, veille au nombre de pages libres qui doivent être suffisantes, au processus endormis depuis plus de 20s. Alors un processus est swappé out.

Pagination – Allocation de la mémoire physique

- Le tableau `cmap[]` (*core map*) décrit l'état des pages physiques.

- Le voleur de page (processus système 2) fait l'algorithme de remplacement de pages. Il applique dans la procédure `pageout()` l'algorithme CLOCK (`Corbato`) qui consiste à faire le tour de la mémoire de tous les processus et repérer les pages LRU. On fait le tour de la mémoire en invalidant les pages, puis on regarde au 2ème tour si les pages sont toujours invalides. Si oui, la page est marquée comme candidate à la libération. Le surcoût du à l'invalidation des pages est faible. D'autres algorithmes à plusieurs mains sont plus efficaces.

Pages: stratégies de remplacements

```
TIT     .OPERATING SYSTEMS THEORY
AUT     .Coffman, Edward G. / Denning, Peter J.
COL     .Prentice Hall series in automatic computation
DAT     .1973
```

- Soit m la taille de la mémoire, et une séquence (dans le temps) de références mémoire r_1, r_1, ..., r_n. Quelle est la meilleure stratégie de remplacement, puisqu'on peut avoir $r_i > m$?

- Exemple en FIFO, $r = 123412512345$ et $m = 3, 4$. FIFO se conduit mal avec m.

```
1 1 1 2 3 4 1 2 5
  2 2 3 4 1 2 5 3
    3 4 1 2 5 3 4

1 1 1 1 2 3 4 5 1 2
  2 2 2 3 4 5 1 2 3
    3 3 4 5 1 2 3 4
      4 5 1 2 3 4 5
```

- LRU (least recently used)

- B_0 `Belady` (largest forwardly used)

- LFU (least frequently used)

Structure `proc` de `tcsh`

```
p_link = 0x801853e0, /* linked list of running processes */
p_rlink = 0x0,
p_nxt = 0x801862a0,  /* linked list of allocated proc slots */
p_prev = 0x801856ac, /* also zombies, and free proc's */
p_addr = 0x801933d8, /* u-area kernel map address */
p_tlbpid = -1,       /* tlb context of this proc */
p_tlbinfo = 0x801940d8, /* per process tlb mappings */
p_tlbindx = 3,       /* index of next wired tlb entry to use */
p_usrpri = 57,       /* user-priority based on p_cpu and p_nice */
p_pri = 40,          /* priority, negative is high */
p_cpu = 31,          /* cpu usage for scheduling */
p_stat = 1,
p_time = 127,        /* resident time for scheduling */
p_nice = 20,         /* nice for cpu usage */
p_slptime = 127,     /* time since last block */
p_cursig = 0,
p_sig = 0,           /* signals pending to this process */
p_sigmask = 2,       /* current signal mask */
p_sigignore = 0x324004, /* signals being ignored */
p_sigcatch = 0x9882003, /* signals being caught by user */
p_flag = 0x13008201,
p_uid = 20059,       /* signals being caught by user */
p_suid = 20059,      /* saved set uid, used to direct tty signals */
p_sgid = 0,          /* saved set group id, used by setgid */
p_pgrp = 3855,       /* name of process group leader */
p_pid = 3855,        /* unique process id */
p_ppid = 3850,       /* process id of parent */
p_xstat = 0,         /* Exit status for wait */
p_progenv = 0,       /* process compatibility mode */
p_sid = 0,           /* session id (for POSIX job control) */
p_ru = 0x0,          /* mbuf holding exit information */
p_poip = 0,          /* page outs in progress */
p_tsize = 51,        /* size of text (clicks) */
p_dsize = 21,        /* size of data space (clicks) */
p_textstart = 0x400000,   /* starting address of user text */
p_datastart = 0x10000000, /* starting address of user data */
```

```
p_textpt = 1,        /* copy of text  page table size */
p_datapt = 1,        /* copy of data  page table size */
p_stakpt = 1,        /* copy of stack page table size */
p_ssize = 9,         /* copy of stack size (clicks) */
p_rssize = 9,        /* current resident set size in clicks */
p_maxrss = 3187,     /* copy of u.u_limit[MAXRSS] */
p_swrss = 0,         /* resident set size before last swap */
p_swaddr = 5856,     /* disk address of u area when swapped */
                     /* event process is awaiting */
p_wchan = 0xffffc000<Address 0xffffc000 out of bounds>,
p_textp = 0x801976e8, /* pointer to text structure */
p_textbr = 0xc2a0e000,/* text  page table base */
p_databr = 0xc2a51000,/* data  page table base */
p_stakbr = 0xc2a52000,/* stack page table base */
p_xlink = 0x801853e0, /* linked list of procs sharing same text */
p_cpticks = 0,       /* ticks of cpu time */
p_pctcpu = 0,        /* %cpu for this process during p_time */
p_fp = 0,            /* generate SIGFPE on all fp interrupts */
p_puac = 1,          /* print/don't print unaligned access msgs */
p_ndx = 28,          /* proc index for memall (because of vfork) */
p_idhash = 0,        /* hashed based on p_pid for kill+exit+... */
p_pptr = 0x8018460c, /* pointer to process structure of parent */
p_cptr = 0x80184aa8, /* pointer to youngest living child */
p_osptr = 0x0,       /* pointer to older sibling processes */
p_ysptr = 0x0,       /* pointer to younger siblings */
p_realtimer = {
it_interval = {tv_sec = 0, tv_usec = 0},
it_value = {tv_sec = 0, tv_usec = 0}},
p_quota = 0x801bd0c0, /* quotas for this process */
p_ttyp = 0x8014cab0,  /* controlling tty pointer  */
p_smsize = 0,         /* size of SM space (clicks) */
p_smbeg = 0,          /* page table offset of first SMS */
p_smend = 0,          /* page table offset of end of */
                      /*       last attached SMS. */
p_smcount = 0,        /* count of SMS attached to proc */
p_sm = 0x0}           /* shared memory related elements */
```

Structure `proc` de `tcsh`

```
(gdb) p jjl_mproc_p_tlbinfo
$4 = {
    {
    lo = {
    tl_struct = {
    tls_g = 0,       /* match any pid */
    tls_v = 1,       /* valid */
    tls_d = 1,       /* dirty (actually writeable) */
    tls_n = 0,       /* non-cacheable */
    tls_pfn = 3424}},/* physical page frame number */
    hi = {
    th_struct = {
    ths_pid = 0,
    ths_vpn = 0xffc01}}}, /* virtual page number */

    {
    lo = {
    tl_struct = {tls_g = 0, tls_v = 1, tls_d = 1,
                 tls_n = 0, tls_pfn = 2759}},
    hi = {
    th_struct = {ths_pid = 0, ths_vpn = 0xffdff}}},

    {
    lo = {
    tl_struct = {tls_g = 0, tls_v = 1, tls_d = 1,
                 tls_n = 0, tls_pfn = 2756}},
    hi = {
    th_struct = {ths_pid = 0, ths_vpn = 0xffc40}}},

    0 ptes <repeats 5 times>}
```

Structures `proc` de `tcsh`

```
(gdb) p jjl_mproc_p_textp
$5 = {
    x_free = {            /* linked list of free text structures */
    xun_freef = 0x80196f58,
    xun_freeb = 0x80197a2c},
    x_daddr = 0xc37c9c00, /* disk addresses of DMTEXT-page segments */
    x_ptdaddr = 1208,     /* disk address of page table */
    x_size = 51,          /* size (clicks) */
    x_caddr = 0x80185a54, /* ptr to linked proc, if loaded */
    x_gptr = 0x80170210,  /* gnode of prototype */
    x_rssize = 39,        /* size in clicks */
    x_swrss = 0,          /* swapped rss */
    x_count = 7,          /* reference count */
    x_ccount = 5,         /* number of loaded references */
    x_lcount = 0,         /* number of processes locking segment */
    x_poip = 0,           /* page out in progress count */
    x_flag = 34}          /* traced, written flags */
```

Structures proc de tcsh

```
(gdb) p jjl_mproc_p_textbr
$6 = {
    {
    pg_prot = 2, pg_fod = 0, pg_swapm = 1, pg_g = 0,
    pg_v = 1, pg_m = 1, pg_n = 0, pg_pfnum = 1980},
    {
    pg_prot = 2, pg_fod = 0, pg_swapm = 1, pg_g = 0,
    pg_v = 1, pg_m = 1, pg_n = 0, pg_pfnum = 1277},
    {
    pg_prot = 2, pg_fod = 0, pg_swapm = 1, pg_g = 0,
    pg_v = 1, pg_m = 1, pg_n = 0, pg_pfnum = 2270},
    {
    pg_prot = 2, pg_fod = 0, pg_swapm = 1, pg_g = 0,
    pg_v = 1, pg_m = 1, pg_n = 0, pg_pfnum = 3800},
    {
    pg_prot = 2, pg_fod = 0, pg_swapm = 1, pg_g = 0,
    pg_v = 1, pg_m = 1, pg_n = 0, pg_pfnum = 3168},
    {
    pg_prot = 2, pg_fod = 0, pg_swapm = 1, pg_g = 0,
    pg_v = 1, pg_m = 1, pg_n = 0, pg_pfnum = 1153},
    {
    pg_prot = 2, pg_fod = 0, pg_swapm = 1, pg_g = 0,
    pg_v = 1, pg_m = 1, pg_n = 0, pg_pfnum = 978},
    {
    pg_prot = 2, pg_fod = 0, pg_swapm = 1, pg_g = 0,
    pg_v = 1, pg_m = 1, pg_n = 0, pg_pfnum = 3939},
...
    {
    pg_prot = 0, pg_fod = 0, pg_swapm = 0, pg_g = 0,
    pg_v = 0, pg_m = 0, pg_n = 0, pg_pfnum = 0} <repeats 49 times>}
```

Structures `proc` de `tcsh`

```
(gdb) p jjl_mproc_p_databr
$7 = {
    {
    pg_prot = 3, pg_fod = 0, pg_swapm = 0, pg_g = 0,
    pg_v = 0, pg_m = 0, pg_n = 0, pg_pfnum = 2036},
    {
    pg_prot = 3, pg_fod = 0, pg_swapm = 0, pg_g = 0,
    pg_v = 0, pg_m = 0, pg_n = 0, pg_pfnum = 0},
    {
    pg_prot = 3, pg_fod = 0, pg_swapm = 0, pg_g = 0,
    pg_v = 0, pg_m = 0, pg_n = 0, pg_pfnum = 3201},
    {
    pg_prot = 3, pg_fod = 0, pg_swapm = 0, pg_g = 0,
    pg_v = 0, pg_m = 0, pg_n = 0, pg_pfnum = 3474},
...

    {
    pg_prot = 3, pg_fod = 0, pg_swapm = 0, pg_g = 0,
    pg_v = 0, pg_m = 0, pg_n = 0, pg_pfnum = 0} <repeats 32 times>,
    {
    pg_prot = 0, pg_fod = 0, pg_swapm = 0, pg_g = 0,
    pg_v = 0, pg_m = 0, pg_n = 0, pg_pfnum = 0} <repeats 49 times>}
(gdb) p jjl_mproc_p_stakbr
$8 = {
    {
    pg_prot = 0, pg_fod = 0, pg_swapm = 0, pg_g = 0,
    pg_v = 0, pg_m = 0, pg_n = 0, pg_pfnum = 0} <repeats 100 times>}
```

Structure `user` de `tcsh`

```
$1 = {
    u_pcb = {  /* General purpose registers saved at context switch time. */
    pcb_regs = {
    -2145890728, 257, 268516740, 2,
    13, 26, 2147473756, 0,
    -8532, 0, -2146586220, 65281,
    0 <repeats 20 times>},
    pcb_pc = 0,
    pcb_sr = 0,
    /* Misc. */
    pcb_sswap = 0x0, /* ptr for non-local resume */
    pcb_resched = 1, /* non-zero if time to resched */
    pcb_sstep = 0,   /* non-zero if single stepping */
    /* single step state info */
    pcb_ssi = {
    ssi_cnt = 0,
    ssi_bp = {
    {bp_addr = 0x0, bp_inst = 0},
    {bp_addr = 0x0, bp_inst = 0}}},
    /* These are use in branch delay instruction emulation *
    pcb_bd_epc = 0,   /* epc register */
    pcb_bd_cause = 0, /* cause register */
    pcb_bd_ra = 0,    /* address to return to if doing bd emulation */
    pcb_bd_instr = 0, /* the branch instr for the bd emulation */
    /* This is use in fp instruction emulation */
    pcb_softfp_pc = 0,/* resulting pc after fp emulation */
    /* Space for the state of all the potential coprocessors. WASTEFUL!*/
    pcb_fpregs = {
    0 <repeats 32 times>}, /* floating point */
    pcb_fpc_csr = 0,      /* floating point control and status reg */
    pcb_fpc_eir = 789,    /* floating point exception instruction reg */
    pcb_ownedfp = -2146258800, /* has owned fp at one time */
    pcb_c2regs = {0 <repeats 32 times>},  /* TBD */
    pcb_c3regs = {0 <repeats 32 times>}}, /* TBD */
```

Structure `user` de `tcsh`

```
    u_procp = 0x80184e58, /* pointer to proc structure */
    u_ar0 = 0xffffdf5c,   /* pointer to proc structure */
    u_comm = {"tcsh", '\000' <repeats 252 times>},
    /* syscall parameters, results and catches */
    u_arg = { /* arguments to current system call */
    2, 0, 2147477928, 4353624,
    0, 0, 0, 0},
    u_ap = 0xffffc350,
    u_qsave = { /* for non-local gotos on interrupts */
    val = {
    536936252, 257, 268516740, 2,
    13, 26, 2147473756, 0,
    -8428, 0, -2146578536, 536936253}},
    u_r = { /* syscall return values */
    u_rv = {
    R_val1 = 0,
    R_val2 = 4},
    r_off = 0,
    r_time = 0},
    u_error = 0 '\000',   /* return error code */
    u_eosys = 3 '\003', /* special action on end of syscall */
    u_narg = 0 '\000',  /* number of arguments in u_arg above */

/* 1.1 - processes and protection */
    u_cred = 0xc3790f00, /* user credentials (uid, gid, etc) */

/* 1.2 memory management */
    u_tsize = 51, /* text size (clicks) */
    u_dsize = 21, /* data size (clicks) */
    u_ssize = 9,  /* stack size (clicks) */
```

Structure user de tcsh

```
/* disk map for data segment */
u_dmap = {
dm_size = 168, /* current size used by process */
dm_alloc = 224,/* amount of physical swap space allocated */
dm_map = {     /* first disk block number in each chunk */
6752,
25968,
44016,
0 <repeats 45 times>}},
u_smap = {     /* disk map for stack segment */
dm_size = 72,  /* current size used by process */
dm_alloc = 96, /* amount of physical swap space allocated */
dm_map = {     /* first disk block number in each chunk */
          21800, 45136,
          0 <repeats 46 times>}},
u_cdmap = { /* shadows of u_dmap, u_smap, for use of parent during fork */
dm_size = 168,
dm_alloc = 224,
dm_map = {44240, 44272, 44336,
          0 <repeats 45 times>}},
u_csmap = {
dm_size = 72,
dm_alloc = 96,
dm_map = {45040, 45072,
          0 <repeats 46 times>}},
/* label variable for swapping */
u_ssave = { /* label variable for swapping */
val = {-2146809824, -2145890728, 5, 268490372,
       0, 26, 2147473492, 0,
       -8684, 0, -2146779520, 65341}},
u_odsize = 63, u_ossize = 2, /* for (clumsy) expansion swaps */
u_outime = 0, /* user time at last sample */
```

Structure user de tcsh

```
/* 1.3 - signal management */
    u_signal = {}, /* disposition of signals */
    u_sigmask = {  /* signals to be blocked */
    0,
    16384,
    0 <repeats 13 times>,
    1,
    0 <repeats 16 times>},
    u_sigonstack = 0, /* signals to take on sigstack */
    u_sigintr = 0,    /* signals that interrupt syscalls */
    u_oldsig = 0,     /* POSIX "old style" signals */
    u_oldmask = 524290, /* saved mask from before sigpause */
    u_code = 0,       /* ``code'' to trap */
    u_sigstack = {    /* sp & on stack state variable */
    ss_sp = 0x0,
    ss_onstack = 0},
    u_sigtramp = 0x426e58 <_atod+51800>, /* signal trampoline code */
    u_trapcause = 0,  /* cause for SIGTRAP */
    u_trapinfo = 0,   /* extra info concerning SIGTRAP */

/* 1.4 - descriptor management */
    u_ofile = {},     /* file structures for open files */
    u_pofile = {      /* per-process flags of open files */
    '\000' <repeats 15 times>,
    "\001\001\001\001\001",
    '\000' <repeats 44 times>},
    u_lastfile = 19,  /* highest numbered open file */
    u_cdir = 0x8017d314, /* current directory */
    u_rdir = 0x0,     /* root directory of current process */
    u_ttyd = 1543,    /* controlling tty dev */
    u_cmask = 18,     /* mask for file creation */
```

```
/* 1.5 - timing and statistics */
    u_ru = {          /* stats for this proc */
    ru_utime = {      /* user time used */
    tv_sec = 0, tv_usec = 187488},
    ru_stime = {      /* system time used */
    tv_sec = 0, tv_usec = 507780},
    ru_maxrss = 148,
    ru_ixrss = 8462, /* integral shared text size */
    ru_ismrss = 0,   /* integral shared memory size*/
    ru_idrss = 4825, /* integral unshared data " */
    ru_isrss = 0,    /* integral unshared stack " */
    ru_minflt = 15,  /* page reclaims */
    ru_majflt = 3,   /* page faults */
    ru_nswap = 0,    /* swaps */
    ru_inblock = 2,  /* block input operations */
    ru_oublock = 1,  /* block output operations */
    ru_msgsnd = 0,   /* messages sent */
    ru_msgrcv = 0,   /* messages received */
    ru_nsignals = 7, /* signals received */
    ru_nvcsw = 120,  /* voluntary context switches */
    ru_nivcsw = 50}, /* involuntary " */
    u_cru = {        /* sum of stats for reaped children */
    ru_utime = {
    tv_sec = 0, tv_usec = 31248},
    ru_stime = {
    tv_sec = 0, tv_usec = 320292},
    ru_maxrss = 64,
    ru_ixrss = 1505,
    ru_ismrss = 0,
    ru_idrss = 930,
    ru_isrss = 0,
    ru_minflt = 20,
    ru_majflt = 13,
    ru_nswap = 0,
    ru_inblock = 0,
    ru_oublock = 0,
    ru_msgsnd = 0,
    ru_msgrcv = 0,
    ru_nsignals = 2,
    ru_nvcsw = 27,
    ru_nivcsw = 22},
```

```
    u_timer = {
    {it_interval = {tv_sec = 0, tv_usec = 0},
     it_value = {tv_sec = 0, tv_usec = 0}},
    {it_interval = {tv_sec = 0, tv_usec = 0},
     it_value = {tv_sec = 0, tv_usec = 0}},
    {it_interval = {tv_sec = 0, tv_usec = 0},
     it_value = {tv_sec = 0, tv_usec = 0}}},
    u_tracedev = 0, /* for syscall tracer */
    u_XXX = {0, 0, 0},
    u_start = {    /* whole timeval instead of secs */
    tv_sec = 700477846, tv_usec = 72145},
    u_acflag = 2,

/* 1.6 - resource controls */

    u_rlimit = {
    {rlim_cur = 2147483647,
     rlim_max = 2147483647},
    {rlim_cur = 2147483647,
     rlim_max = 2147483647},
    {rlim_cur = 88051712,
     rlim_max = 88051712},
    {rlim_cur = 524288,
     rlim_max = 88051712},
    {rlim_cur = 2147483647,
     rlim_max = 2147483647},
    {rlim_cur = 13053952,
     rlim_max = 13053952}},
    u_quota = 0x801bd0c0,
    u_qflags = 0,
    u_prof = {
    pr_base = 0x0,
    pr_size = 0,
    pr_off = 0,
    pr_scale = 0},

/* 1.7 - System V related elements */
    u_smsize = 0,  /* size of SM space (clicks) */
    u_osmsize = 0, /* old SM size for expansion swaps */
    u_lock = 0,    /* memory locking flags (see ../h/lock.h) */
```

Structure `user` de `tcsh`

```
/* 1.9 - Namei caching */
    u_ncache = {   /* last successful directory search */
    nc_prevoffset = 4564, /* offset at which last entry found */
    nc_inumber = 16384,   /* inum of cached directory */
    nc_dev = 2062,        /* dev of cached directory */
    nc_time = 700477853}, /* not used */
    u_nd = {
    ni_dirp = 0xc373ac00<Address 0xc373ac00 out of bounds>,
    ni_nameiop = 64,
    ni_error = 0,
    ni_endoff = 0,
    ni_slcnt = 1,
    ni_pdir = 0x8016eeec,
    ni_iovec = {
    iov_base = 0x0,
    iov_len = 0},
    ni_uio = {
    uio_iov = 0x0,
    uio_iovcnt = 0,
    uio_offset = 24,
    uio_segflg = 0,
    uio_resid = 0,
    uio_flag = 0},
    ni_dent = {
    gd_ino = 2019,
    gd_reclen = 16,
    gd_namelen = 3,
    gd_name = {
    "a7\000a\000lly\000s\000kpriss",
    '\000' <repeats 239 times>}},
    ni_bufp = 0x0,
    ni_cp = 0xc373ac1a<Address 0xc373ac1a out of bounds>},
    u_stack = {
    0}}
```

Entrées dans le noyau (Rappel)

- Appels système (à partir de la librairie C) cf `libc/mips/sys/*.s`

- Exceptions et interruptions (complètement asynchrones). Par exemple, `TLB_MISS` ou interruption horloge toutes les 10ms.

- on arrive à des adresses fixes (0x800000xx) en mode maître du processeur

- on va regarder l'exemple détaillé d'un appel-système.

Appels système

1- dans `sys/mips/asm.h` et dans `libc/gen/getpid.s`

```
/*
 * SYSCALL -- standard system call sequence
 * The kernel expects arguments to be passed with the normal C calling
 * sequence.  v0 should contain the system call number.  On return from
 * the kernel mode, a3 will be 0 to indicate no error and non-zero to
 * indicate an error; if an error occurred v0 will contain an errno.
 */
#define SYSCALL(x)                                      \
LEAF(x);                                                \
        li      v0,SYS_/**/x;                           \
        syscall;                                        \
        beq     a3,zero,9f;                             \
        j       _cerror;                                \
9:

...
SYSCALL(getpid)
        RET
.end getpid
```

2- on reçoit avec des tables toutes prêtes des numéros d'appels-système et l'entrée 0x800000xx qui contient le début du programme `exception` (cf `sys/h/systm.h`, `sys/init_sysent.c`, `sys/mips/locore.s`.

```
extern struct sysent
{
        int     sy_narg;                /* total number of arguments */
        int     (*sy_call)();           /* handler */
        int     sy_mpsafe;              /* safe for multiprocessing use */
} sysent[];
...
/*
```

```
 * System call switch table.
 */
struct sysent sysent[] = {
        0, nosys, 0,            /*   0 = indir */
        1, rexit, 0,            /*   1 = exit */
...
        0, getpid, 1,           /*  20 = getpid */
};
...
VECTOR(exception, M_EXCEPT)    # Copied down to 0x80000080
        .set    noreorder
        .set    noat

        /*
         * WARNING!!!!
         * Address calculation here assumes that 16 bit offset on lw
         * do es NOT have sign bit set! Do not use PUTMSG or any other
         * message routines from here (can destroy stack).
        */
        lui     k0,kstackflag>>16
        lw      k0,+kstackflag&0xffff(k0)
        nop
        beq     k0,zero,1f
        move    k0,AT             # BDSLOT: save at
        .set    at
        .set    reorder
        sw      sp,EF_SP*4-EF_SIZE(sp)
        subu    sp,EF_SIZE
        b       2f

        /*
         * Came from user mode or utlbmiss, initialize kernel stack
         */
1:      sw      sp,KERNELSTACK-EF_SIZE+EF_SP*4
        la      sp,KERNELSTACK-EF_SIZE
        sw      gp,EF_GP*4(sp)
        la      gp,_gp
```

```
        sw      gp,kstackflag        # now on kernel stack (gp always != 0)
        /*
         * This instruction stream can be cleaned up somewhat for write stalls,
         * but for now, left as is so its readable when debugging
        */
2:      sw      k0,EF_AT*4(sp)
        sw      k1,EF_K1*4(sp)       # in case we came from utlbmiss
        sw      a0,EF_A0*4(sp)
        mfc0    a0,C0_EPC
        sw      a1,EF_A1*4(sp)
        sw      a2,EF_A2*4(sp)
        sw      a3,EF_A3*4(sp)
        mfc0    a3,C0_CAUSE
        sw      a3,EF_CAUSE*4(sp)
        sw      s0,EF_S0*4(sp)
        sw      ra,EF_RA*4(sp)
        mfc0    s0,C0_SR
        sw      a0,EF_EPC*4(sp)
        /*
         * Dispatch to appropriate exception handler
         * Register setup:
         *    s0 -- SR register at time of exception
         *    a0 -- exception frame pointer
         *    a1 -- cause code
         *    a3 -- cause register
        */
        and     a1,a3,CAUSE_EXCMASK
        lw      a2,causevec(a1)
        move    a0,sp
        .set    noreorder
        j       a2
        sw      s0,EF_SR*4(sp)        # should clear PE in s0 after here
        .set    reorder
EXPORT(eexception)
        END(exception)
```

3- on a trouvé qu'il s'agissait d'un appel-système grâce à cette table de `sys/mips/trap.c`.

```
int  (*causevec[16])() = {
        /*  0: EXC_INT */           VEC_int,
        /*  1: EXC_MOD */           VEC_tlbmod,
        /*  2: EXC_RMISS */         VEC_tlbmiss,
        /*  3: EXC_WMISS */         VEC_tlbmiss,
        /*  4: EXC_RADE */          VEC_addrerr,
        /*  5: EXC_WADE */          VEC_addrerr,
        /*  6: EXC_IBE */           VEC_ibe,
        /*  7: EXC_DBE */           VEC_dbe,
        /*  8: EXC_SYSCALL */       VEC_syscall,
        /*  9: EXC_BREAK */         VEC_breakpoint,
        /* 10: EXC_II */            VEC_trap,
        /* 11: EXC_CPU */           VEC_cpfault,
        /* 12: EXC_OV */            VEC_trap,
        /* 13: undefined */         VEC_unexp,
        /* 14: undefined */         VEC_unexp,
        /* 15: undefined */         VEC_unexp
};
```

4- on part exécuter le code des appels-système dans `sys/mips/locore.s` et dans sys/mips/trap.c.

```
VECTOR(VEC_syscall, M_SYSCALLSAVE)
#endif
        or      a1,s0,SR_IEC            # enable interrupts
        mtc0    a1,C0_SR
        sw      v0,EF_V0*4(sp)          # u_rval1
        sw      v1,EF_V1*4(sp)          # u_rval2
        move    a1,v0                   # arg2 -- syscall number
        move    a2,s0                   # arg3 -- sr
        jal     syscall                 # syscall(ef_ptr, sysnum, sr, cause)
        bne     v0,zero,full_restore    # doing a sigreturn
        lw      v0,EF_V0*4(sp)          # u_rval1
        lw      v1,EF_V1*4(sp)          # u_rval2
        mtc0    s0,C0_SR                # disable interrupts
        b       exception_exit
        END(VEC_syscall)
```

```
...
syscall(ep, code, sr, cause)
register u_int *ep;
u_int code;
{
    register regparams, nargs;
    register struct sysent *callp;
    register struct proc *p;
    register int *params;
    int opc;
    struct timeval syst;
    extern char *syscallnames[];

    cnt.v_syscall++;
    syst = u.u_ru.ru_stime;
    u.u_error = 0;
    if (!USERMODE(sr))
         panic("syscall");

    opc = ep[EF_EPC];
    ep[EF_EPC] += 4;        /* normal return executes next inst */

    XPRINTF(XPR_SYSCALL, "syscall %n upc 0x%x\n", code, syscall_values,
        opc, 0);

    regparams = EF_A0;
    if (code >= nsysent) {
        callp = &sysent[0];    /* indir (illegal) */
        ep[EF_EPC] = opc;    /* just leave pc at the syscall inst */
    }
    else {
        callp = &sysent[code];
        if (callp == sysent) {
            /*
             * indirect system call (syscall), first param is
             * sys call number
             */
```

```
            code = ep[EF_A0];
            regparams++;
            if (code >= nsysent)
                callp = &sysent[0];
            else
                callp = &sysent[code];
        }
    }
    u.u_eosys = NORMALRETURN;
    params = (int *)u.u_arg;
    for (nargs = callp->sy_narg; nargs && regparams <= EF_A3; nargs--)
        *params++ = ep[regparams++];
    if (nargs) {
        u.u_error = copyin(ep[EF_SP]+4*sizeof(int), params,
                           nargs*sizeof(int));
        if (u.u_error)
            goto done;
    }

    u.u_r.r_val1 = 0;
    u.u_r.r_val2 = ep[EF_V1];

    if (setjmp(&u.u_qsave)) {
        if (u.u_error == 0 && u.u_eosys != RESTARTSYS)
            u.u_error = EINTR;
    } else {
        (*(callp->sy_call))(u.u_arg);
    }

done:
    /*
     * a3 is returned to user 0 if indicate no errors on syscall,
     * non-zero otherwise
     */
    if (u.u_eosys == NORMALRETURN) {
        if (u.u_error) {
            ep[EF_V0] = u.u_error;
```

```
        ep[EF_A3] = 1;
    } else {
        ep[EF_V0] = u.u_r.r_val1;
        ep[EF_V1] = u.u_r.r_val2;
        ep[EF_A3] = 0;
    }
} else if (u.u_eosys == RESTARTSYS)
    ep[EF_EPC] = opc;
/* else if (u.u_eosys == FULLRESTORE) */
    /* returning from sigreturn, force full state restore */

p = u.u_procp;
if (p->p_cursig || ISSIG(p))
    psig();
p->p_pri = p->p_usrpri;
if (runrun) {
    /*
     * Since we are u.u_procp, clock will normally just change
     * our priority without moving us from one queue to another
     * (since the running process is not on a queue.)
     * If that happened after we setrq ourselves but before we
     * swtch()'ed, we might not be on the queue indicated by
     * our priority.
     */
    (void) splclock();
    setrq(p);
    u.u_ru.ru_nivcsw++;
    swtch();
}
/*
 * if single stepping this process, install breakpoints before
 * returning to user mode.  Do this here rather than in procxmt
 * so single stepping will work when signals are delivered.
 */
if (u.u_pcb.pcb_sstep)
    install_bp();
if (u.u_prof.pr_scale) {
```

```
        int ticks = times_to_ticks(&u.u_ru.ru_stime,&syst);
        if (ticks)
            addupc(ep[EF_EPC], &u.u_prof, ticks);
    }
    cpudata[0].c_curpri = p->p_pri;
    /*
     * if u_eosys == FULLRESTORE, then force full state restore
     */
    return (u.u_eosys == FULLRESTORE);
}
```

5- on fait enfin la chose demandée dans l'appel-système!
cf `sys/sys/kern_prot.c`

```
getpid()
{

        u.u_r.r_val1 = u.u_procp->p_pid;
        u.u_r.r_val2 = u.u_procp->p_ppid;
}
```

6- on repart dans `sys/mips/locore.s`

```
/*
 * End of exception processing.  Interrupts should be disabled.
 */
VECTOR(exception_exit, M_EXCEPT)
        /*
        * ENTRY CONDITIONS:
        *    Interrupts Disabled
        *    s0 contains sr at time of exception
        *
        * If we are returning to user mode, check to see if a resched is
        * desired.  If so, fake a RESCHED cause bit and let trap save/restore
        * our state for us.
        */
        and    k0,s0,SR_KUP
        beq    k0,zero,2f            # returning to kernel mode
```

```
        lw      k0,u+PCB_RESCHED
        beq     k0,zero,1f            # no resched requested
        move    a0,sp
        li      a1,SEXC_RESCHED       # software exception
        lw      a3,EF_CAUSE*4(sp)
        b       VEC_trap

1:      sw      zero,kstackflag        # switching to user stack
        lw      gp,EF_GP*4(sp)
2:      lw      a0,EF_A0*4(sp)
        lw      a1,EF_A1*4(sp)
        lw      a2,EF_A2*4(sp)
        lw      a3,EF_A3*4(sp)
        lw      s0,EF_S0*4(sp)
        lw      ra,EF_RA*4(sp)
        lw      k0,EF_EPC*4(sp)
        lw      k1,EF_SR*4(sp)
        mtc0    k1,C0_SR               # PE BIT
        .set    noreorder
        .set    noat
        lw      AT,EF_AT*4(sp)
        lw      sp,EF_SP*4(sp)
        j       k0
        c0      C0_RFE
        .set    at
        .set    reorder
        END(exception_exit)
```

7- Si l'appel système a plus de 4 arguments, les arguments supplémentaires sont comme en C passés dans la pile (utilisateur). Or on est maintenant en mode maître du processeur avec une autre pile. Il faut donc recopier des choses de l'espace utilisateur vers l'espace noyau. C'est fait par `copyin()` de `sys/mips/usercopy.s`. Cette procédure est incroyablement compliquée, car elle peut avoir à copier de gros objets et est super optimisée.

```
/*
 * copyin(user_src, kernel_dst, bcount)
 */
COPYIOFRM=    (4*4)+4             # 4 arg saves plus ra
NESTED(copyin, COPYIOFRM, zero)
        subu    sp,COPYIOFRM
        sw      ra,COPYIOFRM-4(sp)
        bltz    a0,cerror
        .set    noreorder
        li      v0,NF_COPYIO
        sw      v0,nofault        # branch delay slot
        jal     bcopy
        nop
        sw      zero,nofault
        .set    reorder
        move    v0,zero
        lw      ra,COPYIOFRM-4(sp)
        addu    sp,COPYIOFRM
        j       ra
        END(copyin)

/*
 * copyout(kernel_src, user_dst, bcount)
 */
NESTED(copyout, COPYIOFRM, zero)
        subu    sp,COPYIOFRM
        sw      ra,COPYIOFRM-4(sp)
        bltz    a1,cerror
        .set    noreorder
        li      v0,NF_COPYIO
        sw      v0,nofault        # branch delay slot
        jal     bcopy
        nop
        sw      zero,nofault
        .set    reorder
        move    v0,zero
        lw      ra,COPYIOFRM-4(sp)
```

```
        addu    sp,COPYIOFRM
        j       ra
        END(copyout)

NESTED(cerror, COPYIOFRM, zero)
        li      v0,EFAULT
        lw      ra,COPYIOFRM-4(sp)
        addu    sp,COPYIOFRM
        j       ra
        END(cerror)

/*
 * bcopy(src, dst, bcount)
 *
 * NOTE: the optimal copy here is somewhat different than for the user-level
 * equivalents (bcopy in 4.2, memcpy in V), because:
 * 1) it frequently acts on uncached data, especially since copying from
 * (uncached) disk buffers into user pgms is high runner.
 * This means one must be careful with lwl/lwr/lb - don't expect cache help.
 * 2) the distribution of usage is very different: there are a large number
 * of bcopies for small, aligned structures (like for ioctl, for example),
 * a reasonable number of randomly-sized copies for user I/O, and many
 * bcopies of large (page-size) blocks for stdio; the latter must be
 * well-tuned, hence the use of 32-byte loops.
 * 3) this is much more frequently-used code inside the kernel than outside
 *
 * Overall copy-loop speeds, by amount of loop-unrolling: assumptions:
 * a) low icache miss rate (this code gets used a bunch)
 * b) large transfers, especially, will be word-alignable.
 * c) Copying speeds (steady state, 0% I-cache-miss, 100% D-cache Miss):
 * d) 100% D-Cache Miss (but cacheable, so that lwl/lwr/lb work well)
 *      Config  Bytes/  Cycles/ Speed (VAX/780 = 1)
 *              Loop    Word
 *      08V111  35      0.71X   (8MHz, BUS, 1-Deep WB, 1-way ILV)
 *              4       15      1.67X
 *              8/16    13.5    1.85X
 *              32/up   13.25   1.89X
```

```
 *      08MM44  1       26      0.96X   (8MHz, MEM, 4-Deep WB, 4-way ILV)
 *              4       9       2.78X
 *              8       7.5     3.33X
 *              16      6.75    3.70X
 *              32      6.375   3.92X   (diminishing returns thereafter)
 *
 * MINCOPY is minimum number of byte that its worthwhile to try and
 * align copy into word transactions.  Calculations below are for 8 bytes:
 * Estimating MINCOPY (C = Cacheable, NC = Noncacheable):
 * Assumes 100% D-cache miss on first reference, then 0% (100%) for C (NC):
 * (Warning: these are gross numbers, and the code has changed slightly):
 *      Case            08V11                   08M44
 *      MINCOPY         C       NC              C       NC
 *      9 (1 byte loop) 75      133             57      93
 *      8 (complex logic)
 *      Aligned         51      51              40      40
 *      Alignable,
 *      worst (1+4+3)   69      96              53      80
 *      Unalignable     66      93              60      72
 * MINCOPY should be lower for lower cache miss rates, lower cache miss
 * penalties, better alignment properties, or if src and dst alias in
 * cache. For this particular case, it seems very important to minimize the
 * number of lb/sb pairs: a) frequent non-cacheable references are used,
 * b) when i-cache miss rate approaches zero, even the 4-deep WB can't
 * put successive sb's together in any useful way, so few references are saved.
 * To summarize, even as low as 8 bytes, avoiding the single-byte loop seems
 * worthwhile; some assumptions are probably optimistic, so there is not quite
 * as much disadvantage.  However, the optimal number is almost certainly in
 * the range 7-12.
 *
 *      a0      src addr
 *      a1      dst addr
 *      a2      length remaining
 */
#define MINCOPY 8

LEAF(bcopy)
```

```
        xor     v0,a0,a1            # bash src & dst for align chk; BDSLOT
        blt     a2,MINCOPY,bytecopy # too short, just byte copy
        and     v0,NBPW-1           # low-order bits for align chk
        subu    v1,zero,a0          # -src; BDSLOT
        bne     v0,zero,unaligncopy # src and dst not alignable
/*
 * src and dst can be simultaneously word aligned
 */
        and     v1,NBPW-1           # number of bytes til aligned
        subu    a2,v1               # bcount -= alignment
        beq     v1,zero,blkcopy     # already aligned
        lwr     v0,0(a0)
        swr     v0,0(a1)
        addu    a0,v1               # src += alignment
        addu    a1,v1               # dst += alignment

/*
 * 32 byte block, aligned copy loop (for big reads/writes)
 */
blkcopy:
        and      a3,a2,~31         # total space in 32 byte chunks
        subu     a2,a3             # count after by-32 byte loop done
        beq      a3,zero,wordcopy  # less than 32 bytes to copy
        addu     a3,a0             # source endpoint
1:      lw       v0,0(a0)
        lw       v1,4(a0)
        lw       t0,8(a0)
        lw       t1,12(a0)
        sw       v0,0(a1)
        sw       v1,4(a1)
        sw       t0,8(a1)
        sw       t1,12(a1)
        addu     a0,32             # src+= 32; here to ease loop end
        lw       v0,-16(a0)
        lw       v1,-12(a0)
        lw       t0,-8(a0)
        lw       t1,-4(a0)
```

```
        sw      v0,16(a1)
        sw      v1,20(a1)
        sw      t0,24(a1)
        sw      t1,28(a1)
        addu    a1,32             # dst+= 32; fills BD slot
        bne     a0,a3,1b

/*
 * word copy loop
 */
wordcopy:
        and     a3,a2,~(NBPW-1)   # word chunks
        subu    a2,a3             # count after by word loop
        beq     a3,zero,bytecopy  # less than a word to copy
        addu    a3,a0             # source endpoint
1:      lw      v0,0(a0)
        addu    a0,NBPW
        sw      v0,0(a1)
        addu    a1,NBPW           # dst += 4; BD slot
        bne     a0,a3,1b
        b       bytecopy

/*
 * deal with simultaneously unalignable copy by aligning dst
 */
unaligncopy:
        subu    a3,zero,a1         # calc byte cnt to get dst aligned
        and     a3,NBPW-1          # alignment = 0..3
        subu    a2,a3              # bcount -= alignment
        beq     a3,zero,partaligncopy     # already aligned
        lwr     v0,0(a0)          # get whole word
        lwl     v0,3(a0)          # for sure
        swr     v0,0(a1)          # store right piece (1-3 bytes)
        addu    a0,a3             # src += alignment (will fill LD slot)
        addu    a1,a3             # dst += alignment

/*
```

```
 * src unaligned, dst aligned loop
 * NOTE: if MINCOPY >= 7, will always do 1 loop iteration or more
 * if we get here at all
 */
partaligncopy:
        and     a3,a2,~(NBPW-1)    # space in word chunks
        subu    a2,a3              # count after by word loop
        addu    a3,a0              # source endpoint
1:
        lwr     v0,0(a0)
        lwl     v0,3(a0)
        addu    a0,NBPW
        sw      v0,0(a1)
        addu    a1,NBPW
        bne     a0,a3,1b


/*
 * brute force byte copy loop, for bcount < MINCOPY + tail of unaligned dst
 * note that lwl, lwr, swr CANNOT be used for tail, since the lwr might
 * cross page boundary and give spurious address exception
 */
bytecopy:
        addu    a3,a2,a0          # source endpoint; BDSLOT
        ble     a2,zero,copydone  # nothing left to copy, or bad length
1:      lb      v0,0(a0)
        addu    a0,1
        sb      v0,0(a1)
        addu    a1,1              # BDSLOT: incr dst address
        bne     a0,a3,1b
copydone:
        j     ra
        END(bcopy)
```

Appels système – Quelques remarques

- `getpid()` est la caricature de l'appel système. Pour 2 instructions, on a un *overhead* incroyable. Il faut toutefois retenir que ca coûte de faire des tours dans le noyau.

- A la fin d'un appel système, on regarde si on un signal attend. Si oui, on exécute le *handler* du signal.

```
if (issig())
    psig();
```

- A la fin de l'appel système on regarde ensuite si le drapeau `runrun` est positionné. Ceci signifie que le *scheduler* réclame qu'il y ait une commutation de processus.

Commutation de processus

- Le système Unix a son noyau non préemptif. Jamais un appel-système ne peut être empêché d'aller au bout. Il peut s'arrêter par `sleep()` car une ressource dont il a besoin est absente. Il peut abandonner le processeur, par `swtch()`, car il voit le drapeau `runrun` positionné en fin d'appel-système.

- Avec cette seule philosophie, jamais un processus ne faisant d'appels-système n'abandonnerait le cpu. Donc, en fin d'interruption (horloge), dans `hardclock()` de `sys/sys/kern_clock`, le drapeau `runrun` est mis en appelant `setpri()` de `sys/sys/kern_synch`. Et la procédure `trap()` de `sys/mips/trap.c` — qui correspond à `syscall()` — fait la commutation si le processus interrompu est en mode utilisateur et si `runrun` est positionné, en fin de traitement de l'exception.

- La non préemption des appels-système simplifie grandement l'écriture du noyau. Il faut donc se garder uniquement contre les interruptions possibles par les *software priority levels* `splclock()`, `spltty()`, ... définies dans `sys/mips/locore.s`.

Commutation de processus – bis

- L'argument précédent n'est plus vrai sur un multi processeur! Cf Unix sur Sequent, Encore, Alliant.

- Une interruption fait en général très peu, i.e. mettre qques drapeaux ou qques changements légers dans les structures de données, et s'empresse de rendre la main.

Signaux

- La présence d'un signal est testée, en fin de traitement d'exception en mode utilisateur, ou en fin d'appel système par `issig()`.

- La présence d'un signal est aussi testée dans la procédure d'abandon de cpu `sleep()` en fonction de la priorité d'endormissement (2ème argument de `sleep()`).

- Les signaux sont traités en créant au sommet de la pile utilisateur un nouveau *frame* par `sendsig()` de `sys/mips/mips/machdep.c`

- Le retour du signal se fait en revenant par `sigreturn()`.

- L'envoi d'un signal se fait par `kill()` et `psignal()`. Remarquons que `kill()` parcourt la table des `proc` et positionne la présence du signal. La partie "signal" d'un processus doit se trouver dans la structure `proc` et non dans la structure `u`.

Interruptions horloge

- Des alarmes `callout` sont mises en attente des interruptions horloge.

- Les `callout` sont mis en ordre croissant le délai de réveil et chacun stocke le temps incrémental. A chaque *tick* d'horloge, on diminue le temps du premier, et on teste s'il est négatif ou nul.

- On peut manquer une IT horloge, ou en reprendre une pendant son traitement. C'est pourquoi il faut ausi tester si le premier réveil est négatif.

```
/*
 * The callout structure is for
 * a routine arranging
 * to be called by the clock interrupt
 * (clock.c) with a specified argument,
 * in a specified amount of time.
 * Used, for example, to time tab
 * delays on typewriters.
 */

struct  callout {
    int     c_time;             /* incremental time */
    caddr_t c_arg;              /* argument to routine */
    int     (*c_func)();        /* routine */
    struct  callout *c_next;
};
```

Scheduling

- la priorité `p_usrpri` d'un processus descend de 1 tous les 4 *ticks* ($4 \times 1/60$s). La priorité ne peut être plus grande que 127.

$$\texttt{p\_usrpri} = PUSER + \lceil \frac{\texttt{p\_cpu}}{4} \rceil + 2\texttt{p\_nice}$$

- un amortissement est fait par le filtre

$$\texttt{p\_cpu} = \frac{2\text{load}}{2\text{load} + 1}\texttt{p\_cpu} + \texttt{p\_nice}$$

- le *scheduler* `schedcpu()` de `sys/sys/kern_synch.c` recalcule la priorité toutes les secondes, grâce à un réveil

```
timeout(schedcpu, (caddr_t)0, hz);
```

Ordonnancement des processus – *Sceduling*

On a un ensemble de processeurs $P = \{P_1, P_2, \ldots P_m\}$ et un ensemble de ressources $\mathsf{R} = \{R_1, R_2, \ldots R_s\}$. Un système de tâches est donné par un quadruplet $\langle \mathsf{F}, <, [\tau_{ij}], \{\mathsf{R}_i\}, \{w_j\} \rangle$ où:

1. $\mathsf{F} = \{T_1, T_2, \ldots, T_n\}$ ensemble des tâches,

2. $<$ est une ordre partiel entre les tâches (dépendance),

3. $[\tau_{ij}]$ est une matrice $m \times n$ des temps d'exécution. Si tous les processeurs sont identiques, on écrira τ_i,

4. w_i coûts ou priorités,

Soient s_i et f_i les temps de début et de fin de T_i, soit S un ordonnancement particulier. On a plusieurs coûts possibles:

$$\omega(S) = \max_{1 \leq i \leq n} \{f_i(S)\}$$

ou

$$\omega'(S) = \frac{1}{n} \sum_{i=1}^{n} w_i f_i(S)$$

```
TIT     .COMPUTER & JOB/SHOP SCHEDULING THEORY
AUT     .Coffman, Edward G.
COL     .John Wiley & Sons, 1976
```

Créations de processus – Fin de processus

- `fork()` dans `sys/sys/kern_fork.c`, qui appelle `newproc()`. Tout est copié (y compris la structure `u`). Quelques ajustement sont faits dans la structure `proc`.

- `exit()` dans `sys/sys/kern_exit.c` relâce les ressources, sauve la structure `u` pour que le parent puisse la consulter, entre dans l'état ZOMBIE, réveille le parent et réarrange les processus fils.

.

Etats d'un processus – proc.h

```
/* stat codes */
#define SSLEEP  1               /* awaiting an event */
#define SWAIT   2               /* (abandoned state) */
#define SRUN    3               /* running */
#define SIDL    4               /* intermediate state in process creation */
#define SZOMB   5               /* intermediate state in process termination */
#define SSTOP   6               /* process being traced */

/* flag codes */
#define SLOAD   0x0000001       /* in core */
#define SSYS    0x0000002       /* swapper or pager process */
#define SLOCK   0x0000004       /* process being swapped out */
#define SSWAP   0x0000008       /* save area flag */
#define STRC    0x0000010       /* process is being traced */
#define SWTED   0x0000020       /* stpd proc not been waited for by parent */
#define SULOCK  0x0000040       /* user settable lock in core */
#define SPAGE   0x0000080       /* process in page wait state */
#define SKEEP   0x0000100       /* another flag to prevent swap out */
#define SOMASK  0x0000200       /* restore old mask after taking signal */
#define SWEXIT  0x0000400       /* working on exiting */
#define SPHYSIO 0x0000800       /* doing physical i/o (bio.c) */
#define SVFORK  0x0001000       /* process resulted from vfork() */
#define SVFDONE 0x0002000       /* another vfork flag */
#define SNOVM   0x0004000       /* no vm, parent in a vfork() */
#define SPAGI   0x0008000       /* init data space on demand, from inode */
#define SSEQL   0x0010000       /* user warned of sequential vm behavior */
#define SUANOM  0x0020000       /* user warned of random vm behavior */
#define STIMO   0x0040000       /* timing out during sleep */
/* was SDETACH */
#define SNOCLDSTP       0x0100000       /* POSIX child stop flag */
/* was SOUSIG */
#define SOWEUPC 0x0200000       /* owe process an addupc() call at next ast */
#define SSEL    0x0400000       /* selecting; wakeup/waiting danger */
#define SLOGIN  0x0800000       /* a login process (legit child of init) */
#define SPTECHG 0x1000000       /* pt\'es for process have changed */
```

Synchronisation dans le noyau

- `sleep(chan, pri)` a 2 arguments. Le 1er argument est une adresse mémoire quelconque, par exemple

  ```
  sleep((caddr_t)\&runin, PSWP);
  ```

  pour attendre avec une forte priorité qu'un *swap in* soit possible. Si la priorité est faible, on teste les signaux avant de s'endormir.

- si on part sur un signal dans un `sleep()` faible, on retourne grâce à un *goto* non local `longjmp(&u.u_qsave)` vers la fin d'une entrée/sortie, ou vers la fin d'un `wait()`.

- l'abandon du cpu se fait en appelant `swtch()`.

- `wakeup(chan)` réveille les processus endormis sur ce canal. Le réveil est d'autant plus fort que la priorité est forte (i.e. plus proche de 0). La priorité est fixée par le 2ème argument de `sleep()`. Si la priorité de réveil est faible, on teste les signaux.

- `swtch()` se fait dans `sys/mips/mips/swtch.c` qui appelle 2 procédures `save()` et `resume()` de `locore.s` qui agissent comme des *goto* non locaux.

Exercices en TD et à la maison

Finir les exercices de la dernière fois.

Systèmes et réseaux

Cours 6

Jean-Jacques.Levy@inria.fr
Ecole polytechnique

tel: 39-63-56-89
http://pauillac.inria.fr/∼levy/

http://www.poly.polytechnique.fr/poly/...
.../∼www/poly/maj2-syst-96/

Plan

Typedef en C, `lint`

Mémoire partagée

Sémaphores

Synchronisation par le système de fichier (*lock*)

Pipes

Langages à flux de données

Séquentialité et *select*

Exercices.

Typedef en C – `lint`

- `malloc` permet d'allouer des objets dynamiques dans le tas (le *heap* de Pascal).

- `typedef` autorise de nouveaux types

```
typedef int Length;

Lenth x, y[30];
```

- qui peuvent être récursifs

```
typedef struct tnode *Treeptr;

typedef struct tnode {
    char *word;
    int count;
    Treeptr left;
    Treeptr right;
} Treenode;
```

- Pb: comment faire un arbre sans les déclarations de pointeurs précédentes?

- `lint` est un programme qui permet de faire des vérifications intermodules en C. Bien utile parfois pour trouver des erreurs de type. En principe, moins nécessaire avec C ANSI.

Communication entre processus par mémoire partagée

- **`shmget`**, **`shmat`** opérations Système V.

- Une mémoire partagée 2 fois dans un même processus

```
int shmid;

main()
{
    int i, *pint;
    char *addr1, *addr2;
    extern char *shmat();

    shmid = shmget(75, 128*1024, 0777|IPC_CREAT);
    addr1 = shmat(shmid, 0, 0);
    addr2 = shmat(shmid, 0, 0);
    printf ("addr1 0x%x, addr2 0x%x\n", addr1, addr2);

    pint = (int *) addr1;
    for (i = 0; i < 256; i++)
        *pint++ = i;
    *pint = 256;

    pint = (int *) addr2;
    for (i = 0; i < 256; i++)
        print ("index %d, value %d\n", i, *pint++);

    pause();
}
```

Communication entre processus par mémoire partagée

- `shmget`, `shmat` opérations Système V.
- Une mémoire partagée 2 fois dans un même processus

```
int shmid;

main()
{
    int i, *pint;
    char *addr;
    extern char *shmat();

    shmid = shmget(75, 64*1024, 0777);
    addr = shmat(shmid, 0, 0)

    pint= (int *) addr;
    while (*pint == 0)
        ;
    for (i = 0; i < 256; i++)
        printf ("%d\n", *pint++):
}
```

Duplication de *file descriptors*

- Si on écrit un *shell*, on veut lancer des nouveaux programmes avec `exec` et avec l'entrée/sortie standard prédéfinie (à cause de $<$ ou $>$). Or les *file descriptors* traversent `exec`. D'où

```
fd0 = open ("fichier1", 1);
fd1 = open ("fichier2", 2);
fd3 = open ("fichier3", 2);
if ((pid = fork()) == 0) {
    close (0); dup (fd1);
    close (1); dup (fd2);
    close (2); dup (fd3);
    execlp ("prog", ...);
    exit (1);
}
```

- `dup()` duplique le *file descriptor* sur le premier disponible. On peut aussi utiliser `dup2()`.

- Réciproquement, on veut pouvoir remettre au terminal les *file descriptors* standards avec

```
fd0 = open ("/dev/tty", 1);
fd1 = open ("/dev/tty", 2);
fd3 = open ("/dev/tty", 2);
if ((pid = fork()) == 0) {
    close (0); dup (fd1);
    close (1); dup (fd2);
    close (2); dup (fd3);
    execlp ("prog", ...);
    exit (1);
}
```

Pipes

- `pipe()` crée un tampon que l'on peut écrire d'un côté et lire de l'autre de manière asynchrone. Il renvoie 2 *file descriptors* pour lire et écrire.

```
char buf[1024];
int fds[2];

strcpy (buf, "Coucou!");
pipe(fds);
for (;;) {
    write (fds[1], buf, strlen ("Coucou!"));
    read (fds[0], buf, 1024);
}
```

- En général, un *pipe* marche avec un `fork()`

```
pipe (fds);
if ((pid1 = fork()) == 0) {
    close (1); dup (fds[1]);
    ...
}
if ((pid2 = fork()) == 0) {
    close (0); dup (fds[0]);
    ...
}
```

Pipes – 2

- Les shells ne permettent que les *pipes* linéaires. En System V, *pipes* nommés. En C, graphes arbitraires de *pipes*: crible d'Erathosthène, générer les nombres de la forme $2^a3^b5^c$ sans omission, ni répétition (cf les programmes *dataflow* de Kahn et MacQueen, ou les langages LUSTRE de Halbwachs, LUCID de Ashcroft).

- `fopen()` et `fclose()` permettent de prendre l'entrée ou la sortie standard d'une commande sur un *pipe*.

```
FILE *fopen(
        const char *path,
        const char *mode);
```

Canaux de communication (CCS, CSP)

- les canaux de communication ont une longueur de 1.
- il faut des “rendez vous” entre émetteur et récepteur.
- plus simple à implanter (Transputer).
- très belle théorie CSP, CCS, π-calcul, OCCAM.
- version asynchrone avec des queues sur les canaux.


```
TIT     .Communication and concurrency
SLA     .Robin Milner. - New York ; London ; Toronto : Prentice Hall,
        .1989. - XI-260 p. ; 24 cm. - (Prentice Hall international
        .series in computer science).

TIT     .Communicating sequential processes
SLA     .C.A.R. Hoare. - Englewood Cliffs, NJ ; London ;
        .Mexico : Prentice-Hall, 1985. - VIII-256 p. ;
        .24 cm. - (Prentice-Hall international series
        .in computer science).

TIT     .Algebraic theory of processes
SLA     .Matthew Hennessy. - MIT Press, 1988.
```

Déterministes – non déterminisme

- Déterminisme $\Rightarrow$ un seul résultat. Majorité des programmes.

```
while ((c = getchar ()) != EOF) {
...
}
```

- Si on veut attendre à la fois un événement souris ou clavier, on veut une attente multiplexée, i.e.

```
while (((c = getchar ()) != EOF)
     || ((e = MouseEvent() != NULL))) {
...
}
```

  impossible car `getchar()` est bloquant, et comme on évalue alors de droite à gauche, on loupera les événements souris.

- Une solution est d'avoir des entrées/sorties non bloquantes (en fait possibles en Unix), et on fait de l'attente active. Mais on gâche des cycles machine. ($\sim$ la fonction `Button` de MacPascal).

- Une autre solution est d'avoir dans le noyau un *driver* d'événements qui multiplexe (et sérialise) les événements clavier-souris. C'est ce qu'ont la plupart des Unix qui veulent s'interfacer efficacement à un système de fenêtres. Mais c'est théoriquement insatisfaisant (changer le noyau), quoique très efficace en pratique.

Déterministes – non déterminisme

- Remarque: on peut avoir parallélisme et déterminisme: réseaux de Kahn-MacQueen, systèmes purement fonctionnels (confluence).

  `Kahn G.`
  `The semantics of a Simple Language for Parallel Programming,`
  `Proc. of IFIP Congress 74, Amsterdam, North Holland, 1974.`

- Le non déterminisme est introduit par `fork()`.

```
if (fork() == 0) {
    while ((c = getchar()) != EOF) {
        ...
    } else
    while ((e = MouseEvent() != NULL)) {
        ...
    }
```

- La solution `fork()` marche bien (en supposant mémoire partagée), sauf si les événements arrivent en rafale. Alors les lectures se battent avec la communication de contexte des processus.

Primitive non séquentielle

- Un appel-système nouveau `select()`.

- `select()` spécifie le nombre d'événements en attente dans les *file descriptors* décrits par les 3 masques. Un délai maximal peut être donné.

```
nfound = select(nfds, readfds, writefds, execptfds, timeout)

          fd_set *readfds,
          fd_set *writefds,
          fd_set *exceptfds,
          struct timeval *timeout) ;
```

  `select()` regarde l'état des descripteurs fd_1, fd_2, $\ldots fd_n$, et indique si les lectures ou écritures sur ces descripteurs bloquent. Ainsi, on peut savoir s'il y a des caractères disponibles en *input* ou si une *output* est bloquée. Les descripteurs sont représentés par la fonction caractéristique de leur ensemble représentée par un tableau de bits. `nfds` est le nombre de bits de cet ensemble (donc le maximum (+1) de ces *file descriptors*).

- Un bon exemple est le multiplexeur qui attend sur 2 canaux.

Non déterminisme dans d'autres langages.

- En CCS, CSP ou OCCAM, il y a des gardes et on peut attendre sur plusieurs simultanément. Idem pour Esterel (`Berry`) et Squeak (`Pike & Cardelli`).

```
Berry G., Couronne ' P., Gonthier G..
Synchronous programming of reactive systems:
an introduction to Esterel.
In K. Fuchi and M. Nivat, editors,
Programming of Future Generation Computers,
Elsevier Science Publisher B.V. (North Holland), 1988.
INRIA Report 647.

Pike R., Cardelli L.,
Squeak: a Language for Communicating with Mice.
Proc of the ACM-SIGGRAPH Conference. 1985(?)
```

- Certains systèmes autorisent des processus légers (Mach, Topaz, Chorus). Il y a donc un *minifork()* qui ne copie que les registres et pile (et pas tout l'espace `data` qui reste partagé. Alors `select()` devient moins utile, et on programme plus naturellement avec `minifork()` et entrées/sorties bloquantes, sans avoir les inconvénients de la commutation lourde entre processus et de la lourdeur de la communication.

Exercices en TD et à la maison

- Débutants: manipuler quelques *pipes* pour un calcul à la Lustre, sauf le crible d'Erathosthène qui demande trop de processus.

- Chevronnés 1: ajouter les pipes au shell et le traitement des signaux.

- Chevronnés 2: faire la commande *cpustate* qui affiche constamment la charge du cpu. Il faut lire la mémoire à l'adresse de la bonne variable, qu'on obtient en regardant la table des symboles de `/vmunix`. Utiliser *nlist()*.

Systèmes et réseaux

Cours 7

Jean-Jacques.Levy@inria.fr
Ecole polytechnique

tel: 39-63-56-89
http://pauillac.inria.fr/∼levy/

http://www.poly.polytechnique.fr/poly/...
.../∼www/poly/maj2-syst-96/

Réseaux

- le modèle ISO à 7 couches (1: Physique, 2: Liaison, 3: Réseau, 4: Transport, 5: Session, 6: Présentation, 7: Application).

- c'est une approche modulaire qui rend indépendants les programmes traitant des diverses couches.

- plus simple de considérer un modèle simplifié:

  1. Liaison: protocoles physiques, *ethernet, token ring, SLIP*, circuits (longue distance) ou à paquets (réseaux locaux).

  2. Réseau: IP, IDP, ICMP, ARP, RARP.

  3. Transport: UDP, TCP, PEX.

  4. Application: SMTP, TFTP, FTP, TELNET.

Le modèle client – serveur

- Un serveur existe à une certaine “adresse” sur le réseau. Des clients peuvent lui émettre des requêtes.

- Exemples de serveurs: X-windowsX, `rshd`, `rlogind`, `timed`, `mountd`, `nfsd`, `snmpd`, `inetd`, ...

- Un serveur bien utile est le serveur de noms, DNS/BIND en Internet; cf. Grapevine (Alto/Dorado, Xerox Parc)

- La communication avec les serveurs peut se faire à des niveaux d'abstraction plus haut: appel de procédures distantes (*remote procedure calls, RPC*). Cela pose un problème de représentation des données persistantes pour passer les arguments.

- Cela permet de résoudre les problèmes de synchronisation sans modifier le noyau d'Unix

- Ne marche bien que si on utilise `select()` pour attendre les nouveaux clients, si leur nombre est inconnu à l'avance

- Et donc autre méthode si on a les *processus légers*

Autre exemple: *X-windows*, MIT, Projet Athena,
Bob Scheifler, Jim Gettys, 1986

- Le serveur gère l'écran et les événements clavier-souris.

- Les clients sont compilés avec une librairie `Xlib` qui transforme les ordres graphiques ou lectures d'événements en écritures ou lectures sur une connexion par `socket`

- Le système a été relu et corrigé par la communauté académique et est devenu un *standard* gratuit

- Il est compliqué car protocole élémentaire. Il y a au dessus des Toolkits encore non standard: DecWindows, HpToolkit, Interviews, Andrew, OSF-Motif

- Pour qu'il fonctionne, le noyau Unix doit fournir un *driver* d'événements.

- Le serveur envoie au client des événements `Expose` pour recalculer des rectangles. On minimise les *bitmaps* cachés.


```
TIT     .X WINDOW system : C library and protocol reference
SLA     .Robert W. Scheifler, James Gettys, Ron Newman. - [s.l.] :
        .Digital Press, 1988. - XXIX-701 p. ; 24 cm.

TIT     .X WINDOW: Applications programming
SLA     .Eric F. Johnson, Kevin Reichard
        . MIS press, 1989.
```

Un exemple: NFS (*Network File Systems*) `Sun Microsystems, 1984`

- les commandes `/etc/mount` ou `df` permettent de voir sur `cubitus` des lignes:

  ```
  poly.polytechnique.fr:/usr/users/cie1   196895  157405   19801    89%
  /usr/users/cie1
  ...
  ```

  Le sous-arbre `/usr/users/cie1` est en fait le sous-arbre `/usr/users/cie1` de la machine `poly.polytechnique.fr` lui même monté sur le *device* `/dev/ra4d`.

- `read()`, `write()` sont transformés en requêtes au serveur `nfsd` de `poly`. En fait les clients sont 4 processus systèmes `biod` du noyau de `cubitus`.

- `/etc/exports` donne la liste des droits de montage de fichiers, côté serveur. `showmounts` montre les clients actifs.

- NFS est un système sans état. Donc non transitif. Il résiste donc bien aux pannes d'un serveur (modulo un risque de non fiabilité).

Client - Serveur sur Internet

- les adresses IP sont des entiers 32 bits écrits en décimal par octets. Exemple: 192.48.98.1 pour C0306201. Les adresses sont de 4 classes (A $\leq$ 0x70000000, B $\geq$ 0x80000000, C $\geq$ 0xC0000000, D $\geq$ 0xE0000000). Elles se trouvent dans `/etc/hosts` où on trouve aussi leur nom symbolique.

- IP travaille avec des numéros de ports et fournit des *checksums* inconnus au niveau physique. Un port sert à adresser un serveur ou un processus client. Une connexion est caractérisée par un quintuplet:

  $\langle$ protocole, adresse$_1$, port$_1$, adresse$_2$, port$_2$ $\rangle$.

  Par exemple:

  $\langle tcp, 128.10.0.3, 1500, 128.10.0.7, 21\rangle$

- ports bien-connus: FTP = 21 en tcp, TFTP = 69 en udp... (cf `/etc/services`)

- ports éphémères: au delà de 1024 pour les clients.

- 2 modes de transports: datagrammes (UDP), mode connecté (TCP)

Adresses des `sockets`

- domaine `AF_UNIX`: nom de fichier.

```
struct  sockaddr_un {
        short   sun_family;
        char    sun_path[108-4];
#endif
};
```

- domaine `AF_INET`: numéro internet 192.48.98.1 ou poly.polytechnique.fr. Les fonctions `gethostbyaddr` ou `gethostbyname` donnent l'adresse.

```
struct in_addr {
        union {
                struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;
                struct { u_short s_w1,s_w2; } S_un_w;
                u_long S_addr;
        } S_un;

struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};
```

- les adresses peuvent être de longueur variables. Pour ne pas toujours réserver la taille maximale, on indique leur longueur par un argument supplémentaire dans les fonctions aux quelles on passe une adresse.

Adresses des `sockets`

- La commande `netstat -a` permet de voir les services courants. Exemple: 6000 pour X-windows, ...

```
kpr% date; netstat -a |head -10
Sun Feb 23 17:27:44 MET 1992
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  kpriss.inria.fr.1022   margaux.inria.fr.login ESTABLISHED
tcp        0      0  kpriss.inria.fr.6000   toul.inria.fr.1503     ESTABLISHED
tcp        0      0  kpriss.inria.fr.1018   givry.inria.fr.login   ESTABLISHED
tcp        0      0  kpriss.inria.fr.6000   toul.inria.fr.1497     ESTABLISHED
tcp        0      0  kpriss.inria.fr.1019   poly.polytechniq.login ESTABLISHED
tcp        0      0  kpriss.inria.fr.1020   toul.inria.fr.login    ESTABLISHED
...
tcp        0      0  *.6000                 *.*                    LISTEN
tcp        0      0  *.printer              *.*                    LISTEN
tcp        0      0  *.shell                *.*                    LISTEN
tcp        0      0  *.login                *.*                    LISTEN
...
udp        0      0  *.1062                 *.*
udp        0      0  *.ntalk                *.*
udp        0      0  *.talk                 *.*
udp        0      0  *.biff                 *.*
...
Active UNIX domain sockets
Address  Type   Recv-Q Send-Q    Inode     Conn     Refs  Nextref Addr
c37b8280 stream      0      0        0 c36e5200        0        0
...
c3736d80 stream      0      0 8017b0a0        0        0        0 /dev/printer
c3736c00 dgram       0      0 8017bce0        0        0        0 /dev/elcscntls
ckt
c3736a80 stream      0      0 8017cdb8        0        0        0 /tmp/.X11-unix
/X0
```

Les *sockets* d'Unix BSD

- En 4.2BSD, les `sockets` sont apparues (`Leffler, ...`). Leurs buts:
  1. transparence: idem pour opérations locales ou à distance
  2. efficacité: cf plus tard dans le noyau
  3. compatibilité: ne pas changer les programmes existants

- Fonctionnent selon le modèle client-serveur. Exemple: un serveur est `/usr/bin/Xwd` pour *X-windows*. Les clients sont les utilitaires X: `xterm, xclock, xload, ...`

- La séquence typique d'opérations du serveur est

```
fd=socket(domaine, type, protocole);
bind(fd, adresse, longueur_adresse);
listen(fd, max_servis);
fd1 = accept (fd, adresse_client, longueur_adresse_client);
read (fd1, ..., ...);
write {fd1, ..., ...);
```

- Le client lui fait

```
fd = socket(domaine, type, protocole);
connect (fd, adresse_serveur, longueur_adresse_serveur);
read (fd, ..., ...);
write (fd, ..., ...);
```

Les *sockets* (suite)

- `socket` crée un *file descriptor* décrivant une domaine *unix* `AF_UNIX` ou *internet* `AF_INET`, le type datagramme `SOCK_DGRAM` ou circuit virtuel `SOCK_STREAM`, un protocole (en général 0 pour celui par défaut).

- `bind` attache une adresse (port) à un `socket`. Les adresses ≤ 1024 sont réservées au super-utilisateur.

- `listen` donne la longueur maximale de la queue d'attente des demandes de connexions non servies (~ 5). Très utile car en général le serveur fait *fork()* dès qu'il accepte une connexion.

- `accept` est une fonction bloquante qui attend les demandes de connexion et qui retourne un nouveau *file descriptor* qui permettra de faire les entrées/sorties sur la nouvelle connexion.

- `connect` est l'opération duale pour le client. On se sert du *file descriptor* existant pour la nouvelle connexion.

Les *sockets* (suite)

La séquence typique est donc ainsi côté serveur:

```
int fd, newfd;

if ((fd = socket(...)) < 0)
    erreur ("ouverture socket");
if (bind(fd, ...) < 0)
    erreur ("bind");
if (listen (fd, 5) < 0)
    erreur ("listen");
for (; ;) {
    if ((newfd = accept (fd, ...)) < 0)     /* blocant */
        erreur ("accept");
    if (fork() == 0) {
        close (fd);
        .... /* on continue avec newfd */
        exit (0);
    }
    close (newfd);
}
```

Exemple X-windows en AF_UNIX

Client en X-windows pour établir une connexion

```
#include <sys/un.h>
#define X_UNIX_PATH "/tmp/.X11-unix/X"

Display *XOpenDisplay (register char *display)
{
    struct sockaddr_un unaddr;          /* UNIX socket data block */
    struct sockaddr *addr;              /* generic socket pointer */
    int addrlen;                        /* length of addr */
...
    unaddr.sun_family = AF_UNIX;
    sprintf (unaddr.sun_path, "%s%d", X_UNIX_PATH, idisplay);

    addr = (struct sockaddr *) &unaddr;
    addrlen = strlen(unaddr.sun_path) + sizeof(unaddr.sun_family);

    if ((fd = socket (((int) addr->sa_family, SOCK_STREAM, 0)) < 0) {
        return -1;
    }
    if (connect (fd, addr, addrlen) < 0) {
        int olderrno = errno;
        (void) close (fd);
        errno = olderrno;
        return -1;
    }
...
    dpy->fd = fd;
    return (dpy);
    }
```

Exemple X-windows en AF_UNIX

2- Serveur X (établissement du service)

```
#define X_UNIX_PATH     "/tmp/.X11-unix/X"
static struct sockaddr_un unsock;

static int
open_unix_socket ()
{
    int oldUmask;
    int request;

    unsock.sun_family = AF_UNIX;
    oldUmask = umask (0);
    strcpy (unsock.sun_path, X_UNIX_PATH);
    strcat (unsock.sun_path, display);
    unlink (unsock.sun_path);
    if ((request = socket (AF_UNIX, SOCK_STREAM, 0)) < 0) {
        Error ("Creating Unix socket");
        return -1;
    }
    if (bind(request, (struct sockaddr *)&unsock,
                      strlen(unsock.sun_path)+2))
    {
        Error ("Binding Unix socket");
        close (request);
        return -1;
    }
    if (listen (request, 5))
    {
        Error ("Unix Listening");
        close (request);
        return -1;
    }
    (void)umask(oldUmask);
    return request;
}
```

Exemple X-windows en AF_UNIX

3- Service d'un nouveau client

```
long WellKnownConnections;   /* Listener mask */
long LastSelectMask[mskcnt]; /* mask returned from last select call */

void
EstablishNewConnections()
{
    long readyconnections;     /* mask of listeners that are ready */
    int curconn;                  /* fd of listener that's ready */
    register int newconn;         /* fd of new client */
    long connect_time;
    register int i;
    register ClientPtr client;

    readyconnections = (LastSelectMask[0] & WellKnownConnections);
    if (!readyconnections)
        return;
    connect_time = GetTimeInMillis();
    /* kill off stragglers */
    for (i=1; i<currentMaxClients; i++) {
    ...
    }
    while (readyconnections) {
        curconn = ffs (readyconnections) - 1;
        readyconnections &= ~(1 << curconn);
        if ((newconn = accept (curconn,
                               (struct sockaddr *) NULL,
                               (int *)NULL)) < 0)
            continue;
        fcntl (newconn, F_SETFL, FNDELAY);
        ...
     }
     ...
}
```

Le *netclient* de Doligez

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>

extern int errno;

char *usage = "usage : netclient <machine> <tcp_port> [-q]";

int main (argc, argv)
     int argc;
     char *argv [];
{
  long i;
  int s;
  struct hostent *serverhost;
  struct sockaddr_in serveraddr;
  struct servent *service;
  int service_num;
  char buffer [8192];
  int nbytes, sent;
  char addr [4];
  int ret;
  int use_direct_addr;
  int quiet = 0;

  if (argc != 3 && argc != 4){
    fprintf (stderr, "%s\n", usage);
    exit (2);
  }
  if (argc == 4){
    if (!strcmp (argv [3], "-q")){
      quiet = 1;
```

```
  }else{
    fprintf (stderr, "%s\n", usage);
    exit (2);
  }
}
s = socket (AF_INET, SOCK_STREAM, 0);
if (s == -1){
  perror ("socket");
  exit (3);
}
if (s >= 8 * sizeof (int)){
  fprintf (stderr, "socket file number is too high\n");
  exit (3);
}
if (argv [1] [0] >= '0' && argv [1] [0] <= '9'){
  int a0, a1, a2, a3;
  sscanf (argv [1], "%d.%d.%d.%d", &a0, &a1, &a2, &a3);
  addr [0] = a0; addr [1] = a1; addr [2] = a2; addr [3] = a3;
  serverhost = gethostbyaddr (addr, 4, AF_INET);
  if (serverhost == NULL){
    use_direct_addr = 1;
  }else{
    use_direct_addr = 0;
  }
}else{
  serverhost = gethostbyname (argv [1]);
  use_direct_addr = 0;
}
if (serverhost == NULL && !use_direct_addr){
  fprintf (stderr, "unknown host: %s\n", argv [1]);
  exit (3);
}
if (argv [2] [0] >= '0' && argv [2] [0] <= '9'){
  service_num = htons (atoi (argv [2]));
}else{
  service = getservbyname (argv [2], NULL);
  if (service == NULL){
```

```
      fprintf (stderr, "unknown service: %s\n", argv [2]);
      exit (3);
    }
    service_num = service->s_port;
  }
  serveraddr.sin_family = serverhost->h_addrtype;
  serveraddr.sin_port = service_num;
  if (use_direct_addr){
    bcopy (addr, &serveraddr.sin_addr, 4);
  }else{
    bcopy (serverhost->h_addr, &serveraddr.sin_addr,
   serverhost->h_length);
  }
  if (connect (s, &serveraddr, sizeof (serveraddr)) == -1){
    if (errno == ECONNREFUSED){
      if (!quiet) perror ("connect");
      exit (100);
    }else{
      perror ("connect");
      exit (3);
    }
  }
  switch (fork ()){
  case -1:
    perror ("fork");
    exit (3);
  case 0:
    close (1);
    while (1){
      nbytes = read (0, buffer, 8192);
      if (nbytes == -1){
perror ("read (stdin)");
shutdown (s, 1);
exit (3);
      }
      if (nbytes == 0){
shutdown (s, 1);
```

```
exit (0);
      }
      for (sent = 0; sent < nbytes;){
        ret = write (s, buffer + sent, nbytes - sent);
if (ret == -1){
  perror ("write (socket)");
  exit (3);
}
sent += ret;
      }
    }
  default:
    close (0);
    while (1){
      nbytes = read (s, buffer, 8192);
      if (nbytes == -1){
perror ("read (socket)");
close (1);
wait (NULL);
exit (3);
      }
      if (nbytes == 0){
close (1);
wait (NULL);
exit (0);
      }
      for (sent = 0; sent < nbytes;){
ret = write (1, buffer + sent, nbytes - sent);
if (ret == -1){
  perror ("write (stdout)");
  exit (3);
}
sent += ret;
      }
    }
  }
}
```

Datagrammes

- Les datagrammes sont utilisés chaque fois qu'un serveur veut communiquer avec beaucoup de clients de manière épisodique. Exemple: *socket* de contrôle de `talk`. Attention: avec les datagrammes, les messages ne sont pas sûrs d'arriver, ni dans l'ordre. Le protocole est plus bas, donc plus efficace.

- `recvfrom()` et `sendto()` sont alors utilisés.

- les datagrammes se servent principalement du protocole `udp` (*User Datagram Protocol*)

- `talk` se sert de 2 *sockets*: contrôle en datagrammes (avec le démon `talkd`), communication en mode connecté (avec l'interlocuteur qui fait `talk`).

Datagrammes – 2

```
for (;;) {
    extern int errno;

    fromlen = sizeof(from);
    cc = recvfrom(0, (char *)&request, sizeof (request), 0,
                  &from, &fromlen);
    if (cc != sizeof(request)) {
       if (cc < 0 && errno != EINTR)
              perror("recvfrom");
       continue;
    }
    lastmsgtime = time(0);
    swapmsg(&request);
    if (debug) print_request(&request);
    process_request(&request, &response);
    /* can block here, is this what I want? */
    cc = sendto(sockt, (char *) &response,
                sizeof (response), 0, &request.ctl_addr,
                sizeof (request.ctl_addr));
    if (cc != sizeof(response))
        perror("sendto");
}
```

Datagrammes – 3

```
/* open the ctl socket */ /* client: ctl.c */
open_ctl()
{
    int length;

    ctl_addr.sin_port = 0;
    ctl_addr.sin_addr = my_machine_addr;
    ctl_sockt = socket(AF_INET, SOCK_DGRAM, 0);
    if (ctl_sockt <= 0)
        p_error("Bad socket");
    if (bind(ctl_sockt, &ctl_addr, sizeof(ctl_addr), 0) != 0)
        p_error("Couldn't bind to control socket");
    length = sizeof(ctl_addr);
    if (getsockname(ctl_sockt, &ctl_addr, &length) == -1)
        p_error("Bad address for ctl socket");
}
```

```
kpr% talk levy@toul
...
kpr% netstat -a
Active Internet connections (including servers)
tcp        0      0  kpriss.inria.fr.1042   toul.inria.fr.1026     ESTABLISHED
udp        0      0  kpriss.inria.fr.1416   *.*

tou% talk levy@kpriss
tou% netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  toul.inria.fr.1026     kpriss.inria.fr.1042   ESTABLISHED
udp        0      0  toul.inria.fr.1143     *.*
```

Les *sockets* AF_UNIX

- ce sont les connexions entre processus sur une même machine. Le protocole est supposé plus simple. Exemples: X-windows quand on fait

  ```
  % setenv DISPLAY unix:0
  ```

  ou bien dans `lpd`.

```
#define SOCKETNAME "/dev/printer"
    /*
     * Restart all the printers.
     */
    startup();
    (void) unlink(SOCKETNAME);
    funix = socket(AF_UNIX, SOCK_STREAM, 0);
    if (funix < 0) {
        logerr("socket");
        exit(1);
    }
#define    mask(s)    (1 << ((s) - 1))
    omask = sigblock(mask(SIGHUP)|mask(SIGINT)|mask(SIGQUIT)|mask(SIGTERM));
    signal(SIGHUP, cleanup);
    signal(SIGINT, cleanup);
    signal(SIGQUIT, cleanup);
    signal(SIGTERM, cleanup);
    sun.sun_family = AF_UNIX;
    strcpy(sun.sun_path, SOCKETNAME);
    if (bind(funix, &sun, strlen(sun.sun_path) + 2) < 0) {
        logerr("unix domain bind");
        exit(1);
    }
    sigsetmask(omask);
    defreadfds = 1 << funix;
    listen(funix, 5);
    finet = socket(AF_INET, SOCK_STREAM, 0);
```

Exercices en TD et à la maison

- Forts: *remote shell*

- Débutants: se connecter à un serveur WWW ou de news et récupérer le contenu d'un URL ou d'un article de news. Ou faire un serveur *remote echo*. On peut se mettre à deux pour faire ce dernier exemple.

Systèmes et réseaux

Cours 8

Jean-Jacques.Levy@inria.fr
Ecole polytechnique

tel: 39-63-56-89
http://pauillac.inria.fr/∼levy/

http://www.poly.polytechnique.fr/poly/...
.../∼www/poly/maj2-syst-96/

Plan

Datagrammes, *Talk*

Options des socket, *inetd*

Routage, RIP

Serveurs de noms, DNS, nslookup

ATM, Autonet

Généraux byzantins

Terminaux, pseudo-ttys

Exercices.

Datagrammes

- les datagrammes UDP connaissent les noms de ports (à la différence d'IP)
- les datagrammes ne sont pas surs: peuvent ne pas arriver, ou arriver dans le désordre
- les datagrammes sont utiles pour gérer une petite transaction, car l'ouverture du connection TCP coûte (plusieurs aller-retour dans l'établissement de la connexion).

```
kpr% talk levy@toul
...
kpr% netstat -a
Active Internet connections (including servers)
tcp        0      0  kpriss.inria.fr.1042   toul.inria.fr.1026     ESTABLISHED
udp        0      0  kpriss.inria.fr.1416   *.*

tou% talk levy@kpriss
tou% netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  toul.inria.fr.1026     kpriss.inria.fr.1042   ESTABLISHED
udp        0      0  toul.inria.fr.1143     *.*
```

Datagrammes, un exemple: *Talk*

```
main(argc, argv)
        int argc;
        char *argv[];
{

        get_names(argc, argv);
        init_display();
        open_ctl();
        open_sockt();
        start_msgs();
        if (!check_local() )
                invite_remote();
        end_msgs();
        set_edit_chars();
        talk();
}

get_addrs(my_machine_name, his_machine_name)
        char *my_machine_name, *his_machine_name;
{       ...
        /* find the server's port */
        sp = getservbyname("ntalk", "udp");
        if (sp == 0) {
                fprintf(stderr, "talk: %s/%s: service is not registered.\n",
                     "ntalk", "udp");
                exit(-1);
        }
        daemon_port = sp->s_port;
}
u_short daemon_port;    /* port number of the talk daemon */

int    ctl_sockt;
int    sockt;
int    invitation_waiting = 0;

CTL_MSG msg;
```

Datagrammes, un exemple: *Talk*

```
/* open the ctl socket */
open_ctl()
{
    int length;

    ctl_addr.sin_port = 0;
    ctl_addr.sin_addr = my_machine_addr;
    ctl_sockt = socket(AF_INET, SOCK_DGRAM, 0);
    if (ctl_sockt <= 0)
        p_error("Bad socket");
    if (bind(ctl_sockt, &ctl_addr, sizeof(ctl_addr), 0) != 0)
        p_error("Couldn't bind to control socket");
    length = sizeof(ctl_addr);
    if (getsockname(ctl_sockt, &ctl_addr, &length) == -1)
        p_error("Bad address for ctl socket");
}

open_sockt()
{
    int length;

    my_addr.sin_addr = my_machine_addr;
    my_addr.sin_port = 0;
    sockt = socket(AF_INET, SOCK_STREAM, 0);
    if (sockt <= 0)
        p_error("Bad socket");
    if (bind(sockt, &my_addr, sizeof(my_addr)) != 0)
        p_error("Binding local socket");
    length = sizeof(my_addr);
    if (getsockname(sockt, &my_addr, &length) == -1)
        p_error("Bad address for socket");
}

invite_remote()
{       ...
    announce_invite();
```

```
    setitimer(ITIMER_REAL, &itimer, (struct itimerval *)0);
    message("Waiting for your party to respond");
    signal(SIGALRM, re_invite);
    (void) setjmp(invitebuf);
    while ((new_sockt = accept(sockt, 0, 0)) < 0) {
        if (errno == EINTR)
            continue;
        p_error("Unable to connect with your party");
    }
    close(sockt);
    sockt = new_sockt;
    current_state = "Waiting for your party to respond";
    start_msgs();  ...
}

announce_invite()
{
    CTL_RESPONSE response;

    current_state = "Trying to connect to your party's talk daemon";
    ctl_transact(his_machine_addr, msg, ANNOUNCE, &response);
    remote_id = response.id_num;
    if (response.answer != SUCCESS) {
        if (response.answer < NANSWERS)
            message(answers[response.answer]);
        quit();
    }
    /* leave the actual invitation on my talk daemon */
    ctl_transact(my_machine_addr, msg, LEAVE_INVITE, &response);
    local_id = response.id_num;
}

/*
 * SOCKDGRAM is unreliable, so we must repeat messages if we have
 * not recieved an acknowledgement within a reasonable amount
 * of time
 */
```

```
ctl_transact(target, msg, type, rp)
    struct in_addr target;
    CTL_MSG msg;
    int type;
    CTL_RESPONSE *rp;
{
    int read_mask, ctl_mask, nready, cc;
    struct timeval wait;

    msg.type = type;
    daemon_addr.sin_addr = target;
    daemon_addr.sin_port = daemon_port;
    ctl_mask = 1 << ctl_sockt;

    /*
     * Keep sending the message until a response of
     * the proper type is obtained.
     */
    do {
        wait.tv_sec = CTL_WAIT;
        wait.tv_usec = 0;
        /* resend message until a response is obtained */
        do {
            cc = sendto(ctl_sockt, (char *)&msg, sizeof (msg), 0,
                &daemon_addr, sizeof (daemon_addr));
            if (cc != sizeof (msg)) {
                if (errno == EINTR)
                    continue;
                p_error("Error on write to talk daemon");
            }
            read_mask = ctl_mask;
            nready = select(32, &read_mask, 0, 0, &wait);
            if (nready < 0) {
                if (errno == EINTR)
                    continue;
                p_error("Error waiting for daemon response");
            }
```

```
        } while (nready == 0);
        /*
         * Keep reading while there are queued messages
         * (this is not necessary, it just saves extra
         * request/acknowledgements being sent)
         */
        do {
            cc = recv(ctl_sockt, (char *)rp, sizeof (*rp), 0);
            if (cc < 0) {
                if (errno == EINTR)
                    continue;
                p_error("Error on read from talk daemon");
            }
            read_mask = ctl_mask;
            /* an immediate poll */
            timerclear(&wait);
            nready = select(32, &read_mask, 0, 0, &wait);
        } while (nready > 0 && (rp->vers != TALK_VERSION ||
            rp->type != type));
    } while (rp->vers != TALK_VERSION || rp->type != type);
    rp->id_num = ntohl(rp->id_num);
    rp->addr.sa_family = ntohs(rp->addr.sa_family);
}
```

Datagrammes, un exemple: *Talkd*

```
/* talkd */

main(argc, argv)
        int argc;
        char *argv[];
{
        struct sockaddr_in from;
        int fromlen, cc;

       if (getuid()) {
                fprintf(stderr, "Talkd : not super user\n");
                exit(1);
        }
        (void) gethostname(hostname, sizeof (hostname));
        (void) chdir("/dev");
        (void) signal(SIGALRM, timeout);
        (void) alarm(TIMEOUT);
        for (;;) {
                extern int errno;

                fromlen = sizeof(from);
                cc = recvfrom(0, (char *)&request, sizeof (request), 0,
                    &from, &fromlen);
                if (cc != sizeof(request)) {
                        if (cc < 0 && errno != EINTR)
                        perror("recvfrom");
                        continue;
                }
                lastmsgtime = time(0);
                swapmsg(&request);
                if (debug) print_request(&request);
                process_request(&request, &response);
                /* can block here, is this what I want? */
                cc = sendto(0, (char *) &response,
                    sizeof (response), 0, &request.ctl_addr,
```

```
                    sizeof (request.ctl_addr));
                if (cc != sizeof(response))
                        perror("sendto");
        }
}
```

Options des *sockets*, *inetd*

- Pour éviter trop de serveurs en attente, il existe un super-serveur internet pour chaque site: *inetd*. Il lit la table `/etc/services` et lance le serveur correspondant.

```
echo          7/tcp
echo          7/udp
daytime       13/tcp
daytime       13/udp
ftp           21/tcp
telnet        23/tcp
smtp          25/tcp       mail
time          37/tcp       timserver
time          37/udp       timserver
finger        79/tcp
login         513/tcp
who           513/udp      whod
dictionary    10300/tcp    webster
```

- on peut mettre des options sur un *socket* avec `setsockopt()` et `getsockopt()`. Exemples: `SO_KEEPALIVE`, `SO_LINGER`.

- `htonl`, `htons`, `ntohl`, `ntohl` convertissent les entiers cours ou longs entre les machines hotes et la convention réseau (grand *endian*), puisqu'il faut bien un standard de communication entre des machines d'*endianité* différente.

Serveurs de noms

- le réseau est organisé en domaines et sous-domaines. Exemple: `pauillac.inria.fr`. Il y a le domaine `fr`, puis le sous-domaine `inria`, etc.

- pour retrouver l'adresse IP, on peut la prendre dans `/etc/hosts` ou par `gethostbyname()`. Mais si l'adresse est très lointaine on la retrouve par un serveur de noms (DNS ou Domain Nave Server et BIND).

- `nslookup` est une commande qui permet d'avoir des informations en intéractif. Par exemple, les serveurs connaissant l'INRIA.

```
% nslookup
Default Server:  concorde.inria.fr
Address:  192.93.2.39

> set type=any
> inria.fr
Server:  concorde.inria.fr
Address:  192.93.2.39

inria.fr        origin = ns1.nic.fr
        mail addr = hostmaster.inria.fr
        serial=96031908, refresh=21600, retry=3600, expire=3600000, min=172800
inria.fr        nameserver = ns1.nic.fr
inria.fr        nameserver = ns2.nic.fr
inria.fr        nameserver = mirsa.inria.fr
inria.fr        nameserver = imag.imag.fr
inria.fr        nameserver = ns.eu.net
inria.fr        nameserver = princeton.edu
```

Routage

- il y a des passerelles sur le réseau. Il faut trouver les chemins entre 2 adresses IP. Le problème revient à trouver un arbre de recouvrement sur le graphe de l'internet.

- la commande `traceroute` donne le chemin emprunté. Exemple:

```
% /usr/sbin/traceroute poly.polytechnique.fr
traceroute to poly.polytechnique.fr (129.104.252.1), 30 hops max, 40 byte packets
 1  rocq-gwf.inria.fr (128.93.1.100)  4 ms  3 ms  4 ms
 2  rocq-gwr.inria.fr (192.93.1.92)  3 ms  4 ms  5 ms
 3  inria-rocquencourt.rerif.ft.net (192.93.122.1)  4 ms  4 ms  4 ms
 4  boulogne.rerif.ft.net (193.48.55.57)  6 ms  7 ms  7 ms
 5  stlamb12.rerif.ft.net (193.48.53.41)  10 ms  10 ms  12 ms
 6  stlamb1.rerif.ft.net (193.48.53.161)  11 ms  11 ms  14 ms
 7  stlamb3.rerif.ft.net (193.48.53.11)  11 ms  11 ms  11 ms
 8  massy1.rerif.ft.net (193.48.53.18)  11 ms  12 ms  11 ms
 9  ep-palaiseau.rerif.ft.net (193.48.54.26)  17 ms  15 ms  15 ms
10  192.70.27.13 (192.70.27.13)  17 ms  14 ms  16 ms
11  192.70.27.13 (192.70.27.13)  17 ms !H  17 ms !H *
```

- une bonne référence!

```
TIT     .Le routage dans l'Internet
SLA     .Christian Huitema; Eyrolles, 1995
```

Problèmes distribués typiques

- *Le problème des généraux byzantins*

  On a des processus fiables et non fiables. On veut que:

  1. tous les fiables doivent produire le même résultat
  2. si tous les fiables commence avec la même valeur, alors cette valeur doit être le résultat de tous les fiables.


```
TIT     .Parallel program design : a foundation
SLA     .K. Mani Chandy, Jayadev Misra. - Reading, MA US ; Don Mills,
        .Ontario ; Wokingham, England : Addison-Wesley, 1988. -
        .XXVIII-516 p. ; 24 cm.

TIT     .Handbook of Theoretical Computer Science. MIT Press, ed.
        van Leewen, 1990.
        Chapter 18, Leslie Lamport, Nancy Lynch, pp 1159-1199.
```

Consensus en milieu distribué non fiable Généraux byzantins

Enoncé (bis): Quelques généraux félons se sont mêlés à l'armée byzantine qui doit néanmoins avancer en ordre. Il faut donc que:

1. tous les généraux fiables prennent la même décision,
2. si le général en chef n'est pas un traitre, tous les généraux fiables obéissent à l'ordre du général en chef.

Remarque: cet énoncé se ramène à celui du dernier cours. Exercice: montrez le.

Leslie Lamport, Rob Shostak, Marshall Pease
The Byzantine Generals Problem
ACM/TOPLASS, Vol 4, N 3, Jul 1982

Danny Dolev
The Byzantine Generals Strike Again
The Journal of Algorithms,1982

On peut montrer qu'il n'existe de solution que si le nombre de généraux n est plus grand que $3m+1$ où m est le nombre de traîtres.

Généraux byzantins

Solution (`Lamport, Shostak, Pease -- Dolev`)

OM(m) si $m = 0$

(1) Le chef 0 envoie son ordre à tous ses adjoints,
(2) Tous les généraux obéissent à l'ordre du chef, ou battent en retraite si rien reçu.

OM(m) si $m > 0$

(1) Le chef 0 envoie son ordre à tous ses adjoints,
(2) Pour tout i, soit v_i la valeur reçue par le général i. Si rien reçu, $v_i = \mathtt{RETRAITE}$. Le général i fait comme le général en chef pour OM($m-1$) pour les $n-2$ autres lieutenants,
(3) Pour tout i et $j \neq i$, soit v_j la valeur reçue par le lieutenant i du lieutenant j à la 2ème étape (`RETRAITE` si rien reçu). Le général i fait un vote à majorité sur l'ensemble $\{v_1, v_2, \ldots v_{n-1}\}$.

Exercice: Faire l'exemple avec 4 généraux (les 2 cas où le chef ou un lieuten ant est traître). Prouver la correction de l'algorithme.

Exercices en TD et à la maison

- Forts: *remote shell*

- Débutants: continuer à faire un serveur, avec un consensus distribué ou une élection. Faire un serveur d'oral.

Systèmes et réseaux

Cours 9

Jean-Jacques.Levy@inria.fr
Ecole polytechnique

tel: 39-63-56-89
http://pauillac.inria.fr/∼levy/

http://www.poly.polytechnique.fr/poly/...
.../∼www/poly/maj2-syst-96/

Plan

Mémoire partagée

Conditions et moniteurs

Sémantique des conditions

Et si on faisait de la recherche?

Problèmes non traités dans le cours

Terminaux, Pseudo ttys, Streams

Communication par mémoire partagée

Protocoles réseaux

Sécurité sur le réseau: capacités, RSA

Remote procedure calls

Multi processeurs

ATM

Multicast

Transactions atomiques

Mémoire partagée distribuée

Micro noyaux

Communication entre processus par mémoire partagée

- **`shmget`**, **`shmat`** opérations Système V.
- Une mémoire partagée 2 fois dans un même processus

```
int shmid;

main()
{
    int i, *pint;
    char *addr1, *addr2;
    extern char *shmat();

    shmid = shmget(75, 128*1024, 0777|IPC_CREAT);
    addr1 = shmat(shmid, 0, 0);
    addr2 = shmat(shmid, 0, 0);
    printf ("addr1 0x%x, addr2 0x%x\n", addr1, addr2);

    pint = (int *) addr1;
    for (i = 0; i < 256; i++)
        *pint++ = i;
    *pint = 256;

    pint = (int *) addr2;
    for (i = 0; i < 256; i++)
        print ("index %d, value %d\n", i, *pint++);

    pause();
}
```

Communication entre processus par mémoire partagée

- `shmget`, `shmat` opérations Système V.
- Une mémoire partagée 2 fois dans un même processus

```
int shmid;

main()
{
    int i, *pint;
    char *addr;
    extern char *shmat();

    shmid = shmget(75, 64*1024, 0777);
    addr = shmat(shmid, 0, 0)

    pint= (int *) addr;
    while (*pint == 0)
        ;
    for (i = 0; i < 256; i++)
        printf ("%d\n", *pint++):
}
```

Communication par mémoire partagée en Modula-3

```
INTERFACE Thread;

TYPE
  T <: ROOT;          (* A handle on a thread *)
  Mutex = MUTEX;      (* Locked by some thread, or unlocked *)
  Condition <: ROOT;  (* A set of waiting threads *)

(* Initially a Mutex in unlocked and a Condition is empty.
   It is a checked runtime error to pass the NIL Mutex,
   Condition, or T to any procedure in this interface *)

  Closure = OBJECT METHODS apply (): REFANY RAISES {} END;

PROCEDURE NewMutex (): Mutex;
(* A newly-allocated, unlocked mutex. *)

PROCEDURE NewCondition (): Condition;
(* A newly-allocated condition with no threads waiting on it. *)

PROCEDURE Fork (cl: Closure): T;
(* A handle on a newly-created thread that executes cl.apply(). *)

PROCEDURE Join (t: T): REFANY;
(* Wait until t has terminated and return its result. It is a
   checked runtime error to call this more than once for any t. *)

PROCEDURE Wait (m: Mutex; c: Condition);
(* The calling thread must have m locked. Atomically unlocks
   m and waits on c. Then relocks m and returns. *)

PROCEDURE Acquire (m: Mutex);
(* Wait until m is unlocked and then lock it. *)

PROCEDURE Release (m: Mutex);
(* The calling thread must have m locked.  Unlocks m. *)
```

```
PROCEDURE Broadcast (c: Condition);
(* All threads waiting on c become eligible to run. *)

PROCEDURE Signal (c: Condition);
(* One or more threads waiting on c become eligible to run. *)

PROCEDURE Self (): T;
(* Return the handle of the calling thread. *)

EXCEPTION Alerted; (* Approximate asynchronous interrupts *)

PROCEDURE Alert (t: T);
(* Mark t as an alerted thread. *)

PROCEDURE TestAlert (): BOOLEAN;
(* TRUE if the calling thread has been marked alerted. *)

PROCEDURE AlertWait (m: Mutex; c: Condition) RAISES {Alerted};
(* Like Wait, but if the thread is marked alerted at the time of
   call or sometime during the wait, lock m and raise Alerted.
   Implementations may raise additional exceptions. *)

PROCEDURE AlertJoin (t: T): REFANY RAISES {Alerted};
(* Like Join, but if the thread is marked alerted at the time of
   call or sometime during the wait, raise Alerted.
   Implementations may raise additional exceptions. *)

END Thread.
```


```
TIT     .Systems programmin with MODULA-3
SLA     .ed. Greg Nelson. - Englewood Cliffs, NJ : Prentice Hall,
        .1991. -IX-267 p.; 24 cm. - (Prentice Hall series in
        . innotative technology).
```

Pile concurrente

Dépilage

```
VAR nonEmpty := NEW(Thread.Condition);

LOCK m DO
    WHILE head = NIL DO Thread.Wait(m, nonEmpty) END;
    topElement := head;
    head := head.next;
END;
```

Empilage

```
LOCK m DO
    newElement.next := head;
    head := newElement;
    Thread.Signal (nonEmpty);
END;
```

Caution: `WHILE` is safer than `IF` in Pop.

Table concurrente

```
VAR table := ARRAY [0..999] of REFANY {NIL, ...};
VAR i:[0..1000] := 0;

PROCEDURE Insert (r: REFANY) =
  BEGIN
    IF r <> NIL THEN

        table[i] := r;
        i := i+1;

  END;
END Insert;
```

Caution: how not to loose values?

Interblocage (*Deadlocks*)

Thread A locks mutex m_1
Thread B locks mutex m_2
Thread A trying to lock m_2
Thread B trying to lock m_1

Simple technique

Respect a partial order between semaphores. For example, A and B uses m_1 and m_2 in same order.

Les 5 philosophes

```
/* nombre de fourchettes disponibles pour chaque philosophe */
int fork[5] = {2, 2, 2, 2, 2};

Conditions c[5] = {True, True, True, True, True};

Mutex s = 1;


philosopher(int i)
{
    for (;;) {
        think();
        Acquire (s);
            while (forks[i] != 2)
                Wait (s, c[i]);
            --forks[(i+1) % 5];
            --forks[(i-1) % 5];
        Release (s);
        eat();
        Acquire (s);
            ++forks[(i+1) % 5];
            ++forks[(i-1) % 5];
        Release (s);
        Signal (c[(i-1) % 5]);
        Signal (c[(i+1) % 5]);
    }
}
```

Communication en mémoire partagée

- `Signal()` ne garantie pas que le prédicat sur lequel on attend est vrai ($\neq$ conditions de Hoare). `Signal()` relache la condition, mais un autre processus peut renter en section critique, et invalider le prédicat. Donc quand on revient de `Wait()`, on reteste le prédicat. (plus facile à implémenter)

- `Wait()` fait atomiquement l'abandon de la section critique et la suspension du processus. Sinon, il peut y avoir une course au réveil: le processus qui fait `Wait()` abandonne la section critique, un autre processus s'exécute très vite et fait `Signal()`, le 1er processus se suspend, et ne sera jamais réveillé, car le signal s'est perdu.


```
TIT     .Synchronization Primitives for a Multiprocessor:
        .A Formal Specification
SLA     .Digital, Systems Research Center, tech. report 20
        .1997.
AUT     .A.D. Birell, J.V. Guttag, J.J Horning, R. Levin
```

2 exemples de DEA

DEA Algorithmique

1. Algorithmique et combinatoire des mots, [J. Berstel]
2. Géométrie computationnelle, [J.-D. Boissonnat]
3. Introduction au calcul parallèle et distribué, [M. Morvan]
4. Algorithmes probabilistes, [J.-M. Steyaert]

. Pratique du calcul formel.

Filières, responsables et cours

1. Analyse d'algorithmes [J.-M. Steyaert]:
2. Automates et mots [J.-E. Pin]:
3. Calcul formel [D. Lazard]:
4. Combinatoire [R. Cori]:
5. Complexité, codage et cryptographie [J. Stern]:
6. Géométrie, images et robotique [O. Faugeras]:
7. Paralllisme et concurrence [A. Petit]:

. Conception de circuits, [P. Bertin, J. Vuillemin].

2 exemples de DEA –bis–

DEA Sémantique, Preuves et Programmation

1. Structures syntaxiques du lambda calcul, [T. Hardin]
2. Preuves constructives, [G. Dowek]
3. Théories équationnelles, [H. Comon]
4. Récursivité et théorie des domaines, [G. Longo]
5. Sémantique dénotationnelle, [R. Dicosmo]
6. Types et programmation, [D. Rémy - M. Mauny]

Filières, responsables et cours

1. Langages, [G. Cousineau]
(Interprétation abstraite, A. Deutsch;
Sous-typage et langages objets, G. Catagna;
Compilation, X. Leroy; Programmation logique, F. Fages;
Contraintes, L. Puel;)

2. Sémantique, [P.-L. Curien]
(Séquentialité, P.-L. Curien;
Logique linéaire, R. Dicosmo;
Synchrone, N. Halbwachs;
Asynchrone et distribution, J.-J. Lévy)

3. Preuves et spécifications, [J.-P. Jouannaud]
(Constructions, G. Huet;
Démonstration automatique, J. Goubault;
La méthode B, J.-R. Abrial;
Spécifications algébriques, M. Bidoit - V. Donzeau-Gouge)

Laboratoires de recherche

CNET Lannion
CNRS
INRIA, 5 Unités de recherche
ENS
ENS Fontenay
ENS Lyon
ENPC
ENSMP
ENST
LIX

Labri Bordeaux
Imag Grenoble
Marseille Lumigny
Paris 6
Paris 7, (Logique)
Marne la Vallée
ORSAY
Saint Denis
Léonard de Vinci
CNAM

Thomson LCR
Alcatel Marcoussis
GENSET
ILOG
O2 Technology
Chorus Systèmes
Xerox Grenoble
Web Consortium

AT&T Bell labs
Bellcore
Digital SRC, WRL
IBM Almaden, Yorktown
Microsoft Research
Xerox PARC
HPlabs
SUN

Domaines de recherche (exemple de l'INRIA)

Thèmes

1. Réseaux et systèmes

2. Génie logiciel et calcul symbolique

3. Interaction homme-machine, images, données, connaissances

4. Simulation et optimisation de systèmes complexes

Action de développements

DYADE : conception de systèmes d'information avancés
GÉNIE : science de l'information et ingénierie concourante
MÉDIACULTURE : système multimédia distribué pour la documentation culturelle
PRAXITELE : transport urbain public individuel
PRÉVISIA : interopérabilité des systèmes d'information
SYNCHRONE : environnement de développement de systèmes temps-réel
W3C : les services d'information sur Internet
WEBTOOLS : outils logiciels pour le World Wide Web

Thème 1

Parallélisme et architecture :

APACHE - Algorithmique parallèle et partage de charge
API - Architectures parallèles intégrées
CAPS - Compilation, architectures parallèles et systèmes
PAMPA - Environnement de programmation des architectures massivement parallèles
ReMaP - Régularité et parallélisme massif
SLOOP - Simulation, langages orientés objets et parallélisme

Réseaux, systèmes, évaluation de performances :

MÉVAL - Modélisation et évaluation des systèmes informatiques
MISTRAL - Modélisation en informatique et systèmes de télécommunication : recherche et applications logicielles
MODEL - Modélisation de systèmes aléatoires
RESEDAS - Outils logiciels pour les télécommunications et les systèmes distribués
RODEO - Réseaux à haut débit, réseaux ouverts
SATURNE - Systèmes répartis tolérant les fautes et les intrusions
SIRAC - Systèmes informatiques répartis pour applications coopératives
SOLIDOR - Architectures et systèmes distribués extensibles, tolérance aux fautes, programmation objet
SOR - Systèmes d'objets répartis

Thème 1 –bis–

Programmation distribuée et temps-réel :

ADP - Algorithmes répartis et protocoles
EPATR - Environnement de programmation pour applications temps réel
MEIJE - Parallélisme, synchronisation et temps réel
PARA - Parallélisme, mobilité
REFLECS - Systèmes informatiques répartis temps réel tolérant les fautes
SPECTRE - Spécification et programmation des systèmes communicants et temps réel

Thème 2

Sémantique et programmation :

CALLIGRAMME - Logique linéaire, réseaux de démonstration et grammaires catégorielles
COQ - Spécifications et preuves de programmes
CRISTAL - Programmation typée, modularité et compilation
CROAP - Conception et réalisation d'outils d'aide à la programmation
EURECA - Preuve, calcul symbolique et logique
LANDE - Langages déclaratifs
LOCO - Programmation logique avec contraintes
OSCAR - Outils syntaxiques pour la construction et l'analyse de programmes
PROTHEO - Contraintes, déduction automatique et preuves de propriétés de logiciels

Algorithmique et calcul formel :

ALGO - Algorithmes
CODES - Codes et protection de l'information
PRISME - Géométrie, algorithmes et robotique
SAFIR - Systèmes algébriques formels pour l'industrie et la recherche

Thème 3

Bases de données, bases de connaissances, systèmes cognitifs :

ACACIA - Acquisition des connaissances pour l'assistance à la conception par interaction entre agents
AIRELLE - Représentations et langages
ATOLL - Atelier d'outils logiciels pour le langage naturel
DIALOGUE - Dialogue homme-machine à forte composante langagière
OPERA - Outils pour les documents électroniques : recherche et applications
ORION - Modélisation des connaissances pour l'automatisation des tâches
PSYCHOERGO - Psychologie ergonomique pour l'informatique
REPCO - Représentation des connaissances
RODIN - Systèmes de bases de données
SHERPA - Bases de connaissances à objets
SHOOD - Méthodes et outils pour l'intégration de systèmes industriels
SYCO - Systèmes de compréhension et bases de connaissances
VERSO - Bases de données

Thème 3 –bis–

Vision, analyse et synthèse d'images :

AIR - Traitement d'image et données satellites dynamiques
EPIDAURE - Imagerie et robotique médicale
iMAGIS - Modèles, algorithmes, géométrie pour le graphique et l'image de synthèse
ISA - Image, synthèse, analyse
MOVI - Modélisation, localisation, reconnaissance et interprétation en vision parordinateur
PASTIS - Analyse de scènes et traitement d'images symboliques
ROBOTVIS - Vision par ordinateur et robotique
SHARP - Programmation automatique et systèmes décisionnels en robotique
SIAMES - Synthèse d'image, animation, modélisation et simulation
SYNTIM - Analyse et synthèse d'images
TEMIS - Traitement, exploitation et modélisation d'images séquentielles

Thème 4

Automatique, robotique, signal :

AS - Automatique et signal
ATGC - Action transversale génome et calcul
BIP - Conception et contrôle de robots marcheurs et applications
COMORE - Contrôle et modélisation de ressources renouvelables
CONGÉ - Contrôle géométrique des systèmes non linéaires
FRACTALES - Approche fractale pour l'analyse et la modélisation de signaux complexes
ICARE - Instrumentation, commande et architecture des robots évolués
MÉTA 2 - Méta automatique et méthodes de l'automatique
MIAOU - Mathématiques et informatique de l'automatique et de l'optimisation pour l'utilisateur
PROMATH - Programmation mathématique
SAGEP - Simulation, analyse et gestion des systèmes de production
SOSSO - Applications et outils de l'automatique
SYSTOL - Modélisation statistique et applications biomédicales

Thème 4 –bis–

Modélisation et calcul scientifique :

ALADIN - Algorithmes adaptés au calcul numérique intensif
CAIMAN - Calcul scientifique, modélisation et analyse numérique
ESTIME - Estimation de paramètres
IDOPT - Identification et optimisation de systèmes en physique et en environnement
M3N - Multi-modèles et méthodes numériques
MODULEF - Méthodes et outils pour le calcul scientifique
NUMATH - Analyse mathématique et traitement numérique de modèles non linéaires
OMEGA - Méthodes numériques probabilistes
ONDES - Modélisation, analyse, simulation des équations des ondes
SINUS - Simulation numérique dans les sciences de l'ingénieur
SYSDYS - Systèmes dynamiques stochastiques