El sistema operatiu: una màquina virtual

José Ramón Herrero Zaragoza
Josep Lluís Marzo i Lázaro
Enric Morancho Llena

PID_00214800

UOC
Universitat Oberta de Catalunya

Índex

Introducció

En aquest mòdul didàctic refermem el concepte de *sistema operatiu*, introduït en el mòdul anterior, i introduim el concepte de *màquina virtual*.

Expliquem els diferents mecanismes que provoquen l'execució de codi propi del sistema operatiu: els directes (crides al sistema, interrupcions i excepcions), i els indirectes (biblioteques i aplicacions de sistema).

També enumerem els requisits que ha de complir el maquinari per poder instal·lar-hi un sistema operatiu multiusuari capaç de protegir cada usuari de la resta d'usuaris.

Finalment, fem una breu introducció a la planificació del processador.

Objectius

Els materials didàctics d'aquest mòdul contenen les eines necessàries per a assolir els objectius següents:

1. Refermar el concepte de sistema operatiu introduït en el mòdul anterior.

2. Adquirir el concepte de *màquina virtual*.

3. Comprendre les diferències entre els diferents mecanismes directes d'entrada al sistema operatiu.

4. Conèixer els mecanismes indirectes d'entrada al sistema operatiu.

5. Ser conscient dels requisits que ha de complir el maquinari per a poder instal·lar-hi un sistema operatiu multiusuari capaç de protegir cada usuari de la resta d'usuaris.

6. Entendre el funcionament bàsic de la planificació del processador.

1. Programari de sistema i màquines virtuals

1.1. El concepte de *màquina virtual*

La finalitat bàsica del sistema operatiu és amagar el **maquinari** (*hardware*) del sistema computador mitjançant una capa de programari per mitjà de la qual accedim al maquinari per a treballar. El sistema operatiu defineix una **màquina virtual** en la qual els usuaris poden treballar d'una manera més còmoda de la que ho farien si treballessin directament amb els elements que componen el sistema computador.

Aquesta màquina virtual és molt convenient per als usuaris programadors perquè els ofereix una visió "idealitzada" del sistema computador. Els programadors no hauran de preocupar-se dels detalls específics del maquinari (per exemple, el tipus de teclat, el format del disc dur, etc.) i podran centrar-se en la programació dels aspectes directament relacionats amb el problema que han de resoldre. Aquesta màquina virtual està definida per la interfície de crides ofertes pel sistema operatiu. Aquest últim també té la important tasca de transformar les abstraccions ofertes (com ara fitxers) en referències físiques concretes (com ara pistes al disc dur).

Hi ha moltes maneres d'amagar aquest maquinari i d'agrupar i organitzar les funcions que tindrà la "nova" màquina. Cadascuna d'aquestes combinacions determina un tipus diferent de sistema operatiu. Des del punt de vista de l'usuari que accedeix als recursos oferts per la màquina, aquests recursos vénen donats per l'entorn d'accés; aquest entorn està determinat pel sistema operatiu i no pas pel maquinari.

El fet d'ocultar el maquinari als usuaris i als programadors té, entre altres, dos objectius principals, que són els següents:

- L'abstracció del sistema computador com a un maquinari determinat per a tenir una visió més global i senzilla de la màquina.
- Proporciona un mode de funcionament nou i més segur quan s'executa el codi que pertany al sistema operatiu. Aquest aspecte és molt important per a evitar que els usuaris tinguin accés a certs elements del sistema que podrien afectar el funcionament normal i que d'alguna manera han d'estar supervisats.

Ara bé, els usuaris finals del sistema computador, que bàsicament desitgen executar programes, poden trobar que la màquina virtual oferta pel sistema operatiu es troba en un nivell massa baix. Per a facilitar la tasca d'aquests usu-

aris, la resta de programari de sistema també els ofereix una màquina virtual a un nivell molt més alt, que és **l'intèrpret de comandes** (també anomenat intèrpret d'ordres o *shell*).

L'intèrpret de comandes és el programa més habitual que configura la interfície amb què dialoga l'usuari. Per simplificar-ho, podem dir que l'intèrpret de comandes és l'interlocutor entre els usuaris i el sistema operatiu.

En funció del tipus d'usuari, aquest intèrpret pot estar més o menys especialitzat, o pot ser més o menys restrictiu. Per exemple, un programador que ha de fer tasques de desenvolupament o de gestió del sistema tindrà un intèrpret de comandes genèric que li permetrà fer moltes accions diferents; en canvi, un administratiu pot tenir un intèrpret de comandes molt restrictiu que només li permeti executar les aplicacions pròpies de la seva tasca.

1.2. Capes dins del programari de sistema

A la figura 1 podem veure les capes del programari sistema amb més detall. Dins del programari de sistema distingim tres capes, que són el **sistema operatiu** (nucli), les **biblioteques del sistema** i les **aplicacions de sistema** (entre elles, l'intèrpret de comandes). Tots els nivells contenen codi útil i accessible per als usuaris, encara que l'accés és diferent en cada nivell.

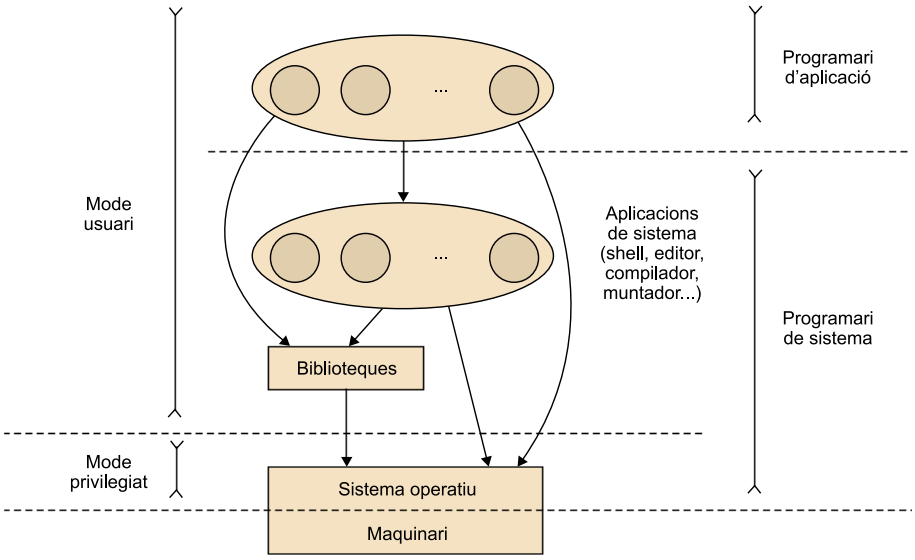

Figura 1. Classificació dels tipus de programari existents a un sistema computador

La manera d'accedir a aquests nivells del programari de sistema és la següent:

1) El **sistema operatiu** és la capa que es troba més a prop del maquinari i conté les rutines de gestió del sistema més relacionades amb els seus recursos físics. Aquestes rutines es poden utilitzar per mitjà de les crides al sistema. Per a garantir que únicament el codi del sistema operatiu pugui accedir a determinats

recursos del sistema computador, aquest codi s'executarà en una modalitat d'execució privilegiada; el codi corresponent a les altres capes s'executarà en una modalitat d'execució no privilegiada (mode d'execució usuari).

2) En un nivell superior del sistema operatiu trobem les **biblioteques de rutines**. Aquestes biblioteques ofereixen als programadors dos tipus de rutines; algunes són d'ús molt comú, però no formen part del sistema operatiu (biblioteques del llenguatge), mentre que d'altres permeten als programadors invocar crides al sistema de forma independent del llenguatge màquina (biblioteques del sistema).

3) L'**intèrpret de comandes** és accessible mitjançant la interfície d'usuari (en mode text o en mode gràfic). En aquest nivell, el sistema espera la introducció de comandes de manera interactiva per part d'usuaris reals o bé per part de programes formats per grups de comandes (aquests programes reben el nom d'*scripts* o *shellscripts*), els quals són interpretats per l'intèrpret de comandes.

1.2.1. Les crides al sistema

Com hem dit, un sistema operatiu proporciona un entorn per a l'execució de programes. Una part d'aquest entorn consisteix en un conjunt de serveis que són accessibles pels processos[1] que s'estan executant en el sistema. Aquestes funcionalitats són com una extensió de les instruccions que pot executar el processador i estan en la línia d'una concepció del sistema com una màquina virtual que pot fer coses més complicades que les que pot fer el maquinari en si mateix.

[(1)]Tal com vam introduir en el mòdul anterior, un procés no és més que un programa en execució.

Els programadors d'aplicacions invoquen els serveis del sistema operatiu des dels seus programes fent crides al sistema[2].

[(2)]Aquestes crides al sistema també s'anomenen API (Application Programming Interface) del sistema operatiu.

Encara que cada sistema operatiu pot oferir serveis molt específics, de manera general podem agrupar-los en les classes següents:

1) Gestió de processos. En aquest apartat englobem les crides que fan referència a la creació i l'eliminació de processos, però també podem trobar-hi altres serveis, com ara la suspensió i la reanimació de processos. També podem modificar atributs d'execució, com ara la prioritat o el temps de quota de CPU assignat. Tal com veurem més endavant, totes aquestes crides es troben sota el control del nucli del sistema operatiu, que determina en última instància si la petició realitzada per un procés pot ser servida d'acord amb els seus privilegis.

2) Senyalització entre processos. En un sistema multitasca, el nexe natural d'unió entre totes les aplicacions en execució és el sistema operatiu. Així, el vehicle apropiat per a comunicar i sincronitzar processos és el nucli del sistema operatiu. Senyals, semàfors, regions crítiques i altres funcions són exemples concrets d'aquesta classe de serveis.

3) Gestió de dispositius d'entrada/sortida. Els serveis relacionats amb el sistema d'entrada/sortida són dels més complets i també dels més utilitzats. Tenen com a funcions principals crear, canals d'entrada/sortida[3], obrir-los i tancar-los, així com escriure'ls i llegir-los. Els sistemes operatius també proporcionen un conjunt de crides per a conèixer l'estat d'un canal o canviar-li els atributs.

(3)Els *canals* bàsicament permeten treure dades des d'un procés o cap a aquest.

4) Gestió directa dels recursos del sistema. Aquest grup de serveis és molt específic de cada sistema operatiu, però un exemple comú és la gestió de la memòria. Algunes de les operacions que s'efectuen amb la memòria del sistema són capturar, alliberar, protegir, compartir, etc.

5) Gestió del sistema d'arxius. Juntament amb el bloc de serveis d'entrada/sortida, també és un grup de serveis molt utilitzat. La gestió del sistema de fitxers inclou les funcionalitats més conegudes també pels usuaris de l'intèrpret de comandes en relació amb la gestió d'arxius i directoris, com ara crear i eliminar entrades, canviar de directori, mostrar el directori actual, etc. També permet fer les operacions típiques amb fitxers, com ara crear-ne, eliminar-ne, copiar-ne, canviar-ne el nom, etc.

6) Proteccions. El sistema operatiu ha de combinar els atributs entre el procés i el servei que demana per a determinar si aquest servei es pot donar o no. És una de les parts més utilitzades pels administradors dels sistemes, especialment com a crides del sistema, però no ho és tant pels usuaris.

7) Funcions de temps. Totes les referències temporals dels processos provenen de crides al sistema operatiu. Per exemple, la data, l'hora, etc. També es poden capturar dades sobre el temps d'execució d'un procés, el temps de CPU o altres funcions estadístiques d'aquesta classe.

1.2.2. Les biblioteques

Els llenguatges d'alt nivell ens permeten fer operacions que el llenguatge màquina no pot servir directament, com per exemple:

- operacions matemàtiques (`ln`, `sin`, `cos`, `sqrt`, etc.)
- manipulació de cadenes de caràcters (`strcpy`, `strcat`, etc.)
- operacions d'entrada/sortida amb format (`readln`, `printf`, `fopen`, etc.)

Els llenguatges d'alt nivell permeten efectuar operacions d'aquest estil directament, però perquè es puguin executar s'han d'afegir fragments de codi nous que les resolguin; són les biblioteques de rutines.

Les biblioteques són un conjunt de mòduls objecte organitzats adequadament. Les subministra o bé el fabricant del sistema operatiu o bé la institució que ha desenvolupat l'entorn de compilació. Les rutines que contenen estan catalogades i se sap perfectament com s'han d'utilitzar; tenen definits els noms dels procediments, els paràmetres necessaris i la manera en què es passen, així com els possibles fitxers complementaris[4]. No és habitual disposar del codi font de les biblioteques.

(4)Un exemple de fitxers complementaris són els fitxers capçalera (per exemple, `/usr/include/fcntl.h`) del llenguatge C.

Els procediments de les biblioteques són molt genèrics i estan pensats perquè molts programadors de diferents tipus d'aplicacions puguin fer-ne ús. Dins de les aplicacions de sistema, generalment es disposa d'eines per a construir noves biblioteques. El programador pot donar format de biblioteca a aquells procediments més generals i més utilitzats i comprovats per un programador, o un equip de programadors, i utilitzar-los així més fàcilment en diferents aplicacions.

Podem agrupar les biblioteques en les tres famílies següents:

1) Biblioteques del sistema: permeten invocar les crides al sistema de forma independent del llenguatge màquina del nostre sistema computador. D'aquesta forma, el codi font de les nostres aplicacions serà portable entre sistemes computadors on hi hagi instal·lat el mateix sistema operatiu.

2) Biblioteques del llenguatge: són les més genèriques i normalment són estandarditzades per a cada llenguatge de programació d'alt nivell, almenys en un conjunt de procediments. Les àrees més comunes són la gestió de fitxers i de dispositius d'entrada/sortida, les funcions matemàtiques, la manipulació de cadenes de caràcters, etc.

3) Biblioteques especialitzades: són aquelles que, de manera opcional, s'afegeixen al sistema per tal de resoldre problemes més específics que no poden solucionar les biblioteques estàndard. Un bon exemple són les biblioteques especialitzades en càlculs matemàtics (càlcul matricial, estadístic, criptogràfic, etc.) o les biblioteques per a visualització gràfica. També se subministren per a la utilització de maquinari molt específic, com és el cas dels entorns industrials.

1.2.3. L'intèrpret de comandes

Els usuaris d'un sistema operatiu acostumen a treballar amb el sistema utilitzant un intèrpret de comandes (*shell*), ja sigui en **mode text** o en **mode gràfic**.

Hi ha un seguit de conceptes comuns a tots els intèrprets de comandes, com ara l'establiment d'una sessió de treball, l'autenticació de l'usuari i la introducció de les comandes (ja sigui utilitzant el teclat o utilitzant el ratolí).

Aquest document no descriu el funcionament de cap intèrpret de comandes.

Vegeu també

Dins dels recursos de la vostra aula disposeu dels enllaços necessaris per a iniciar-vos en l'intèrpret de comandes típic dels sistemes Linux.

1.3. El sistema operatiu des del punt de vista del programador

La visió de l'usuari del sistema operatiu està definida principalment per les aplicacions de sistema i, concretament, per la interacció amb l'intèrpret de comandes. En canvi, el programador en té un punt de vista molt diferent; allà on l'usuari veu les facilitats ofertes pel sistema operatiu, el programador veu els recursos físics i, mitjançant crides al sistema operatiu, ha d'aconseguir crear un programa que doni facilitats a un futur usuari.

El programador disposa de les dues vies següents per a utilitzar els serveis del sistema operatiu:

- Les crides al sistema, fan que en temps d'execució es transfereixi el control a un codi que pertany al nucli del sistema operatiu. Per tant, el programador no ha d'implementar dins del programa aquests serveis, únicament ha d'invocar-los.

- Les biblioteques de sistema, que són un conjunt de procediments d'ús comú que es poden afegir a les aplicacions mitjançant una simple referència al procediment en qüestió. Això fa que el codi que necessitem s'enllaci amb el nostre durant el procés de compilació. En general, el codi de les biblioteques és de nivell més alt i és més independent del maquinari que s'utilitza que el codi de les crides al sistema.

Vegeu també

L'annex d'aquest mòdul fa un repàs del seguit de passos necessaris per a generar un fitxer executable (compilació, enllaçat o muntatge) i executar-lo.

2. Els mecanismes d'entrada al sistema operatiu

En aquest apartat discutirem els mecanismes per a transferir l'execució al sistema operatiu, des d'un nivell més baix (proper al maquinari) fins a un nivell més alt (més abstracte). Alguns d'aquests mecanismes respondran directament a la voluntat del procés en execució, però d'altres en seran aliens.

2.1. Mecanismes directes (*traps*, excepcions i interrupcions)

Hi ha tres mecanismes bàsics mitjançant els quals es pot transferir l'execució al sistema operatiu:

- **Crides al sistema operatiu** (també anomenades *system calls* o *traps*): les crides al sistema són provocades per les aplicacions d'usuari quan demanen un servei al sistema operatiu. En un punt concret del codi d'usuari es fa una transferència explícita del control al sistema operatiu mitjançant l'execució de la instrucció del llenguatge màquina que permet fer-ho (en el cas dels processadors Intel és la instrucció `int`). Des d'aquest punt de vista són esdeveniments síncrons. També poden rebre el nom d'interrupcions *software*. El sistema operatiu defineix quins són els paràmetres d'entrada i de sortida de cada crida al sistema, així com la forma de passar aquests paràmetres (per registres, per la pila, etc.).

- **Interrupcions**: les interrupcions són provocades pels dispositius del maquinari quan volen informar d'algun canvi d'estat (per exemple, la pulsació d'una tecla, la finalització d'una operació d'entrada/sortida, etc.). Normalment són successos asíncrons aliens al procés en execució. Típicament reben el nom d'interrupcions *hardware*.

- **Excepcions**: les excepcions es produeixen quan el maquinari detecta algun tipus d'anomalia. Per exemple, si el processador està executant la instrucció de llenguatge màquina *dividir* i el denominador té el valor 0, el processador genera una excepció perquè aquesta operació no està definida. Les excepcions són ruptures de seqüència no previstes; de fet són interrupcions, però provocades directament per l'execució del codi mateix de l'usuari en curs.

Tot i aquestes diferències, el tractament que reben aquests esdeveniments és molt similar:

- **Salvar l'estat del procés d'usuari en execució**: en el moment de produir-se l'esdeveniment que causa la crida al sistema / interrupció / excepció, el més probable és que hi hagi algun procés d'usuari en execució. Per tal de poder executar la rutina d'atenció del sistema operatiu, cal aturar

l'execució d'aquest procés d'usuari. Ara bé, el més probable és que, un cop executada aquesta rutina, calgui reprendre l'execució del procés. Per a poder fer-ho, cal salvar l'estat del procés (contingut dels registres del processador, etc.) i "fer una foto" del procés, de forma que més endavant el procés pugui continuar la seva execució com si res no hagués passat.

- **Determinar quina rutina del sistema operatiu ha de tractar aquest esdeveniment**: existeixen diverses alternatives per a realitzar aquesta determinació. Linux resol aquest problema en temps d'execució (cada cop que es produeix un *trap*/interrupció/excepció) mitjançant un vector d'interrupcions que emmagatzema en cada posició una posició de memòria, que és la posició on es troba el començament de la rutina d'atenció associada a aquella interrupció.
  - Cada possible excepció i interrupció té associada una posició fixa d'aquest vector. Quan es produeix una interrupció o una excepció, el maquinari consulta l'entrada corresponent del vector d'interrupcions i determina on s'ha de transferir el control.

  - El cas de les crides al sistema és lleugerament diferent. Com que el nombre de crides al sistema ofertes per un sistema operatiu pot ser molt gran (337 en el cas de la versió 2.6.32 de Linux), s'ha optat per un mecanisme híbrid maquinari-programari. Totes les crides als sistema tenen associada la mateixa entrada del vector d'interrupcions (la 0x80); la rutina d'atenció al *trap* 0x80 és la que determina on cal transferir el control en funció d'un registre del processador que conté l'identificador de crida al sistema que es vol executar.

- **Executar la rutina d'atenció**: es transfereix el control a la rutina del sistema operatiu associada a aquest esdeveniment.
  - En el cas de les crides al sistema, aquesta rutina implementarà el servei demanat per l'usuari (per exemple, llegir del teclat, executar un programa, etc.).

  - En el cas de les interrupcions, la rutina d'atenció actua en conseqüència (per exemple, si s'ha premut una tecla, desa el caràcter llegit en una memòria intermèdia; si ha finalitzat una lectura de disc, informa al procés que l'estava esperant, etc.).

  - En el cas de les excepcions, la rutina d'atenció intentarà solucionar el problema. A partir d'aquí no es pot generalitzar, ja que de vegades el sistema pot solucionar el problema i de vegades no queda cap més alternativa que avortar l'execució del procés.

- **Continuar l'execució d'un procés d'usuari**: en alguns casos serà el mateix que estava en execució abans de produir-se l'esdeveniment, mentre que

en altres casos serà un altre procés. Per a fer-ho cal restaurar l'estat (que prèviament ha estat salvat) i continuar la seva execució.

Tot i que aquests mecanismes puguin recordar les rutines convencionals (cal salvar l'estat, es produeix una ruptura de la seqüenciació implícita, etc.), hi ha unes diferències molt importants:

- El codi de la rutina d'atenció als *traps*/interrupcions/excepcions no forma part dels executables d'usuari, sinó que forma part del nucli del sistema operatiu.

- L'adreça de memòria on es troba el codi de la rutina d'atenció es determina en temps d'execució (en el cas de les rutines convencionals, l'adreça de la rutina és un paràmetre de la instrucció de llenguatge màquina que invoca una rutina).

- Una altra diferència té a veure amb el mode d'execució i la veurem més endavant en aquest mòdul. Totes aquestes rutines d'atenció formen part del nucli del sistema operatiu i tenen accés complet al maquinari; en canvi, les rutines escrites per l'usuari no tindran accés a determinades parts del maquinari.

2.2. Mecanismes indirectes

2.2.1. Biblioteca del sistema

Des del punt de vista del programador, el mecanisme d'entrada al sistema mitjançant *traps* té un gran problema: no és portable perquè depèn del llenguatge màquina propi del processador. Si volem facilitar que el codi font dels programes sigui fàcilment portable entre arquitectures diferents on hi hagi instal·lat el mateix sistema operatiu, cal disposar d'una forma independent del maquinari per a poder realitzar les crides al sistema operatiu. La solució ve donada per la biblioteca del sistema. Les rutines d'aquesta biblioteca permeten expressar de forma estàndard les crides al sistema.

Per exemple, la figura 2 mostra la interfície (en llenguatge C) de la crida al sistema `write` del sistema operatiu Unix.

```
ssize_t write(int fd, const void *buf, size_t count);
```

Figura 2. Interfície de la crida al sistema `write` del sistema operatiu Linux

La rutina `write` de la biblioteca del sistema Unix per a l'arquitectura IA-32 invoca la crida al sistema tal i com es fa en aquesta arquitectura (pas de paràmetres a través de registres determinats, instrucció de llenguatge màquina `int`). La figura 3 (a dalt) mostra el codi assemblador d'aquesta rutina.

```
000001cd <write>:
      1cd:   push   %ebp
      1ce:   mov    %esp,%ebp
      1d0:   push   %ebx
      1d1:   sub    $0x4,%esp
      1d4:   mov    $0x4,%eax
      1d9:   mov    0x8(%ebp),%ebx
      1dc:   mov    0xc(%ebp),%ecx
      1df:   mov    0x10(%ebp),%edx
      1e2:   int    $0x80
      1e4:   mov    %eax,0xfffffff8(%ebp)
      1e7:   cmpl   $0xffffff82,0xfffffff8(%ebp)
      1eb:   jbe    1f4 <write+0x27>
      1ed:   movl   $0xffffffff,0xfffffff8(%ebp)
      1f4:   mov    0xfffffff8(%ebp),%eax
      1f7:   add    $0x4,%esp
      1fa:   pop    %ebx
      1fb:   leave
      1fc:   ret
```

```
__write:
0x120009fd0: addq zero, 0x4, v0
0x120009fd4: call_pal callsys
0x120009fd8: beq a3, 0x120009ff0
0x120009fdc: br gp, 0x120009fe0
0x120009fe0: ldah gp, 0(gp)
0x120009fe4: lda gp, 19472(gp)
0x120009fe8: lda at, 26144(gp)
0x120009fec: br zero, 0x120015210
0x120009ff0: ret zero, (ra), 1
```

Figura 3. Codi de la rutina de biblioteca `write` de Unix a dues arquitectures: IA-32 (a dalt) i Alpha (a baix)

D'aquesta forma, el programador pot invocar la crida al sistema `write` com si fos una rutina qualsevol del llenguatge C. Com que l'executable es genera sobre l'arquitectura IA-32, es farà servir la biblioteca de sistema corresponent a l'arquitectura IA-32.

La figura 3 (a baix) mostra el codi assemblador de la rutina `write` de la biblioteca del sistema Unix per a l'arquitectura Alpha. Com que l'executable es genera sobre aquesta arquitectura, es farà servir la biblioteca de sistema de

l'arquitectura Alpha. D'aquesta forma, el programador no necessita conèixer com es realitza la crida al sistema en llenguatge màquina i pot invocar-la d'una forma portable.

Els altres dos mecanismes (interrupcions i excepcions) no estan a disposició directa de l'usuari.

2.2.2. Biblioteques del llenguatge

Alguns llenguatges de programació ofereixen serveis més propers a les necessitats del programador i, per tant, permeten que el programador sigui més productiu.

En l'Unix, per exemple, les crides d'entrada/sortida (`read/write`) treballen amb cadenes de bytes. Si volem escriure un valor enter en base 10 per pantalla, abans és necessari convertir el valor enter a la cadena de dígits decimals que representa el valor.

En canvi, el llenguatge C ofereix una biblioteca d'entrada/sortida estàndard (la biblioteca `stdio`, amb rutines com `printf`, `scanf`, etc.) que permet realitzar entrada/sortida amb format.

La figura 4 mostra un exemple de com escriure en l'Unix un nombre enter per pantalla en base 10. Si utilitzem directament crides al sistema (codi de l'esquerra), cal transformar aquest valor numèric a la cadena de caràcters que representa el seu valor. Si utilitzem la rutina de biblioteca `printf` (codi de la dreta), aquesta rutina realitza la conversió i invoca la crida al sistema corresponent.

```
char s[10];
int i=9;

s[9]='\n';
while (n > 0) {
       i--;
       s[i] = '0' + (n%10);
       n=n/10;
}
write(1, s[i], 10-i);
```

```
int n;

printf("%d\n", n);
```

Figura 4. Com escriure a Unix un nombre enter per pantalla en base 10: utilitzant crides al sistema (esquerra) i utilitzant rutines de la biblioteca del llenguatge (dreta)

La biblioteca del llenguatge també ofereix rutines d'ús comú que estan implementades sense utilitzar crides al sistema. Per exemple, rutines trigonomètriques, generadors de nombres pseudoaleatoris, manipulació de cadenes de caràcters (*strings*), etc.

La figura 5 mostra un exemple en què apareix una aplicació que utilitza rutines d'una biblioteca del llenguatge. Hi podem veure els passos necessaris fins a arribar al codi del sistema operatiu que implementa el servei sol·licitat.

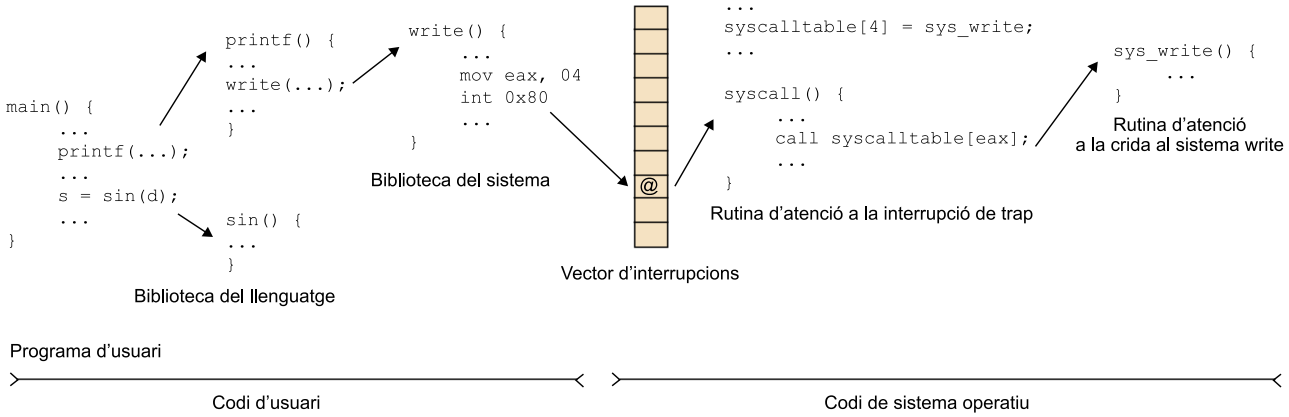

Figura 5. Entrada indirecta al codi del sistema operatiu utilitzant una rutina de la biblioteca del llenguatge

2.2.3. Aplicacions

Els usuaris no programadors volen poder resoldre les seves necessitats sense haver de programar. El programari de sistema els ofereix un seguit d'aplicacions ja implementades que resolen les seves necessitats més bàsiques. Aquestes aplicacions, com no pot ser de cap altra forma, estan implementades utilitzant les crides al sistema i les biblioteques del llenguatge i del sistema.

Per exemple, la comanda `cp` de l'Unix (fa la còpia d'un fitxer origen en un fitxer de destí) està implementada utilitzant un seguit de crides al sistema: obrir el fitxer d'origen, obrir el fitxer de destí, llegir el contingut del fitxer d'origen, escriure'l en el fitxer de destí i tancar els fitxers involucrats.

La capa d'aplicacions de sistema ofereix una gran quantitat d'aplicacions que resolen la gran majoria de necessitats dels usuaris finals no programadors: crear/esborrar/reanomenar/copiar fitxers ordinaris i directoris, aplicacions d'ordenació de fitxers, consultar informació sobre el sistema, aturar/reprendre/finalitzar un procés, etc.

3. Suport del maquinari per a implementar el sistema operatiu

Hem vist que el sistema operatiu ofereix una màquina virtual als usuaris més senzilla d'utilitzar que la màquina física. Ara bé, què passaria si un usuari insistís a treballar directament amb la màquina física tot ignorant el sistema operatiu? Si el sistema operatiu vol garantir la fiabilitat dels serveis que ofereix, no pot consentir aquesta situació. Un usuari que treballi directament amb la màquina física pot provocar errors irrecuperables o pot accedir a recursos (posicions de memòria, sectors de disc, etc.) que no li pertanyen i posar en compromís la confidencialitat i integritat de les dades d'altres usuaris.

Ara bé, al cap i a la fi, tot el codi en execució en un processador no és més que seqüències d'instruccions de llenguatge màquina. En principi, qualsevol usuari podria escriure un programa que utilitzés instruccions del llenguatge màquina, que treballin directament amb la màquina física i intentar executar-lo. Per a poder neutralitzar aquesta estratègia, el sistema operatiu necessita un cert suport per part del maquinari.

- Dins del repertori d'instruccions del llenguatge màquina, l'arquitectura del computador identifica un subconjunt d'**instruccions "privilegiades"** que poden posar en compromís l'estabilitat de la màquina o la confidencialitat de les dades, com per exemple la instrucció que permet aturar el processador, la que permet inhibir el tractament de les interrupcions, les que permeten llegir/escriure els registres de control dels dispositius d'entrada/sortida, etc. Veurem que els programes d'usuari no podran utilitzar directament aquestes instruccions privilegiades. La resta d'instruccions del llenguatge màquina sí que podran ser utilitzades lliurement pels programes d'usuari.

- El processador ofereix dos (o més) modes d'execució, que són **el mode d'execució privilegiat i el mode no privilegiat** (usuari). En el mode privilegiat, el processador podrà executar qualsevol instrucció de llenguatge màquina. En el mode usuari, el processador únicament podrà executar instruccions no privilegiades; si intenta executar-ne una de privilegiada, el processador produirà una excepció.

- **El mode d'execució usuari s'utilitzarà per a executar codi d'usuari i el mode d'execució privilegiat s'utilitzarà per a executar el codi del sistema operatiu**. El processador haurà de controlar en cada moment en quin mode d'execució es troba. Per fer-ho, només haurà de ser conscient del moment en què es produeixen els esdeveniments directes d'entrada al sistema operatiu (interrupcions, excepcions i crides al sistema) i del moment en què s'executa la instrucció de llenguatge màquina que indica el retorn

des d'una rutina d'atenció (en l'arquitectura IA-32 és la instrucció `iret`). La figura 6 mostra el diagrama d'estats corresponent.

- **El vector d'interrupcions** (VI) determina quines rutines donen servei a les interrupcions/excepcions/crides al sistema. Òbviament, cal garantir que un procés d'usuari no pugui modificar-lo. Si el VI es pot llegir/modificar amb instruccions convencionals d'accés a la memòria, cal garantir que els processos d'usuari no pugin fer-ho. Cal que els processadors disposin d'una unitat de gestió de la memòria (MMU, Memory Management Unit) que limiti el rang o els rangs d'adreces de memòria que pot accedir cada procés. Si un procés intenta accedir fora del rang o dels rangs que li corresponen, la MMU generarà una excepció d'accés a la memòria invàlid.

- Per tal de garantir que el sistema operatiu prengui el control de la màquina periòdicament, cal un dispositiu temporitzador que generi interrupcions periòdicament (típicament, entre 100 i 1.000 cops per segon).

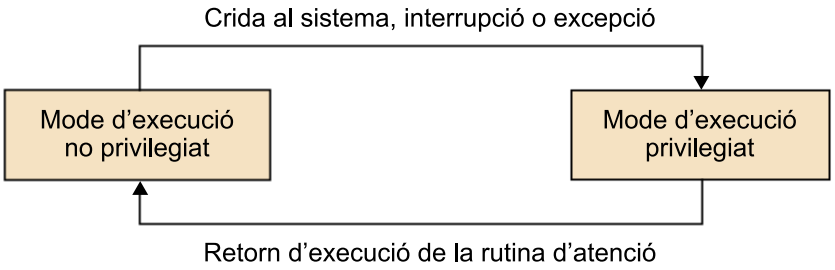

Figura 6. Diagrama que mostra les transicions entre el mode d'execució privilegiat i el no privilegiat

Observacions:

- El sistema operatiu MS-DOS tenia greus problemes d'estabilitat perquè el processador per al qual es va dissenyar (8088/8086) tenia força limitacions (no disposava de diversos modes d'execució, no tenia instruccions privilegiades i no tenia MMU). Per tant, qualsevol programa en execució a la màquina tenia el control total d'aquesta.

- Gràcies a la MMU, les instruccions de llenguatge màquina d'accés a la memòria (`load/store`) no són privilegiades. És la MMU la que determina si els accessos a la memòria que fa un procés són vàlids o no; si no ho són, la MMU generarà una excepció que, en la majoria dels casos, provocarà que el sistema operatiu avorti l'execució del procés.

Vegeu també

En el mòdul 3, tractarem la qüestió de l'accés a la memòria amb més detall.

4. Introducció a la planificació del processador

4.1. El concepte de procés

La noció de *procés*[5] és fonamental per a entendre el funcionament intern dels sistemes de computació moderns que són capaços d'executar moltes activitats de manera simultània (sistemes operatius multiprocés). Tanmateix, no és senzill donar-ne una definició exacta. A continuació en donem algunes de les més utilitzades:

(5)El terme *procés* aplicat al món de la computació fou utilitzat per primer cop pel sistema operatiu Multics en els anys seixanta. Des d'aleshores també s'utilitza el terme *tasca* com a sinònim.

- un programa en execució
- l'esperit animat d'un procediment
- el centre de control d'un procediment en execució
- l'entitat a la qual s'assignen els recursos d'un ordinador
- la instanciació d'un programa

Encara que, com hem dit, no hi ha cap acord sobre la definició de *procés*, l'expressió *un programa en execució* és una de les definicions més acceptades. Així, podem formular les afirmacions següents:

- Un **programa** és la descripció detallada per a la resolució d'un problema de manera general, en abstracte.
- Un **procés** és l'aplicació en concret del programa a un cas particular i en un moment determinat, amb l'assignació d'uns recursos concrets. Un procés és irrepetible.

De fet, un mateix programa pot tenir moltes instanciacions (còpies) que s'executen de manera concurrent (simultània) en el mateix sistema informàtic. Per exemple, imaginem que en un sistema multiusuari un conjunt de programadors estan utilitzant el mateix editor de text.

Vegeu també

Al mòdul 3 veurem si és necessari replicar el mateix codi a la memòria tantes vegades com usuaris hi ha executant-lo.

Analogies de procés

Podríem dir que un programa és a un procés com el futbol és a un partit. El futbol és un esport en què s'han definit unes normes de funcionament, les quals resolen totes les possibles incidències i ho fan de manera general. Quan assignem recursos a aquest concepte (camp, entrenador, jugadors, etc.), instanciem el concepte de futbol i s'inicia un nou "procés": el partit. Durant aquella execució, el partit es desenvolupa d'una manera determinada i dóna uns resultats concrets, però quan finalitza deixa d'existir. El futbol, en canvi, ha estat aliè a aquesta execució i permet tantes altres instàncies del joc com siguin necessàries. La mateixa idea es pot aplicar a la descripció acurada d'una interpretació musical (la partitura) i l'execució (el concert) corresponent, o a una recepta de cuina i la realització d'un plat utilitzant la recepta.

Hi ha sistemes operatius que accepten que es puguin definir diferents fils d'execució concurrent (*threads*) dins d'un procés.

Vegeu també

La definició de diferents fils d'execució concurrent dins d'un procés es tractarà en el mòdul didàctic de processos.

4.2. Planificació del processador

En un sistema operatiu multiprocés hi ha diversos processos que competeixen per executar-se i fer ús del processador o dels processadors disponibles a l'ordinador. En aquest subapartat assumim que l'ordinador disposa d'un únic processador; en un moment donat es podrà estar executant un únic procés, i la resta de processos hauran d'esperar per a poder fer-ne ús.

El sistema operatiu decideix en cada moment quin procés pot fer ús del processador; la part del sistema operatiu que du a terme aquesta tasca és el **planificador del processador**. El planificador associa a cada procés un estat que descriu el seu grau d'activitat.

Els estats bàsics que contempla qualsevol planificador del processador són:

- Run (execució): el procés està utilitzant el processador.
- Ready (preparat): el procés no està utilitzant el processador perquè algun altre procés l'està utilitzant.
- Blocked (bloquejat): el procés no està utilitzant el processador perquè el procés està esperant que finalitzi una petició que ha fet al sistema operatiu (per exemple, una operació d'entrada/sortida, la sincronització amb un altre procés, etc.). Aquest estat també pot rebre el nom de *wait* (espera).

La figura 7 mostra un diagrama amb aquests estats, així com les transicions entre ells. A continuació detallem les causes que provoquen aquestes transicions:

- Ready a Run: el planificador ha decidit que el procés pot fer ús del processador.
- Run a Ready: el planificador ha decidit que el procés ha de deixar de fer ús del processador.
- Run a Blocked: el procés ha invocat alguna crida al sistema que no s'ha pogut realitzar immediatament (per exemple, una lectura de teclat). Aquest tipus de crides al sistema reben el nom de *crides al sistema bloquejants*.
- Blocked a Ready: s'ha completat l'execució de la crida al sistema sol·licitada pel procés, amb la qual cosa el procés està preparat per a tornar a utilitzar el processador.

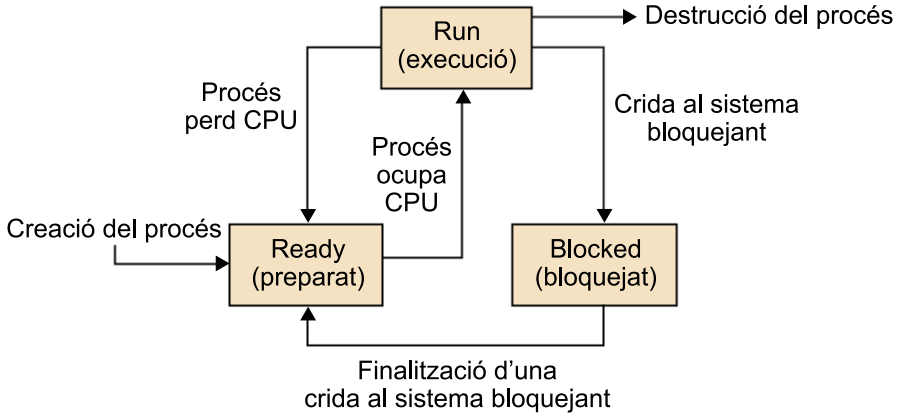

Figura 7. Versió simplificada del diagrama d'estats dels processos des del punt de vista del planificador del processador

Quan el procés que es troba en l'estat Run deixa d'utilitzar el processador (ja sigui voluntàriament perquè ha invocat una crida al sistema bloquejant, o bé involuntàriament perquè el planificador del processador ha decidit que un altre procés faci ús del processador), es produeix un canvi de context. El processador atura l'execució d'un procés i passa a executar un altre procés. La rutina del sistema operatiu que implementa el canvi de context ha de desar tota la informació necessària per, més endavant, poder reprendre l'execució del procés. Bàsicament, aquesta informació està formada pels registres del processador.

El planificador del processador és responsable de garantir que els processos en execució en el sistema utilitzin el processador de forma equitativa. No és objectiu d'aquesta assignatura veure els diferents criteris que pot fer servir un planificador per a realitzar aquesta tasca.

Els diagrames d'estats dels planificadors del processador dels sistemes operatius actuals tenen més estats i transicions que els que hem presentat a la figura 7. Aquests estats addicionals poden mostrar si el procés està executant codi en mode usuari o en mode sistema, el motiu que ha provocat el bloqueig del procés, etc.

Resum

En aquest mòdul didàctic hem vist el sistema operatiu com una màquina virtual que amaga als usuaris la complexitat de la gestió dels recursos de maquinari i programari que componen un sistema computador. Hem presentat les diferents capes de programari que podem trobar en un sistema informàtic i les hem analitzat des de les més properes al maquinari (rutines d'atenció a interrupcions, excepcions i crides al sistema) fins a les més llunyanes (aplicacions de sistema), passant per les biblioteques de rutines (del sistema i del llenguatge). Hem vist que el codi del sistema operatiu respon a esdeveniments (interrupcions, excepcions i crides al sistema). Per a complementar el tema, també s'han llistat els requisits que ha de complir el maquinari per poder-hi instal·lar un sistema operatiu multiusuari que protegeixi cada usuari de la resta d'usuaris. Finalment, hem fet una breu introducció a la planificació del processador.

Activitats

1. Utilitzant algun dels recursos disponibles a l'aula, inicieu una sessió de treball a l'intèrpret de comandes de Linux i intenteu fer les següents proves:

a) Utilitzeu les comandes bàsiques de gestió de fitxers i directoris (`more`, `cat`, `cp`, `mv`, `rm`, `mkdir`, `cd`, `rmdir`).
b) Utilitzeu la comanda `ps` per a veure els processos actius associats a la sessió i tots els processos en execució en el sistema.
c) Comproveu el funcionament dels filtres utilitzant el metacaràcter |.
d) Executeu comandes en mode primer pla (*foreground*) i en mode segon pla (*background*).
e) Genereu l'executable d'algun programa senzill escrit en C (per exemple, un "Hello world"). Haureu de fer servir les utilitats que hi ha a Linux per a editar un programa escrit en C, compilar-lo i muntar-lo.

2. Les versions actuals de Linux basades en l'arquitectura IA-32 implementen les crides al sistema utilitzant la instrucció `sysenter` en lloc de la instrucció `int`. Investigueu quins avantatges té la utilització de `sysenter` respecte de la utilització d'`int`.

Exercicis d'autoavaluació

1. Quines diferències hi ha entre invocar una rutina d'un programa i invocar una crida al sistema?

2. En aquest mòdul hem mencionat alguns escenaris en què es genera una excepció. Quins són?

3. Dieu quines són les diferències més rellevants entre els modes d'execució del processador (mode privilegiat i mode no privilegiat).

4. En quin mode d'execució es troba el processador quan...

a) un usuari ordinari executa codi propi d'una aplicació de sistema (com ara l'intèrpret de comandes)?
b) un usuari privilegiat (l'administrador de la màquina) executa codi propi d'una aplicació de sistema?

5. Quines de les següents instruccions del llenguatge màquina són privilegiades?

a) Llegir el registre de control del disc.
b) Accedir a la memòria.
c) Sumar dos registres de propòsit general del processador.
d) Inhibir l'atenció a les interrupcions.
e) Fer que el processador passi a un mode de baix consum energètic.

6. Des del punt de vista del planificador del processador, quina diferència hi ha entre que un procés implementi una sincronització utilitzant una espera activa o una sincronització utilitzant una operació bloquejant?

7. Què es necessita per a convertir un programa en procés?

8. Per quins motius l'MS-DOS era un sistema operatiu inestable?

9. Des del punt de vista del planificador del processador, perquè és necessària l'existència d'un dispositiu temporitzador que generi interrupcions periòdicament?

10. Un cop llegit l'annex d'aquest mòdul, contesteu les preguntes:
a) Podeu explicar les funcions del muntador?
b) Amb quines altres eines del sistema té relació?
c) Imagineu que heu escrit un únic fitxer de programa font; per què l'heu de muntar?

Solucionari

Exercicis d'autoavaluació

1. Invocar una crida al sistema provoca un canvi de mode d'execució; la determinació de la posició de memòria on es troba la rutina es fa en temps d'execució i el codi de la rutina no forma part del fitxer executable.

2. En aquest mòdul hem vist: la divisió per zero, l'intent d'executar una instrucció privilegiada del llenguatge màquina en mode d'execució no privilegiat i un accés a la memòria invàlid.

3. En mode privilegiat, el processador pot executar qualsevol instrucció del llenguatge màquina. En mode no privilegiat, únicament es poden executar instruccions no privilegiades; si s'intenta executar una instrucció privilegiada en aquest mode d'execució, es produirà una excepció.

4. En tots dos casos es troba en mode d'execució no privilegiat. El processador es troba en mode d'execució privilegiat únicament quan executa codi d'atenció a interrupcions, excepcions i crides al sistema. Les aplicacions de sistema, independentment de quin usuari les executi, sempre s'executen en mode d'execució no privilegiat.

5. L'accés a la memòria i la suma de registres de propòsit general són no privilegiades; les altres són privilegiades.

6. En el primer cas, el procés està consumint CPU perquè fa la transició entre els estats Run i Ready. En el segon cas, el procés no està consumint CPU perquè durant tot el temps d'espera tindrà assignat l'estat Blocked.

7. Perquè un programa generi un procés nou li hem de donar "vida"; li hem d'assignar recursos, l'hem de copiar a la memòria i hem de fer que el processador n'iniciï l'execució.

8. L'MS-DOS era un sistema operatiu inestable perquè el processador per al qual estava dissenyat no disposava de mode d'execució "privilegiat" i no tenia MMU. Per tant, qualsevol programa en execució a la màquina tenia el control total d'aquesta.

9. Per a evitar que un procés d'usuari de càlcul intensiu (que efectua molt poques crides al sistema) pugui monopolitzar la CPU. El temporitzador dóna l'oportunitat al planificador del processador d'entrar en execució, amb la qual cosa pot decidir forçar un canvi de context amb un altre procés.

10.a) El muntador combina diferents mòduls objecte i les biblioteques per a construir el programa executable. La feina més important que porta a terme és resoldre les referències entre les diferents parts i establir la correspondència entre les adreces de les diferents variables i els procediments.
b) El muntador té relació amb les biblioteques i amb els mòduls objecte generats durant la fase de compilació.
c) Sempre s'ha d'utilitzar el muntador perquè sempre s'han d'afegir les capçaleres per al carregador i les rutines de les biblioteques (és difícil trobar una aplicació sense cap rutina d'entrada/sortida de les biblioteques del sistema).

Glossari

biblioteca *f* Conjunt de rutines d'ús comú ja implementats, disponibles per a tots els usuaris. Aquest conjunt de rutines codificades com a mòduls objecte està organitzat en fitxers que contenen el codi amb un índex que permet afegir aquest codi al del programador. Aquest pas es fa en la fase de muntatge del fitxer executable.

carregador *m* Eina que permet carregar un programa a la memòria per tal que pugui ser executat; per a fer-ho cal obtenir memòria, copiar-hi l'executable i donar-li control.

compilador *m* Eina que permet transformar un programa escrit en llenguatge d'alt nivell en un programa escrit en un llenguatge conegut per la màquina, la qual cosa permetrà que el pugui executar.

crida al sistema *f* Eina que permet a l'usuari utilitzar uns serveis que s'han d'executar dins del nucli del sistema.

depurador *m* Eina que permet controlar l'execució d'un programa pas per pas analitzant-ne el contingut de les variables; d'aquesta manera ens facilita la detecció dels errors i la seva correcció.

excepció *f* Transferència de control al sistema operatiu provocada per l'execució d'una instrucció d'usuari no prevista, com per exemple una fallada de pàgina de memòria, una divisió per zero, etc.

intèrpret de comandes (*shell*) *m* Interfície per mitjà de la qual l'usuari interacciona amb el sistema operatiu. L'intèrpret de comandes es basa en el diàleg entre l'usuari i el sistema; el sistema declara la seva disponibilitat presentant el l'indicador i l'usuari demana serveis introduint comandes. La majoria dels intèrprets de comandes admeten la programació de programes petits o scripts.
sin intèrpret d'ordres

interrupció *f* Transferència de control al nucli del sistema operatiu. Típicament és provocada de manera asíncrona pels dispositius del sistema per mitjà del maquinari (interrupcions *hardware*).

llibreria *f* Nom que també reben les biblioteques a causa d'una traducció incorrecta del terme anglès de *library*, un *false friend*, al català o al castellà.

muntador *m* Eina que s'encarrega d'agrupar els mòduls i les biblioteques necessaris per a construir un programa executable.

procés (tasca) *m* Programa en execució.

programa *m* Descripció detallada en forma d'instruccions que permet resoldre un problema determinat. Els programes han d'estar escrits en fitxers de text i han de seguir unes normes sintàctiques determinades per un llenguatge de programació.

programa font *m* Fitxer que conté les instruccions del programa escrites en un llenguatge informàtic.

programa objecte *m* Fitxer que conté un mòdul d'un programa escrit en llenguatge màquina i la informació necessària per a ser muntat amb altres programes.

trap *m* Mecanisme de maquinari que ens permet fer crides explícites als serveis del sistema operatiu.

Bibliografia

Silberschatz, A.; Galvin; Gagne, G. (2008, 8a. ed.). *Operating Systems Concepts*. John Wiley & Sons.

Tanembaum, A. (2009). *Modern Operating Systems*. Prentice-Hall.

Documentació disponible als recursos de l'aula sobre l'intèrpret de comandes de Linux.

Annex

A) Les fases d'execució d'un programa

AA) El llenguatge informàtic

Els programadors d'aplicacions escriuen els seus programes en fitxers de text que segueixen les normes d'un llenguatge informàtic determinat. Aquestes normes determinen la gramàtica del llenguatge de programació. Això és degut al fet que s'intenta utilitzar un llenguatge tan proper com sigui possible al llenguatge natural per tal de descriure la solució del problema. Per a les persones de parla anglesa els llenguatges de programació són força naturals, atès que la majoria estan escrits en la seva llengua. El fitxer generat és de text i, per tant, no es pot executar de manera immediata.

A l'hora de resoldre un problema determinat, els programadors d'aplicacions informàtiques han d'escriure les instruccions o les sentències d'un programa seguint unes normes. El format que utilitzen s'anomena **llenguatge informàtic**.

La figura 8 presenta un exemple d'ús de llenguatge informàtic. El programa que es representa efectua la multiplicació de dues variables `a` i `b` i deixa el resultat a `c`. Suposem que les variables `a` i `b` estan inicialitzades als valors corresponents al càlcul desitjat.

```
c = 0;
for (i = 0; i &#8249; b; i++)
      c = c + a;
```

Figura 8. Programa d'exemple que calcula el producte dels enters `a` i `b`. Noteu que el programa presentat és únicament un exemple i no conté tot el codi necessari.

Un **llenguatge de programació o informàtic** és una convenció de sentències i estructures de dades que, mitjançant un procés de traducció o interpretació, es pot convertir en llenguatge màquina i es pot executar. Amb els llenguatges de programació es poden escriure algorismes.

La idea dels llenguatges de programació és que els programes siguin independents de l'arquitectura física de l'ordinador i del sistema operatiu en què s'executin finalment. És a dir, que siguin programes **portables a altres sistemes** i que siguin útils sense haver d'introduir-hi canvis. Organismes com l'American National Standards Institute (ANSI) i la International Standards Organization (ISO) s'ocupen de la normalització dels diferents llenguatges. El

sistema operatiu ha de proporcionar, doncs, les eines necessàries per a portar a terme tot el procés de creació d'una aplicació informàtica que es pugui executar en el sistema del mateix sistema operatiu.

Classifiquem els diferents tipus de llenguatges de programació de la manera següent:

- **Llenguatge d'alt nivell**. Quan les instruccions s'escriuen en un format de text molt proper al llenguatge natural, encara que amb unes regles molt més estrictes, diem que estem utilitzant un llenguatge d'alt nivell.
  L'exemple anterior està escrit en un llenguatge d'alt nivell. La màquina únicament pot executar el llenguatge màquina propi i, per tant, hi ha d'haver un procés que transformi el programa anterior a un format conegut per la màquina. Aquest procés s'anomena, de manera genèrica, **compilació**.

- **Llenguatge assemblador**. Per a simplificar l'ús directe del llenguatge màquina s'han dissenyat llenguatges en els quals cada instrucció màquina correspon a una instrucció en un llenguatge mnemotècnic més intel·ligible. Un d'aquests llenguatges és el llenguatge assemblador, que permet escriure un codi molt eficient pel fet que es poden escriure les instruccions nadiues del processador controlant tota l'execució a un nivell molt baix. Cal dir, però, que el llenguatge màquina és cada dia més sofisticat i que és pràcticament inviable programar-lo "a mà", de manera que aquesta tasca tan complicada quedarà reservada als compiladors i als optimitzadors de codi.

- **Codi màquina**. Finalment, les úniques instruccions que reconeix un processador són les que estan codificades en el seu codi particular. Aquest codi és binari, en el sentit que s'interpreten directament les seqüències de zeros i uns. Es pot executar de manera directa i rep el nom de codi màquina. Normalment, una sentència de llenguatge d'alt nivell genera diverses instruccions de codi màquina.

A la figura 9 veiem com seria l'aspecte del programa de l'exemple anterior escrit en llenguatge d'alt nivell, en llenguatge assemblador i en codi màquina. A l'hora de codificar les instruccions `CALL` i `RET` s'utilitza el mateix codi que per a la instrucció `BEQ` i s'utilitza el camp d'origen per a distingir de quina instrucció es tracta: 0 per a `BEQ`, 1 per a `CALL` i 2 per a `RET`.

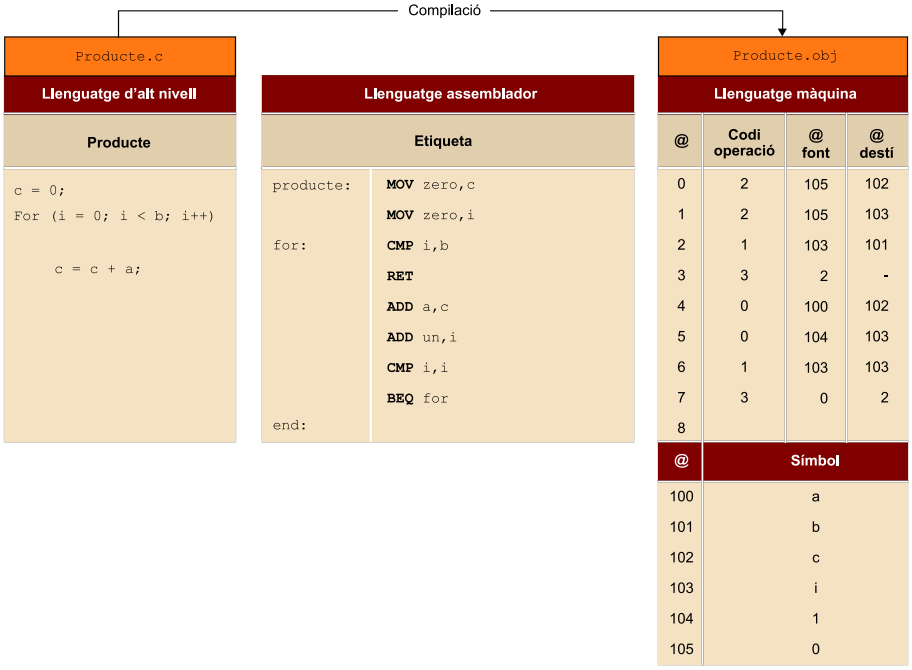

Figura 9. Tres versions d'una rutina d'exemple en llenguatge d'alt nivell, en llenguatge assemblador i en llenguatge màquina. Aquest exemple es basa en la idea d'una màquina senzilla amb un joc d'instruccions molt reduït. Per exemple, no disposa de la instrucció producte

Hem representat el codi màquina en notació decimal per a cada camp. El codi d'operació només ocupa 2 bits i els codis de la variable d'origen i de destí ocupen 7 bits cadascun. Sempre fan un adreçament directe a la posició de memòria de l'operand, per la qual cosa els valors constants zero i u s'han d'enregistrar en una posició de memòria.

La màquina no disposa d'una operació de salt incondicional. Per tal de simular-la (anar a l'etiqueta `for:`), fem que el resultat de la comparació prèvia sigui sempre cert, cosa que aconseguim comparant qualsevol posició de memòria amb ella mateixa (en aquest exemple, la variable `i`).

Vegeu també

Podeu veure el valor de les posicions de memòria de cada variable a la taula de símbols que hem definit en aquest annex.

AB) Les eines per a la creació d'una aplicació informàtica

Dins els programes d'utilitat del sistema operatiu, n'hi ha un ampli ventall de catalogats com a programes de suport als llenguatges de programació, entre els quals podem trobar les eines següents:

- **Editors de text**: formen part de les utilitats del sistema que permeten l'edició de fitxers que contenen informació en format de text. Aquests fitxers poden ser de missatges, de configuració del sistema o de petits programes que utilitzen l'intèrpret de comandes del sistema operatiu (scripts o *shellscripts*), però també poden ser fitxers de codi d'alt nivell d'una aplicació informàtica. Recordem que els programes escrits en alt nivell no són sinó fitxers de text escrits amb les normes d'un llenguatge d'alt nivell.

- **Compiladors-assembladors**: són programes que transformen els programes editats en format de text a un format que sigui executable per la màquina. Els fitxers resultants s'anomenen fitxers executables, i el seu con-

tingut està representat en el format binari, que el maquinari pot executar de manera directa. Tant els programes font d'alt nivell com els editats en assemblador estan escrits en format de text. No és habitual fer la transformació de programes d'alt nivell a executables i que la transformació es pugui fer en un sol pas. Més endavant detallarem aquest procés.

- **Biblioteques**: per a ajudar a crear programes, el sistema operatiu permet la gestió de biblioteques. Aquestes permeten estendre les funcions que es poden utilitzar de manera directa i, així, estalvien al programador la feina feixuga de programar molts procediments d'entrada/sortida o de gestió de fitxers que es repeteixen per a qualsevol tipus d'aplicació. Utilitzant les biblioteques aprofitem un conjunt de rutines que ja està molt depurat i experimentat.

- **Muntadors (enllaçadors o *linkers*)**: són aplicacions que agrupen diferents mòduls objecte i les biblioteques del sistema per a obtenir un programa executable únic. La seva tasca principal és resoldre les referències creuades entre els diferents elements que no han estat resoltes en les primeres fases de compilació. Encara que de vegades es troben com a aplicacions separades dels sistemes operatius, normalment s'executen de manera automàtica en la fase final dels entorns integrats de compilació.

- **Depuradors**: estan formats per una sèrie d'aplicacions que permeten l'execució controlada d'un programa a fi de resoldre possibles errors. Per a fer viable la utilització d'un depurador és necessari fer les fases prèvies en aquesta modalitat. El compilador i el muntador han de preparar el codi resultant per a aquest fi.

- **Executadors-carregadors**: són l'últim pas del procés i permeten l'execució dels fitxers executables resultants dels passos anteriors després d'assignar-los recursos del sistema. En els sistemes operatius antics aquesta ordre era explícita, mentre que actualment, en indicar el nom d'un fitxer executable s'interpreta que s'ha d'iniciar la seva execució.

AC) El procés de creació d'un programa executable

Utilitzant les eines que hem descrit podem crear i executar una aplicació. Per a poder-ho fer hem de passar per un conjunt de fases que desenvolupem tot seguit.

ACA) L'edició

En aquesta fase el programador ha de transcriure totes les idees per a solucionar els problemes en un fitxer de text anomenat **fitxer font**. Les instruccions del programa s'han d'escriure seguint les normes estrictes del llenguatge de programació que s'empri.

L'eina que s'utilitza és qualsevol processador de text, que, de fet, pot ser molt senzill, ja que no calen grans prestacions quant al format del text. En canvi, és convenient que pugui gestionar més d'un fitxer simultàniament o que disposi de bones eines de cerca i de manipulació del text. Hi ha editors específics per a alguns llenguatges de programació que poden resoldre certs problemes sintàctics del codi o poden presentar les parts d'un programa en diferents colors.

Exemple

Es poden fer servir els recursos de l'editor de text per a diferenciar les paraules reservades, els noms de les variables, els noms dels procediments, etc.

ACB) La compilació (assemblatge)

Amb el procés de traducció (compilació) d'un programa escrit en llenguatge d'alt nivell, anomenat programa font, n'obtenim un d'equivalent en llenguatge màquina que anomenem **programa objecte**. Si el programa font és l'assemblador, aquesta fase també s'anomena **assemblar**, i en aquest cas la traducció és més senzilla, atès que, com hem dit, a cada instrucció en assemblador li correspon una instrucció màquina. A partir d'aquesta fase no hi ha cap diferència entre els objectes que provenen d'un llenguatge d'alt nivell i els que són de format assemblador.

Nota

Aquest programa equivalent tindrà la mateixa informació, però en un format diferent.

L'estructura d'un programa objecte ja conté els codis corresponents a les instruccions màquina, però el programa objecte no pot resoldre totes les adreces que inclou a causa de les dues raons següents:

- En primer lloc, perquè els programes d'una aplicació habitualment estan escrits en diferents fitxers o mòduls. Així, si hem de fer referència a una variable o a un procediment que està escrit en un altre mòdul, no la podem resoldre en aquest pas.

- En segon lloc, perquè les rutines d'ús molt comú, com ara els càlculs matemàtics més o menys complexos, el control de determinades operacions d'entrada/sortida amb fitxers o dispositius i altres procediments molt genèrics ja estan generalment programades, compilades i incloses en un conjunt de fitxers anomenats biblioteques.

Les **biblioteques** són un conjunt de mòduls objecte organitzats adequadament. El mateix fabricant del programari de suport als llenguatges de programació sol facilitar aquest fitxer. Gràcies a les biblioteques, el programador s'estalvia molta feina i es redueixen considerablement la mida del programa i

el temps de programació. Normalment el fabricant no facilita el programa font de les biblioteques, però sí que proporciona les instruccions i els paràmetres necessaris per a utilitzar-les.

Tant els mòduls objecte com les biblioteques indiquen les adreces de manera relativa a l'inici del mòdul. Després d'aquesta primera fase de compilació disposem d'un conjunt de mòduls objecte amb referències d'adreçament relatives al mòdul i també altres referències externes sense resoldre. En aquesta fase es creen tres estructures:

- **La taula de símbols**, que indica la ubicació de cada símbol definit en un mòdul objecte (variable, procediment o constant).
- **La taula de referències externes**, que indica quines instruccions fan referència a símbols no definits dins d'un mòdul objecte.
- **La taula de reubicació**, que indica quines referències a símbols s'hauran de modificar en el procés de muntatge. No és objecte d'aquesta assignatura, però, donar els detalls de com es tracten aquestes taules ni de la manera en què s'utilitzen.

Exemple de compilació

Considerem que hi ha dos mòduls, `producte.c` i `principal.c`. A `principal.c` (figura 10) trobem un bucle que imprimeix els quadrats dels n primers nombres. En l'exemple hem pres el valor n = 10. Aquest programa fa una sèrie de crides a un segon mòdul en què hem escrit el codi per al producte. Recordeu que només és un exemple didàctic en el qual no ho hem resolt tot. Primerament podem veure, de manera molt simplificada, com és la compilació de `producte.c`. Presentem el codi assemblador només per a intuir el significat del codi màquina. Les constants 0 i 1 s'han carregat en posicions de memòria, ja que la màquina disposa únicament d'adreçament directe.

És important veure que ara hi ha una sèrie de referències externes no resoltes, com les adreces de les variables `a` i `b` i les dels procediments `producte` i `printf`.

```
n = 10;
for (i = 0; i &#8249; n; i++){
      a = i;
      b = i;
      producte();
      printf(_i = %d quadrat =...
}
```

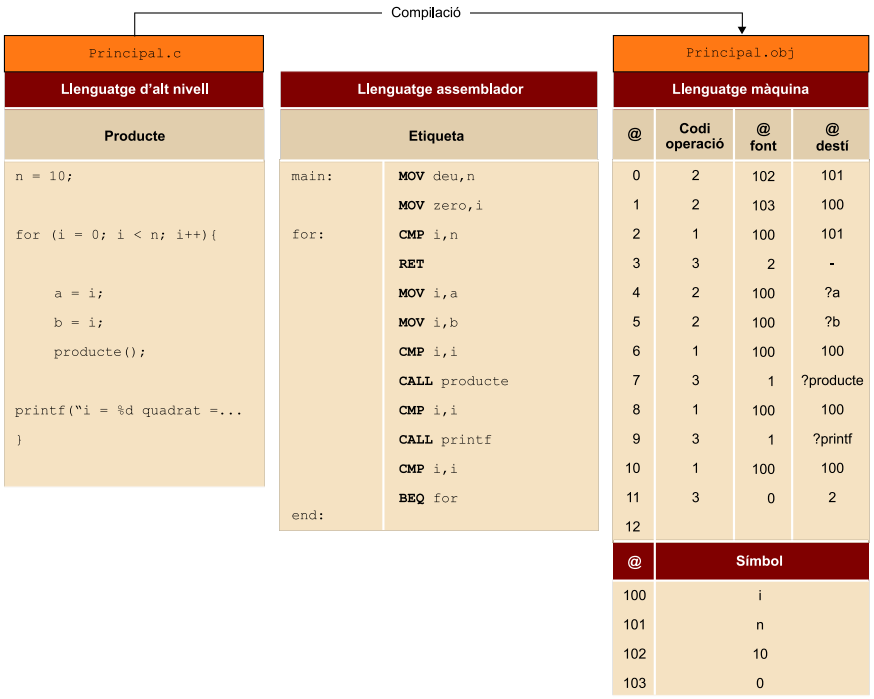

Figura 10. Codi en alt nivell, assemblador i llenguatge màquina del programa `principal.c`

ACC) El muntatge (enllaçat o linkatge)

El muntatge és el procés encarregat d'agafar tots els mòduls corresponents a un programa, i també les biblioteques si cal, i agrupar-los per a construir el programa executable. Els programes que fan aquestes funcions s'anomenen muntadors.

De fet, la feina principal de l'enllaçador és resoldre les adreces externes entre els mòduls, i també amb la biblioteca. El resultat final de l'enllaçador és el programa executable definitiu amb un espai lògic d'adreces completament construït.

La **capçalera d'un fitxer executable** conté informació relativa a la manera en què s'ha de carregar en la memòria, com ara la grandària de la pila i de les dades no inicialitzades o dinàmiques, el nombre de segments (si n'hi ha), l'especificació de si el codi pot ser compartit amb altres processos, etc. També indica si es genera codi per a la seva depuració i, en aquest cas, adjunta informació amb el nom de les variables, dels procediments i de les adreces que siguin necessàries. És habitual que la capçalera contingui una marca, que és la **signatura del fitxer**. Amb aquest identificador, el sistema operatiu sap quan un fitxer és realment executable, sempre que la versió del sistema operatiu sigui correcta o el codi màquina generat sigui l'apropiat.

L'enllaçat dels mòduls de l'exemple proposat és el que podeu veure al gràfic de la figura 11.

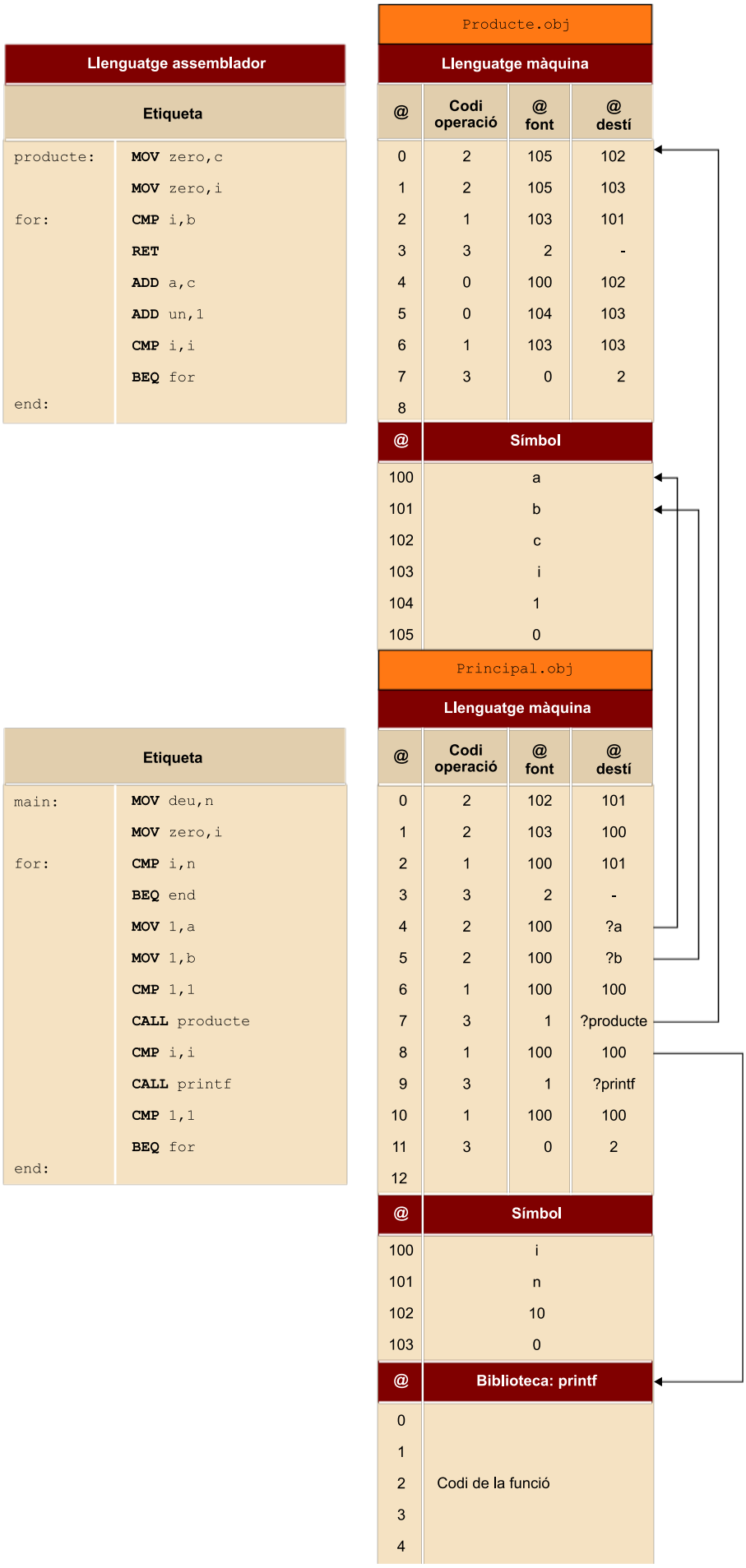

Figura 11. Enllaçat dels mòduls d'exemple

Podem veure que a l'hora de generar el fitxer executable sorgeixen diferents problemes:

- Les adreces de les tres parts que s'han d'enllaçar comencen per 0.
- Hi ha dos símbols locals amb el mateix nom: `i`.
- Hi ha variables i procediments encara sense muntar.
- Les adreces dels salts incondicionals (`BEQ`) poden canviar.

Per a solucionar aquests problemes, el fitxer executable podria ser semblant al que presentem a la figura 12.

Com podem veure, s'han resolt les referències externes. El compilador no pot simplificar la doble definició de la constant del zero i la manté per separat. L'ordre dels mòduls és irrellevant, de manera que podríem crear un executable equivalent alterant l'ordre dels mòduls. També cal subratllar que el codi de la biblioteca s'ha gestionat com un mòdul més i que s'hi ha afegit el codi de la rutina de biblioteca `printf`. D'altra banda, si alterem un sol mòdul objecte, és clar que la resta d'objectes són perfectament vàlids per al seu muntatge.

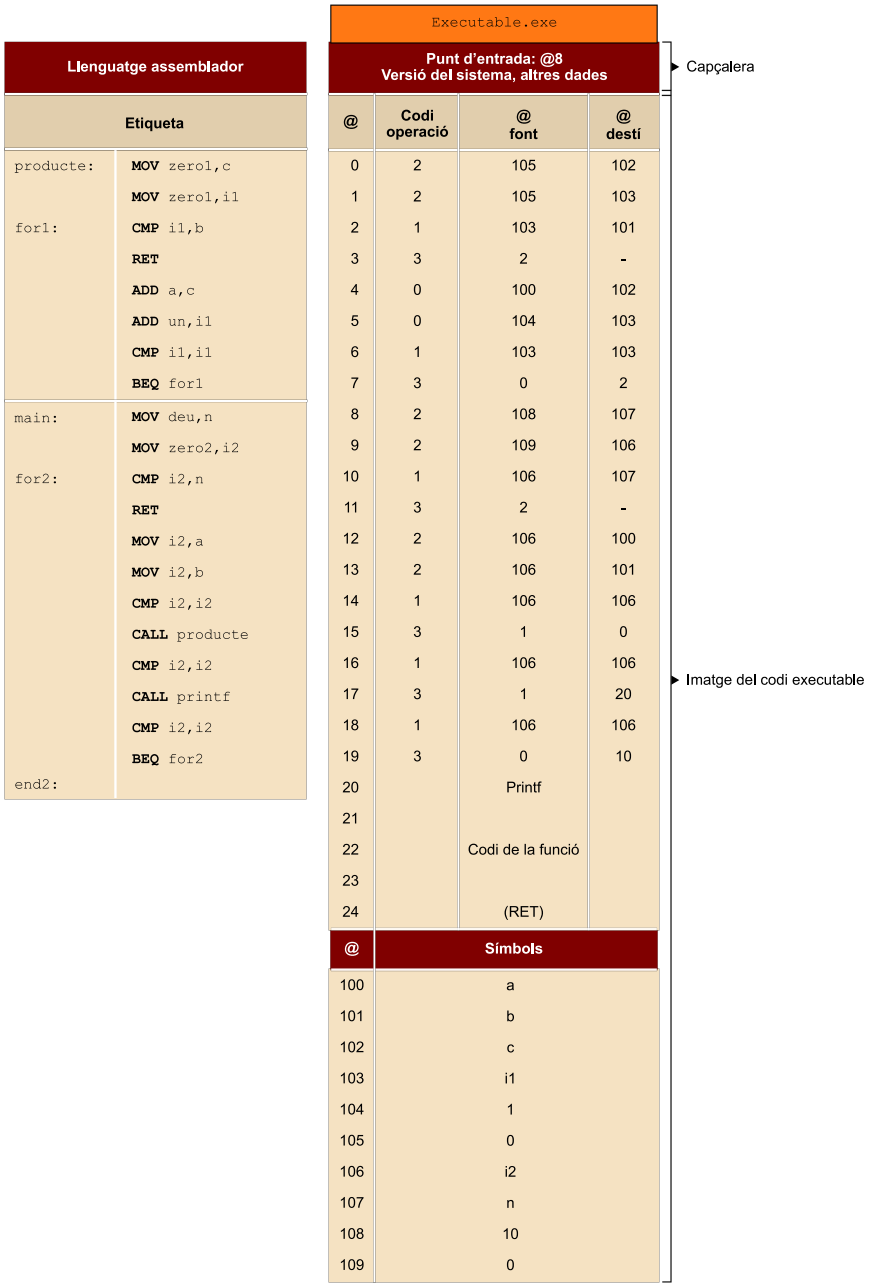

Figura 12. Fitxer executable corresponent al programa d'exemple

ACD) La càrrega

La tasca principal del carregador és buscar els recursos que demana el programa executable. Entre aquests recursos hi ha la memòria. El **carregador** ha de buscar memòria lliure, copiar-hi el codi i també assignar-hi espai per a dades i pila.

Tot aquest procés es fa adaptant les adreces lògiques dels programes executables a les adreces físiques de la memòria real del sistema. Finalment, el carregador transfereix el control a la primera instrucció del programa i marca el procés com a preparat.

Quan molts usuaris criden el mateix programa, si aquest permet que s'utilitzi en modalitat compartida, el carregador mirarà si hi ha alguna còpia en la memòria del programa. Si la resposta és afirmativa, farà que el nou procés tingui tot el seu entorn propi excepte la part de memòria on resideix el codi, que serà compartida amb altres processos. Recordem l'exemple dels programadors que utilitzen alhora l'editor.

ACE) L'execució (depuració)

Mentre es porta a terme el procés de desenvolupament d'un programa o d'una aplicació informàtica, sol ser molt interessant seguir l'execució pas per pas, cosa que és factible gràcies als programes depuradors (*debuggers*).

Els **programes depuradors** tenen com a única missió controlar l'execució d'un programa i permeten que el programador pugui veure una execució pas per pas (instrucció per instrucció) o per intervals de programa tot visualitzant constantment l'entorn que es va produint. El programador fixa els punts on s'ha d'aturar l'execució i en cada parada pot comprovar si l'entorn és el correcte i, fins i tot, pot modificar-lo.

Així es poden veure i corregir errors que difícilment es trobarien d'una altra manera i s'obté un programa final més eficient (figura 13).

El fet de partir el programa font en diferents fitxers presenta diversos avantatges tant per al programador com per al sistema:

- D'una banda, perquè una aplicació informàtica pot tenir desenes de milers de línies de codi, i editar un fitxer d'aquestes dimensions pot resultar poc pràctic.
- El procés de compilació també en surt beneficiat, perquè en la primera fase només ha de traduir els mòduls de codi font que s'hagin modificat. En la fase final o d'enllaç sí que ha d'actuar en tots els mòduls objecte, tant els que s'han modificat com els antics.

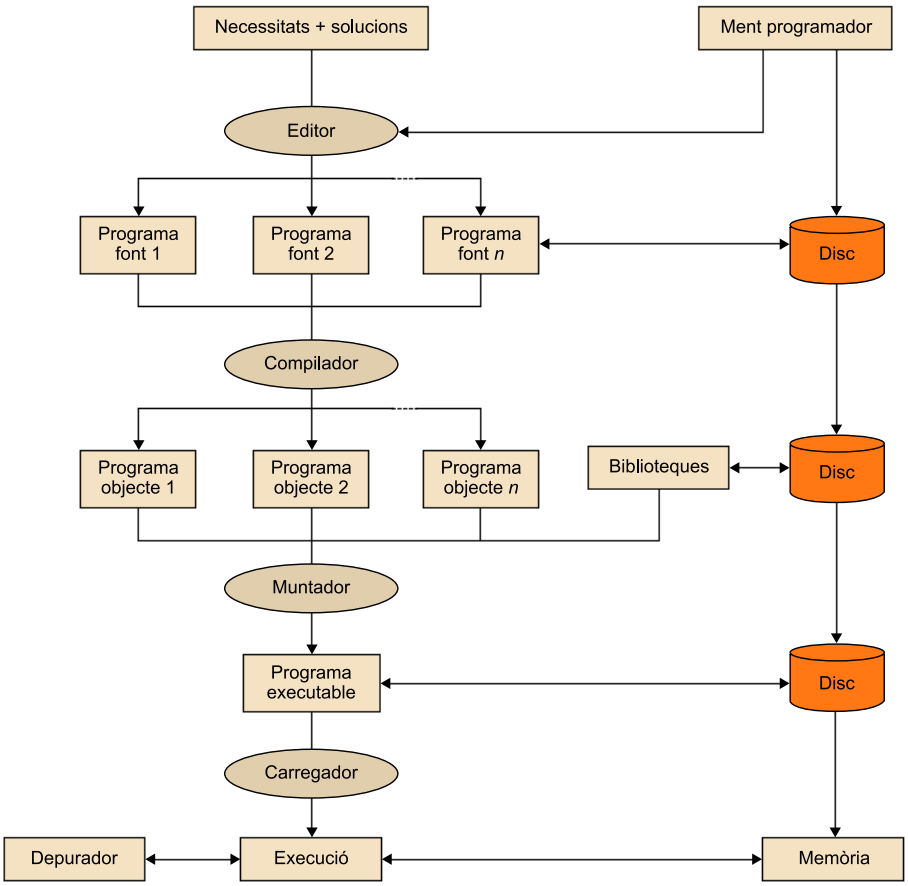

Figura 13. Etapes de la creació d'un programa executable

ACF) Interpretació enfront de compilació

Els intèrprets intenten unificar tot el procés de creació d'un programa (figura 14). A mesura que es va llegint cada instrucció del llenguatge d'alt nivell es transforma en el conjunt corresponent d'instruccions màquina, que s'executa immediatament. L'interpretador controla directament totes les fases de la creació i l'execució del programa. És un pas necessari per a l'execució, ja que no es genera cap altre fitxer i sempre es treballa amb el fitxer font.

La interpretació de programes va ser el primer procés que va aparèixer, atès que els ordinadors eren poc potents i el procés de compilació massa costós. Era massa lent per a aplicar-lo contínuament durant el procés de creació i depuració de l'aplicació. Amb l'augment progressiu de la potència dels sistemes informàtics, la fase de compilació és cada dia més ràpida i permet treballar de manera contínua en aquesta modalitat. En els darrers temps, amb l'evolució de les xarxes de comunicacions i la creació d'aplicacions pensades per a aquest entorn, s'està tornant a l'execució interpretada. Els procediments s'envien en codi d'alt nivell fins a les màquines remotes que els executen mitjançant l'ús

d'intèrprets. Aquesta modalitat té dos grans avantatges: el codi d'alt nivell és més compacte que el codi màquina i, a més, és independent de la plataforma en la qual s'executa.

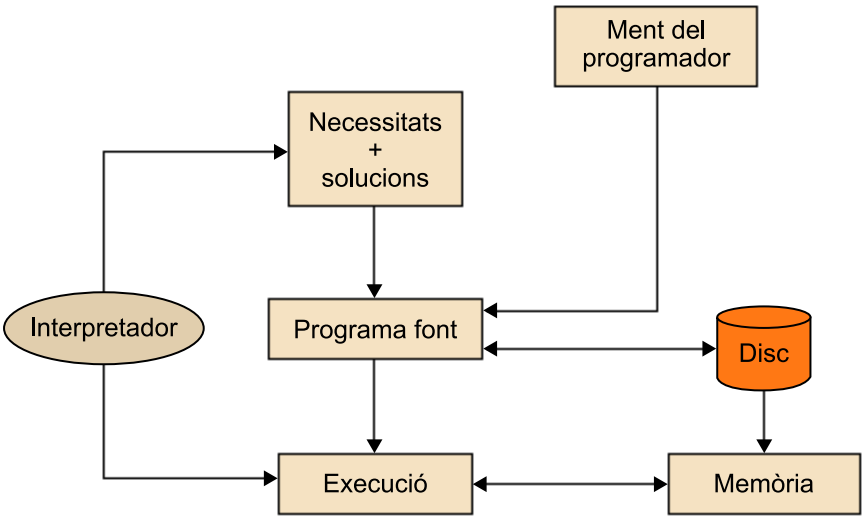

Figura 14. Procés d'interpretació

B) Els espais d'adreces d'un procés

Ara analitzarem el mecanisme de càrrega d'un programa a la memòria i la manera en què aquesta memòria es diferencia de la memòria assignada a altres processos o de l'assignada al sistema operatiu mateix.

BA) L'espai lògic i l'espai físic

Partim de l'espai definit en el fitxer executable, el qual pot tenir una estructura d'una o dues dimensions (espai lineal o segmentat). Aquests espais s'han de transportar des dels fitxers executables fins a la memòria que gestiona el sistema operatiu. L'estructura que té aquesta memòria pot ser molt diferent en funció del model escollit pel sistema operatiu.

Un programa reubicable es pot executar en adreces diferents, que són assignades pel carregador. Per aquest motiu es distingeix entre adreces lògiques i adreces físiques:

- **Adreces lògiques**: Són les referenciades pel programador. Solen ser identificadors que reconeixen les diferents parts d'un programa dins del seu espai d'adreces. Són referències locals en el sentit que l'entorn en què s'hagin d'executar no té importància.

- **Adreces físiques**: Són les adreces assignades en temps de càrrega, és a dir, l'indret on residirà realment el procés durant la seva execució.

BB) La càrrega a la memòria: reubicació

La propietat de la reubicació fa referència a la capacitat de carregar i després executar un programa en qualsevol posició de memòria. És la propietat contrària a l'obligatorietat de conèixer totes les adreces de manera fixa en temps de compilació. Òbviament, hi ha un mecanisme de traducció entre les adreces lògiques i les físiques. En funció de com es faci, podem distingir dos tipus de reubicació:

- **Reubicació estàtica**: es realitza abans o durant la càrrega del programa a la memòria per a ser executat. Durant el procés de creació d'un programa executable, normalment es pren l'adreça lògica 0 com a adreça inicial del programa. D'aquesta manera, totes les altres adreces del programa seran adreces lògiques relatives a l'adreça final de càrrega del programa. Quan es carrega la imatge del programa, totes les adreces marcades com a reubicables es transformen en l'adreça física final. La referència per a poder-ho fer és l'adreça lògica inicial.
  Un cop s'han modificat totes aquestes adreces per a reubicar-les, no es poden distingir les adreces reubicades de les originals. En altres paraules, no es pot tornar a reubicar durant l'execució. Si per alguna raó el procés ha de sortir del seu espai físic assignat, o bé torna exactament al mateix lloc, o bé s'ha d'iniciar el procés de reubicació des del principi, és a dir, s'ha de tornar a carregar.
  A causa del problema anterior, la reubicació estàtica està pràcticament limitada a certs sistemes operatius en què el mecanisme de càrrega de programes a la memòria és molt determinat i el codi ha d'anar a particions de memòria predeterminades.

- **Reubicació dinàmica**: en aquest cas, la transformació entre les adreces físiques i lògiques s'efectua en temps d'execució. Com abans, els programes executables resultants de les compilacions adopten l'adreça inicial de referència 0.
  Aquestes imatges dels programes executables es carreguen directament a qualsevol posició de memòria, i és una part del maquinari la que s'encarrega de fer la feina de reubicació d'adreces lògiques a físiques.
  Quan el procés s'està executant, abans d'accedir realment a la memòria física es reubiquen tots els accessos a la memòria. Aquest procés es porta a terme utilitzant els registres de base. Cada referència que fa un procés durant la seva execució és transformada per la suma de l'adreça lògica més el registre de base per a obtenir l'adreça física. Aquest mecanisme és totalment transparent per al programador.
  La transformació d'adreces depèn en gran mesura del maquinari del sistema. Per aquest motiu no en podem donar més detalls sense parlar d'una arquitectura d'ordinador determinada; però, de fet, això no és objecte d'aquesta assignatura.

C) Mapa conceptual

El mapa conceptual de la figura 15 reflecteix les relacions que hi ha entre els diferents conceptes que hem presentat en aquest mòdul.

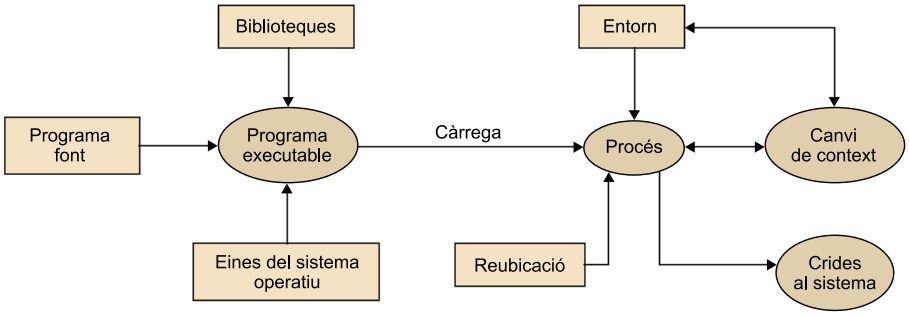

Figura 15. Mapa conceptual