La gestió de la memòria

José Ramón Herrero Zaragoza
Enric Morancho Llena
Dolors Royo Vallés

PID_00169369

UOC
Universitat Oberta de Catalunya

Índex

Introducció

En aquest mòdul didàctic presentem els principis de la gestió de la memòria principal, un dels recursos més importants de qualsevol sistema operatiu multiprogramat. La **gestió de la memòria** s'encarrega bàsicament d'assignar la memòria física del sistema computador, que és finita, als processos que la sol·liciten. Cap programa no es pot executar si no se li ha assignat memòria principal per a emmagatzemar-hi el seu codi, les seves dades i la seva pila d'execució.

Primer s'introdueixen els **espais d'adreces**, que són el lògic i el físic. Mentre que l'espai físic és únic, cada procés en execució tindrà associat un espai lògic; cada espai lògic serà independent dels de la resta de processos. El sistema de gestió de la memòria amagarà l'espai físic als usuaris i la visió que aquests tindran de la memòria serà l'espai lògic. Per tant, serà necessari un mecanisme que tradueixi, de forma transparent per a l'usuari, els accessos a l'espai lògic en accessos a l'espai físic.

La coexistència de múltiples espais d'adreces lògics que pertanyen a diferents processos residents a la memòria principal introdueix la necessitat de protegir els espais d'adreces. En un entorn multiprogramat, el gestor de la memòria hauria de disposar de **mecanismes de protecció de la memòria**, és a dir, hauria de garantir que un procés no pogués accedir a la memòria física assignada a un altre procés o al sistema operatiu. També és desitjable que, si els processos ho autoritzen, accepti la **compartició de la memòria** i permeti, així, espais comuns per als processos interessats.

Presentem la implementació d'un gestor de memòria senzill que, amb l'ajut del maquinari, compleix la majoria dels requeriments indicats. Un cop analitzades les seves limitacions, descrivim el gestor de memòria virtual. Aquest darrer sistema serà capaç d'independitzar els processos de la quantitat de memòria física instal·lada a la màquina, amb la qual cosa permetrà executar processos que no caben a la memòria física.

Finalment, reflexionem sobre les possibilitats d'ampliar, en temps d'execució, l'espai lògic d'un procés. També comentem alguns errors de programació típics relacionats amb l'ús de la memòria.

Objectius

Els materials didàctics d'aquest mòdul contenen les eines necessàries perquè l'estudiant assoleixi els objectius següents:

1. Justificar l'existència de diversos espais d'adreces i dels mecanismes de traducció d'adreces en temps d'execució.

2. Conèixer les funcionalitats que han d'oferir els sistemes de gestió de la memòria en un sistema operatiu multiprocés de propòsit general.

3. Saber com es pot augmentar, en temps d'execució, l'espai lògic dels processos.

1. Espais d'adreces

La memòria física (RAM) del computador és un vector de mots (típicament de la mida d'un byte), cadascuna de les quals s'identifica mitjançant una adreça (@).

En els sistemes operatius multiprocés, els processos en execució (i el sistema operatiu mateix) comparteixen la memòria física. Cada procés (i el sistema operatiu) té assignat un conjunt d'adreces físiques per a emmagatzemar-hi el seu codi, les seves dades i la seva pila d'execució. Ara bé, el sistema operatiu ha de garantir que cap altre procés podrà accedir al conjunt d'adreces assignat a la resta de processos (i al sistema operatiu).

Una possible solució passa per considerar dos tipus d'espais d'adreces (l'espai físic i l'espai lògic) i un mecanisme de traducció d'adreces lògiques a adreces físiques:

- **Espai físic**: està determinat per la memòria física instal·lada en el sistema computador. Les adreces de memòria d'aquest espai s'anomenen adreces físiques. Assumirem que el rang d'adreces físiques de l'espai físic és lineal i que comença a l'adreça física 0 i finalitza a l'adreça Memòria_Física_Instal·lada - 1.

- **Espai lògic**: aquest espai és la visió que el sistema operatiu ofereix de l'espai físic. Cada procés té associat un espai lògic, independent dels de la resta de processos, a través del qual pot accedir al seu codi, a les seves dades i a la seva pila d'execució. Les adreces de memòria d'aquest espai s'anomenen adreces lògiques. Assumirem que aquest espai és lineal. Potencialment, el rang d'adreces lògiques de l'espai lògic comença a l'adreça lògica 0 i finalitza a l'adreça $2^{Mida\_Bus\_Adreces}$-1, on Mida_Bus_Adreces està determinada per la mida del bus d'adreces del processador (típicament, 32 o 64 bits en els processadors actuals). Cal notar que per a cada procés existeix un rang d'adreces lògiques vàlides, que dependrà de la quantitat de memòria que utilitzi el procés. La traducció d'adreça lògica a adreça física dependrà del procés que la realitzi; per exemple, l'adreça lògica 0x0bc00000[1] pot ser traduïda a adreces físiques diferents per a cada procés del sistema.

(1)En aquest mòdul, mostrem les adreces lògiques i les adreces físiques codificades en hexadecimal.

Hi ha dues alternatives per a implementar aquesta traducció d'adreces:

- **Estàtica** (en temps de càrrega): en carregar un fitxer executable a la memòria es realitza la traducció de cada adreça lògica a l'adreça física corres-

ponent. Per tant, cada adreça lògica es tradueix una única vegada al llarg de l'execució del programa.

- **Dinàmica** (en temps d'execució): en carregar un fitxer executable a la memòria no es realitza cap tipus de traducció. L'execució del procés farà que el processador generi adreces lògiques i un maquinari especialitzat (la MMU, Memory Management Unit) traduirà cada adreça lògica a l'adreça física corresponent. A diferència del cas anterior, en què cada adreça lògica es traduïa exactament un cop, en aquest cas cada adreça lògica es tradueix tantes vegades com sigui referenciada.

Els sistemes operatius multiprocés actuals utilitzen mecanismes basats en la traducció dinàmica perquè són més flexibles que els basats en la traducció estàtica.

2. Un primer mecanisme de traducció dinàmica

A continuació presentem un mecanisme senzill de traducció dinàmica d'adreces. Tot i que aquest mecanisme permet aïllar l'espai lògic d'un procés (i del sistema operatiu) dels de la resta de processos, veurem que té algunes limitacions que el fan inapropiat per als sistemes computadors actuals.

Cada procés tindrà associat un espai lògic. El rang d'adreces vàlides de l'espai lògic del procés i-éssim serà [0, $mida_i$-1], on $mida_i$ és la mida de l'espai lògic del procés i-éssim (podem assumir que coincideix amb la suma de les mides de les seves regions de codi, dades i pila d'execució). Cada procés es carregarà en un seguit de posicions consecutives de la memòria física; $base_i$ serà l'adreça de memòria física base a partir de la qual es carrega el procés i-èssim.

La MMU d'aquest sistema realitzarà dues tasques:

- Comprovarà que les adreces lògiques (@L) generades per un procés es trobin dins del rang d'adreces vàlides per al procés. En cas que una @L no sigui vàlida, la MMU generarà una excepció del tipus "accés a memòria invàlid".

- Traduirà les adreces lògiques en adreces físiques (@F) mitjançant una suma de l'adreça lògica amb l'adreça base.

La MMU es programarà amb dos registres de control:

- Base: indica l'adreça física inicial assignada al procés en execució.
- Mida: indica la mida en bytes de l'espai lògic del procés en execució.

La figura 1 mostra el maquinari que realitza aquestes tasques. El processador genera adreces lògiques per a accedir al codi, a les dades o a la pila emmagatzemades a la memòria física. La MMU comprova si l'adreça lògica generada és vàlida per al procés (és a dir, si pertany al rang [0 ... mida-1]) utilitzant un comparador. Si no és dins del rang, la MMU genera una excepció del tipus "adreça lògica invàlida" i no realitza cap traducció. Si és dins del rang, suma l'adreça lògica a l'adreça física base i obté l'adreça física.

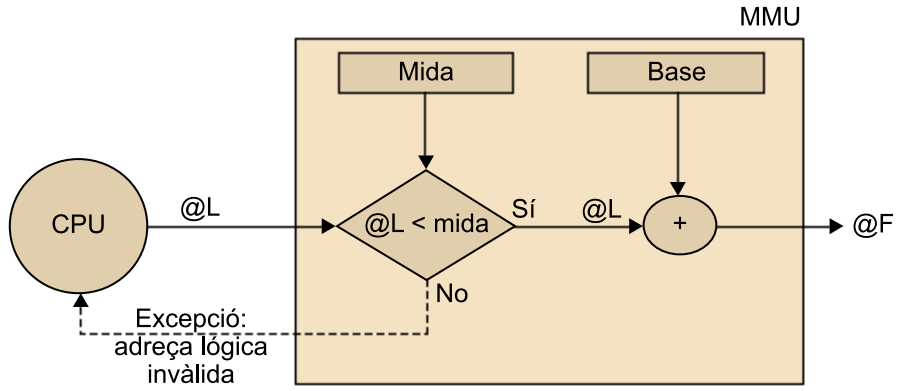

Figura 1. Esquema d'una MMU senzilla que realitza traducció dinàmica d'adreces i que permet aïllar l'espai lògic d'un procés del de la resta de processos

Per tal de garantir que el mecanisme aïlli el rang d'adreces d'un procés del de la resta de processos, les instruccions del llenguatge màquina que permetin modificar el valor dels registres de control de la MMU hauran de ser privilegiades. Cada vegada que canviï de context caldrà modificar el valor d'aquests registres de control.

Vegeu també

A l'apartat 3 del mòdul 2, trobareu més informació relacionada amb les instruccions privilegiades del llenguatge màquina.

La figura 2 presenta un exemple en el qual assumim que la memòria física instal·lada és d'1 MB (1.024 KB) i que està ocupada pel sistema operatiu (256 KB a partir de l'adreça $0x00000^2$), pel procés 1 (128 KB a partir de l'adreça 0x80000) i pel procés 2 (64 KB a partir de l'adreça 0xC0000).

(2)Com que l'espai físic és de 2^{20} posicions, 5 dígits hexadecimals ens permeten codificar qualsevol adreça de l'espai.

Volem crear un procés (procés 3) a partir d'un executable que defineix les regions següents: codi (64 KB), dades inicialitzades (16 KB), dades no inicialitzades (32 KB) i pila d'execució (16 KB). Si es crea un procés i es carrega aquest fitxer executable a la memòria, el sistema operatiu ha de saber quanta memòria necessita. La memòria necessària ve donada per la mida de les regions de codi, dades (inicialitzades i no inicialitzades) i pila. En aquest cas, serien necessaris 128 KB.

Vegeu també

Consulteu l'exemple de compilació a l'annex del mòdul didàctic 2.

El sistema operatiu ha de buscar memòria física per a encabir-hi aquest procés. Per fer-ho, necessitarà alguna estructura de dades que representi l'espai lliure a la memòria física. Assumim que el sistema operatiu decideix carregar el procés en el rang d'adreces físiques que comença a partir de l'adreça 0x40000. El sistema operatiu haurà de guardar en alguna estructura de dades la mida (128 KB) i l'adreça física base (0x40000) associades a aquest nou procés.

Estructura de dades del SO amb la informació necessària per traduir les adreces lògiques de cada procés (i del propi SO)

	Base	Mida
SO	0x00000	256 KB
Procés 1	0x80000	128 KB
Procés 2	0xC0000	64 KB
Procés 3	0x40000	128 KB

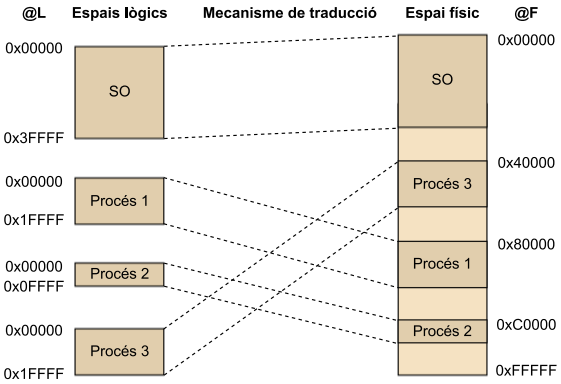

Figura 2. Exemple en què el sistema operatiu i tres processos comparteixen l'espai físic, i en el qual el mecanisme de traducció dinàmica presentat a la secció 2 tradueix les adreces lògiques en adreces físiques

Noteu que cada procés (i el sistema operatiu) té un espai lògic independent de la resta de processos. Tot i que aparentment aquests espais lògics estan superposats (per exemple, a tots els espais lògics existeix l'adreça lògica 0x00FFF), el mecanisme de traducció els transforma en regions disjuntes de l'espai físic.

Noteu que el codi emmagatzemat a la memòria física utilitzarà adreces lògiques (dins de l'espai lògic del procés corresponent) per a codificar les direccions de salt i les adreces de rutines.

2.1. Limitacions d'aquest sistema

A causa de la seva simplicitat, aquest sistema de gestió de la memòria presenta un seguit de problemes:

1) Contigüitat: l'esquema presentat requereix que la memòria física en què es carrega un procés sigui contigua. Per exemple, a l'exemple de la figura 2 no serà possible carregar un procés que necessiti 200 KB perquè, tot i que disposem de 448 KB (128 KB + 128 KB + 192 KB) de memòria lliure, no és possible trobar un fragment contigu de 200 KB lliures. Aquest problema es coneix amb el nom de fragmentació externa.

Fragmentació externa

El problema de la fragmentació externa també apareix en situacions quotidianes, com ara l'aparcament de cotxes en filera en un carrer que no senyalitzi les places d'aparcament. Tot i que poden existir diversos forats disponibles per a estacionar-hi un vehicle, per a poder aparcar-lo cal que existeixi un forat prou gran per a encabir-l'hi. La suma de les mides del forats disponibles pot ser molt superior a la mida del vehicle, però no podrem aparcar el vehicle si l'espai lliure necessari no està disponible de forma contigua.

2) Dificultat de creixement dinàmic de l'espai lògic d'un procés: alguns processos poden necessitar augmentar dinàmicament la mida del seu espai lògic per a poder invocar procediments recursius o per a crear estructures de dades en temps d'execució. A causa de la contigüitat d'aquest sistema de gestió de la memòria, per a augmentar l'espai lògic del procés caldria disposar de memòria física lliure contigua a la que té assignada el procés o bé, en carregar el procés, reservar més memòria de la necessària. En general, això no sempre serà factible.

3) No permet la compartició de porcions de codi o dades entre diversos processos: si diversos processos estan executant el mateix programa (per exemple, l'intèrpret de comandes o un editor de text), no és raonable que el codi estigui replicat a la memòria física per a cada procés que l'està executant. Seria més efectiu, si fos possible, que el codi estigués carregat un únic cop a la memòria física; els processos compartirien aquest codi i cadascun treballaria amb les seves regions de dades i de pila privades. També seria factible que diversos processos volguessin compartir un fragment de la seva regió de dades. L'esquema presentat en aquest apartat no permet la compartició de memòria entre processos.

4) No permet la càrrega d'un procés que no càpiga a la memòria física.

Per tant, aquest sistema de gestió de la memòria no és adequat per als sistemes operatius multiprocés de propòsit general. A l'apartat 3 donarem les pinzellades bàsiques d'un sistema de gestió de la memòria amb traducció dinàmica que resol aquests problemes.

3. Memòria virtual

Existeixen diverses implementacions possibles d'un sistema de gestió de la memòria que resolgui els problemes del sistema presentat a l'apartat anterior. En aquest apartat presentarem un sistema de gestió de la memòria basat en paginació, tot i que hi ha altres alternatives. Veurem que aquest sistema permetrà independitzar els processos de la quantitat de memòria física instal·lada a la màquina, amb la qual cosa serà possible executar processos que requereixin més memòria de la disponible. Es diu que aquests sistemes ofereixen memòria virtual.

3.1. Paginació

De forma transparent per al programador, un sistema de gestió de la memòria basat en paginació divideix l'espai lògic del procés en trossos de mida fixa, que anomenem pàgines[3]. L'espai físic també es divideix en trossos de la mida d'una pàgina, anomenats *frames*. Aquest sistema de traducció ubicarà cada pàgina en un *frame* de memòria física.

[3] L'arquitectura IA-32 d'Intel utilitza pàgines d'una mida de 4 KB (2^{12} bytes).

La figura 3 mostra un esquema del maquinari que es necessita per a implementar la paginació.

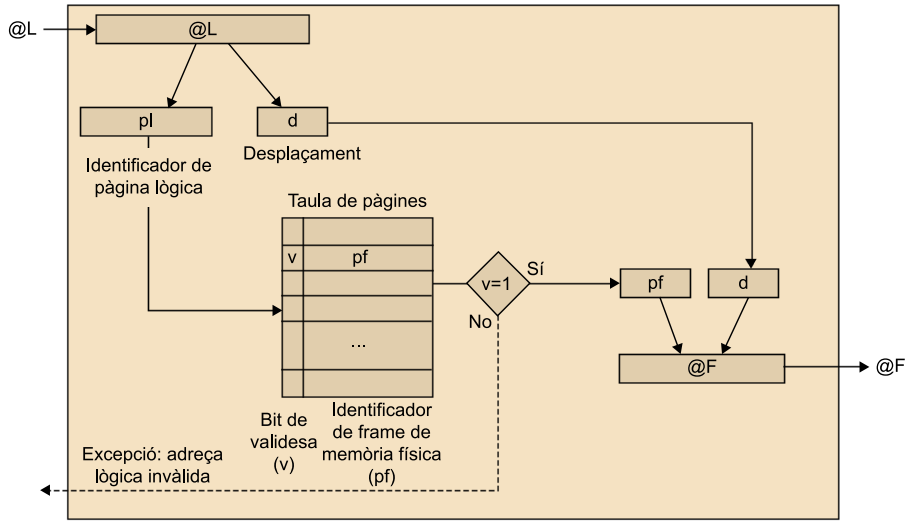

Figura 3. Esquema d'una MMU que implementa paginació pura

El mecanisme de traducció divideix les adreces lògiques en dos components, que són l'identificador de pàgina lògica i el desplaçament dins de la pàgina. Utilitzant una taula, anomenada taula de pàgines, es determina l'identificador de *frame* a l'espai físic on es troba la pàgina lògica. Finalment, es construeix l'adreça física concatenant l'identificador de *frame* i el desplaçament.

Divisió de les adreces lògiques

Com que el processador codifica les adreces de memòria en base 2 i la mida de pàgina és una potència de 2 ($2^{log2(mida\_pagina)}$), la divisió de les adreces lògiques en identificador de pàgina lògica i desplaçament dins de la pàgina es realitza de forma trivial. El desplaçament són els log2(mida_pagina) bits baixos de l'adreça lògica i l'identificador de pàgina lògica són la resta de bits de l'adreça lògica.

La taula de pàgines és pròpia de cada procés i té tantes entrades com pàgines pugui arribar a tenir l'espai lògic d'un procés ($2^{mida\_bus\_adreces}$/mida_pàgina). En cadascuna de les entrades s'emmagatzema un identificador de *frame* i un seguit de bits de control, entre els quals hi ha, com a mínim, el bit de validesa, que indica si aquesta pàgina lògica pertany a l'espai lògic del procés. Altres bits de control possibles serien els bits de protecció (read, readwrite, execute), que indiquen quines operacions és possible realitzar a la pàgina[4].

(4)Com que la granularitat d'aquestes proteccions és la de pàgina, dins de la mateixa pàgina no serà possible barrejar informació corresponent a regions diferents.

En cas d'intentar accedir a una pàgina lògica marcada com a invàlida a la taula de pàgines, es provocarà una excepció que notificarà aquest fet al sistema operatiu. Normalment, aquesta excepció provocarà que el sistema operatiu faci finalitzar immediatament aquest procés perquè es considera que el procés presenta un error de programació.

Aquest sistema de gestió de la memòria presenta alguns problemes pràctics que compliquen la seva implementació real:

- La mida de la taula de pàgines pot ser molt gran. Per exemple, si la mida del bus d'adreces és de 32 bits i la mida de la pàgina és de 4 KB, la taula de pàgines té 2^{20} entrades (1.048.576 entrades[5]). Si cada entrada ocupa 4 bytes, això representa 4 MB. Aquesta mida resulta impracticable i calen organitzacions que permetin reduir la mida de la taula de pàgines, com per exemple estructurar la taula de pàgines en dos nivells[6].

(5)Tots els processos del sistema tindran una taula de pàgines d'aquesta mida, independentment de la quantitat de pàgines que ocupi el procés. El bit de validesa indicarà quines entrades són les que realment fa servir el procés.

(6)És la solució adoptada pels processadors que implementen l'arquitectura IA-32 d'Intel. No és l'objectiu d'aquest document presentar aquesta solució.

- A causa de la seva mida, la taula de pàgines no es pot emmagatzemar en registres i cal emmagatzemar-la a la memòria[7]. Per tant, cada referència a la memòria lògica implicaria dos accessos a la memòria física: un a la taula de pàgines i un altre a l'adreça física calculada. Aquest sobrecost és inacceptable, per la qual cosa el maquinari proporciona la TLB (Translation Lookaside Buffer), una memòria cau especialitzada a emmagatzemar les entrades de la taula de pàgines més utilitzades recentment.

(7)La MMU tindrà un registre de control que indicarà a partir de quina posició de memòria física es troba la taula de pàgines del procés en execució.

Un sistema operatiu que utilitzi la paginació necessitarà un seguit d'estructures de dades per a dur a terme la gestió de la memòria. A més de les taules de pàgines de cada procés (emmagatzemades a l'espai de dades del sistema operatiu), caldrà alguna estructura que indiqui si els *frames* de la memòria física estan lliures o ocupats (taula de *frames*) i algun procediment per a obtenir *frames* lliures.

3.1.1. Ubicació de les regions de codi, dades i pila dins de l'espai lògic

En el primer esquema presentat a l'apartat 2 ubicàvem les regions de codi, dades i pila contiguament dins de l'espai lògic.

Emprant la paginació, les regions de codi i de dades s'ubiquen típicament en un extrem de l'espai lògic, mentre que la de pila s'ubica a l'altre; totes les pàgines intermèdies estaran marcades com a invàlides. Aquesta distribució facilitarà el fet que les regions de dades i de pila del procés puguin créixer dinàmicament.

Emprant la paginació, aquest "forat" a l'espai lògic no implica un desaprofitament de memòria física perquè totes les pàgines lògiques corresponents al forat estan marcades com a invàlides, amb la qual cosa no tindran associat cap *frame* de memòria física. En canvi, a l'esquema presentat a l'apartat 2, aquesta disposició sí que provocaria un desaprofitament de memòria física.

La figura 4 presenta un exemple en el qual assumim que l'espai lògic pot arribar a tenir 8 pàgines i l'espai físic té 16 *frames*. Carreguem dos executables, el primer amb dues pàgines de codi, una de dades i una de pila i el segon amb tres de codi, una de dades i una de pila. Ara bé, el segon executable està carregat dos cops (dos usuaris diferents estan executant aquest programa), i així demostrarem que no cal duplicar el codi a la memòria física.

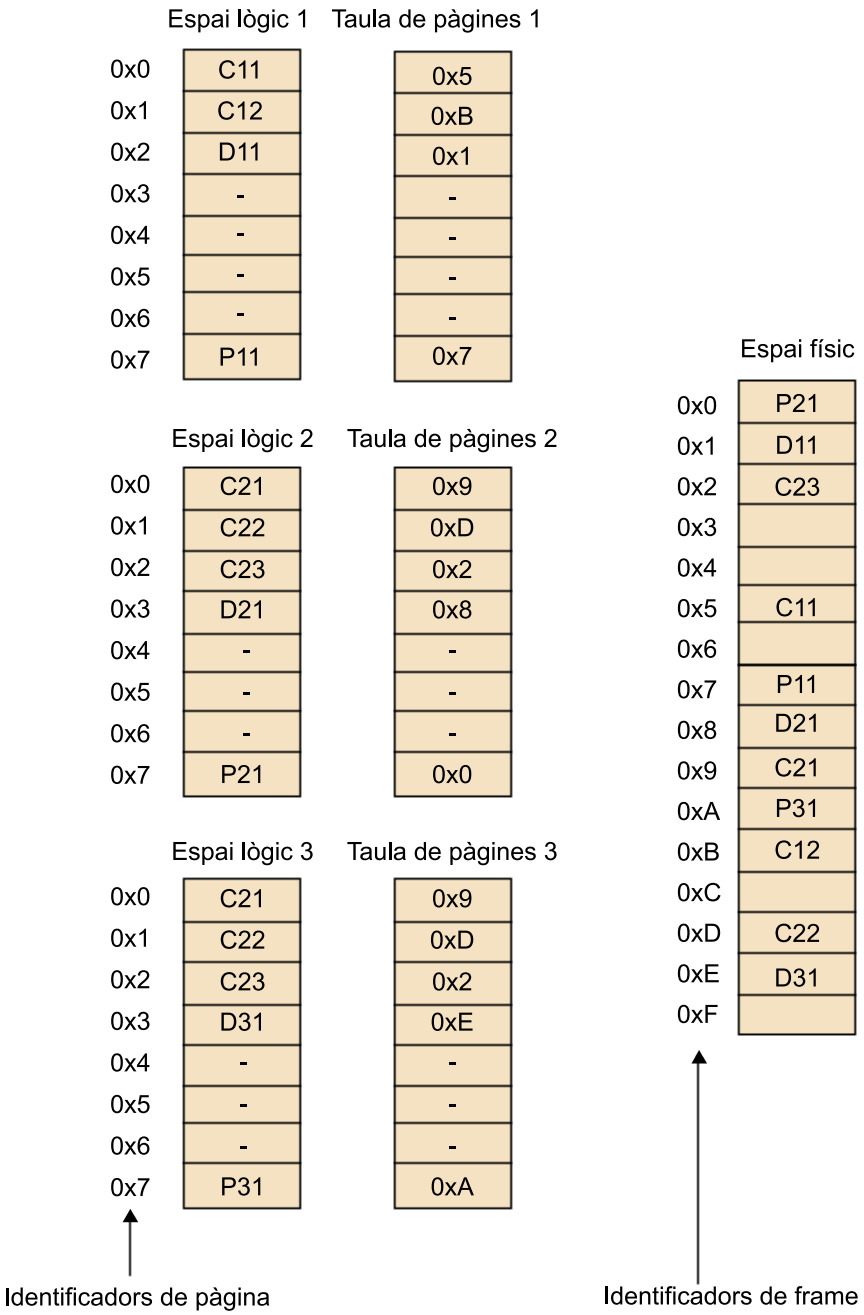

Figura 4. Exemple de càrrega de tres processos en un sistema de gestió de la memòria paginat. Les pàgines invàlides s'han marcat amb el símbol -.

La figura mostra els espais lògics corresponents als tres processos, així com una possible càrrega dels processos a l'espai físic i les taules de pàgines corresponents. Noteu que la paginació permet que els dos processos comparteixin les pàgines de codi.

3.1.2. Evita la paginació les deficiències del sistema presentat a l'apartat 2?

Un mecanisme de gestió de la memòria basat en la paginació permet solucionar la majoria de problemes descrits per al sistema anterior:

1) No cal que pàgines contigües a l'espai lògic hagin de ser-ho també a l'espai físic. Per tant, la paginació no presenta el problema de la fragmentació externa; quan es carrega un executable a la memòria, és possible utilitzar qualsevol *frame* lliure de memòria física, independentment de si és contigu o no amb la

resta de *frames*. En canvi, la paginació presenta un altre problema, ja que la unitat mínima d'assignació és la pàgina, amb la qual cosa les peticions de memòria de mida no múltiple de la mida de pàgina provocaran un cert desaprofitament de memòria. Aquest problema es coneix com a fragmentació interna i és inherent a tots els sistemes d'emmagatzemament (ja sigui en memòria, en disc...) que divideixen l'espai en blocs d'una mida fixa.

Fragmentació interna

El problema de la fragmentació interna també apareix en situacions quotidianes, com ara les reserves de taules als restaurants. Típicament, en una taula no es barregen comensals corresponents a reserves diferents (és a dir, la taula és la unitat mínima d'assignació d'espai). Per tant, tot i que el restaurant pot tenir cadires lliures, aquestes no es poden assignar a noves reserves si aquestes cadires lliures pertanyen a taules ja assignades a altres reserves.

2) La paginació permet que l'espai lògic d'un procés pugui créixer i decréixer. Gràcies al "forat" produït en ubicar les dades i la pila a cada extrem de l'espai lògic, disposem de pàgines lògiques per a poder variar la mida tant de la regió de dades com de la pila sense desaprofitar memòria física.

3) La paginació permet que diversos processos comparteixin *frames* de l'espai físic. Només cal que les entrades corresponents de les taules de pàgines dels processos facin referència a aquests *frames* compartits.

En canvi, la paginació no permet carregar processos que no càpiguen a l'espai físic. Al subapartat següent veurem com podem estendre la paginació per a poder carregar aquests processos.

3.2. Paginació sota demanda

Fins ara, tots els esquemes que hem presentat partien de la base que quan un procés s'executa és necessari que estigui carregat totalment a la memòria principal, però això no ha de ser necessàriament així. La memòria virtual és un esquema de gestió de la memòria amb el qual un procés pot estar només parcialment carregat (resident) a la memòria principal. Amb aquest esquema, la suma dels espais lògics dels processos que s'estan executant pot ser més gran que la memòria física disponible.

L'esquema també permet executar processos que tinguin un espai lògic més gran que l'espai físic, cosa que amb els esquemes anteriors era impossible. Aquest fet s'aconsegueix mantenint una imatge de l'espai lògic del procés a la memòria secundària (disc) en una zona anomenada àrea d'intercanvi (*swap area*). En cada instant, a la memòria principal només hi ha carregada la part del procés que és necessària per a la seva execució, o bé la part del procés que el sistema operatiu permet carregar a la memòria. La decisió de quina part del procés es carrega a la memòria i quan es carrega la pren el sistema operatiu, i ho fa en funció de la càrrega i la disponibilitat del sistema.

La memòria virtual es pot implementar estenent la idea de paginació (figura 5). Aquesta extensió rep el nom de **paginació sota demanda**. Cal afegir un bit (bit de presència) a cada entrada de la taula de pàgines per a indicar si la pàgina en qüestió està carregada o no a la memòria física. En el cas que no ho estigui, es produirà una excepció del tipus "fallada de pàgina" (*page fault*) i la rutina d'atenció haurà d'anar a buscar la pàgina a l'àrea d'intercanvi, carregar-la a la memòria principal, fer que es repeteixi l'accés a la memòria que ha provocat la fallada i fer continuar l'execució del procés.

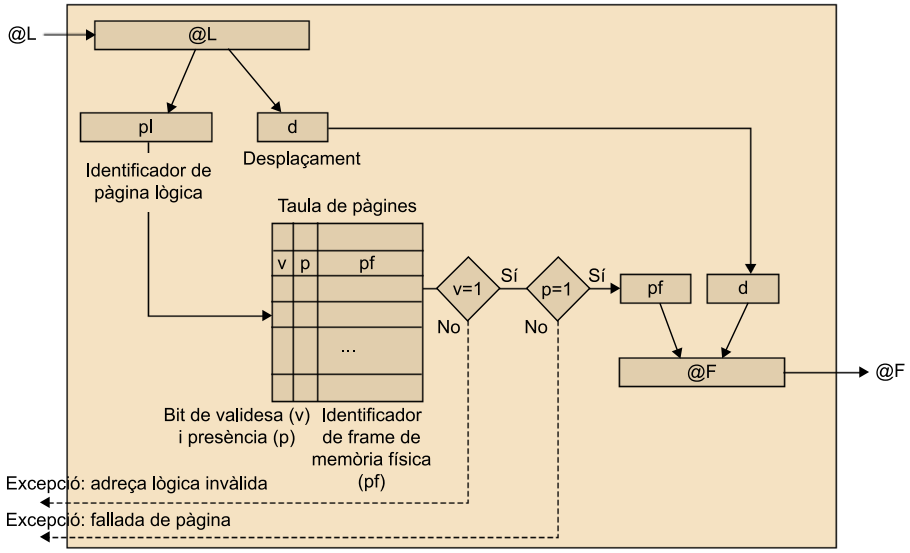

Figura 5. Extensió de la MMU de la figura 3 per tal de permetre l'ús de memòria virtual

A més a més, és necessari que el sistema operatiu disposi d'algun tipus d'estructura de dades que indiqui com esta distribuït el procés a l'àrea d'intercanvi.

La gestió més habitual és que en un principi no hi hagi cap pàgina carregada a la memòria física i que les pàgines del procés es carreguin a mesura que es van necessitant. Quan s'executa un procés, cada referència a la memòria lògica suposa l'execució dels passos següents:

- Primerament es comprova si la pàgina corresponent a l'adreça és vàlida amb el bit de validesa (v).

- A continuació es determina si la pàgina està carregada a la memòria comprovant el bit de presència (p). En cas que no es trobi a la memòria, es genera una fallada de pàgina i el sistema operatiu ha de fer les gestions necessàries per tal d'anar a buscar la pàgina a l'àrea d'intercanvi i carregar-la a la memòria principal en un *frame* lliure. La cosa es complica encara més si no queda cap *frame* lliure, ja que aleshores el sistema de gestió de la memòria ha de treure alguna pàgina de la memòria principal i portar-la al disc, sempre que hi hagi modificacions. A més a més, ha d'actualitzar la taula de pàgines del procés al qual pertany la pàgina que s'ha emportat a

l'àrea d'intercanvi. Un cop el *frame* ja ha quedat lliure, es pot assignar al procés que s'estava executant, tot actualitzant la seva taula de pàgines.

- Finalment, si la pàgina lògica és vàlida i present, es genera l'adreça física.

Els sistemes de memòria virtual han de gestionar les pàgines de la manera més eficient possible a fi que el temps d'execució dels processos no sigui excessivament alt a causa d'un nombre molt gran de fallades de pàgina. És important establir bones polítiques[8], com ara les següents:

[8] En aquesta assignatura no entrarem en detall a considerar aquestes polítiques.

- **Polítiques d'assignació de pàgines**: determinen quantes pàgines pot tenir carregades un procés a la memòria física en un moment donat, és a dir, determina el que s'anomena *working set*.

- **Polítiques de substitució de pàgines**: aquestes polítiques determinen quin *frame* convé portar al disc quan s'ha de portar una pàgina a la memòria física i no queden *frames* lliures. Cal escollir entre els *frames* del mateix procés que s'està executant?, o bé cal considerar els *frames* d'altres processos?, pàgines que només s'hagin llegit? (en aquest cas s'hauria de considerar que en cada entrada de la taula de pàgines o de la taula de *frames* hi ha un bit que indica que la pàgina s'ha modificat), etc.

- **Polítiques de càrrega de pàgines**: determinen quan es carreguen les pàgines a la memòria. Hi ha la possibilitat de carregar pàgines només quan hi ha una fallada de pàgina (paginació sota demanda), o bé es pot intentar avançar la càrrega de pàgines aprofitant la localitat espacial dels programes (*prefetching*).

Consideracions finals:

- Aquest mecanisme és totalment transparent per a l'usuari.

- S'ha comprovat que les referències a la memòria fetes per un programa no es distribueixen uniformement per totes les pàgines de l'espai lògic. A causa de la localitat espacial i temporal de les referències a la memòria, la majoria de referències es concentren en un conjunt reduït de pàgines. Per tant, la paginació sota demanda no incrementa necessàriament el temps d'execució del programa.

- La memòria virtual permet incrementar el grau de multiprogramació perquè, com que cada procés necessita menys memòria física, serà possible carregar més processos. De totes formes, augmentar excessivament aquest grau pot afectar el rendiment del sistema operatiu (*thrashing*).

- Al llarg de l'execució d'un procés, una mateixa pàgina es pot intercanviar del disc a la memòria i de la memòria al disc diverses vegades. Noteu que

cada cop que torna a la memòria física pot utilitzar un *frame* diferent de memòria física (el sistema de gestió permet mobilitat).

4. Modificació dinàmica de l'espai lògic

En temps d'execució, els processos poden haver de modificar la mida del seu espai lògic. Ara bé, la forma de fer-ho depèn de la regió (codi/dades/pila) de la qual es vulgui variar la mida.

- **Regió de codi**: Existeixen diverses possibilitats per a modificar dinàmicament la regió de codi de l'espai lògic d'un procés, com ara *overlays*, codi automodificable o biblioteques carregades en temps d'execució[9]. Llevat de les biblioteques carregades en temps d'execució, aquestes opcions estan en desús o s'utilitzen únicament en escenaris molt concrets. En aquesta assignatura no considerarem cap d'aquestes opcions.

(9) En els sistemes Windows també reben el nom de DLL (*dynamic-link libraries*).

- **Regió de pila d'execució**: La pila és transparent per al programador d'aplicacions d'alt nivell. En crear cada procés, el sistema operatiu li assigna una mida inicial de pila. Si en algun moment aquesta mida resulta insuficient (el procés executa una instrucció `push` que intenta escriure més enllà de la mida assignada a la pila), la MMU generarà una excepció. La rutina d'atenció a l'excepció comprovarà si el problema és degut a la mida de la pila i, si és possible[10], n'augmentarà la mida i permetrà que el procés repeteixi l'execució de la instrucció `push` i continuï la seva execució. Per tant, el creixement de la regió de pila és completament transparent per al programador i tampoc ens n'ocuparem.

(10) El sistema operatiu pot limitar la mida màxima que pot arribar a assolir la pila d'execució dels processos.

- **Regió de dades**: La mida de la regió de dades és responsabilitat del programador. Aquest disposarà de crides al sistema per tal de poder modificar dinàmicament aquesta regió de l'espai lògic. Això pot resultar útil perquè la mida d'algunes estructures de dades pot dependre d'un valor desconegut en temps de compilació i que no es coneixerà fins al temps d'execució del programa. En aquesta assignatura ens centrarem en com modificar dinàmicament la mida d'aquesta regió.

4.1. Modificació dinàmica de la mida de la regió de dades

El sistema operatiu acostuma a oferir crides al sistema per a augmentar i disminuir la regió de dades de l'espai lògic del procés.

En el cas de l'Unix, totes dues tasques es poden fer amb la crida al sistema `brk`. Malauradament, aquesta crida té una interfície poc intuïtiva i presenta alguns problemes de portabilitat. Per tant, no és habitual que els programes utilitzin directament aquesta crida al sistema.

La biblioteca del llenguatge C ofereix diverses rutines per a modificar la mida de la regió de dades: `calloc`, `malloc`, `free` i `realloc`. Les rutines `calloc`, `malloc` i `realloc` permeten ampliar la regió de dades, però difereixen en la interfície. La rutina `free` permet alliberar la memòria obtinguda prèviament amb `calloc/malloc/realloc`.

La interfície de la rutina `malloc` és `void *malloc(size_t size)`. I, per exemple, rep com a paràmetre la quantitat de bytes en què volem ampliar la regió de dades. Si no és possible realitzar aquesta operació, la rutina retorna el punter `NULL`; altrament, retorna l'adreça lògica de la memòria corresponent al primer byte que s'ha afegit a la regió de dades i la resta de bytes es troben a posicions de memòria consecutives. La interfície de la rutina `free` és `void free(void *ptr)` i rep com a paràmetre una adreça de memòria obtinguda prèviament amb `calloc/malloc/realloc`.

A continuació mostrem un parell de fragments de codi en què es creen dinàmicament un vector (figura 6) i una matriu (figura 7).

```
int *vector;
vector = malloc(N*sizeof(int)); /* Demana espai per a N enters */
if (vector == NULL) error("Memòria no disponible");
for (i=0; i<N; i++)
      vector[i] = ....; /* Omple el vector */
...
free(vector);      /* Allibera l'espai */
```

Figura 6. Fragment de codi que crea dinàmicament un vector d'enters d'N posicions

```
int **matrix;

/* Demana espai per a un vector d'N punters, un per a cada fila */
matrix = malloc(N*sizeof(int *));
if (matrix == NULL) error("Memòria no disponible");
for (i=0; i<N; i++) {
   /* Demana espai per a la fila i-èssima (M enters) */
   matrix[i] = malloc(M*sizeof(int));
   if (matrix[i] == NULL) error("Memòria no disponible");
}
for (i=0; i<N; i++)
  for (j=0; j<M; j++)
    matrix[i][j] = ...;  /* Omple la matriu */
...
for (i=0; i<N; i++)
   free(matrix[i]);     /* Allibera la fila i-èssima */
free(matrix);         /* Allibera el vector de punters
```

Figura 7. Fragment de codi que crea dinàmicament una matriu d'enters d'N files i M columnes.

5. Errors de programació relacionats amb l'accés a la memòria

Finalment farem algunes consideracions sobre els errors de programació relacionats amb l'accés a la memòria, és a dir, els errors de programació que provoquen que un procés accedeixi a una adreça de memòria a la qual no hauria d'accedir. Són especialment típics de programes escrits en llenguatge C o C++. La figura 8 mostra alguns exemples d'aquests errors.

```
int vector[10]; unsigned int n; char s[10]; int *k; char *p;

/* Error de programació si n >= 10 */
vector[n] = valor;

/* Error de programació perquè la memòria intermèdia s és massa petita (buffer overflow) */
sprintf(s, "El valor de vector[3] és %d\n", vector[3]);

/* Error de programació perquè no té sentit referenciar variables locals d'un procediment
un cop aquest ha acabat */
int *rut()
{
 int n;
 n = valor;
 return(&n);
}
...
k = rut();
n = *k;

/* Error de programació perquè s'accedeix a memòria alliberada */
p = malloc(100);
...
free(p);
p[4] = valor;
```

Figura 8. Exemple d'alguns errors de programació relacionats amb l'accés a la memòria

En general, el compilador de llenguatge C no pot detectar aquests tipus d'errors. A més a més, poden ser molt difícils de localitzar pel programador perquè:

- És possible que, per a la majoria de dades d'entrada, aquests errors no afectin el comportament "correcte" del programa.

- Poden manifestar-se moltes sentències més enllà de la que ha realitzat l'accés erroni.
- Poden manifestar-se (o no) en funció de les versions del sistema operatiu/compilador/biblioteques/etc., dels paràmetres de compilació, de les dades d'entrada del procés, etc.

Als apartats anteriors hem vist que la MMU detecta accessos a la memòria invàlids, però hem de tenir en compte que:

- La MMU no podrà detectar un error de programació que provoqui l'accés a una altra adreça vàlida del procés. Per tant, el procés pot llegir o modificar dades que no tenen res a veure amb l'operació que s'està realitzant. Per exemple, un accés a la memòria incorrecte pot estar modificant dades de control de la biblioteca d'entrada/sortida[11], cosa que pot provocar un error d'execució el següent cop que utilitzem una rutina d'aquesta biblioteca.

(11) Aquesta biblioteca guarda informació a l'espai lògic de cada usuari que la fa servir.

- En un sistema basat en la paginació, si la mida de les regions de codi/dades/pila no és múltiple de la mida de pàgina, podem tenir pàgines marcades com a vàlides en les quals no totes les adreces d'aquestes pàgines són "correctes" des del punt de vista del programador. La MMU no podrà detectar un accés incorrecte dins d'aquesta pàgina.

Els accessos incorrectes poden produir-se en qualsevol mode d'execució (privilegiat i no privilegiat), però el tractament és diferent.

- **Mode d'execució no privilegiat**: Si la MMU determina que un accés és invàlid, la MMU provocarà una excepció. La rutina del sistema operatiu que atén aquesta excepció fa avortar el procés. En el cas de l'Unix, la rutina d'atenció acostuma a crear un fitxer[12] anomenat `core` en què s'emmagatzema l'estat complet del procés (contingut dels registres i de la memòria, etc.) en el moment que es provoca l'excepció. Aquest fitxer es podrà examinar posteriorment amb el depurador (*debugger*). Quan un procés creat des de l'intèrpret de comandes ha estat avortat a causa d'un accés a memòria invàlid, els intèrprets de comandes de l'Unix acostumen a mostrar el missatge *Segmentation fault. Core dumped* (Error de segmentació).

(12) L'usuari pot indicar si vol que es generin aquests fitxers i quina mida màxima poden tenir.

- **Mode d'execució privilegiat**: Sempre que alguna crida al sistema ha d'accedir a algun rang d'adreces lògiques del procés[13], abans el sistema operatiu comprova per programari que aquest rang d'adreces sigui vàlid per al procés. El codi del sistema operatiu accedirà a aquestes adreces únicament si la comprovació indica que tot el rang d'adreces és vàlid; ara bé, tot i que el rang d'adreces sigui vàlid, això no vol dir que l'accés a la memòria sigui correcte des del punt de vista del programador. En cas que alguna adreça del rang sigui invàlida, la crida al sistema no es realitzarà i retornarà el codi d'error adequat. Si, malgrat tot, la MMU detecta un accés a la memòria invàlid en mode d'execució privilegiat, per precaució el sistema operatiu

(13) Per exemple, si un procés invoca la crida al sistema `write` de l'Unix, la rutina d'atenció haurà de llegir dades d'un fitxer i escriure'ls a l'espai lògic de l'usuari, en el rang d'adreces especificat per una adreça lògica inicial i una mida.

acostuma a deixar de prestar servei i apareix un *Kernel panic* (Linux) o la *Blue screen of death* (Windows).

Per a ajudar als programadors a detectar o corregir aquests tipus d'errors, existeixen eines com ara:

- Depuradors (*debuggers*), que permeten analitzar els fitxers `core`.
- Versions sofisticades de les biblioteques d'assignació de memòria dinàmica.

És important que els programes no presentin aquests tipus d'errors. Cal pensar que una quantitat significativa d'atacs a sistemes informàtics aprofiten un tipus particular d'aquests errors (els *buffer overflows*).

Resum

En aquest mòdul didàctic hem vist la manera en què el sistema operatiu, amb l'ajut del maquinari, pot oferir una visió de la memòria "idealitzada" als processos en execució. Mentre que la memòria física és limitada i compartida per tots els processos, la visió que tindrà cada procés és que disposa d'un espai d'adreces no compartit amb cap altre procés, independentment de l'ocupació actual de la memòria física, i que pot tenir una mida més gran que la memòria física instal·lada. El sistema operatiu realitza aquesta tasca de forma transparent per als processos utilitzant mecanismes de traducció dinàmics (en temps d'execució) implementats dins d'un maquinari especialitzat anomenat MMU (Memory Management Unit).

Activitats

1. Expliqueu la diferència que hi ha entre fragmentació interna i fragmentació externa.

2. Es vol reduir la fragmentació interna en un sistema de gestió de la memòria basat en la paginació reduint la mida de les pàgines (per exemple, des de 4 KB a 1 KB). Quin impacte té aquesta reducció en la mida de la taula de pàgines dels processos?

3. A la figura 3 no s'especifica el nombre de bits que circula per cada línia del circuit de traducció. Determineu-lo assumint que la mida de pàgina és de 4 KB, que el bus d'adreces és de 32 bits i que la quantitat màxima de memòria física instal·lada és de 2 GB.

4. L'operador & del llenguatge C s'aplica a una variable i ens retorna l'adreça de memòria on s'emmagatzema. Aquesta adreça és lògica o física?

5. Busqueu informació sobre la manera en què el Linux estructura l'espai lògic en els processadors IA-32 d'Intel. On ubica el Linux les regions de codi, dades i pila d'un procés? Per què creieu que, a partir de l'adreça lògica 0xC0000000, tots els espais lògics tenen el mateix contingut (codi, dades i pila del sistema operatiu)? Per què creieu que el codi del programa no es comença a carregar a partir de l'adreça lògica 0x00000000?

6. Busqueu informació sobre l'estructura de dos nivells de la taula de pàgines aplicada pels processadors que implementen l'arquitectura IA-32 d'Intel.

7. Busqueu informació sobre quins són els bits de control presents a les entrades de les taules de pàgina dels processadors que implementen l'arquitectura IA-32 d'Intel.

8. En un sistema paginat amb memòria virtual volem controlar mitjançant un bit (*dirty bit*) si una pàgina ha estat modificada. En quina taula situaríeu aquest bit, a la taula de pàgines del procés, a la taula de *frames* o a la taula de pàgines del disc?

9. En un sistema paginat, podem afirmar que la MMU detectarà tots els accessos a la memòria en una posició de memòria "incorrecta" (per exemple, un accés més enllà de la mida d'un vector)?

10. Busqueu informació sobre els *stack overflows*. En què consisteixen? Per què són perillosos? Busqueu informació sobre l'*Execute-Disable (NX) bit* que alguns processadors Intel incorporen a la taula de pàgines.

Exercicis d'autoavaluació

1. Disposem d'una màquina amb un espai lògic de processador de 1.024 Kbytes i una memòria física instal·lada de 128 Kbytes. El sistema de gestió de la memòria del sistema operatiu instal·lat en aquesta màquina es basa en la tècnica de la paginació sota demanda, en què les pàgines tenen una mida de 2 Kbytes. Inicialment el procés no té cap pàgina carregada a la memòria. Suposem que en una pàgina només hi haurà o bé codi, o bé dades del procés, o bé la pila d'execució del procés.

Nota

Per *Espai lògic de processador* entenem la mida màxima que pot arribar a tenir l'espai lògic d'un procés.

En aquesta màquina volem executar un programa. La capçalera de l'executable d'aquest programa ens indica que el codi del programa ocupa 10 Kbytes, que tenim 6 Kbytes de dades inicialitzades, que les dades no inicialitzades ocuparan 15 Kbytes i que la pila té 12 Kbytes. La capçalera de l'executable ocupa 128 bytes. Amb aquesta informació, contesteu les preguntes següents:

a) Podeu dir aproximadament la mida que tindrà el fitxer executable del programa que hem descrit? Si la mida de les pàgines de la memòria fos d'1 Kbyte, quina seria la mida del fitxer executable?

b) Un cop carregat aquest programa i creat l'espai lògic del procés corresponent, i suposant que totes les pàgines que ocupa el nostre procés fossin presents a la memòria física, quina seria la quantitat de memòria física que ocuparia el nostre procés? Responeu la mateixa pregunta considerant que les pàgines fossin d'1 Kbyte.

c) Quina és la fragmentació interna que produiria aquest procés en el cas que les pàgines fossin de 2 Kbytes? I si fossin d'1 Kbyte?

d) Quin efecte tindria un canvi en la quantitat de memòria física instal·lada que té aquesta màquina sobre els espais d'adreces següents: fitxer executable, espai lògic del processador i espai lògic del procés?

2. Suposem que una màquina té un sistema de gestió de la memòria en què la traducció d'adreces s'efectua dinàmicament (en execució) i que ofereix memòria virtual. Contesteu les preguntes següents tot justificant breument la vostra resposta.

a) Pot ser més gran la mida de l'espai lògic d'un procés que la quantitat total de la memòria física instal·lada a la màquina?

b) Quin efecte tindria un augment de la quantitat de memòria física instal·lada sobre la mida de l'espai lògic del processador?

c) És possible que la mateixa adreça lògica de dos processos diferents sigui traduïda a la mateixa adreça física en un mateix moment?

d) És possible que una adreça lògica d'un programa en execució sigui traduïda a adreces físiques diferents en diferents instants de la mateixa execució?

e) És necessari que la pila d'execució d'un procés s'emmagatzemi en adreces lògiques consecutives? I en adreces físiques consecutives?

f) Compilem un mateix programa de dues maneres diferents; en la primera indiquem que la mida inicial de la pila serà de 10 K i en la segona indiquem que serà de 20 K. Quina creieu que serà la diferència de mida entre els dos fitxers executables generats?

g) Permet ampliar el grau de multiprogramació el fet de disposar de la memòria virtual?

3. Contesteu les preguntes següents:

a) Pot l'espai lògic del processador ser més gran que la suma de les mides de la memòria física instal·lada i del dispositiu d'emmagatzematge (àrea d'intercanvi) de pàgines tretes de la memòria física?

b) Pot l'espai lògic del procés ser més gran que la suma de les mides de la memòria física instal·lada i de la mida del dispositiu d'emmagatzematge (disc) de pàgines tretes de la memòria física?

c) Té qualsevol procés d'usuari parts del seu espai lògic de procés que han de trobar-se permanentment a la memòria física durant l'execució del procés?

d) Si durant l'execució concurrent de dos processos tots dos intenten accedir a la mateixa adreça lògica, vol dir això que estan compartint la memòria?

Solucionari

Exercicis d'autoavaluació

1.a) Un executable està format pel codi i les dades inicialitzades. La pila i les dades no inicialitzades es creen en temps d'execució. Així doncs, la mida de l'executable és igual a la mida de la capçalera + la mida del codi + la mida de les dades inicialitzades (128 bytes + 10 Kbytes + 6 Kbytes). En el cas de pàgines de memòria d'1 Kbyte, la mida de l'executable seria la mateixa, perquè és independent de la mida de la pàgina.

b) Amb pàgines de 2 Kbytes, la mida de la memòria física ocupada seria igual a 22 pàgines (44 Kbytes), i amb pàgines d'1 Kbyte la mida seria igual a 43 pàgines (43 Kbytes).

c) Amb pàgines de 2 Kbytes tindríem una fragmentació interna d'1 Kbyte i amb pàgines d'1 Kbyte no hi hauria fragmentació interna.

d) No els afectaria de cap manera.

2.a) Sí, ja que el sistema de gestió de la memòria permet executar processos sense que tot l'espai lògic sigui a la memòria física simultàniament.

b) Cap, ja que l'espai lògic del processador està determinat pels bits d'adreça de memòria de què disposa el processador.

c) Sí. En el cas que tots dos processos estiguin compartint codi o dades.

d) Sí, ja que el sistema de gestió permet que una pàgina es pugui moure dintre de la memòria física.

e) Cal que es trobi en adreces lògiques consecutives, però no cal que estigui en adreces físiques consecutives.

f) Els executables tindran la mateixa mida, ja que a l'executable es guarda informació de la mida inicial de la pila, però no es reserva espai.

g) Sí, ja que en un sistema no sencer no cal tenir tot l'espai lògic dels processos simultàniament a la memòria física, i per tant permet que n'entrin més.

3.a) Pot ser més gran perquè l'espai lògic del processador només depèn del nombre de bits que té el bus d'adreces del processador.

b) L'espai lògic d'un procés no pot ser més gran que la suma de les mides d'aquestes dues zones.

c) Un sistema de gestió de la memòria que sigui no sencer permet que tot l'espai lògic d'un procés (codi, dades i pila) sigui al disc durant l'execució. A mesura que es van referenciant pàgines, la rutina d'atenció a la fallada de pàgina ja les anirà portant del disc a la memòria física. Les estructures de dades del sistema operatiu (taules de pàgines i altres informacions relatives al procés) no formen part de l'espai lògic de procés.

d) No necessàriament, perquè processos diferents tenen programacions de gestors de la memòria diferents; per tant, tots dos poden generar la mateixa adreça lògica, però la traducció dinàmica pot fer que aquesta adreça lògica vagi a parar a adreces físiques diferents.

Glossari

bit de presència *m* Bit utilitzat en els sistemes de gestió de la memòria virtual per a indicar si el fragment de l'espai lògic del procés al qual es fa referència està carregat a la memòria.

bit de validesa *m* Bit utilitzat en els sistemes de gestió de la memòria per a indicar si el fragment de l'espai lògic al qual es fa referència pertany a l'espai lògic del procés.

fallada de pàgina *f* Excepció generada per la MMU en un sistema de gestió de la memòria que ofereix memòria virtual quan el processador genera una adreça lògica que es tradueix a una pàgina vàlida però que no està carregada (no és present) a la memòria física.

fragmentació externa *f* Espais de memòria no utilitzables que es creen en sistemes de gestió de la memòria en què els processos es carreguen en posicions de la memòria física contigües. Té lloc quan no hi ha un espai de memòria contigu lliure prou gran per a carregar-hi un procés, malgrat que hi hagi trossos lliures més petits la suma de les capacitats dels quals seria suficient per a carregar-lo.

fragmentació interna *f* Espais de memòria no utilitzables que es creen quan la memòria es divideix en parts de mida fixa (per exemple, *frames* de memòria principal o blocs de disc) i una d'aquestes parts s'assigna a un conjunt de dades més petit que la capacitat de la part en qüestió.

intercanvi (*swapping*) *m* Acció de portar pàgines d'un procés des de la memòria secundària (disc) a la memòria principal, o *swap in*. El procés invers, és a dir, el fet de portar pàgines d'un procés des de la memòria principal a la secundària, s'anomena *swap out*.

memòria virtual *f* Sistema de gestió de la memòria que permet executar un procés tot i tenir només part del procés carregada a la memòria. Mitjançant tècniques com la paginació sota demanda, el sistema anirà carregant altres parts del procés a mesura que es necessitin.

paginació *f* Sistema de gestió de la memòria que divideix l'espai lògic i l'espai físic en blocs de mida fixa i assigna a cada bloc de l'espai lògic (pàgina) un bloc de l'espai físic (*frame*).

***thrashing* (hiperpaginació)** *m* Situació que ocorre quan la memòria disponible en un moment donat per a l'execució d'un procés és inferior a la mínima necessària (*working set*) per a la seva execució, la qual cosa provoca una elevada activitat d'intercanvi. Un sistema es troba en *thrashing* si es passa més temps intercanviant pàgines que executant codi de les aplicacions. Un causa habitual és que el grau de multiprogramació és excessivament elevat per a la configuració del maquinari.

Bibliografia

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

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