Rutines Dependents

Fitxers

1. Estructures de dades per gestió de fitxers

1. Com l'accés és seqüencial, cada cop que s'obre un fitxer es crea un ↑`L/E` nou. On podem guardar-lo?

    - A nivell BFS:

      NO, per què a nivell BFS l'accés ja és directe, especificant el número de registre o bloc de 512 bytes:

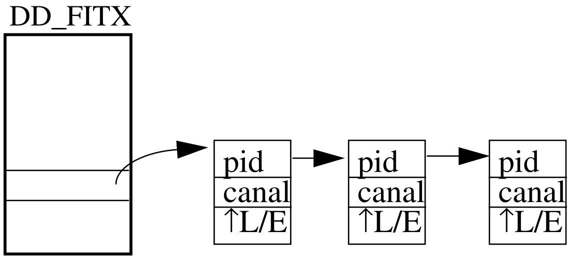
      llegir_fitxer_bfs (int id_fitxer, char * buff, int n_reg)

      Ha d'estar per tant a un nivell superior ⇒ a nivell sistema

    - A nivell SISTEMA: Dues possibles opcions són:

      - **Opció A:**
        Tots els processos que obren un mateix fitxer comparteixen el mateix Descriptor de Dispositiu de Fitxer (DD_FITX).

        Una cua associada al DD_FITX guardaria el valor del ↑L/E de cada procés i canal.

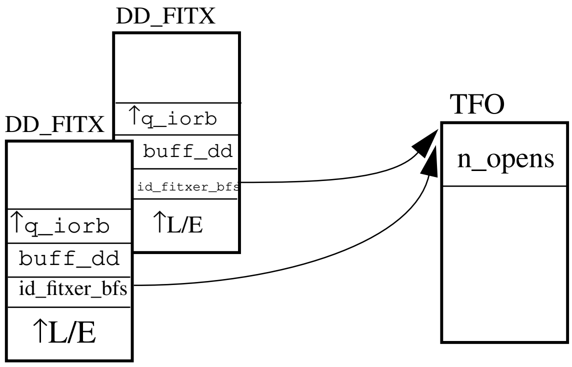
        

        Els accessos serien molt lents a l'hora de realitzar L/E.

      - **Opció B:** (Aquesta és la solució adoptada)
        Crear un **nou DD_FITX cada cop que s'obre** un fitxer, cadascun amb el seu ↑`L/E`.

        Si dos DD_FITX corresponen a un mateix fitxer compartiran l'entrada a la Taula de Fitxers Oberts (TFO) del BFS: la informació comuna, tal com el propietari o les proteccions, es guarda un sol cop a la TFO (que està o nivell BFS); l'índex de l'entrada es fa servir com identificador del fitxer per a les operacions del nivell BFS.

.



2. Degut a que es crea un nou DD_FITX cada cop que s'obre un fitxer podem tenir molts DD_FITXs. La cua d'*iorb*s i la cua d'*io_fin* no es guarden al DD_FITX sinó que són globals, comuns per a tots els fitxers, estaran al nivell de sistema i al DD_FITX guardarem **apuntadors** a elles.

3. Les transferències de disc són de 512 bytes. Els bytes en excés respecte al que hagi demanat l'usuari es guarden al DD_FITX (al camp *buff_dd*) de forma que en posteriors operacions de lectura/escriptura els caracters s'agafen d'allí, evitant així accessos al disc innecessaris (tècnica de **Buffering**).

Nivell SISTEMA

DD_FITX

```
/* Part comú */
char nom[12];
int caract;
int opens;                 /* herencies */
int sem_mutex;
int sem_gestor_fitx;
int (* p_obrir)();
...
/* Part no comú */
struct cua * q_io_fin;
struct cua * q_iorb;
int modalitat;             /* Modalitat d'accés (L/E)*/
char buff_dd[512];
long pL_E;
int id_fitxer_bfs;
long longitud;           /* Longitud en bytes del fitxer */
```

- **id_fitxer_bfs** és un identificador intern del SO usat per a accedir a aquest fitxer. És vàlid mentre el fitxer romangui obert.

- **pL_E** és el punter de Lectura/Escriptura: indica el desplaçament des de l'inici del fitxer. Permet calcular el número de registre (pL_E div 512) i la posició dintre del registre (pL_E mod 512).

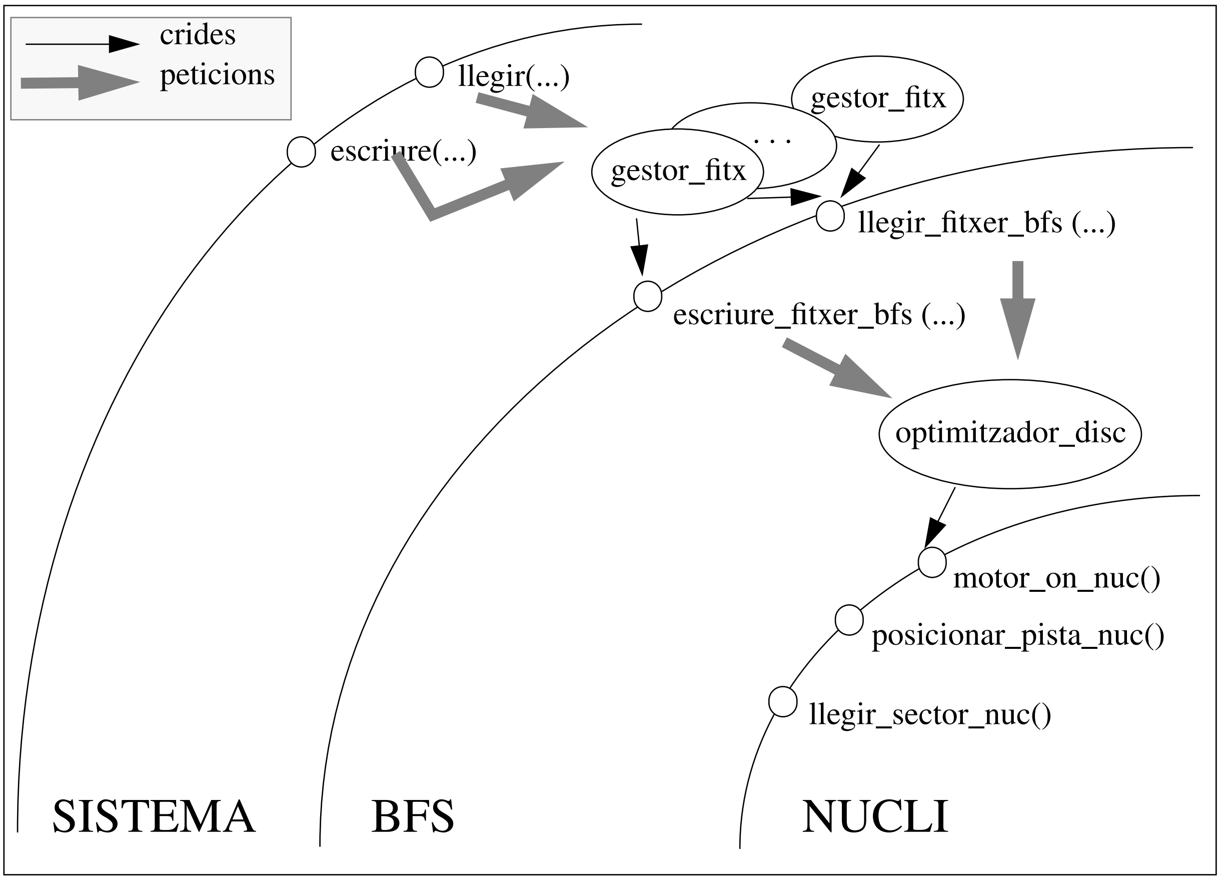
4. Tindrem gestors a nivell BFS i SISTEMA.

   - Hi haurà un gestor a nivell BFS que anomenarem **Optimitzador de Disc**.

     Aquest gestor atendrà les peticions d'accés al disc tractant d'optimitzar els accessos. Si no es realitzessin peticions de transferència de disc de forma concurrent no caldria l'optimitzador de disc.

   - Hi hauran N **gestors de fitxer** a nivell SISTEMA.

     Aquests gestors en general serviran les peticions de E/S sobre fitxers a partir de la informació que hi al buff_dd. Si el nombre de bytes a llegir excedeix 512 caldrà realitzar diverses crides a la rutina *llegir_fitxer_bfs*. Per evitar que una petició de lectura amb tamany molt gran col.lapsi el sistema s'introdueixen diversos gestors de fitxer a nivell SISTEMA.



Els gestors de cada nivell hauran de treballar amb estructures de dades diferents:

- Els **gestors de fitxer** de nivell sistema treballen amb la cua d'*iorb*s i la cua d'*io_fin* de fitxers que es troben a nivell SISTEMA, servint les peticions mitjançant crides a les rutines del nivell BFS quan sigui necessari (llegir i/o escriure).

  Com aquests gestors treballen indistintament amb diversos DD_FITXs cal tenir un camp a l'*iorb* per indicar al gestor de fitxers amb quin DD_FITX ha de treballar: *struct dd * eldd*

- Les rutines de nivell BFS donaran el seu servei fent peticions al gestor de BFS **optimitzador de disc**. Caldrà per això tenir unes cues específiques per a fer les peticions i per al retorn de resultats a aquest nivell. Anomenarem aquestes cues *q_iorb_bfs* (que encua estructures de tipus *struct iorb_bfs*) i *q_io_fin_bfs*.

Nivell BFS

```
struct iorb_bfs
{
   int cara;
   int pista;
   int sector;
   char * pbuff_dd;
   int operacio;
   int sem_proces;
}
```

2. Obrir fitxer

La rutina obrir de nivell sistema crida a obrir_fitxer_bfs si el nom correspon al nom d'un fitxer.

```
int obrir (int canal, char * nom, int mod, char prot)

int obrir_fitxer_bfs (char * nom, int crear, int uid_proc,
                      int mod, long * lon, int * id_fitxer)
```

Les estructures de dades que afecten i es veuen afectades per l'obertura d'un fitxer són la Taula de Fitxers Oberts (TFO) i el Directori (DIR). Els camps d'aquestes estructures són:

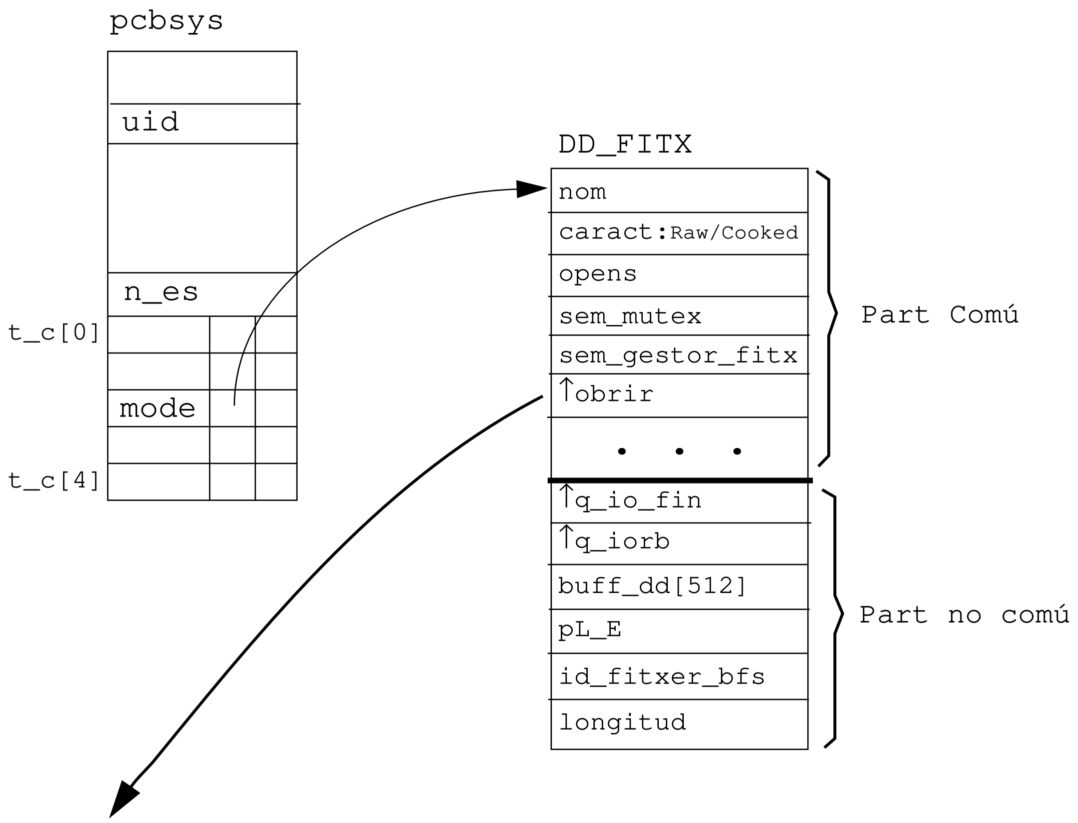
```
struct entrada_tfo
{
   /*Informació dinàmica */
   int n_opens;
   int pendent_esborrar;
   int num_entrada_DIR;

   /*Informació estàtica:
      còpia entrada_dir */
   struct entrada_dir ent_dir;
}
```

```
struct entrada_dir
{
   char nom[8];
   char extensio[3];
   char proteccio;
   int propietari;
   long tamany;
   int primer_cluster;
   ...
}
```

- **n_opens** indica el nombre de vegades que el fitxer ha estat obert ⇒ Cada cop que obrim el fitxer cal incrementar el nombre d'opens ja què per un cert fitxer sempre utilitzem la mateixa entrada de la TFO.

- **pendent_esborrar** és una marca que indica que el fitxer està pendent de ser esborrat per què s'ha fet la crida a esborrar quan el fitxer encara estava obert: s'esborrarà al tancar-lo l'últim procés que el tenia obert

- **num_entrada_DIR** indica on es pot trobar la informació al disc: permet accedir directament a l'entrada de directori del disc quan calgui actualitzar la seva informació: tamany, data darrera modificació, ...




```
obrir_fitxer (eldd,mode,prot)

{
   ...
   if (mode ⊂ write)
      crear = prot;
    else
      crear = NOCREAR;
   ...
   aux = buscar_pcb(quisoc());
   uid_proc = aux->uid;
   err = obrir_fitxer_bfs(nom, crear, uid_proc, mode,
                              &tam, &id_fitxer);
   if (err < 0) return (err);

   eldd.id_fitxer = id_fitxer;
   eldd.longitud = tam;
   eldd.L_E = 0;
   ...
}
```

```
obrir_fitxer_bfs (nom, crear, uid_proc, mod, lon, id_fitxer)
char * nom;        /* Fitxer a obrir */
int crear;          /* Utilitzat si el fitxer ¬$: crear-lo o no */
int uid_proc;      /* uid del proces que fa la crida */
int mod;           /* modalitat amb la que s'accedira al fitxer */
long * lon;        /* Valor retorn: longitud (bytes) del fitxer */
int * id_fitxer;   /* Valor retorn: Identificador per a futurs
                        accessos a aquest fitxer.
                        Vàlid mentre el fitxer romangui obert */
{
 int n_entry, n_entry_dir;
 struct entrada_dir entry_dir;

 /* Si el fitxer ja està obert:
        validar accés i utilitzar la mateixa entrada de la TFO */
 if ((n_entry = buscar_entrada(TFO, nom)) >= 0)
 {
  if (TFO[n_entry].pendent_esborrar) return(-1);
  if ((validar_acces(uid_proc,mod,&TFO.ent_dir) < 0) return(-4);

  TFO[n_entry].n_opens++;          /* Nombre de vegades que s'ha obert */
  (*lon) = TFO[n_entry].ent_dir.tamany;
  (*id_fitxer) = n_entry;
  return (0);
 }

 /* El fitxer no estava obert: buscar i omplir entrada de TFO */
 if ((n_entry = obtenir_entrada_lliure(TFO)) < 0) return(-3);
     /* Excedit el nombre maxim de fitxers oberts */

 /* Si el fitxer existeix: copiar informació del directori */
 if (!crear) {
  if (buscar_fitxer_al_DIR(nom, &n_entry_dir, &entry_dir) <0)
        return(-1);                  /* Fitxer no existent */
  if ((validar_acces(uid_proc, mod, &entry_dir) < 0) return(-4);
  copiar_entrada(&entry_dir, &TFO.ent_dir);
  TFO[n_entry].n_opens = 1;
  TFO[n_entry].pendent_esborrar = 0;
  TFO[n_entry].id_fitxer = n_entry;
  (*lon) = TFO[n_entry].ent_dir.tamany;
  (*id_fitxer) = n_entry;
  return (0);
 }

 /* El fitxer no existeix: crear entrada a la TFO i al DIR*/
 if (buscar_fitxer_al_DIR(nom, &n_entry_dir, &entry_dir) == 0)
          return(-5);                /* Fitxer ja existent */
 if ((n_entry_dir = obtenir_entrada_lliure(DIR)) < 0)
          return(-2);                /* Directori ple */

 crear_entrada_dir(&entry_dir, nom, uid_proc);
 copiar_entrada(&entry_dir, &TFO.ent_dir);
 TFO[n_entry].n_opens = 1;
 TFO[n_entry].pendent_esborrar = 0;
 TFO[n_entry].id_fitxer = n_entry;
 (*lon) = 0;
 (*id_fitxer) = n_entry;
 actualitzar_DIR(DIR, &entry_dir);
 return (0);
}
```

3. Llegir fitxer

A partir de la rutina dependent *llegir_fitx* es fa la petició als gestors de fitxer. Tots aquests esperen del mateix semàfor *sem_gestor_fitx* amb el qual un d'ells atendrà la petició.

```
llegir_fitx (eldd, buff, lon)
struct dd * eldd;
char * buff;
long * lon;
{
   obtenir id_io;
   obtenir i omplir iorb;
   encuar(eldd.q_iorb, iorb);
   signal (sem_gestor_fitx);
   return (id_io);
}
```

Després d'encuar la petició retorna. El funcionament és **asíncron**.

```
gestor_fitx()
{
   for (;;)
   {
      wait (sem_gestor_fitx);
      extreure iorb;

      operacio_es();

      lliberar_iorb;
      encuar io_finalitzada;
      signal (id_io);
   }
}
```

Veurem tot seguit com realitza el gestor de fitxer l'operació d'E/S.

Quan el gestor fa la crida a la rutina de nivell BFS es realitzen lectures de 512 bytes.

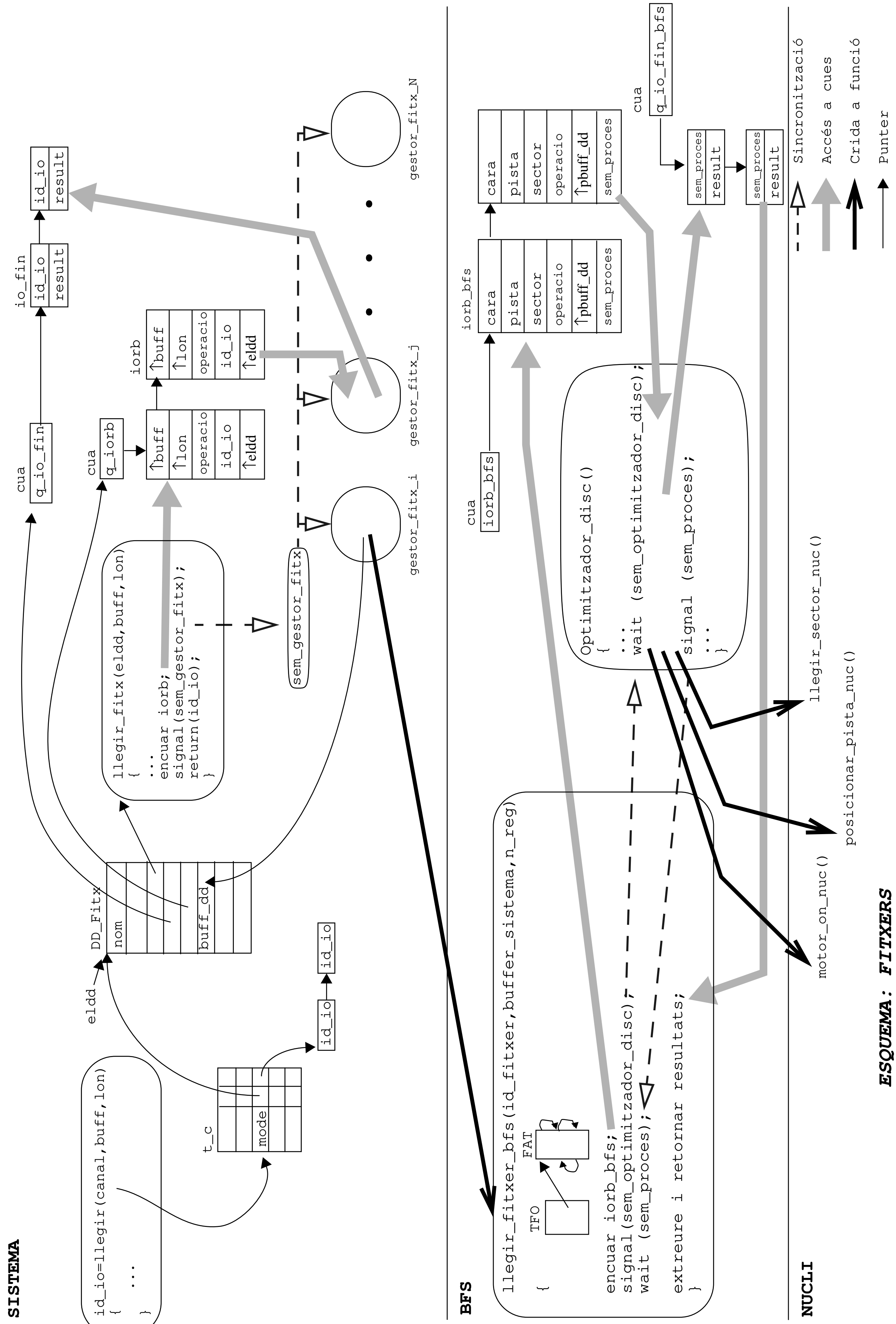
```
operacio_es()
{
  if (∃ suficients caracters a buff_dd per atendre peticio)
    copiar caracters de buff_dd a buffer usuari
   else {                          /* crides a BFS (síncrones) */
    while (Hi hagin registres per llegir)
    {
     eldd = piorb->eldd;
     /* n_reg ⇒ Acces sequencial en funcio de ↑L/E */
     nreg = seg_reg(eldd->pL_E);
     res = llegir_fitxer_bfs((eldd->id_fitxer_bfs,
                   eldd->buff_dd, n_reg);
     passar caracters de buffer_dd a buffer usuari;
     actualitzar punters;
    }
   }
}
```

Llegir_fitxer_bfs té un comportament síncron. És aquí on es calcula l'adreça de disc a partir de la TFO i la FAT.

```
int llegir_fitxer_bfs (id_fitxer, buffer, n_reg)
{
 if (!existeix id_fitxer a TFO) return (-1);
       /* Identificador no valid o Fitxer no obert */
 else {
   n_cluster = recorrer_FAT(1er cluster, n_reg);
   calcul_adreça_disc(n_cluster, &cara, &pista, &sector);
   obtenir, omplir i encuar iorb_bfs;
   signal (sem_optimitzador_disc);
   wait (sem_proces);
   extreure resultat de la cua q_io_fin_bfs;
   return (resultat);
 }
}
```

L'optimitzador de disc serveix una per una les peticions que fan concurrentment els gestors de fitxer.

```
optimitzador_disc()
{
 for (;;)
 {
 wait (sem_optimitzador_disc);
 extreure iorb_bfs ;        <-- segons política utilitzada
                                (SCAN, C-SCAN, FCFS, ... )
 operacio_es_nuc();         <-- Fa servir rutines del nucli:
                                motor_on_nuc;/* si parat */
                                posicionar_pista_nuc;
                                llegir_sector_nuc;
 lliberar iorb_bfs;
 encuar resultat a q_io_fin_bfs;
 signal (sem_proces);
 }
}
```


SISTEMA
id_io=llegir(canal,buff,lon)
{
...
}
t_c
mode
id_io
id_io
eldd
DD_Fitx
nom
buff_dd
llegir_fitx(eldd,buff,lon)
{
...
encuar iorb;
signal(sem_gestor_fitx);
return(id_io);
}
cua
q_io_fin
cua
q_iorb
iorb
↑buff
↑lon
operacio
id_io
↑eldd
io_fin
id_io
result
sem_gestor_fitx
gestor_fitx_i
gestor_fitx_j
gestor_fitx_N
BFS
llegir_fitxer_bfs(id_fitxer,buffer_sistema,n_reg)
{
TFO
FAT
encuar iorb_bfs;
signal(sem_optimitzador_disc);
wait (sem_proces);
extreure i retornar resultats;
}
cua
iorb_bfs
iorb_bfs
cara
pista
sector
operacio
↑pbuff_dd
sem_proces
Optimitzador_disc()
{
...
wait (sem_optimitzador_disc);
signal (sem_proces);
...
}
cua
q_io_fin_bfs
sem_proces
result
NUCLI
llegir_sector_nuc()
posicionar_pista_nuc()
motor_on_nuc()
Sincronització
Accés a cues
Crida a funció
Punter
ESQUEMA: FITXERS