T4-Entrada/Sortida

SO-Grau 2010-2011

Llicència

Índex

- Conceptes
- Gestió de l'E/S
- Gestió de l'E/S a linux
  - Subsistema d'E/S a linux
  - Operacions d'E/S a linux
- Optimitzacions de l'E/S

Índex

- **Conceptes**
  - ***Què és l'E/S***
  - ***Dispositius d'E/S***
  - ***Accés als dispositius d'E/S***
- **Gestió de l'E/S**
  - ***Tipus de dispositius***
  - ***Independència dels dispositius***
  - ***Subsistema d'E/S***
  - ***Exemples d'operacions***
  - ***Exemples de dispositius***
- Gestió de l'E/S a linux
  - Subsistema d'E/S a linux
  - Operacions d'E/S a linux
- Optimitzacions de l'E/S

1. Què és l'E/S?
2. Dispositius d'E/S
3. Accés als dispositius d'E/S

CONCEPTES

Què és l'E/S?

- **Definició**: transferència d'informació entre un procés i l'exterior
  - Entrada: de l'exterior al procés
  - Sortida: del procés a l'exterior

  (sempre des del punt de vista del procés)

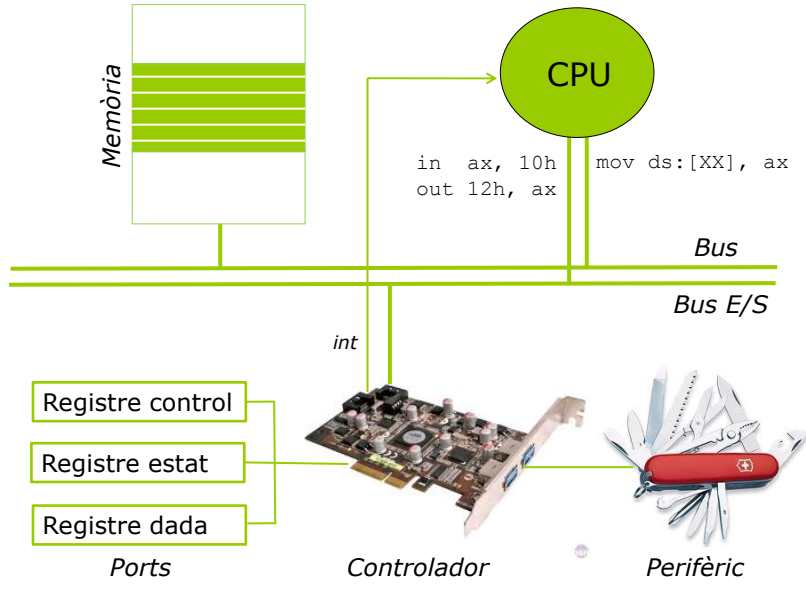
- De fet, bàsicament, els processos realitzen càlcul i E/S
- Molts cops, inclús, l'E/S és la tasca principal d'un procés:
  p.ex. navegació web, intèrpret de comandes, processador de texts

- **Gestió de l'E/S**: administrar el funcionament dels dispositius d'E/S (perifèrics) per a un ús correcte, compartit i eficient dels recursos

Dispositius d'E/S

...

Accés als dispositius d'E/S



1.8

1. Tipus de dispositius
2. Independència dels dispositius
3. Subsistema d'E/S
4. Exemples d'operacions
5. Exemples de dispositius

GESTIÓ DE L'E/S

"Administrar el funcionament dels dispositius d'E/S per a un ús correcte, compartit i eficient dels recursos"

Tipus de dispositius

- Dispositius d'interacció amb l'usuari (pantalla, teclat, mouse), emmagatzemament (disc dur, DVD, pen drive), transmissió (mòdem, xarxa amb cable, xarxa sense cable) o inclús d'altres més especialitzats (controlador d'un avió, sensors, robots) … moltíssima varietat!
- Classificació segons:
  - Tipus de dispositiu: lògic, físic, de xarxa
  - Velocitat d'accés: teclat vs disc dur
  - Flux d'accés: mouse (byte) vs DVD (bloc)
  - Exclusivitat d'accés: disc (compartit) vs impressora (dedicat)
  - Altres…
- Dilema: estandardització vs nous tipus de dispositius

Independència dels dispositius

Independència: principis de disseny

- **Uniformitat** de les operacions d'E/S
  - Accés a tots els dispositius mitjançant les mateixes crides al sistema
  - Augmenta la simplicitat i la portabilitat dels processos d'usuari
- **Dispositius virtuals**
  - El procés no especifica sobre quin dispositiu treballa: utilitza dispositius virtuals
- **Redireccionament**
  - El sistema operatiu permet canviar l'assignació dels dispositius virtuals abans d'iniciar un procés
  - Permet que el mateix programa s'executi sobre diferents dispositius sense canviar el codi

Independència dels dispositius (1)

- Possibilitat de realitzar E/S sobre un dispositiu sense fixar quin serà

```
% comanda < disp1 > disp2
```

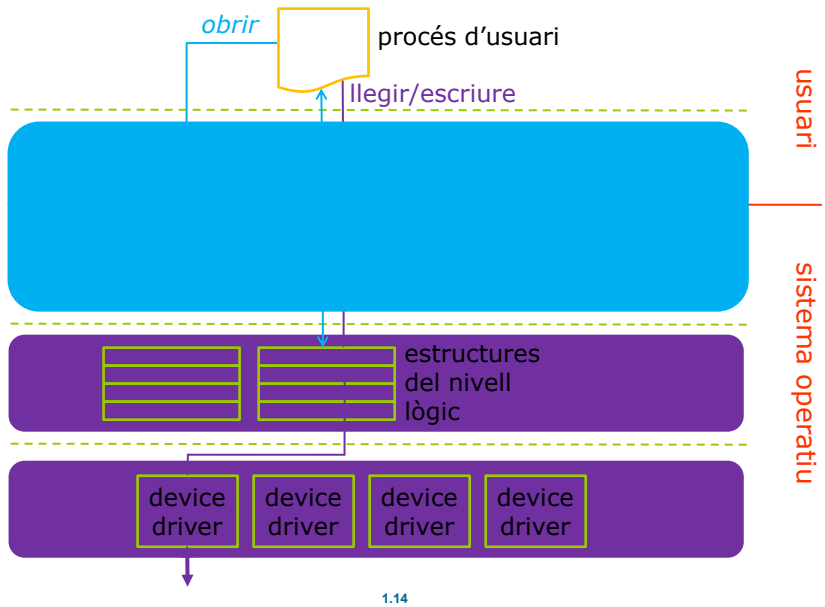
- Per això, habitualment, disseny en tres nivells: virtual, lògic i físic
- **Virtual**: Aïlla l'usuari de la complexitat de gestió dels dispositius físics
  - Estableix correspondència entre nom simbòlic i l'aplicació d'usuari, a través d'un dispositiu virtual
    - A Linux s'anomena canal o descriptor de fitxer (*file descriptor*)
      - És un enter
      - Els processos tenen 3 canals estàndard: entrada (0), sortida (1), error (2)
  - Proporciona les funcions genèriques d'accés als dispositius
    - A Linux són les crides al sistema d'E/S
    - Cal una funció específica que faci aquesta associació: *obrir (open)*
  - Per tant, el nivell virtual permet que una aplicació sigui independent dels dispositius físics que hagi d'utilitzar

Independència dels dispositius (2)

- **Lògic**: Implementa tasques independents dels dispositius
  - Estableix correspondència entre nom simbòlic i el controlador adequat
  - Gestiona dispositius que poden tenir, o no, representació física
    - P.ex. pipe, dispositiu nul
  - Manipula blocs de dades de tamany independent
  - Proporciona una interfície uniforme al nivell físic
- **Físic**: Implementa a baix nivell les operacions abstractes sol·licitades
  - Tradueix paràmetres del nivell abstracte a paràmetres concrets
    - P.ex. En un disc, traduïr pL/E a cilindre, cara, pista i sector
  - Inicialitza dispositiu. Comprova si lliure; altrament posa petició a cua
  - Realitza la programació de l'operació demanada
    - Pot incloure examinar l'estat, posar motors en marxa (disc), …
  - Espera, o no, a la finalització de l'operació
  - Retorna els resultats o informa sobre algun eventual error

1.13

Independència dels dispositius (3)



1.14

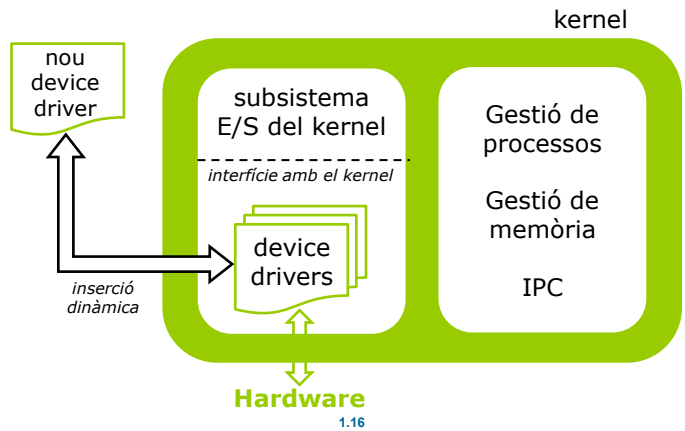
Independència dels dispositius (4)

- D'aquesta manera…
- …es poden fabricar nous dispositius sense modificar el kernel
  - Cal proporcionar (el fabricant) les rutines d'accés a baix nivell als dispositius
    - *Device driver* (pot incloure codi en assemblador)
  - Cal proporcionar una interfície d'accés a les operacions d'E/S
    - *System calls, APIs*
- …es poden afegir nous dispositius sense modificar el kernel
  - Cal que el SO ofereixi un mecanisme d'inserció dinàmica de codi
    - Mòduls*, plug & play*
- …es poden escriure programes que realitzin E/S sobre dispositius (virtuals) sense especificar quins (físics). En el moment d'executar el programa es determina sobre quins dispositius treballa
  - Possibilitat de redireccionament dels canals d'entrada/sortida
    - P. Ex. Que la entrada sigui la consola o un fitxer

1.15

Subsistema d'E/S

- Aïlla, la resta del kernel, de la complexitat de la gestió dels dispositius
- A més, protegeix el kernel en front d'un codi escrit per "altres"
- S'identifiquen unes operacions comunes (interfície) i les diferències específiques s'encapsulen en mòduls del SO (*device driver*)



1.16

Mòduls d'E/S carregats dinàmicament

- Mòdul: sistema que ofereix el kernel per afegir codi i dades al kernel sense necessitat de recompilar-lo
- El subsistema de gestió de l'E/S ha de permetre:
  - Encapsulament de les operacions (*devide driver*, *API*)
  - Mecanisme de càrrega dinàmica de codi (mòduls, *plug & play*)
- El fabricant proporciona, juntament amb el dispositiu:
  - Les rutines de baix nivell que gestionen el controlador
  - Un conjunt de rutines que implementen les diferents operacions, segons la interfície definida en cada sistema operatiu

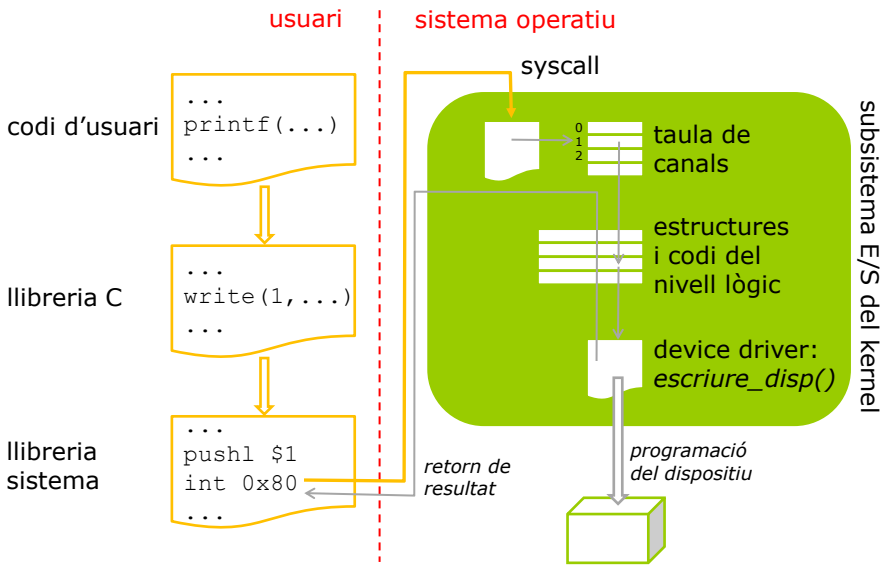
Mòduls d'E/S carregats dinàmicament (II)

- Funcionament:
  - Instal·lar (inserir) en temps d'execució les rutines del *driver*
    - El dispositiu s'identifica amb 2 nombres: *major* (tipus de dispositiu) i *minor* (instància de dispositiu del tipus major)
      - `insmod fitxer_amb_codi_acces_dispositiu`
  - Crear un dispositiu lògic (nom d'arxiu en el FS) i lligar-lo amb el *driver*
    - A través del *major* i del *minor*
    - Utilitzant la comanda *mknod*
      - `mknod /dev/mydisp c major minor`
  - El dispositiu ja està llest per ser utilitzat, mitjançant el nom d'arxiu
    - `open("/dev/mydisp", ...);`

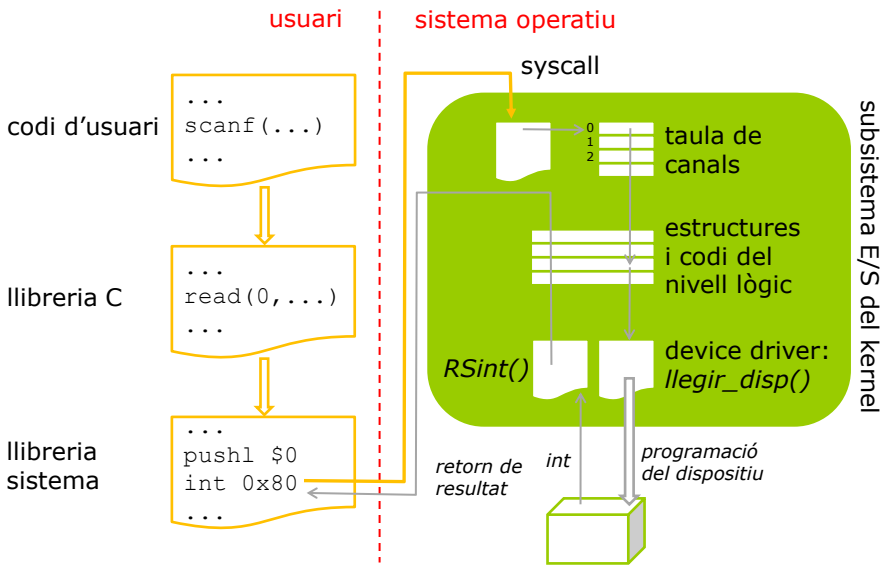
Device Driver

- Continguts d'un *Device Driver* (DD)
  - Informació general sobre el DD: nom, autor, llicència, descripció, ...
  - Implementació de les funcions genèriques i específiques d'accés als dispositius
  - Estructura de dades amb una llista d'apuntadors a les funcions
  - Funció d'inicialització
    - S'executa en instal·lar el DD
    - Registra el DD en el sistema, associant-lo a un *major* i *minor*
    - Associa les funcions genèriques al DD registrat
  - Funció de desinstal·lació
    - Desregistra el DD del sistema i les funcions associades
- Instal·lació del *Device Driver*
  - Es compila, si escau, en un format determinat: .ko (*kernel object*)
  - S'inserta dinàmicament al kernel
    - `insmod device_driver_name.ko`

Exemple: Escriptura a un dispositiu



Exemple: Lectura d'un dispositiu



1.21

Ops bloquejants i no bloquejants (1)

- Operació d'E/S **bloquejant**: Executa l'operació i retorna quan hagi acabat
  - Si pot completar l'operació amb les dades demanades, ho fa. Altrament ho farà amb les disponibles. Retorna el nombre de dades
  - Si no hi ha dades disponibles, bloqueig
    - Es mou el procés a una cua d'espera
    - Es posa en execució el primer procés de la cua de preparats
    - Quan arriba la dada es produeix una interrupció
      - La RSI recull la dada i posa el procés a la cua de preparats
    - Quan toqui, el procés de la cua de preparats es posarà en execució i li seran lliurades les dades retornades per la crida al sistema
  - Si lectura, quan *EOF* acaba la lectura i retorna 0
    - El significat d'*EOF* depèn del tipus de dispositiu
  - Normalment les operacions són bloquejants per defecte

1.22

Ops bloquejants i no bloquejants (2)

- Operació d'E/S **no bloquejant**
  - Executa l'operació i retorna tant si hi ha dades com si no
  - Retorna el nombre de dades llegides o escrites
- Operació d'E/S **no bloquejant asíncrona**
  - Es realitza una petició de dades, però no necessàriament es requereixen pel mateix moment; es segueix executant codi
  - Quan arriben les dades, pot passar:
    1. Es fixa un valor en alguna variable de l'aplicació
    2. Es produeix una interrupció software (excepció)

Exemples de dispositius (1)

- Funcionament d'alguns dispositius lògics: consola, fitxer, pipe, socket

1. **Consola**
   - Objecte a nivell lògic que representa el conjunt teclat+pantalla
   - "Habitualment" els processos el tenen a l'E/S estàndard per defecte
   - Lectura: El procés es bloqueja fins que l'usuari pitja i valida (*enter*) les tecles. Quan es pitja ^D provoca un *EOF*
   - Escriptura: S'escriu el text a pantalla sense bloquejar el procés
2. **Fitxer de dades**
   - Objecte a nivell lògic que representa informació emmagatzemada a disc
   - El sistema manté, a nivell lògic, un punter de Lectura/Escriptura que permet realitzar accés seqüencial als fitxers
   - Lectura: El procés es pot bloquejar fins que s'acaba la lectura. Retorna el nombre de bytes llegits. Si *EOF* no es bloqueja i retorna 0 caràcters llegits
   - Escriptura: El procés es bloqueja (o no) fins que acaba l'escriptura. Retorna el nombre de bytes escrits o -1 en cas d'error.

Exemples de dispositius (2)

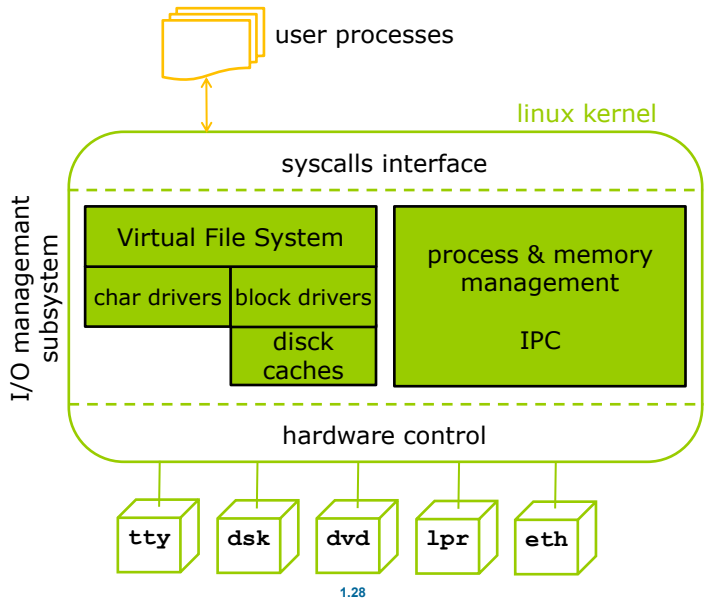
- Pipe
  - Objecte a nivell lògic que implementa un buffer temporal amb funcionament FIFO. Les dades de la pipe s'esborren a mesura que es van llegint. Serveix per intercanviar informació entre processos
  - Pipe sense nom, connecta processos amb parentest (herència)
  - Pipe amb nom permet connectar qualsevols processos (file system)
  - Lectura: Es bloqueja el procés fins que hi hagi dades a la pipe, tot i que no necessàriament la quantitat de dades demanades. Quan no hi ha cap procés que pugui escriure i la pipe està buida provoca un EOF
  - Escriptura: El procés escriu, a no ser que la pipe estigui plena. En aquest cas es bloqueja fins que es buidi. Quan no hi ha cap procés que pugui llegir el procés rep una excepció del tipus SIGPIPE
- Socket
  - Objecte a nivell lògic que implementa un buffer temporal amb funcionament FIFO. Serveix per intercanviar informació entre processos que es trobin en diferents computadores connectades per alguna xarxa
  - Funcionament similar a les pipes, tot i que la implementació interna és molt més complexa (implementació del protocol IP)

Índex

1. Subsistema d'E/S a linux
   1. Estructures de dades
   2. Espai de noms
2. Operacions d'E/S a linux
   1. Descripció de les operacions bàsiques d'E/S
   2. E/S i execució concurrent de processos
   3. Exemples

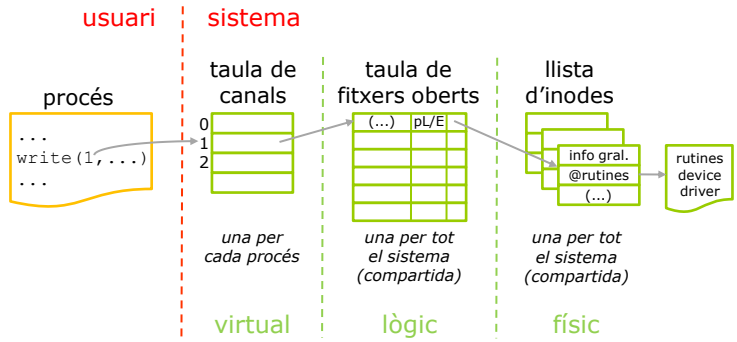
GESTIÓ DE L'E/S A LINUX

Subsistema d'E/S a linux



Estructures de dades

- 3 nivells:
  - **Taula de canals**: per cada procés, indica a quins fitxers pot accedir
    - S'accedeix al fitxer mitjançant el canal, que és un índex a la TC
  - **Taula de fitxers oberts**: indica els fitxers oberts en tot el sistema
  - **Taula (llista) d'inodes**: conté informació sobre cada objecte físic obert, incloent les rutines del DD



1.29

Espai de noms: VFS (1)

- Tots els dispositius tenen un nom al VFS (*tema 5*)
  - Això permet que qualsevol procés hi pugui accedir
  - Generalment es troben al directori `/dev`
- Per tant, es poden considerar com **fitxers especials**. Cada un té associat:
  - **major**: identificador de tipus de dispositiu
  - **minor**: instància de dispositiu del tipus "major"
  - **tipus**: [*char / block*]
- Comanda per a la creació d'un fitxer especial:
  - `mknod nom c|b major minor`
- Crida al sistema per a la creació d'un fitxer especial
  - `mknod(pathname, mode, dev);`
    - on `mode` codifica el tipus c|b i les proteccions del fitxer
    - on `dev` codifica el *major* i el *minor*

1.30

Espai de noms: VFS (2)

- Alguns exemples:

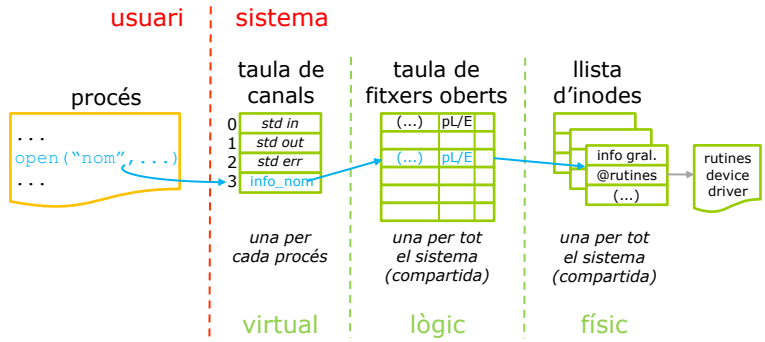
dev name	type	major	minor	description
`/dev/fd0`	b	2	0	floppy disk
`/dev/hda`	b	3	0	first IDE disk
`/dev/hda2`	b	3	2	second primary partition of first IDE disk
`/dev/hdb`	b	3	64	second IDE disk
`/dev/tty0`	c	3	0	terminal
`/dev/null`	c	1	3	null device

Operacions bàsiques d'E/S a linux (1)

- Per tant, com s'associa un nom a un dispositiu virtual?
- `fd = open(nom, mode);`
  - Vincula nom simbòlic a dispositiu virtual (descriptor de fitxer o canal)
    - A més, permet fer un sol cop les comprovacions de proteccions. Un cop verificat, ja es pot executar *read/write* múltiples cops però ja no es torna a comprovar
  - Se li passa el nom simbòlic del dispositiu (nom del fitxer en el VFS) i retorna el dispositiu virtual (*fd: file descriptor* o canal) a través del qual es podran realitzar les operacions de lectura/escriptura

Operacions bàsiques d'E/S a linux (2)

- Open (cont): Efecte sobre les estructures de dades internes del sistema
  - Ocupa un canal lliure de la TC. Sempre serà el primer disponible
  - Ocupa una nova entrada de la TFO
  - Associa aquestes estructures al DD corresponent (major del nom simbòlic). Pot passar que diferents entrades de la TFO apuntin al mateix DD



Operacions bàsiques d'E/S a linux (3)

- `n = read(fd, buffer, count);`
  - Demana la lectura de *count* caràcters del dispositiu associat al canal *fd*
    - Si n'hi ha *count*, o més, disponibles, en llegirà *count*
    - Si n'hi ha menys, llegirà els que hi hagi
    - Si no n'hi ha cap, dependrà del funcionament del dispositiu
      - Es pot bloquejar, esperant a que n'hi hagi
      - Pot retornar amb cap caràcter llegit
    - Quan *EOF*, l'operació retorna sense cap caràcter llegit
      - El significat d'*EOF* depèn del dispositiu
  - Retorna *n*, el nombre de caràcters llegits

Operacions bàsiques d'E/S a linux (4)

- `n = write(fd, buffer, count);`
  - Demana l'escriptura de *count* caràcters al dispositiu associat al canal *fd*
    - Si hi ha espai per *count*, o més, n'escriurà *count*
    - Si n'hi ha menys, escriurà els que hi càpiguen
    - Si no hi ha espai, dependrà del funcionament del dispositiu
      - Es pot bloquejar, esperant a que hi hagi espai
      - Pot retornar amb cap caràcter escrit
  - Retorna *n*, el nombre de caràcters escrits

Operacions bàsiques d'E/S a linux (5)

- `newfd = dup(fd);`
  - Duplica un canal
  - Ocupa el primer canal lliure, que ara contindrà una còpia de la informació del canal original *fd*
  - Retorna *newfd*, l'identificador del nou canal
- `newfd = dup2(fd, newfd);`
  - Igual que *dup*, però el canal duplicat és forçosament *newfd*
- `close(fd);`
  - Allibera el canal *fd* i les estructures associades dels nivells inferiors
  - Cal tenir en compte que diferents canals poden apuntar a la mateixa entrada de la TFO (p.ex. si *dup*), per tant, tancar un canal significaria decrementar -1 el comptador d'apuntadors a aquella entra newfdda
  - El mateix passa amb els apuntadors a la llista d'inodes

Exemple: Operacions bàsiques d'E/S

- Lectura per l'entrada estàndard i escriptura per la sortida estàndard

```
while ((n = read(0, &c, 1)) > 0)
    write(1, &c, 1);
```

  - Observacions:
    - Generalment, els canals estàndard ja estan oberts per defecte
    - Es llegeix fins EOF (`n==0`), que depèn del tipus de dispositiu
    - Es pot executar redireccionant com vulguem
- Igual, però llegim blocs de caracters

```
char buf[SIZE];
...
while ((n = read(0, buf, SIZE)) > 0)
    write(1, buf, n);
```

  - Observacions:
    - Compte! Cal escriure `n` chars
    - Quines diferències hi ha en quant a rendiment?

Operacions d'E/S a linux (6)

- Altres operacions interessants:
  - `pipe(fd_vector); // Dispositiu de comunicacions FIFO`
    - Crea una *pipe* **sense** nom. Retorna 2 canals, un de lectura i un altre d'escriptura
    - No utilitza cap nom del VFS (*pipe* sense nom) i, per tant, no executa la crida al sistema *open*
    - Només podrà ser usada per comunicar el procés que la crea i qualsevol descendent (directe o indirecte) d'aquest (pq heretaran els canals)
  - Recordar que:
    - Lectura: Es bloqueja el procés fins que hi hagi dades a la *pipe*, tot i que no necessàriament la quantitat de dades demanades. Quan no hi ha cap procés que pugui escriure i *pipe* buida, provoca *EOF*
    - Escriptura: El procés escriu, a no ser que la *pipe* estigui plena. En aquest cas es bloqueja fins que es buidi. Quan no hi ha cap procés que pugui llegir el procés rep una excepció del tipus *SIGPIPE*
  - Cal tancar els canals que no s'hagin de fer servir! Altrament bloqueig
  - Per crear una *pipe* **amb** nom, cal fer: mknod + open

Exemple: Ús bàsic d'una pipe

```
int fd[2];
...
pipe(fd);
pid = fork();
if (pid == 0) {                          // Codi del fill
    while (read(0, &c, 1) > 0) {
        // Llegeix la dada, la processa i l'envia
        write(fd[1], &c, 1);
    }
}
else {                                   // Codi del pare
    while (read(fd[0], &c, 1) > 0) {
        // Rep la dada, la processa i la escriu
        write(1, &c, 1);
    }
}
...
```

Teniu disponible aquest codi a: pipe_basic.c

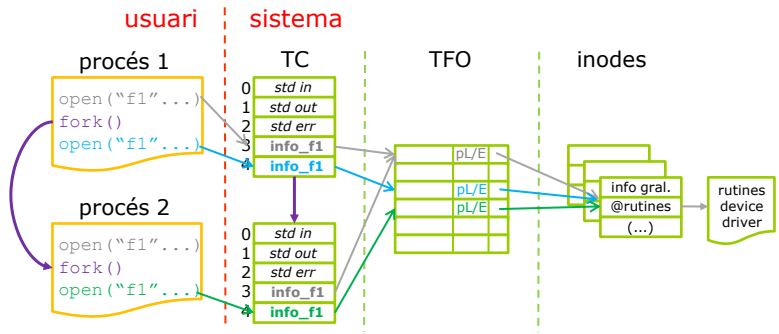
- Compte! El pare ha de tancar `fd[1]` si no es vol quedar bloquejat!

Comunicació entre processos (bis)

- Recordatori del vist en el Tema 2
- Els processos poden ser independents o cooperar entre si
- Per poder cooperar, els processos necessiten comunicar-se
  - **Interprocess communication (IPC) = Comunicació entre processos**
- Comunicacació **sense** dades, mitjançant ESDEVENIMENTS
  - **Signals** (Tema 2)
- Comunicació **AMB** dades, hi ha 2 models principalment
  - **Memòria compartida (Shared memory)** (Tema 3)
    - Els processos utilitzen variables que poden llegir/escriure
  - **Pas de missatges (Message passing)**
    - Els processos utilitzen funcions per enviar/rebre dades
    - Per exemple: pipes

E/S i execució concurrent (1)

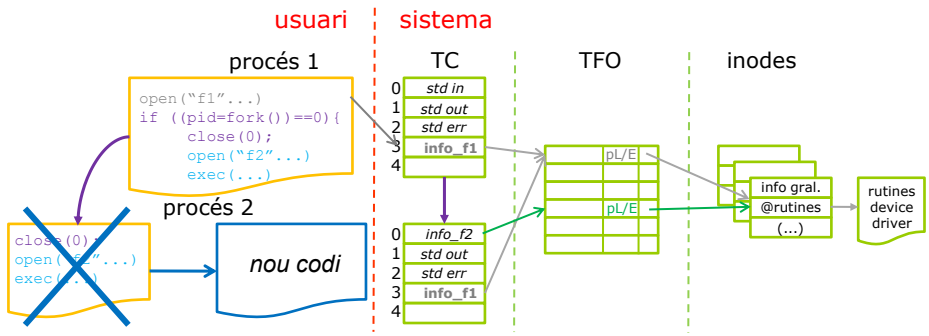
- E/S i *fork*
  - El procés fill hereta una **còpia** de la taula de canals del pare
    - Totes les entrades obertes apunten al mateix lloc de la TFO
  - Permet **compartir** l'accés als dispositius oberts abans del *fork*
  - Els següents *open* ja seran independents



1.41

E/S i execució concurrent (2)

- E/S i *exec*
  - La nova imatge **manté** les estructures internes d'E/S del procés
  - El fet d'executar *fork+exec* permet realitzar redireccions abans de canviar la imatge



1.42

E/S i execució concurrent (3)

- Efecte dels *signals* sobre l'E/S: depèn del sistema operatiu
  - A Unix: Si el procés bloquejat en una operació d'E/S, es desbloqueja
    - Cal controlar `errno`, que valdrà `-EINTR`
    - Cal protegir les crides al sistema
  - A Linux: Rep el signal, el tracta, i continua l'operació d'E/S
    - Si *siginterrupt(SIGXXX, 1)* es comportarà igual que Unix

- Exemple de com protegir la crida al sistema segons el model de Unix:

```
while ( (n = read(...)) == -1 && errno == -EINTR );
```


Exemple 1

- Què fa aquest fragment de codi?

```
...
fd = open("fitxer.txt", O_RDONLY);
pid = fork();
while ((n = read(fd, &car, 1)) > 0 )
       if (car == 'A') numA++;
sprintf(str, "El número d'As és %d\n", numA);
write(1, str, strlen(str));
...
```

Teniu disponible aquest codi a: exemple1.c

Exemple 2

- Què fa aquest fragment de codi?

```
...
pid = fork();
fd = open("fitxer.txt", O_RDONLY);
while ((n = read(fd, &car, 1)) > 0 )
       if (car == 'A') numA++;
sprintf(str, "El número d'As és %d\n", numA);
write(1, str, strlen(str));
...
```

Teniu disponible aquest codi a: exemple2.c

Exemple 3

- Què fa aquest fragment de codi?

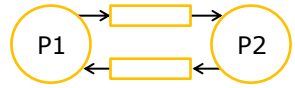
```
...
pid = fork();
if ( pid == 0 ) {
    close(0);
    fd1 = open("/dev/disp1", O_RDONLY);
    close(1);
    fd2 = open("/dev/disp2", O_WRONLY);
    execv("programa", "programa", (char *)NULL);
}
...
```

Exemple 4

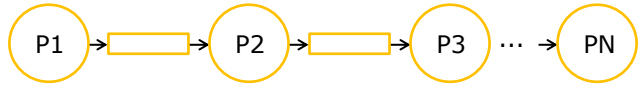
```
...
pipe(fd);
pid1 = fork();
if ( pid1 != 0 ) {                          // PARE
    pid2 = fork();
    if ( pid2 != 0 ) {                      // PARE
        close(fd[0]); close(fd[1]);
        while (1);
    }
    else {                                  // FILL 2
        close(0); dup(fd[0]);
        close(fd[0]); close(fd[1]);
        execv("programa2", "programa2", NULL);
    }
}
else {                                      // FILL 1
        close(1); dup(fd[1]);
        close(fd[0]); close(fd[1]);
        execv("programa1", "programa1", NULL);
}
```


Exercicis a classe (1)

- Escriviu un fragment de codi que creï dos processos *p1* i *p2*, i els connecti mitjançant dues *pipes* pels canals d'E/S estàndard: en la primera *p1* hi escriurà i *p2* hi llegirà, en la segona *p2* hi escriurà i *p1* hi llegirà

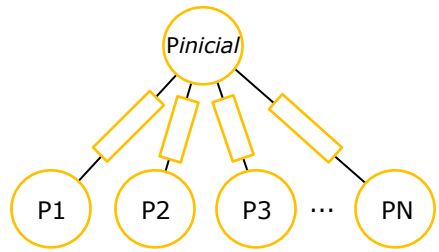


- Escriviu un fragment de codi que creï una seqüència de *N* processos en cadena: cada procés *p_i* crea un sol descendent *p_i+1*, fins arribar a *p_N*. Cada procés s'ha de comunicar en cadena amb l'ascendent i el descendent mitjançant una *pipe* pel canal d'E/S estàndard, de manera que el que escrigui el primer procés pugui ésser enviat per cadena fins al darrer procés



Exercicis a classe (2)

- Escriviu un fragment de codi que creï *N* processos en seqüència: el procés inicial crea tots els descendents *p_1* fins arribar a *p_N*. Cada procés s'ha de comunicar amb el procés pare mitjançant una *pipe*; el pare ha de poder escriure a totes les *pipes* (a través dels canals 3..N+2) i els fills han de llegir de la *pipe* per la seva entrada estàndard.



Índex

- Conceptes
- Gestió de l'E/S
- Gestió de l'E/S a linux
  - Subsistema d'E/S a linux
  - **Operacions d'E/S a linux**
    - *Descripció de les operacions bàsiques d'E/S*
    - *E/S i execució concurrent de processos*
    - *Exemples*
    - ***Dispositius amb accés específic***
    - ***Modificacions de les operacions***
- **Optimitzacions de l'E/S**
  - ***Buffering***
  - ***Caching***
  - ***Spooling***
  - ***Planificació de l'E/S***
  - ***Lectures anticipades***
  - ***Escriptures retardades***
  - ***Gestió de l'energia***

Dispositius amb accés específic (1)

- **Consola**
  - Dispositiu lògic que representa un conjunt teclat+pantalla
    - Històricament era l'únic mètode d'accés, ara es diu consola als terminals que treballen en mode text
  - Els sistemes actuals poden tenir múltiples consoles actives
  - El sistema manté, per cada consola, un *buffer* intern per guardar els caràcters teclejats. Això permet esborrar caràcters abans de ser tractats
  - Posix defineix una funcionalitat particular a certs caràcters especials:
    - ^H: esborrar un caràcter
    - ^U: esborrar una línia
    - ^D: *EOF* o final de fitxer (final de l'entrada)
  - Pot tenir també un *buffer* d'escriptura per cada consola i implementar certes funcions especials, del tipus "pujar N línies", "avançar esquerra"
  - Cada controlador pot implementar la consola tan complicada com vulgui. P.ex. que es pugui modificar text del mig de la línia
  - Habitualment (si no hi ha hagut una redirecció) constitueix l'entrada i sortida estàndard dels processos

Dispositius amb accés específic (2)

- **Consola**: Funcionament
  - Lectura
    - El *buffer* guarda caràcters fins que es pitja el *CR*
    - Si hi ha procés bloquejat esperant caràcters, li dona els que pugui
    - Altrament ho guarda; quan un procés en demana li dona directament
    - ^D provoca que la lectura actual acabi amb els caràcters que hi hagi en aquell moment, encara que no n'hi hagi cap:

```
while ( (n=read(0, &car, 1)) > 0 )
        write(1, &car, 1);
```

      Això, per convenció, s'interpreta com a final de fitxer (*EOF*)
  - Escriptura
    - Escriu un bloc de caràcters
      - Pot esperar que s'escrigui el *CR* per ser visualitzat per pantalla
    - El procés no es bloqueja
  - Aquest comportament bloquejant pot ser modificat mitjançant crides al sistema que canvien el comportament dels dispositius

Dispositius amb accés específic (3)

- **Pipe**
  - *Buffer* temporal amb funcionament FIFO. Les dades de la *pipe* es van esborrant a mesura que es van llegint
  - Lectura
    - Si hi ha dades, agafa les que necessita (o les que hi hagi)
    - Si no hi ha dades, es queda bloquejat fins que n'hi hagi alguna
    - Si la *pipe* està buida i no hi ha cap possible escriptor (tots els canals d'escriptura de la *pipe* tancats), el procés lector rep *EOF*
      - Cal tancar sempre els canals que no es facin servir
  - Escriptura
    - Si hi ha espai a la *pipe*, escriu les que té (o les que pugui)
    - Si la *pipe* està plena, el procés es bloqueja
    - Si la *pipe* està plena i no hi ha cap possible lector, rep *SIGPIPE*
  - Aquest comportament bloquejant pot ser modificat mitjançant crides al sistema que canvien el comportament dels dispositius

Dispositius amb accés específic (4)

- **Dispositius de xarxa**
  - Tot i ser un dispositiu d'E/S, la xarxa té un comportament que no pot ser cobert amb les operacions genèriques d'accés als dispositius
  - La gestió dels dispositius de xarxa es realitza de manera separada
    - P.ex: `/dev/eth0` no disposa ni de *inode* ni de *device driver*
  - Disposen de crides al sistema específiques, i hi ha múltiples propostes
  - La més habitual, és la interfície **socket** de xarxa
    - Habitualment implementant el protocol IP
  - Tot i què hi ha moltes altres propostes d'interfícies i protocols de comunicació de xarxa que també estan implementades a linux (AppleTalk, IPX, ...) o que són específiques d'altres sistemes operatius, com Windows NT, Unix, …

Dispositius amb accés específic (5)

- **Socket**: Funcionament
  - Funciona similar a una *pipe*, excepte que per comptes de dos canals (fd[2]) utilitza un únic canal per la lectura i l'escriptura del *socket*
  - Es crea un *socket*, que es connecta al *socket* d'un altre procés en una altra màquina, connectada a la xarxa
    - Es pot demanar connexió a un *socket* remot
    - Es pot detectar si una altra es vol connectar al *socket* local
    - Un cop connectats, es poden enviar i rebre missatges
      - *read* /*write* o, més específicament, *send* /*recv*
  - Es disposa de crides addicionals que permeten implementar servidors concurrents o, en general, aplicacions distribuïdes
    - P.ex: *select*
    - P.ex: *sendto, recvfrom* per usar el protocol UDP
  - Habitualment, s'executen aplicacions amb un model client-servidor

Dispositius amb accés específic (6)

- **Socket**: Exemple (*pseudo-codi*)
  - Client

```
...
sfd = socket(...)
connect(sfd, ...)

write/read(sfd, ...)
```

  - Server

```
...
sfd = socket(...)
bind(sfd, ...)
listen(sfd, ...)
nfd = accept(sfd, ...)

read/write(nfd, ...)
```

Modificacions de les operacions (1)

- Existeix una crida d'escapament, o porta trasera, que permet fer ajustos **depenents** d'un dispositiu: *ioctl* (a linux)
  - Permet que una aplicació accedeixi a qualsevol funcionalitat que pugui ser implementada per un *device driver*, sense necessitat de crear una nova crida al sistema específica
  - `ioctl(fd, cmd [, ptr]);`
    - *fd* és l'identificador del dispositiu
    - *cmd* selecciona la comanda a executar; depén de cada dispositiu
    - *ptr*, si escau, és un apuntador a memòria a una estructura de dades genèrica el contingut de la qual depén del tipus de *request* (i de dispositiu)

Modificacions de les operacions (2)

- Exemple d'ioctl: Canvi de les característiques del teclat

```
#include <sys/ioctl.h>
#include <termio.h>

struct termio valors;

/* llegeixo els valors actuals */
ret=ioctl(0, TCGETA, &valors);

/* modifico el flag de canonical: el dispositiu ja no és
canònic */
valors.c_lflag = valors.c_lflag & ~ICANON;

/* modifico el nombre màxim de caracters que espero a
l'entrada */
valors.c_cc[VMIN] = 2;

/* afegeixo límit de temps */
valors.c_cc[VTIME] = 10;

/* actualitzo els valors */
ret=ioctl(0, TCSETA, &valors);
```

```
struct termio {
        tcflag_t  c_iflag; // input modes
        tcflag_t  c_oflag; // output modes
        tcflag_t  c_cflag; // control modes
        tcflag_t  c_lflag; // local modes
        cc_t      c_cc[N]; // control chars
};
```

Teniu disponible aquest codi a: teclat.c

Modificacions de les operacions (3)

- *fcntl*
  - Modifica el comprotament d'un canal (*file descriptor*)
  - `fcntl(fd, cmd, [, args]);`
    - *fd* és l'identificador del dispositiu
    - *cmd* selecciona la comanda a executar
    - *args*, si escau, és qualsevol informació addicional que requereixi la comanda; pot ser de tipus long o apuntador a memòria
  - Exemples
    - modificar el fet que el canal es mantingui obert en executar una crida al sistema del tipus *exec* (*close-on-exec*)
    - establir diferents tipus de bloquejos (a zones especificades) sobre diferents tipus d'accés (lectura, escriptura, ...) a un fitxer
    - establir el comportament de les operacions d'E/S sobre aquest canal en rebre *signals*

1. Buffering
2. Caching
3. Spooling
4. Planificació de l'E/S
5. Lectures anticipades
6. Escriptures retardades
7. Gestió de l'energia

OPTIMITZACIÓ DE L'E/S

Optimització de l'E/S (1)

- **Buffering**
  - *Buffer*: Àrea de memòria que guarda dades temporalment
- Consisteix en mantenir un (o varis) *buffers* en el sistema per donar suport a l'execució d'algunes operacions d'E/S

- Objectius:
  1. Amortiguar la diferència de velocitat entre productor i consumidor
  2. Adaptar la transferència de dades entre dispositius que, físicament, utilitzen blocs de tamany diferent
  3. Soportar la "semàntica de la còpia" en les operacions d'E/S

Optimització de l'E/S (2)

- **Caching**
  - *Cache*: Zona de memòria ràpida que conté còpies de dades
- Consisteix en utilitzar la còpia de la dada (*cache*), per comptes de la dada original, per augmentar la velocitat d'accés

- Exemples:
  - Caches de disc: Mecanisme software que permet mantenir a memòria dades que normalment estarien a disc
    - Si cal llegir varis cops el mateix bloc, es carrega un sol cop i es llegeix de la cache
    - Si cal escriure varis cops el mateix bloc, s'escriu a la cache
    - En algun moment, periòdicament, s'escriu a disc

Optimització de l'E/S (3)

- **Spooling**
  - *Spool*: Buffer que conté la sortida cap a un dispositiu que no pot acceptar fluxos de dades intercalats (P.ex: impressora, cinta)
- Permet que vàries aplicacions puguin escriure simultàniament a un dispositiu, tipus impressora, sense que es barregin els treballs
- Cal que el sistema implementi un mecanisme de cues de treballs pendents i un procés (*daemon*) que vagi processant un treball darrera l'altre
- Opcionalment, en funció del dispositiu físic, el sistema pot implementar també algun tipus d'optimització addicional en la planificació dels treballs

Optimització de l'E/S (4)

- **Planificació de les operacions d'E/S**
  - Aprofitant que les E/S són asíncrones
    - No es processen al moment
    - Es guarden en una cua i es van reactivant segons arriben les interrupcions de final d'operació
  - …es reordenen per tal de millorar el rendiment
    - Mínim moviment (rebobinat) de la unitat de cinta
    - Mínim moviment del capçal del disc
  - S'intenten solapar operacions a blocs consecutius a disc, per tal de realitzar les operacions amb una única programació del controlador
  - El sistema disposa d'una rutina (*strategy*) que és l'encarregada d'implementar diferents polítiques d'optimització, en funció del dispositiu i del comportament del propi sistema

Optimització de l'E/S (5)

- Altres optimitzacions: Lectura
  - S'intenta mantenir a memòria una còpia de part de les dades que hi ha a disc (*cache*). A més, aprofitant la característica que molts arxius es llegeixen seqüencialment:
    - **Lectures anticipades** (*read ahead*): es porta el bloc demanat, més altres blocs consecutius
      - Es programen menys operacions de L/E
      - Es produeix menys moviment del capçal
    - Però si l'accés és *random*: problema!
      - Hi ha un flag associat a l'arxiu que indica si cal fer *read ahead*
      - Quan *lseek* el flag es posa a 0, quan *read* es torna a posar a 1

Optimització de l'E/S (6)

- Altres optimitzacions: Escriptura
  - Es guarden les dades a memòria (*cache*) i es realitza l'**escriptura retardada**, a intervals, aprofitant moments en què el dispositiu està parat
    - Les escriptures no són crítiques, ja que els processos no s'esperen a que acabin
    - S'escriu preferiblement en intervals en què el dispositiu està parat
    - Això permet agrupar diferents escriptures d'un mateix bloc
  - Es guarden les escriptures fins al "darrer moment"

threads de kernel

- *bdflush*: escriu quan les caches estan plenes

…tot i que cal minimitzar danys en cas de fallada del sistema

- *kupdate*: escriu cada cert interval de temps, definit com a part de la configuració del sistema

Gestió de l'energia (1)

- Un PC actual consumeix uns 200 watts d'energia
  - Amb una eficiència del 85%: el 15% es perd en calor
- Es calcula que hi ha constantment uns 100 milions d'ordinadors funcionant
  - 20.000 megawatts (equivalent a unes 20 centrals nuclears)
- Els principals consumidors d'energia són: pantalla, disc dur i CPU
  - Aprox: 60%, 15-20% i 10-15% respectivament
- Aspectes hardware
  - Bateries dels portàtils: tipus, consum, duració, contaminació
  - CPU, memòria i perifèrics amb diferents modes de treball: en marxa, inactius i apagats

Gestió de l'energia (2)

- El principal responsable d'implementar polítiques de reducció del consum energètic és el sistema operatiu
  - Tot i que les aplicacions també es poden programar segons models d'execució de menor consum
- Dues estratègies, a nivell de sistema operatiu:
  - Apagar components del sistema: principalment perifèrics
  - Degradar la qualitat del servei que rep un procés
- Addicionalment, es pot dissenyar una interfície que permeti la cooperació entre el sistema operatiu i les aplicacions d'usuari
  - El sistema informa sobre l'estat dels dispositius i l'aplicació utilitza aquesta informació per realitzar unes tasques o unes altres

Gestió de l'energia. Exemples (3)

- Pantalla
  - Apagat automàtic, que habitualment es deixa que ho configuri l'usuari; el reactivat és quasi instantani
  - Il·luminat de zona parcial de la pantalla, la que ocupi la finestra activa
- Disc dur
  - Cal mantenir-lo constantment girant a alta velocitat: cost energètic!
    - Es pot parar, però el cost de reactivar-lo és elevat en quant a temps i consum energètic
    - Cal implementar una molt bona política de predicció d'ús
  - Es pot complementar l'optimització mitjançant una bona cache de disc i mitjançant les escriptures retardades
  - Cooperació sistema-aplicació: En programes tipus processador de texts es pot configurar que es salvi el document periòdicament però preferiblement en moments en els quals el disc estigui en estat actiu

1.69

Gestió de l'energia. Exemples (4)

- CPU
  - La CPU dels portàtils es pot posar en estat inactiu mitjançant software, i es torna a activar mitjançant una interrupció
  - La CPU pot reduir la velocitat a la meitat, reduint el consum en ¾ parts
  - Cal implementar polítiques de predicció de manera que el sistema configuri adequadament el mode de treball de al CPU
- Altres components
  - Memòria: Buidar la cache i desactivar-la temporalment, o copiar-la completament a disc en estat d'hibernació
  - La comunicació inalàmbrica consumeix molta energia, però si es desconnecta es perdria la informació: modificar el *router* per a què guardi la informació en un *buffer* i s'esperi a que es connecti la WiFI
  - Quan puja la temperatura, valorar si posar en marxa el ventilador o posar els components en mode de baix consum

1.70