6 - Le système de gestion de fichiers

F. Boyer, UJF-Laboratoire Lig, Fablenne.Boyer@imag.fr

- Interface d'un SGF
- Implémentation d'un SGF
  - Gestion de la correspondance entre la structure logique et la structure physique d'un fichier

Ce cours a été conçu à partir de…

- Cours de E. Berthelot
  - http://www.iie.cnam.fr/EBerthelot/
- Cours de A. Sylberschatz
  - www.sciences.univ-nantes.fr/info/perso/permanents/attiogbe/SYSTEME/CoursSysteme.html
- Cours de A. Griffaut
  - http://dept-info.labri.fr/~griffault/Enseignement/SE/Cours
- Cours de H. Bouzourfi, D. Donsez
  - http://www-adele.imag.fr/~donsez/cours/#se

Interface d'un SGF

- Concept de fichier
- Structuration
- Méthodes d'accès
- Protection
- Liens physiques et symboliques
- Montage de SGF
- Partage de fichiers

Concept de fichier

- Espace d'adressage *logiquement* contigu
- Types (Unix):
  - Ordinaire
    - Caractère (ascii)
    - Binaire
  - Répertoire
    - Contenu = { <nom symbolique, identification disque, adresse descripteur sur le disque> }
  - Spécial
    - Caractère (E/S série)
    - Bloc (disque)

Structure logique d'un fichier

- Aucune - séquence d'octets
- Enregistrements simples
  - Longueur fixe
  - Longueur variable (lignes)
- Enregistrements complexes
  - Document formatté
  - Fichier exécutable relogeable

Notion d'enregistrement pas forcément gérée par le SGF

Attributs d'un fichier

- Nom symbolique
- Type
- Localisation
- Taille
- Protection
- Dates

Protection : listes d'accès et groupes (Unix)

- 3 modes d'accès: read, write, execute
- 3 classes d'utilisateurs

	RWX		
a) owner access	7	⇒	1 1 1
b) group access	RWX 6	⇒	1 1 0
c) public access	RWX 1	⇒	0 0 1

- L'administrateur crée les groupes et ajoute les utilisateurs dans les groupes.

Attacher un groupe à un fichier : chgrp G game

Opérations sur les fichiers

- La création
- La destruction
- L'ouverture
- La fermeture
- La lecture
- L'écriture
- Le positionnement
- Le renommage

Accès à un fichier (unix)

- **Appels système :**
  - int open(char *name, int flag, mode_t mode)
  - int read(int fd, void *buf, size_t nbytes)
  - int write(int fd, void *buf, size_t nbytes)
  - int lseek((int fd, void *buf, size_t nbytes)
  - int fsync(int fd)
  - …
- **Fonctions librairie C**
  - FILE *fopen(char *name, char *type)
  - size_t fread(void *buf, size_t nbytes, size_t count, FILE *f)
  - size_t fwrite(void *buf, size_t nbytes, size_t count, FILE *f)
  - char * fgets(char *s, int nbytes_max, FILE *f)
  - int fputs(char *s, FILE *f)
  - int fflush(FILE *f)
  - …

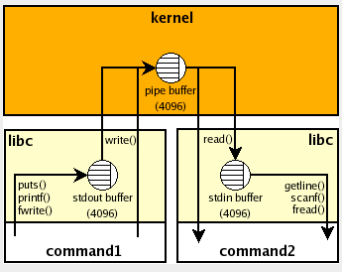
Différences appels système / fonctions systèmes

- **Appels systèmes: les entrées/sorties sont effectuées immédiatement (au niveau des buffers noyau ou sur le périphérique si le flag O_SYNC est positionné)**
  - fsync() provoque le vidage du buffer noyau sur le périphérique
- **Fonctions systèmes: les données passent par des buffers de la libC ; elles sont automatiquement flushées au niveau du noyau à chaque retour à la ligne (\n)**
  - fflush() provoque le vidage du buffer libC sur les buffers de niveau noyau

```
int f;
char msg[256];
if  ((f=open( "myfile.txt",
            O_CREAT|O_TRUNC|O_WRONLY))==0) {
 int n= write(f,msg,32);
 …
  close(f);
}
```

```
FILE *f;
char msg[256];
if ((f=fopen("myfile.txt","w"))!=NULL) {
    int n = fwrite(msg,sizeof(char),32,f);
    …
    fclose(f);
}
```


Usage des Buffers



http://www.pixelbeat.org/programming/stdio_buffering/

Copie de fichiers (exemple)

```
char buf[512];
int nb, nb_write, nb_read, src, dest;
…
if (src = open(« source », O_RDONLY) == -1) …
if (dest = open(« dest », O_WRONLY) == -1) …

while ((nb_read = read(src,buf, sizeof(buf))) != NULL){
  for (nb_write = 0; nb_write < nb_read; ) {
      if (nb = write(dest,buf(nb_write],nb_read-nb_write) ==-1)…
      nb_write += nb;
  }
}
…
```

Correct?

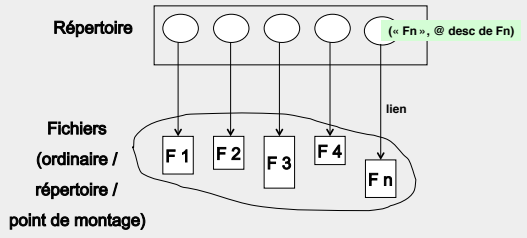
```
#include <stdio.h>
#include <string.h>
void main(void) {

    char* buf = "hello world";
    FILE *f = fopen( "test", "w");
    fwrite( buf, 1, strlen( buf), f);
}
```


Fichiers mappés

- Un fichier mappé en mémoire correspond à un fichier dont le contenu est chargé dans la mémoire virtuelle du processus courant
- Permet de modifier le fichier en lisant et en écrivant directement dans la mémoire (au lieu de passer par les buffers de la libC ou du noyau)

Concept de répertoire

- Un ensemble d'entrées contenant des informations sur des fichiers.

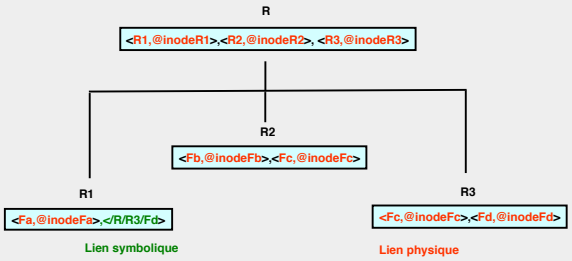


Attributs d'un répertoire

- Nom symbolique
- Localisation
- Taille
- Protection
- Dates

Graphes acycliques

→Partage de fichiers ou de répertoires
En utilisant un mécanisme de lien physique / symbolique



Opérations sur un répertoire

- La création
- La destruction
- L'ouverture
- La fermeture
- La lecture
- Le renommage
- La création d'un lien
- La destruction d'un lien

Utilisation du renommage

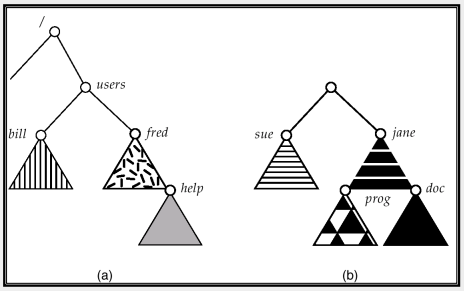
- Le renommage de fichier est atomique (Unix)
- Permet de mettre en oeuvre des traitements transactionnels

```
// Applicative program
   ...
   f_#.. = begin-session(myFile)
   <work on f_#..>
   commit-session(f_#.., myFile)
   ...


int begin-session(int f) {
   < copy f  in f_#..>
   <return f_#'s descriptor>
}

void commit-session(File new, File f) {
   fsync(new);
   move(new,  f);
}
```

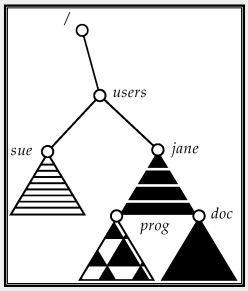

Montage de Système de fichiers (Partition)

- Partition : ensemble de fichiers soumis à des règles de gestion communes (sauvegardes, …)
- Une partition doit être *montée* sur un point de montage avant d'être accédée
  - Montage implicite (ex: clé USB, CD, ..)
  - Montage manuel
    - $mount -t <type > -o options /dev/repUSB /mnt/rep-montage
    - Si le montage est décrit dans le fichier /etc/fstab, la commande peut être simplifiée $mount /dev/repUSB ou mount /mnt/rep-montage
    - Types principaux: ext2 (par défaut), ext3, FAT16, FAT32, NFS, ..
- Plusieurs partitions peuvent être montées sur une arborescence commune

(a) Existant (b) Partition non montée



Montage

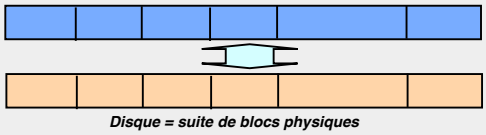


(Unix) mount <file system type> <identification partition> <point de montage>
mount –t ext2 /dev/sda1 /users

Implémentation d'un SGF

- Liaison ressource logique – ressource physique
  - Chargement et sauvegarde d'un fichier
  - Gestion des accès

Fichier = suite contiguë de blocs logiques



Disque = suite de blocs physiques

Allocation de l'espace disque

- Fichier :
  - Suite de blocs logiques contigüs(0..K)
  - Suite de blocs physiques non forcément contigüs

Méthode d'allocation : comment les blocs physiques sont alloués sur disque ?

détermine

Algorithmes d'accès à un élément du fichier :

AccesDirect(file f, int noEnreg) → (noBlocPhysique, depl)

Allocation par zones contigües de l'espace disque

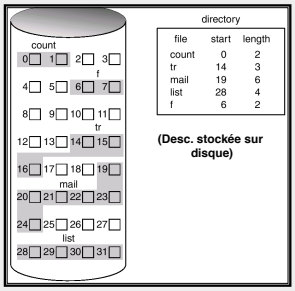
- Chaque fichier occupe une suite de blocs contigüs sur disque

+ Simple – seulement le no du bloc de départ et la longueur sont requis

+ Accès direct simple

- Fragmentation de l'espace (libérations)

- Les fichiers ne peuvent pas grossir



A. Sylberschatz

Accès direct

```
//Variable nbeb : nombre d'enregistrements par blocs phys. pour f
// Variable start : no du premier bloc phys. pour f

AccesDirect (file f, noEnreg i) :
    int B = i div f.nbeb;     // B = indice bloc logique
    return (f.start + B, i mod f.nbeb);
```


Allocation par zones contigües étendue

- Un groupement = une suite de blocs contigüs
- Un fichier = un ou plusieurs groupements indéxés

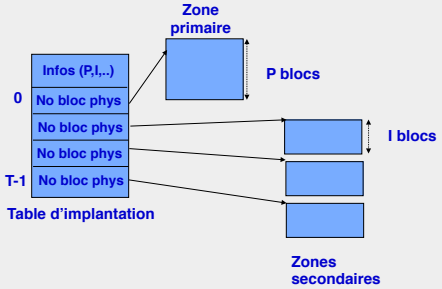
+ Un fichier peut grossir

- Fragmentation interne (unité d'allocation = le groupement)

- Taille de la table d'implantation

Utilisé par Veritas File System

Allocation par zones contigües étendue



Variante : zones secondaires de taille variable (WindowsNT, MacOS)

Accès direct

AccesDirect(file f, noEnreg i):
B = i div f.nbeb // indice bloc logique

0 <= B <= P-1 : // contiguité des blocs en zone primaire
return (f.TImpl[0] + B, i mod f.nbeb)

P <= B <= T-1 :
Z = (B-P) div l // l : taille zones secondaires
return (f.TImpl[Z+1] + (B-P) mod l, i mod f.nbeb)

Allocation par blocs

- **Allocation « à l'unité »**
- **Bloc : espace de taille fixe**

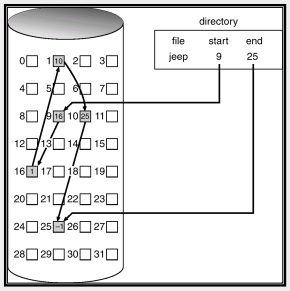
Allocation par blocs chaînés

- Chaque fichier est une liste de blocs chaînés sur disque.

\+ Simple – seulement le no du bloc de départ et de fin sont requis.

\+ Pas de fragmentation
\+ Les fichiers peuvent grossir

\- Accès séquentiel uniquement
\- Place utilisée pour les pointeurs



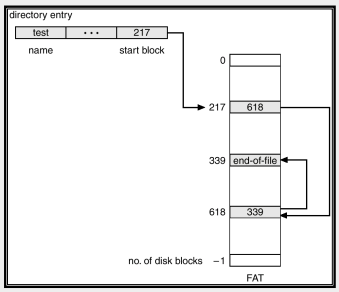
File-Allocation table (FAT)

Variante de l'allocation chaînée (MS-DOS, OS/2, W95, W98)
FAT[i] = numéro du bloc suivant du bloc i

Table globale à tous les fichiers, placée en début de disque

Contient une entrée par bloc du disque

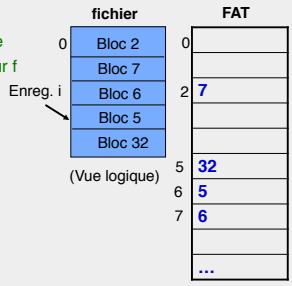
\+ Accès direct possible
\- Parcours table



A. Sylbetschatz

FAT

```
AccesDirect (file f, noEnreg i) :
  B = i div f.nbeb;  // indice bloc logique
  j = f.start ; // no du 1er bloc phys. pour f
  while B > 0 {
        j = FAT[j];
        B = B – 1;
  }
  return (j, i mod f.nbeb);
```



Ex : nbeb=8, i=26, B=3, j=2

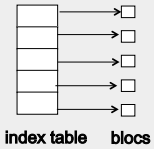
Allocation par blocs indéxés

- Chaque fichier occupe une suite de blocs quelconque.
- Un index référence chaque bloc.

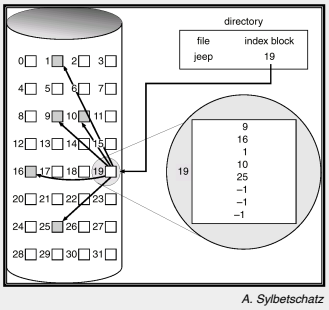
\+ Accès direct

\- Place utilisée pour l'index (index de taille fixe)

\- Fichiers de taille limitée



Allocation par blocs indéxés



A. Sylbetschatz

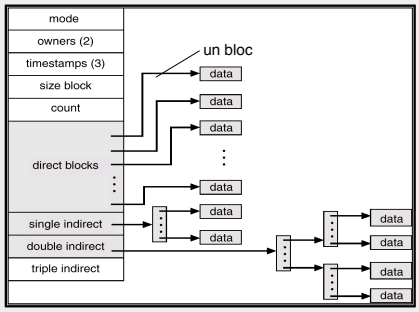
Techniques de fragmentation d'index

- **Chainage des fragments d'index**
  - La dernière entrée du fragment d'index d'indice *i* désigne le bloc contenant le fragment d'index d'indice *i+1*
- **Index multi-niveaux**
  - Les fragments d'index d'indice 0..K sont référencés dans un index de niveau supérieur
  - L'index de niveau supérieur peut lui-même être fragmenté
  - Organisation de type B-tree

Schéma combiné: le nœud d'information (i-node)

Une table de taille raisonnable décrit un fichier pouvant être gros.

En général, nombre de blocs direct = 10.



A. Sylberschatz

I-node Unix

- Type de fichier (ordinaire, catalogue, spécial)
- Nom et groupe du propriétaire
- Informations de protection
- Nombre de liens physiques
- Informations de localisation physique
- Taille
- Dates (création, modification, consultation, etc)

I-node

```
AccesDirect (file f, noEnreg i)
  B = i div nbeb;  // indice bloc logique
  if (B < 10) j = f.TABDIRECT[B]
  sinon
     B = B – 10
     si B < p alors   // p : nombre de blocs de l'index indirect_1
           lire_bloc (INDIRECT_1, TAB_LOCALE)
           j = TAB_LOCALE[B]
     sinon
           B = B – p
           si B < p * p alors // p*p : nombre de blocs de l'index indirect_2
                 lire_bloc(INDIRECT_2, TAB_LOCALE)
                 lire_bloc(TAB_LOCALE[B div p], TAB_LOCALE)
                 j = TAB_LOCALE[B mod p]
```

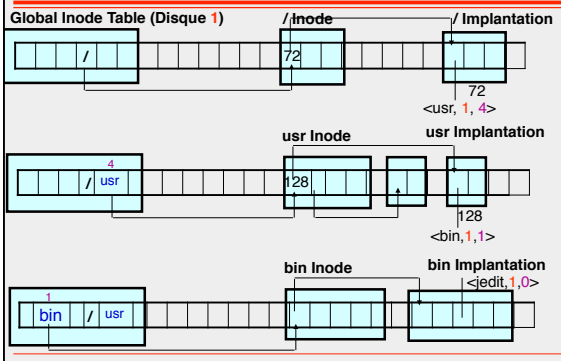

I-node

```
           sinon  // B < (p*p + p*p*p)
                 B = B – p * p
                 lire_bloc(INDIRECT_3, TAB_LOCALE)
                 lire_bloc(TAB_LOCALE[B div (p*p)], TAB_LOCALE)
                 B = B mod (p*p);
                 lire_bloc(TAB_LOCALE[B div p], TAB_LOCALE)
                 j = TAB_LOCALE[B mod p]   // numéro de bloc
  return (j, i mod f.nbeb)
```

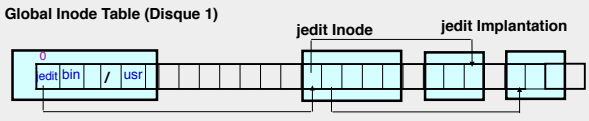
I-node Unix

- Temps d'accès rapide pour les fichiers de petite taille
- Un fichier qui occupe moins de 10 Ko a des numéros de blocs indirects nuls. Tous les accès disques seront immédiats
- Ces fichiers représentent 90% des fichiers !!!
- Pour les fichiers de taille comprise entre 10Ko et 266Ko, il y a un accès externe supplémentaire
- Pour les fichiers de taille comprise entre 266Ko et 64Mo, il y a deux accès externes
- Pour les fichiers de taille comprise entre 64Mo et 16Go, il y a trois accès supplémentaires

Exemple (/usr/bin/jedit)



Exemple (/usr/bin/jedit)



Exemple de SGF basés sur les inodes

- Ext2
  - SGF historique de Linux
  - Peu de fragmentation
  - Non robuste (arrêt intempestif / pb matériel / crash du noyau --> fsck + perte d'information)
  - Utilisé pour les supports de stockage flash

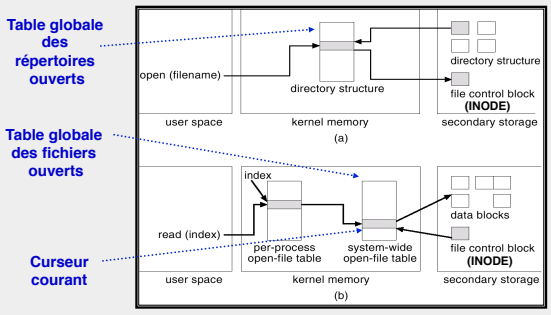
Exemple de SGF basés sur les inodes

- Ext3
  - SGF journalisé
  - Journalisation des méta-données [et des données]

    "mount -o data=journal"
    Journals all data and metadata, so data is written twice. This is the mode which all prior versions of ext3 used.

    "mount -o data=ordered"
    Only journals metadata changes, but data updates are flushed to disk before any transactions commit. Data writes are not atomic but this mode still guarantees that after a crash, files will never contain stale data blocks from old files.
- Ext4
  - Volume allant jusqu'à un exbioctet (2^{60} octets)
  - Journalisé
  - Allocation par *extent* qui permet la pré-allocation d'une zone contiguë pour un fichier

Structures d'accès à un fichier



A. Sylbetschatz

Lien entre fichiers et entrées-sorties

- Par convention, les flots d'ES standards (entrée, sortie, erreur) sont associés aux descripteurs 0, 1 et 2
- Les flots d'ES standards peuvent être ré-orientés vers des fichiers
- Dans le langage de commande, on ré-oriente les flôts d'ES standards au moyen de < et >

  cat fich > fich1c

  cat /dev/null > fich

Tubes

- Les tubes (*pipes*) permettent de faire communiquer des processus
- Un tube est un fichier anonyme qui sert de tampon entre deux processus fonctionnant en producteur-consommateur

  *cat *.c | grep var*
- Un tube a une capacité limitée (8Ko), au delà de laquelle un processus est bloqué pour produire dans le tube

.

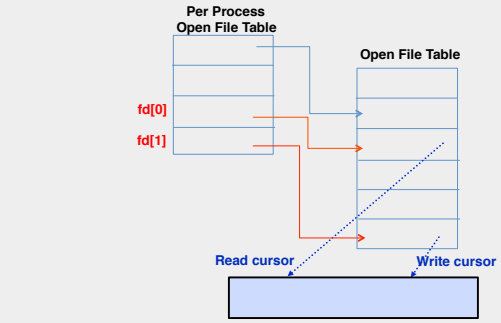
Création d'un tube

- **L'appel système *pipe()* crée un tube, dont l'entrée et la sortie sont associées à des descripteurs choisis par le système**

```
int fd[2];
pipe (fd);    // fd[0] for reading, fd[1] for writing
...
```

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

main() {
  int fd[2];
  pid_t childpid;

      pipe(fd);
      ...
```

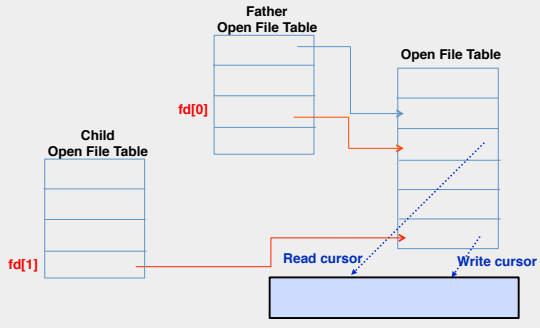

Création d'un tube



Exemple

```
...
main() {
  int fd[2];
  pid_t childpid;
  char readbuffer[126];
      pipe(fd);
      if((childpid = fork()) == -1)
          { perror("fork"); exit(1); }

      if(childpid == 0) {
          close(fd[0]);
          write(fd[1], « Hello », (strlen(« hello »)+1));
          exit(0);
      } else {
          close(fd[1]);
          nbytes = read(fd[0], buffer, sizeof(buffer));
          printf(« received : %s", buffer);
    }
. . }
```


Exemple



La copie de descripteurs

- ***dup(desc)*** **recopie** ***desc*** **dans le descripteur disponible de plus petit index**
- ***dup2 (desc, i)*** **recopie desc dans le descripteur d'index i**
- **Attention, les descripteurs copiés partagent le même curseur**
  - *Contrairement à open() qui crée une entrée dans la PPOFT et dans la OFT*

La copie de descripteurs

- **La copie de descripteurs permet de rediriger les flôts d'entrée-sortie**
- **Attention, les redirections effectuées sont transmises aux fils**
- **Exemple de redirection du flot de sortie**

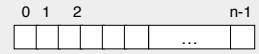
```
#include <fcntl.h>
    int fd;
    char *filename = "/tmp/file";
    fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, …);
    dup2(fd , 1);
    …
    close(fd);
```


Gestion de l'espace libre sur disque

- **Vecteur de bits**
- **Liste chaînée**
- **Liste de regroupement**

Vecteur de bits

- **Vecteur de bits (*n* blocs)**



bit[i] = { 0 ⇒ block[i] libre ; 1 ⇒ block[i] occupé }

+ Simple
+ Efficace pour l'allocation de *n* blocs contigus
- Doit résider en mémoire (taille importante)
→Variante : vecteur de comptage qui liste les couples (adr bloc, nb blocs libres successifs)

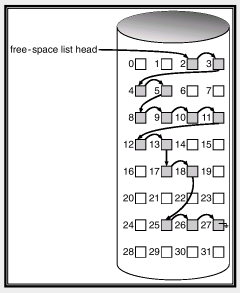
Solution utilisée dans Linux, exFAT

Liste chaînée des blocs libres

+ Simple

- Inefficace pour l'allocation d'un grand nombre de blocs

Solution utilisée dans MS-DOS



A. Sylbetschatz

Liste de regroupement de blocs libres

- Stocke les adresses des *n* blocs libres dans le premier bloc libre.
- Le dernier bloc contient les adresses de *n* autres blocs libres.

+ codage compact

+ recherche d'un grand nombre de blocs libres rapide

- gestion complexifiée

Exemple avec n=4

0	1	2	3	4	5	6	7	8
1,3,4,6	libre		libre	libre		8,12,..		