Systèmes d'exploitation: les entrées/sorties

Luigi Santocanale

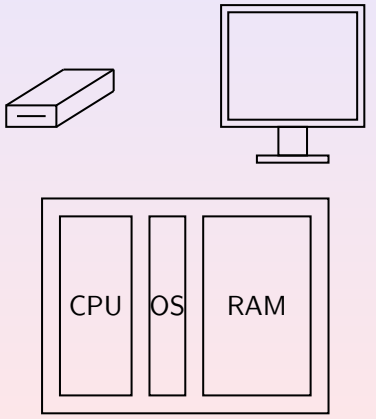
Laboratoire d'Informatique Fondamentale,
Centre de Mathématiques et Informatique,
39, rue Joliot-Curie - F-13453 Marseille

13 octobre 2004

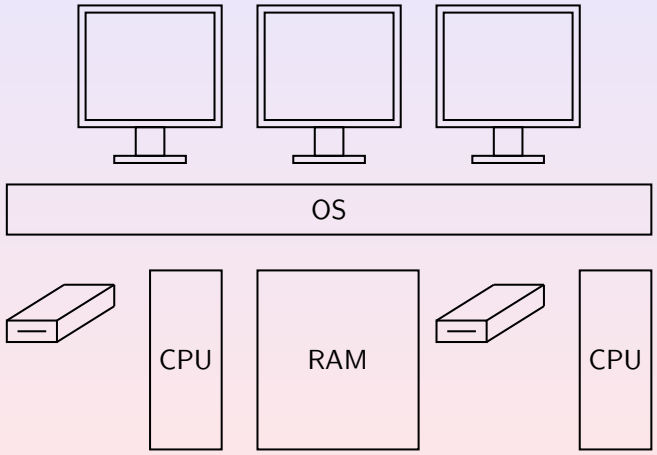
Plan

Plan

OS : machine virtuelle

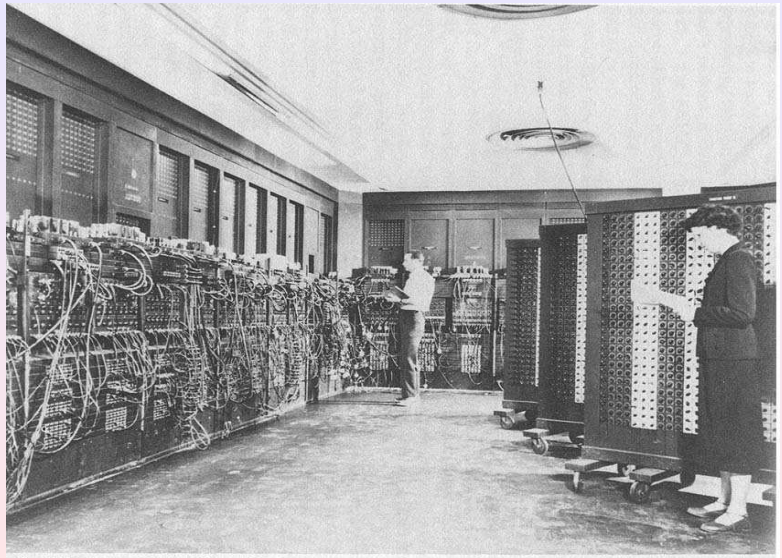


OS : allocateur de ressources



Plan

1945 – 1955 : les tubes à vide



The ENIAC.
Smithsonian Institution Photo No. 53192.

1945 – 1955 : les tubes à vide

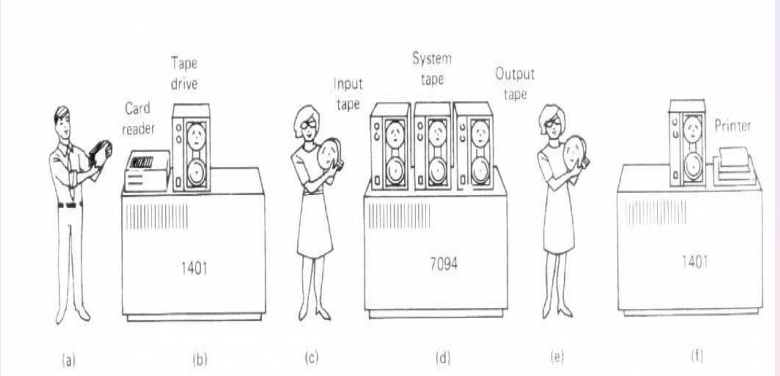
ENIAC : 20000 tubes à vide, 160m^2

- constructeur = programmateur = utilisateur
- lots d'heures allouées au programmateur
- programmation par câblage,
  pas de langages de programmation
- pas de mémoire
- calculs : tables des sinus et cosinus

1955 – 1965 : les transistors

- mainframes IBM 7094
- écriture du source sur carte perforé
- appui de « petit » ordinateurs IBM 1401 pour collectionner les taches sur bande magnétique
- opérateur humain transfère les bandes magnétiques vers/de l'ordinateur

Le traitement par lots (batch) (Tanenbaum 1992)



- traitement d'un ensemble de travaux
- transfert sur bande magnétique
- opérateur insère et récupère les bandes
- l'ordinateur compile et charge automatiquement
- impression off-line

1965 – 1980 : Circuits intégrés

- IBM system 360 : famille d'ordinateurs homogènes
- calcul scientifique vs. calcul commerciale : le pb des E/S
- la multiprogrammation :
  - plusieurs taches en mémoire
  - mise en attente des jobs bloqués en E/S
- spooling (Simultaneous Peripheral Operation On Line) : chargemanet automatique des nouvelles taches (jobs)

Limites : manque d'interactivité

Le partage du temps

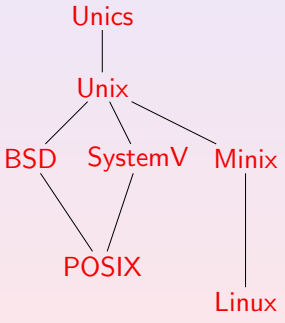
- On découpe le temps en tranche
- Les utilisateurs interagissent sur des terminaux
- MIT, Bell Labs, General Electric :
  développement de MULTICS.
  Modèle : le système de distribution de l'électricité

1980 – . . . : les ordinateurs personnels

- Circuits LSI (Large Scale Factors) :
  baisse de prix.
- Mini-ordinateur →Micro-ordinateur : chacun a son ordinateur.
- Développement d'applications conviviaux, « User-friendly »:
  destinés a l'utilisateur sans diplôme en informatique.
- MS-DOS pour IBM 8088, UNIX pour Motorola 68000.
- Versions ultérieures de MS-DOS intègrent des éléments de UNIX.

Plan

Unix : historique



1969 : portage de MULTICS sur un PDP-7
par Ken Thompson chez Bell Labs

1970 : portage de UNICS sur un PDP-11

arrivée de D. Ritchie, réécriture en C,
portage de UNIX sur autres machines

1977 : développement à l'université de Berkeley

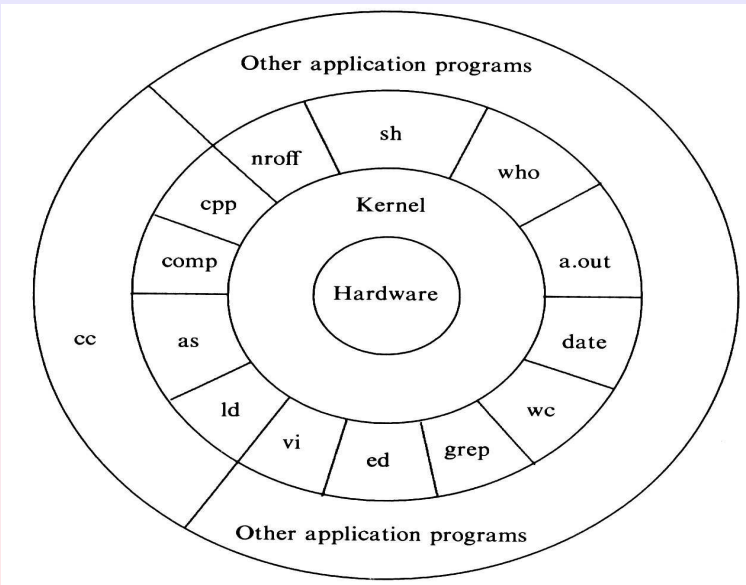
1983 : faillite de AT&T,
droits commerciales sur System V

1984 : norme IEEE 1003

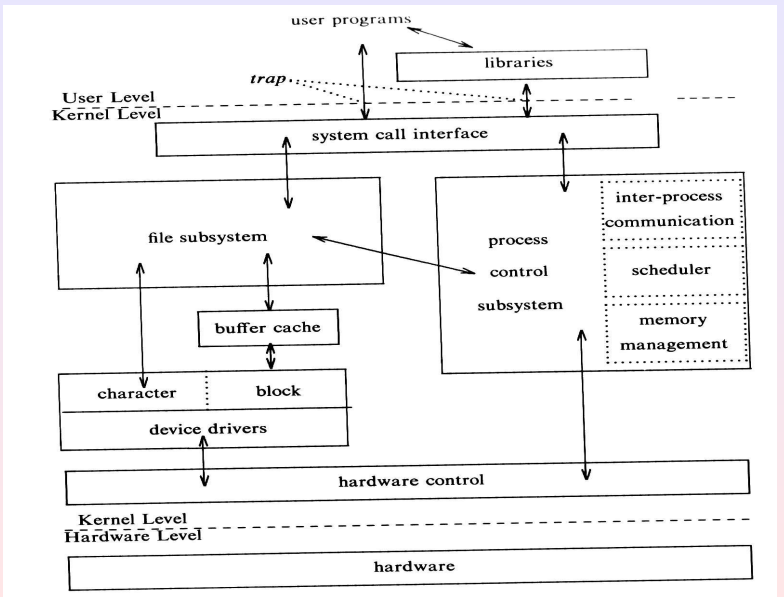
1985 : didactique : UNIX minimal

1991 : implantation de UNIX sur les PCs

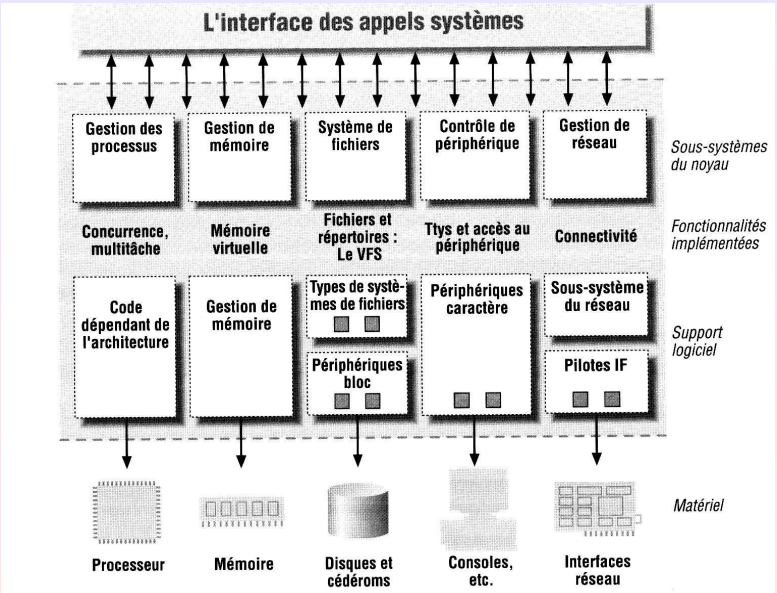
Unix 1986 (Bach 1986)



Unix 1986, le noyau (Bach 1986)



Linux 2000, le noyau (Rubini & Corbet, 2000)



Étudier le systèmes d'exploitation UNIX ?

1. Utilisateur :
   savoir utiliser le système UNIX.
2. Administrateur-programmeur :
   savoir gérer un système UNIX : programmation shell-script
3. Programmeur :
   savoir bien utiliser les appels système (interface POSIX) dans un programme.
4. Théorie :
   les problèmes posés par la conception d'un système d'exploitation.
5. Développeur de systèmes:
   problèmes posé par l'implantation d'un système d'exploitation.

Ici: on s'occupe de 2,3,4.

Plan

fopen, fclose

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

FILE * fopen (const char * *fic*, const char * *mode*)
fic : le nom du fichier a ouvrir
mode : le mode d'ouverture: "r", "w","a", "r+", "w+", "a+".
Retourne : pointer sur struct FILE, NULL si erreur.

int fclose (FILE * *ptr_file*)
ptr_file : pointeur sur le fichier ouvert qu'on veut fermer
Retourne : 0 ou EOF si erreur

Modes d'ouverture

"r" (read) : lecture, positionnement au début.
"w" (write): écriture, positionnement au début avec création/écrasement du fichier.
"a" (append) : écriture, positionnement à la fin, pas d'écrasement du fichier.
"r+" : lecture/écriture, positionnement au début, pas d'écrasement du fichier existent.
"w+" : lecture/écriture, positionnement au début. écrasement du fichier.
"a+" : lecture/écriture, positionnement à la fin, pas d'écrasement du fichier.

fprintf, fscanf, fgetc, fgets

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

int fprintf (FILE * *flot*, const char *format*, ...)
Retourne : nombre d'octets écrits, ou valeur négatif si erreur.

int fscanf (FILE * *flot*, const char *format*, ...)
Retourne : nombre de paramètres reconnus, ou EOF si erreur de flot.

int fgetc (FILE * *flot*)
Retourne : le caractère ou EOF.

char * fgets (char * s *tampon*, int *maxtaille*, FILE * *flot*)
Retourne : *tampon* ou NULL si erreur ou EOF.
Remarques : Caractère fin de ligne \n stocké dans le tampon.

Programme : comptage.c

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 :
 4 : void compter(char nom[], FILE *f)
 5 : {
 6 :   int c , cars = 0, lignes = 0;
 7 :
 8 :   while( (c = fgetc(f)) != EOF ){
 9 :     cars ++;
10 :     if (c == '\n') lignes ++;
11 :   }
12 :   printf("%s : %d caractères, %d lignes.\n", nom, cars, lignes);
13 : }
14 :
```


Programme : comptage.c

```
15 : int main(int argc, char *argv[])
16 : {
17 :   FILE * f;
18 :
19 :   if (argc != 2)
20 :     exit(EXIT_FAILURE);
21 :
22 :   if ( (f=fopen(argv[1],"r")) == NULL)
23 :     {
24 :       perror("fopen");
25 :       exit(EXIT_FAILURE);
26 :     }
27 :   compter(argv[1],f);
28 :   fclose(f);
29 :   exit(EXIT_SUCCESS);
30 : }
```


fflush

```
#include <stdio.h>
int fflush(FILE * flot)
```

flot : flot ouvert en écriture.

Retourne : 0/EOF

Remarques : Vide le tampon du flot *flot*.

stdin, stdout, stderr, printf,scanf, getc, gets

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

FILE * stdin, stdout, sdterr

int printf (const char *format*, ...)
Remarques : équivalent à `fprintf(stdout,...)`.

int scanf (const char *format*, ...)
Remarques : équivalent à `fscanf(stdin,...)`.

int getc (FILE * *flot*)
Remarques : équivalent à `fgetc(stdin)`.

char * gets (FILE * *flot*)
Remarques : Ne pas utiliser !!!

fread

```
#include <stdio.h>
size_t fread(void * ptr, size_t taille,
size_t nb_objets, FILE * ptr_fic)
```

ptr : un pointeur à un tampon en mémoire (déjà allouée)
taille : nombre d'octets pour chaque objets
nb_objets : le nombre d'objets qu'on veut lire, tel que

```
nb_objets * taille < size_of(ptr)
```

ptr_fic : le fichier ouvert qu'on veut lire

Retourne : nombre d'objets lus ($\leq$ *nb_objets*).

fwrite

```
#include <stdio.h>
size_t fwrite(void * ptr, size_t taille,
size_t nb_objets, FILE * ptr_fic)
```

ptr : un pointer à un tampon en mémoire
taille : nombre d'octets pour chaque objets
nb_objets : le nombre d'objets qu'on veut écrire
ptr_fic : le fichier ouvert qu'on veut lire

Retourne : nombre d'objets écrits ($\leq$ *nb_objets*).

Programme : copier.c

```
 1 : #include <stdlib.h>
 2 : #include <stdio.h>
 3 : #define TAILLETAMPON 1024
 4 :
 5 : int main(int argc, char *argv[])
 6 : {
 7 :   FILE *source, *dest;
 8 :   char tampon[TAILLETAMPON];
 9 :   size_t nolu;
10 :
11 :   if (argc != 3)
12 :     {
13 :       printf("Usage : %s source destination\n",argv[0]);
14 :       exit(EXIT_FAILURE);
15 :     }
```


Programme : copier.c

```
16 :   if( (source = fopen(argv[1],"r")) == NULL
17 :        || (dest = fopen(argv[2],"w")) == NULL )
18 :     {
19 :       perror("fopen");
20 :       exit(EXIT_FAILURE);
21 :     }
22 :   while
23 :     ((nolu = fread(tampon,sizeof(char),sizeof(tampon),source)) > 0)
24 :     fwrite(tampon,sizeof(char),nolu,dest) ;
25 :
26 :   fclose(source); fclose(dest);
27 :   exit(EXIT_SUCCESS);
28 : }
```

fseek

```
#include <stdio.h>
int fseek(FILE * ptr_fic, long int offset,
int origine)
```

ptr_fic : le descripteur du fichier ouvert
offset : déplacement par rapport à origine
origine : SEEK_SET, début du fichier.
SEEK_CURR, position courante.
SEEK_END, fin du fichier.
Retourne : 0 succès, -1 erreur.

Plan

open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char * fic, int mode, mode_t droits)
```

fic : le nom du fichier à ouvrir
mode : disjonction bit-à-bit de:
un de : O_RDONLY, O_WRONLY, O_RDWR.
une combinaison de:
O_APPEND, O_TRUNC
O_CREAT, O_EXCL
O_NONBLOCK
...
droits : voir les permissions.
Retourne : erreur -1, descripteur de fichier sinon.

Les permissions sur un fichier

	USeR	GRouP	OTHers
Read	S_IRUSR	S_IWUSR	S_IXUSR
Write	S_IRGRP	S_IWGRP	S_IXGRP
eXecute	S_IROTH	S_IWOTH	S_IXOTH

```
S_IRWXU = S_IRUSR|S_IWUSR|S_IXUSR
S_IRWXG = S_IRGRP|S_IWGRP|S_IXGRP
S_IRWXO = S_IROTH|S_IWOTH|S_IXOTH
```

creat

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat(const char * fic, mode_t droits)
```

fic : le nom du fichier à ouvrir,
droits : voire les permissions.

Remarques : Équivalent à :

```
open(fic, O_WRONLY|O_CREAT|O_TRUNC,droits)
```


close

```
#include <unistd.h>
int close(int desc)
```

desc : le descripteur du fichier qu'on veut fermer

Retourne : 0/-1

read

```
#include <unistd.h>
ssize_t read(int desc, void * ptr, ssize_t nb_octets)
```

desc : descripteur d'un fichier ouvert en lecture
ptr : un pointer à un tampon en mémoire (déjà allouée)
nb_octets : le nombre d'octets qu'on veut lire, tel que

```
nb_octets < sizeof(ptr)
```

Retourne : nombre d'octets lus/-1

write

```
#include <unistd.h>
ssize_t write(int desc, void * ptr,
ssize_t nb_octets)
```

desc : le descripteur du fichier ouvert en écriture
ptr : un pointer à un tampon en mémoire (déjà allouée)
nb_octets : le nombre d'octets qu'on veut écrire

Retourne : nombre d'octets écrits/-1

Programme : copierunix.c

```
 1 : #include <stdlib.h>
 2 : #include <fcntl.h>
 3 : #include <unistd.h>
 4 : #include <stdio.h>
 5 : #define TAILLETAMPON 1024
 6 :
 7 : int main(int argc, char *argv[])
 8 : {
 9 :   int source, dest;
10 :   char tampon[TAILLETAMPON];
11 :   ssize_t nolu;
12 :
13 :   if (argc != 3)
14 :     {
15 :       printf("Usage : %s source destination\n",argv[0]);
16 :       exit(EXIT_FAILURE);
17 :     }
```


Programme : copierunix.c

```
18 :   if( (source = open(argv[1],O_RDONLY)) == -1
19 :        || (dest = open(argv[2],O_WRONLY)) == -1 )
20 :     {
21 :       perror("open");
22 :       exit(EXIT_FAILURE);
23 :     }
24 :   while((nolu = read(source,tampon,sizeof(tampon))) > 0)
25 :     write(dest,tampon,nolu) ;
26 :
27 :   close(source); close(dest);
28 :   exit(EXIT_SUCCESS);
29 : }
```


lseek

```
#include <unistd.h>
off_t int lseek(int desc, off_t offset, int origine)
```

desc : le descripteur du fichier ouvert en écriture
offset : déplacement par rapport à l'origine

Retourne : position courante à partir de l'origine du fichier,
`(off_t) -1` si erreur