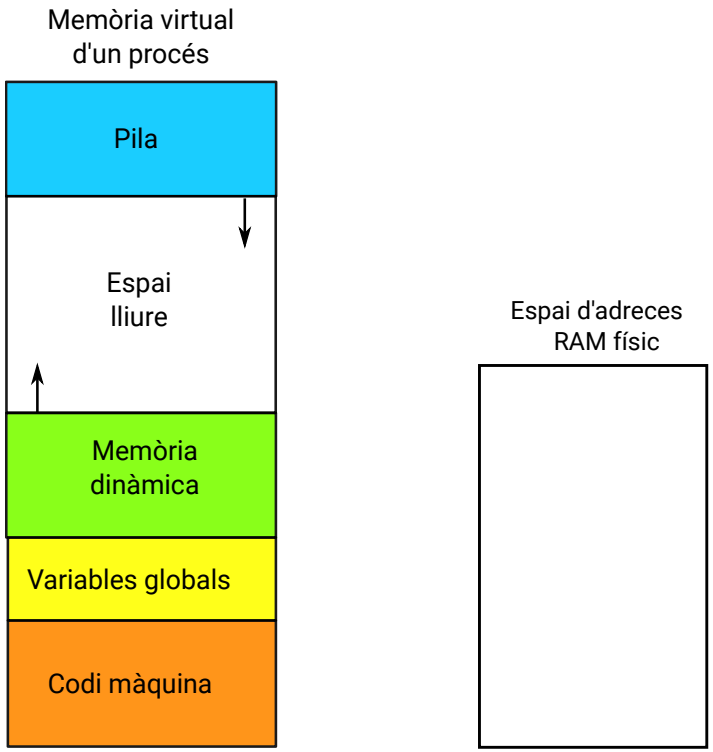
Memòria virtual i traducció d'adreces

Sistemes Operatius I

Grau d'Enginyeria Informàtica

Introducció

Cada procés "creu" que té assignat tot l'espai de memòria possible (64 bits) i que a més és "lineal". Però no és així...



Executeu el codi `exemple_malloc.c`. Observar que al malloc podem demanar més memòria que RAM física hi ha disponible.

Introducció

Estem acostumats a pensar que una direcció de memòria és només això: una direcció de memòria física.

- Per exemple, creiem que un punter apunta a la direcció de memòria RAM en què està emmagatzemada la variable o que el registre de comptador de programa apunta a la direcció de memòria RAM que s'està executant.
- Tot això és només una ficció! Executeu el codi `exemple.c` i `exemple_fork.c`: comproveu si els valors que surten per pantalla tenen sentit físic. No en tenen!

Introducció

Aquest "entorn de memòria virtual" s'aconsegueix gràcies a la col·laboració estreta entre maquinari i el sistema operatiu:

- El maquinari i el sistema operatiu s'encarreguen de gestionar un sistema que "tradueix" qualsevol direcció que genera un procés a una direcció física de memòria RAM.
- El sistema de traducció d'adreces és un concepte ben senzill però molt potent. Anem a veure'l! Com a programadors, però, és millor no pensar en tot aquest esquema a l'hora de programar...

Introducció

A l'actualitat, gràcies al sistema de traducció d'adreces es poden implementar un munt de funcionalitats molt importants. Algunes d'elles són:

- Aïllament de processos o protecció dels processos entre sí. A més, permet a aplicacions construir "sandboxes" per executar aplicacions externes.
- Auto-aïllament d'un procés. Les diferents parts del codi estan protegides entre sí. Per exemple, la zona de codi executable no es pot modificar pel mateix procés.
- Execució d'un procés. Un procés pot executar-se sense tenir tot el codi executable a memòria RAM.
- Comunicació interprocés. Permet tenir una zona de memòria compartida entre processos perquè puguin compartir dades.

Introducció

Més funcionalitats importants:

- Codi executable compartit. Diferents processos poden compartir codi; per exemple llibreries comunes.
- Gestió de memòria de la pila o memòria dinàmica. El sistema operatiu ubica memòria per aquestes zones a mesura que creixen.
- Fitxers mapats a memòria. Es pot accedir a un fitxer accedint a posicions de memòria com si fos un vector (i.e. sense fer servir les funcions read i write).

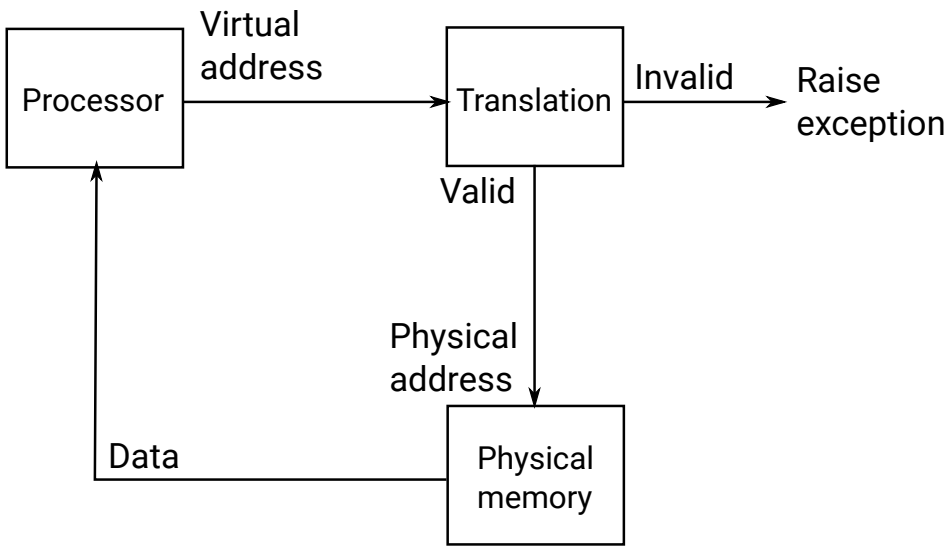
Contingut

El contingut d'aquestes transparències és

1. Concepte de traducció d'adreça. Els conceptes bàsics associats.
2. Traducció d'adreça. Com es pot dissenyar el maquinari per proveir de la màxima flexibilitat al sistema operatiu?
3. Fitxers mapats a memòria i la memòria virtual.
4. Protecció d'adreces a nivell de programari.

Concepte de traducció d'adreça

La majoria de sistemes (els que nosaltres utilitzem) tenen un maquinari especialitzat que realitza aquesta traducció; el maquinari està gestionat pel sistema operatiu. Totes (totes!) les direccions de memòria generades pels processos són "traduïdes" a una adreça de memòria física.



Concepte de traducció d'adreça

Respecte la traducció

- Un procés genera direccions de memòria virtual; cal realitzar la traducció a una adreça física.
- La traducció acostuma a fer-se a nivell de maquinari, la Memory Mapping Unit (MMU). El sistema operatiu s'encarrega de gestionar i configurar aquest maquinari perquè realitzi aquesta tasca de traducció de forma correcta.
- En traduir una adreça, es comprova que l'adreça virtual és està mapada a una adreça física. Si no ho està, es produeix una excepció!

Concepte de traducció d'adreça

En produir-se l'excepció...

- El sistema operatiu comprova si l'adreça virtual pertany al procés.
- Si no pertany al procés, el sistema operatiu el mata (li envia el senyal SIGSEGV).
- Si pertany al procés, el sistema operatiu ha de fer les "gestions" necessàries perquè l'adreça virtual es mapi a una adreça física perquè el procés pugui accedir a la dada demanada.

Concepte de traducció d'adreça

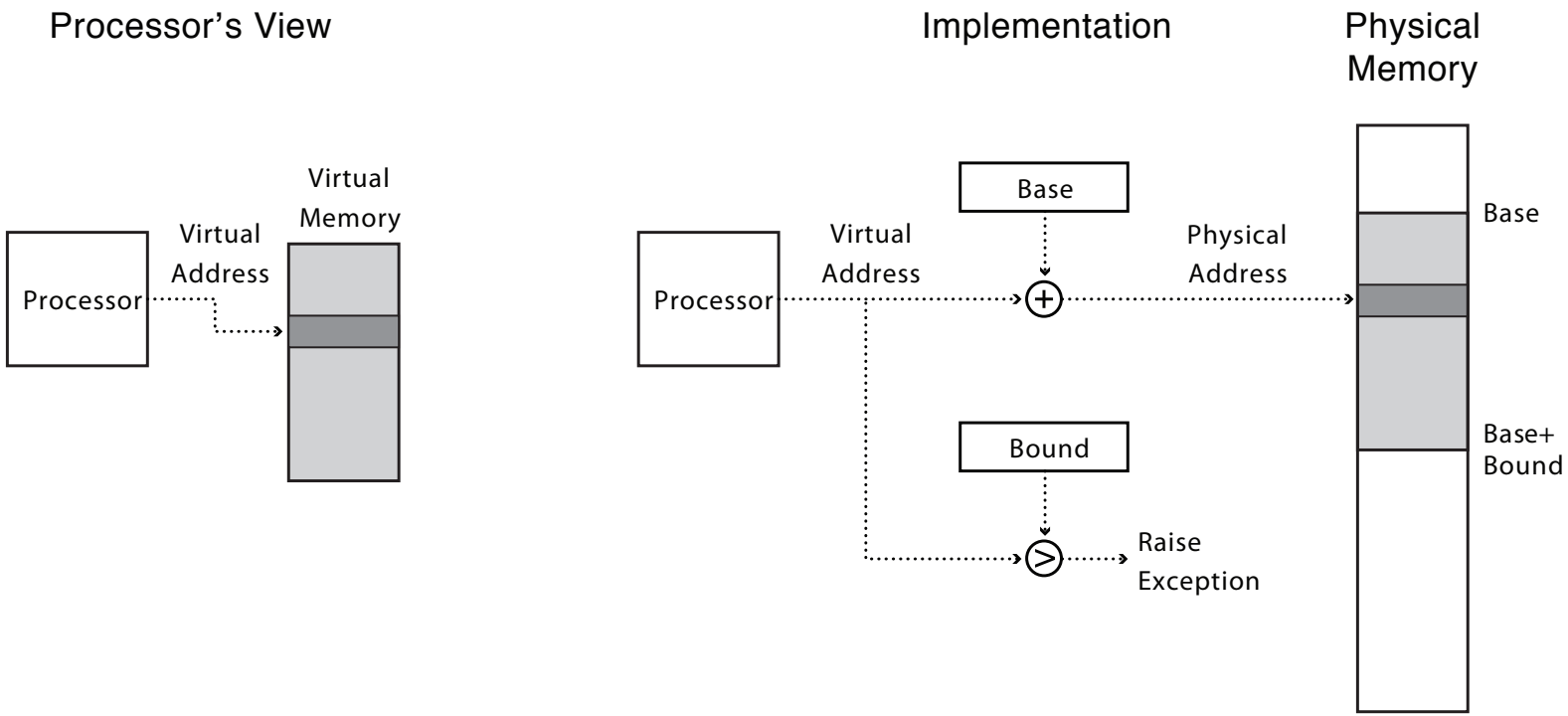
Com es pot implementar aquesta traducció? Un vector? Un arbre? Una *hash table*? Totes les respostes són certes, totes s'han implementat en sistemes reals. Esquemes de traducció que es veuran:

- Sistemes "antics": base i limit, memòria segmentada
- Sistema "actual": memòria paginada

Traducció d'adreça: *base* i *limit*

Un dels esquemes bàsics de traducció d'adreces fa servir dos registres: *base* i *limit*.

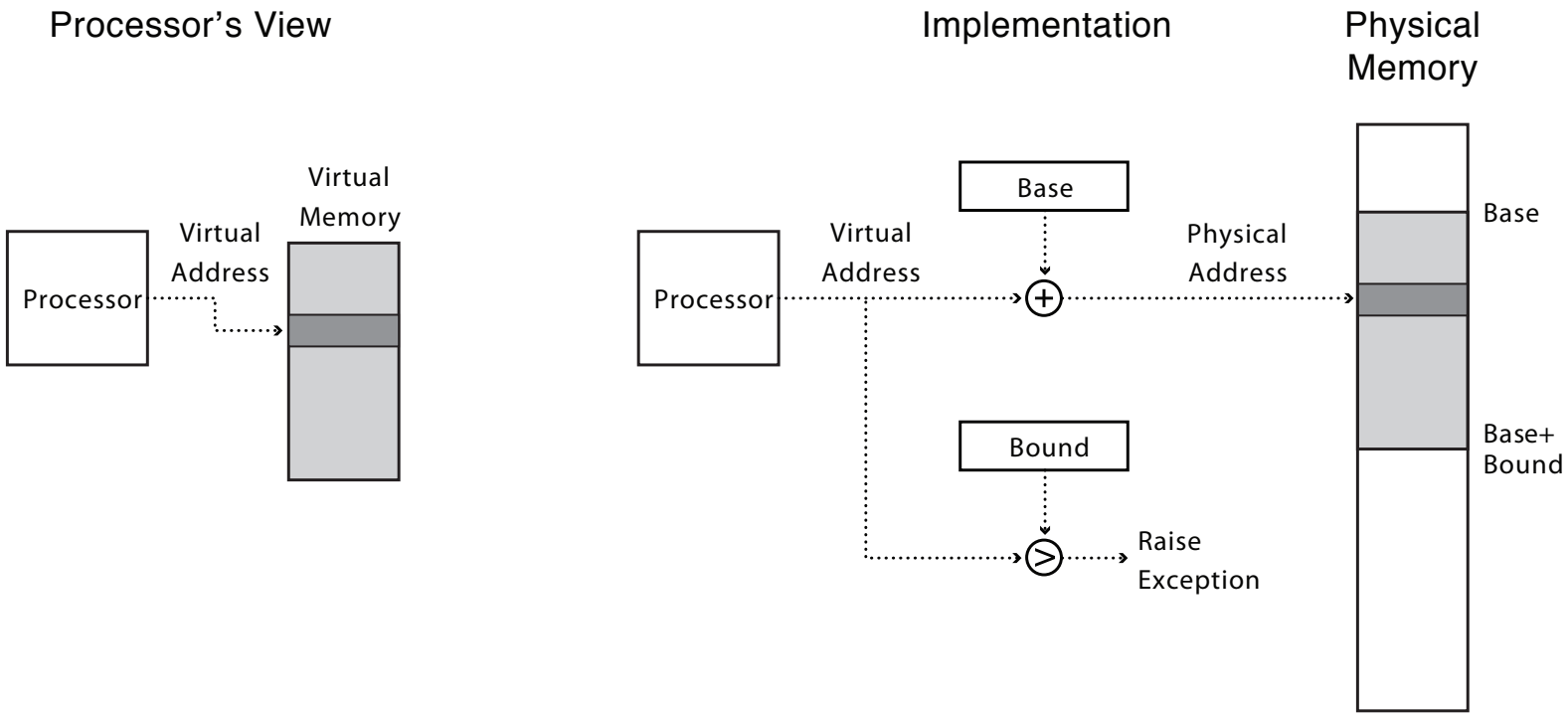
- Aquests registres només poden ser modificats per instruccions privilegiades.
- Cada procés té els seus propis registres base i limit.



Traducció d'adreça: *base* i *limit*

Traducció realitzada amb els registres *base* i *limit*

- Cada cop que s'accedeix a memòria, se suma la base a l'adreça. Si se supera el límit, es produeix una excepció.



Traducció d'adreça: *base* i *limit*

Esquema amb dos registres: *base* i *limit*. Té importants defectes

- Pila i memòria dinàmica expansible? Amb només dos registres cal preveure la memòria màxima que ocuparà el procés en el moment de carregar-lo de disc a memòria. Aquesta previsió és difícil de fer.
- Fragmentació de memòria? Amb aquest esquema tot el procés ocupa un espai continu, "lineal", que no es pot fragmentar en trossos.
- Auto-aïllament? Un procés pot sobreescriure la seva zona de codi executable, per exemple.
- Compartició de memòria? Amb aquest esquema no es pot aconseguir que diversos processos "comparteixin" una zona de memòria (per exemple, la de les instruccions màquina).
- Comunicació interprocés? Diversos processos no es poden comunicar entre sí compartint una zona de memòria.

Traducció d'adreça: memòria segmentada

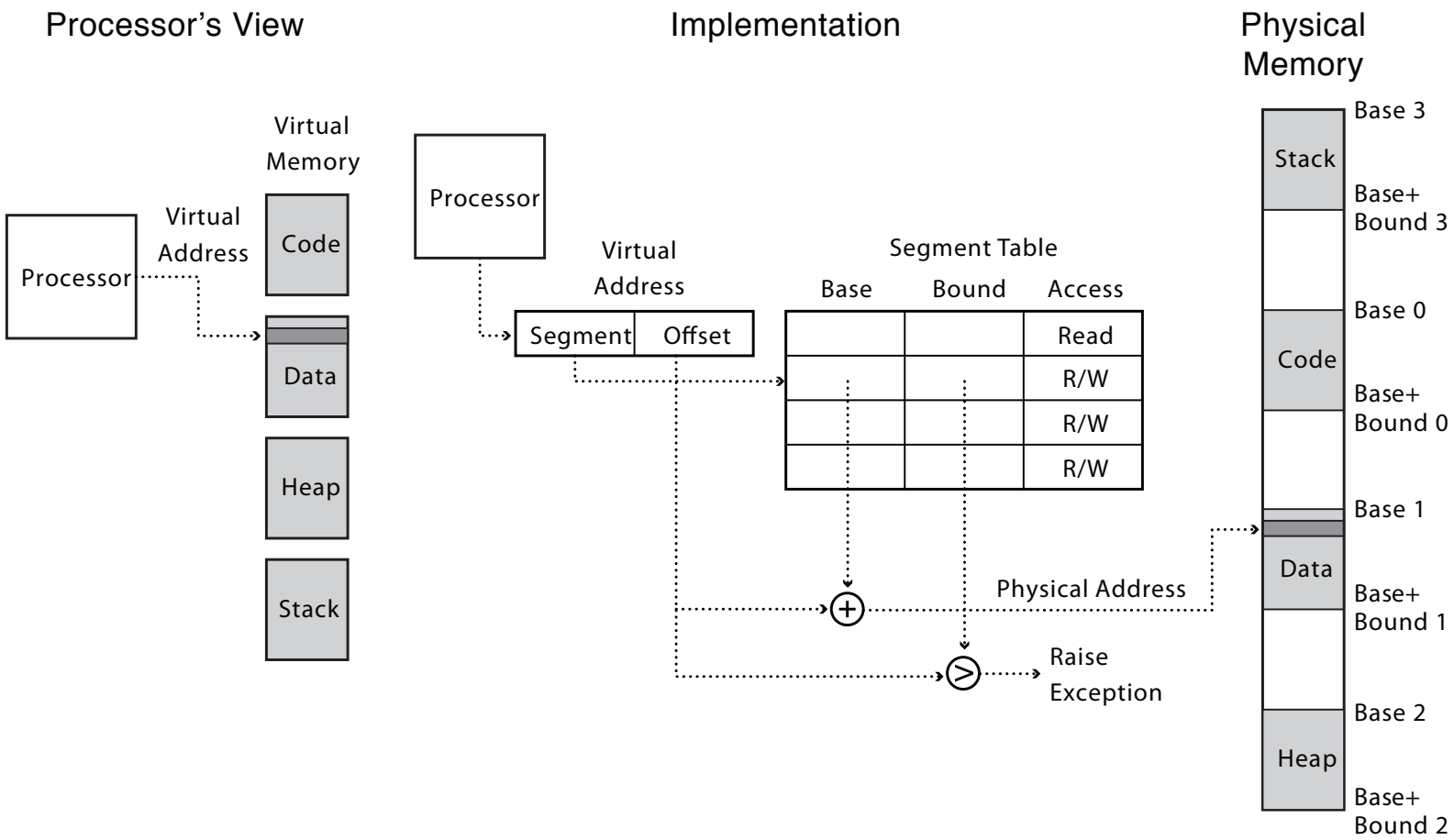
Molts dels problemes anteriors es poden solucionar de forma senzilla amb un canvi petit: en comptes de tenir un únic parell de registres *base* i *limit* per procés, el maquinari dóna suport per a múltiples registres *base* i *limit* per a cada procés.

És el que s'anomena segmentació.

- Cada parell de registres base i limit té associada una porció de l'espai d'adreces anomenat segment.
- Cada segment s'emmagatzema de forma contigua a memòria física i pot tenir una mida variable.

Traducció d'adreça: memòria segmentada

Cada adreça virtual té dos components: el número de segment (és a dir, el *base* i *limit* associat) i l'*offset* dins del segment.



Traducció d'adreça: memòria segmentada

Al sistema de memòria segmentada

- A una adreça virtual es bits alts estan associats al número de segment; els bits baixos a l'*offset*. El número de segments total possible depèn del nombre de bits alts assignats a representar el segment.
- A nivell de maquinari es poden assignar diferents permisos d'accés (lectura, escriptura, execució) a cada segment. El sistema operatiu s'encarrega de gestionar-ho.
- Si un procés produeix una adreça de memòria virtual invàlida (que no pertany al procés), es produeix una excepció i el sistema operatiu genera un senyal SIGSEGV que s'envia al procés.

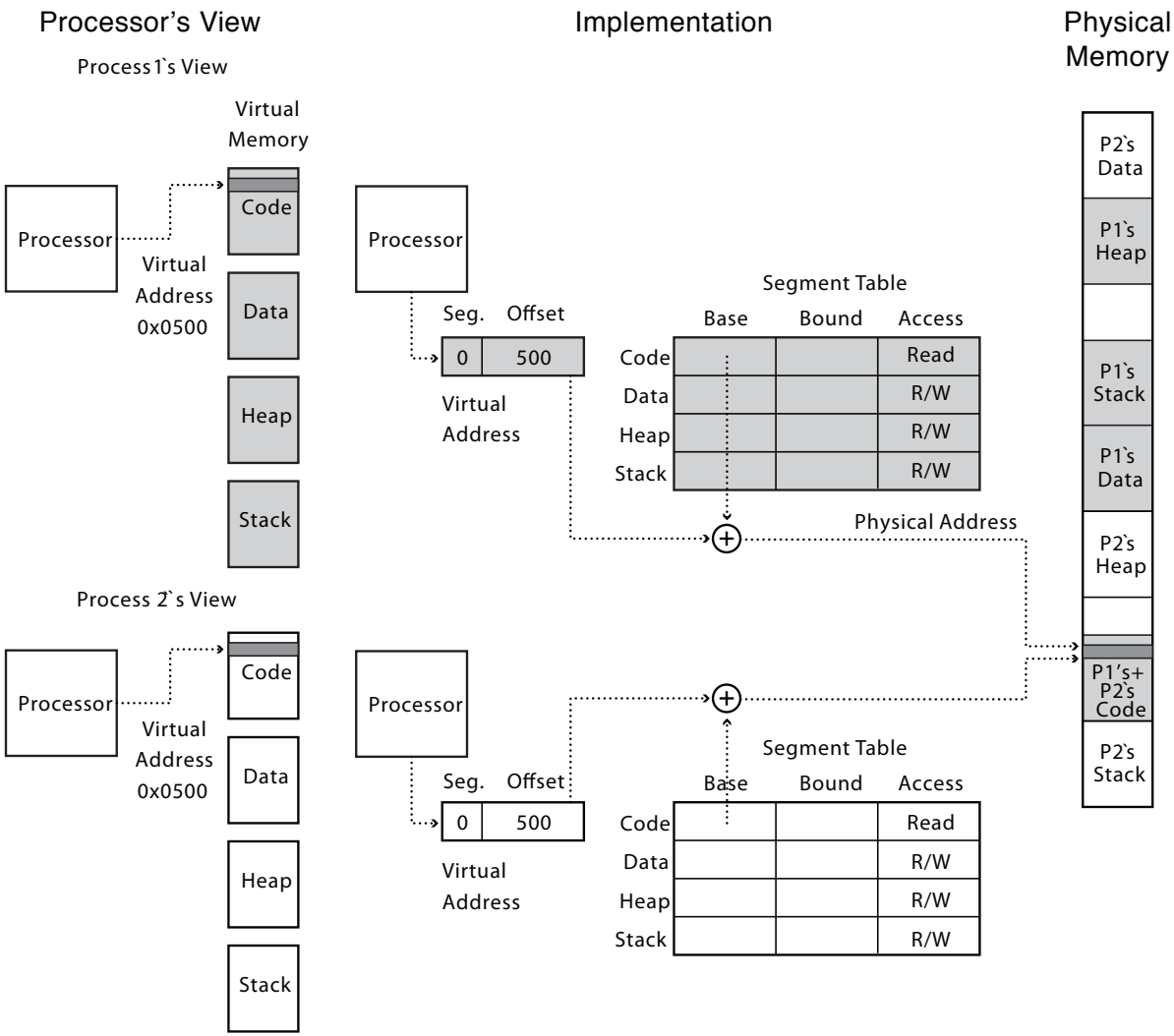
Traducció d'adreça: memòria segmentada

La memòria segmentada té importants avantatges respecte un esquema amb un sol parell de registres *base* i *limit*.

- Auto-aïllament. Permet que un procés es protegeixi a sí mateix diverses parts del codi.
- Fragmentació de memòria. L'espai de memòria d'un procés pot estar fragmentat en diversos trossos. Els segments estan associats a regions "gruixudes" com, per exemple, el codi, la pila, o la zona de memòria dinàmica.
- Compartició de segments. Diversos processos poden compartir codi (llibreries, etc.) compartint els registres *base* i *limit* d'un segment.
- Comunicació interprocés. Diversos processos poden comunicar-se entre sí compartint una zona de memòria.

Traducció d'adreça: memòria segmentada

Esquema de compartició de segments entre diversos processos:



Traducció d'adreça: memòria segmentada

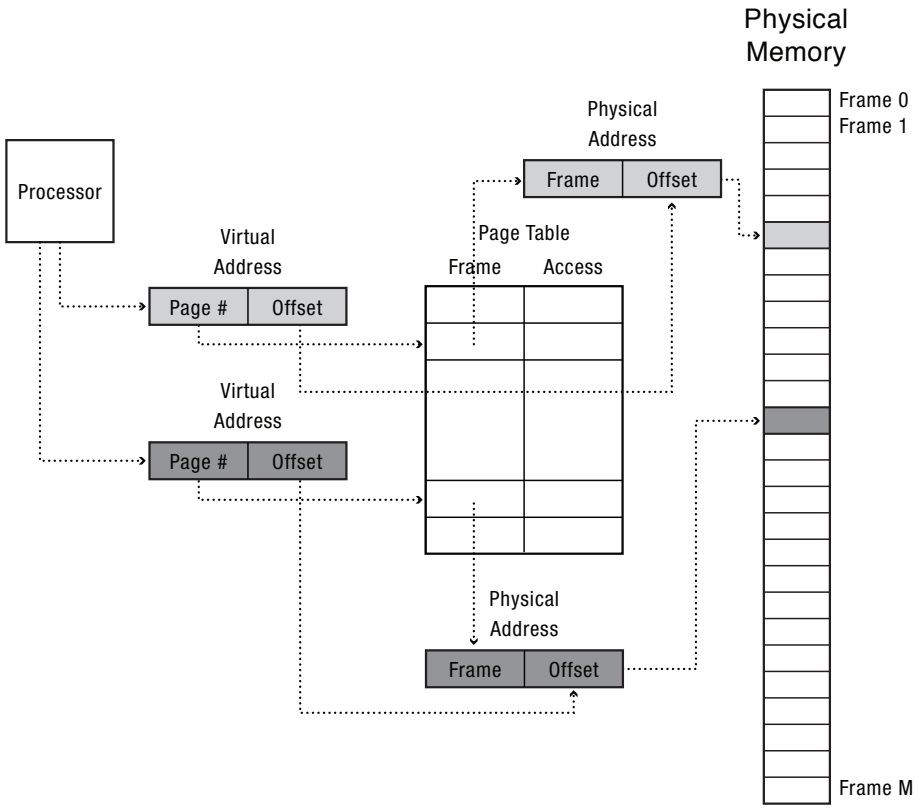
Aquest sistema té desavantatges. Per exemple, pot passar això:

1. A mesura que "passa el temps" la memòria física es divideix en trossos ocupats i lliures.
2. En crear un nou procés hem de crear nous segments. Es possible que no hi hagi cap zona lliure prou gran per a un segment. Però si "sumem" les regions lliures sí que hi pot haver una zona lliure prou gran per al segment.
3. En el cas anterior, el sistema operatiu pot "compactar" les regions per a unificar les regions lliures. Cal canviar els registres *base* i *limit* per a cada segment i moure les dades a memòria física. Les adreces virtuals no canvien.
4. L'operació de compactació és costosa! Un ordinador podria trigar un segon (!) a realitzar aquest procés.

Traducció d'adreça: memòria paginada

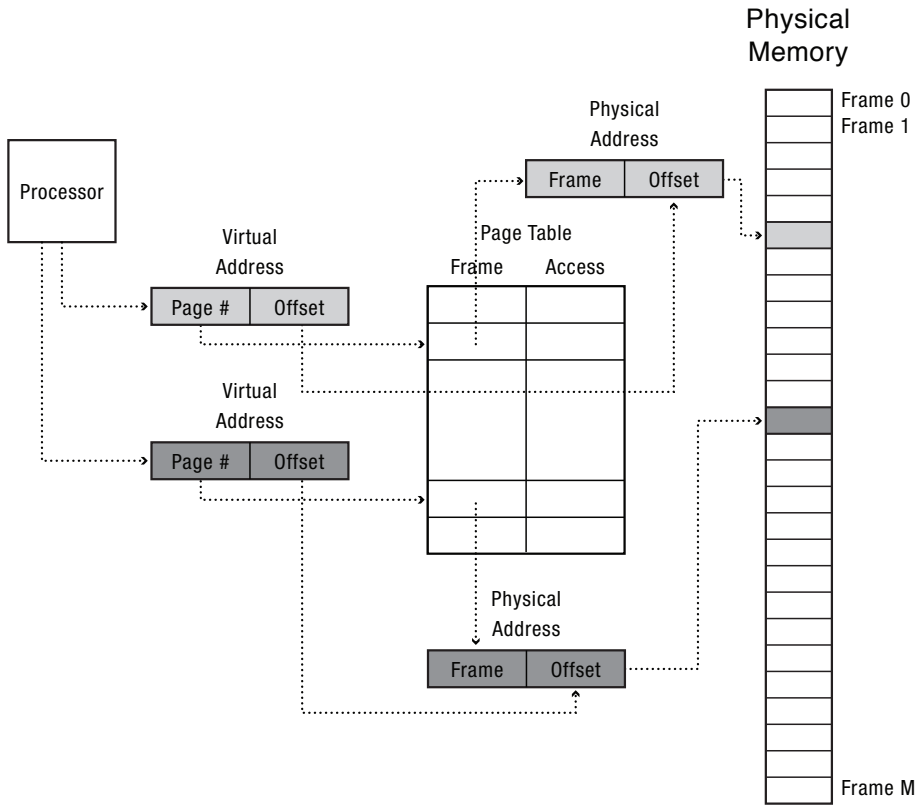
Una alternativa a memòria segmentada és la memòria paginada.

- La memòria física es divideix en blocs de mida fixa anomenats marcs de pàgina.
- La traducció es realitza amb una taula. Cada procés té la seva pròpia taula i el sistema operatiu s'encarrega de gestionar-les.



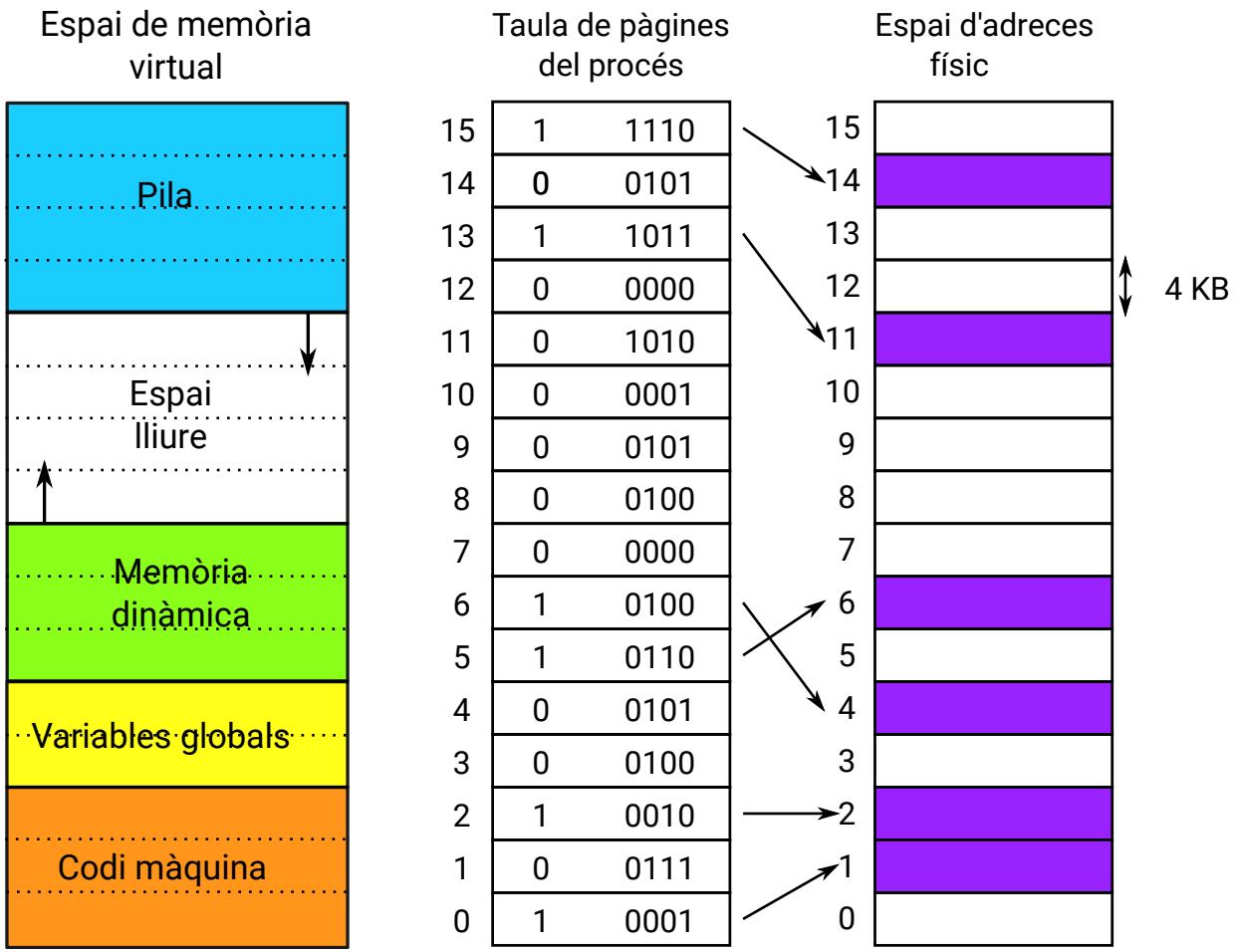
Traducció d'adreça: memòria paginada

- Els marcs de pàgina són de mida fixa i potència de 2: es fan servir els "bits alts" de l'adreça com a índex a la taula i el "offset" indica la posició dintre del marc.
- Per fer la traducció només cal fer una "substitució" dels bits alts de l'adreça virtual pels bits indicats a la taula.



Traducció d'adreça: memòria paginada

Exemple de mapat d'una adreça virtual a una adreça física amb arquitectura de 16 bits (veure següent transparència).



Traducció d'adreça: memòria paginada

Respecte l'esquema de la transparència anterior:

- Es un esquema de 16 bits amb marcs de pàgina de 4Kbytes (4096 bytes). Això dóna un total de $2^{16}/4096 = 16 = 2^4$ marcs de pàgina. Es faran servir doncs els 4 bits superiors de l'adreça per fer referència a un marc[1].
- Per exemple, l'adreça virtual 0000 1111 0000 1111 es mapa a 0001 1111 0000 1111, i l'adreça virtual 0001 1111 0000 1111 produeix una excepció.
- Observar que la memòria virtual d'un procés acostuma a estar "dispersa" entre la memòria física com si fos un mosaic.

[1] Al dibuix anterior, el primer bit fa referència a si el marc virtual es troba a memòria física. Si hi és, els 4 bits següents fan referència al marc físic en què es troba. Si no hi és, els 4 bits no signifiquen res (és merda!)

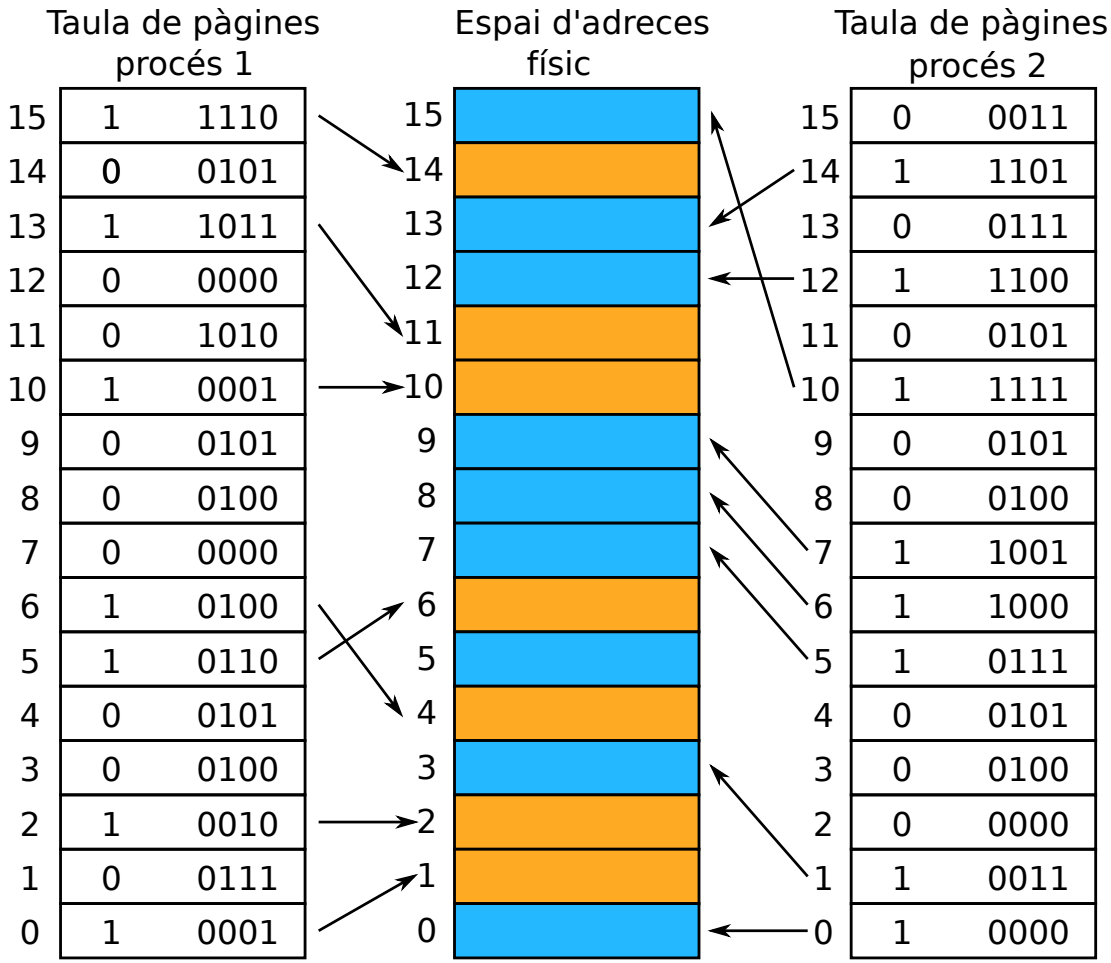
Traducció d'adreça: memòria paginada

Unes conclusions del dibuix:

- Cada entrada a la taula conté la "traducció" al marc físic així com informació sobre les propietats associades (lectura, escriptura, execució, si està disponible o no, ...). Si un procés fa una operació invàlida, es produeix una excepció.
- Cada procés creu que té disponible "tot" l'espai de memòria possible: en sistemes de 32 bits són 4GB. A l'actualitat, els sistemes de 64 bits només es "veuen" realment 48 bits per motius electrònics.
- El sistema operatiu s'encarrega de gestionar les taules dels processos. Les taules s'emmgatzemen a memòria RAM.

Traducció d'adreça: memòria paginada

Exemple amb dos processos: la memòria virtual permet gestionar la protecció de memòria entre aquests.



Traducció d'adreça: memòria paginada

Recordem el codi `exemple_fork.c`

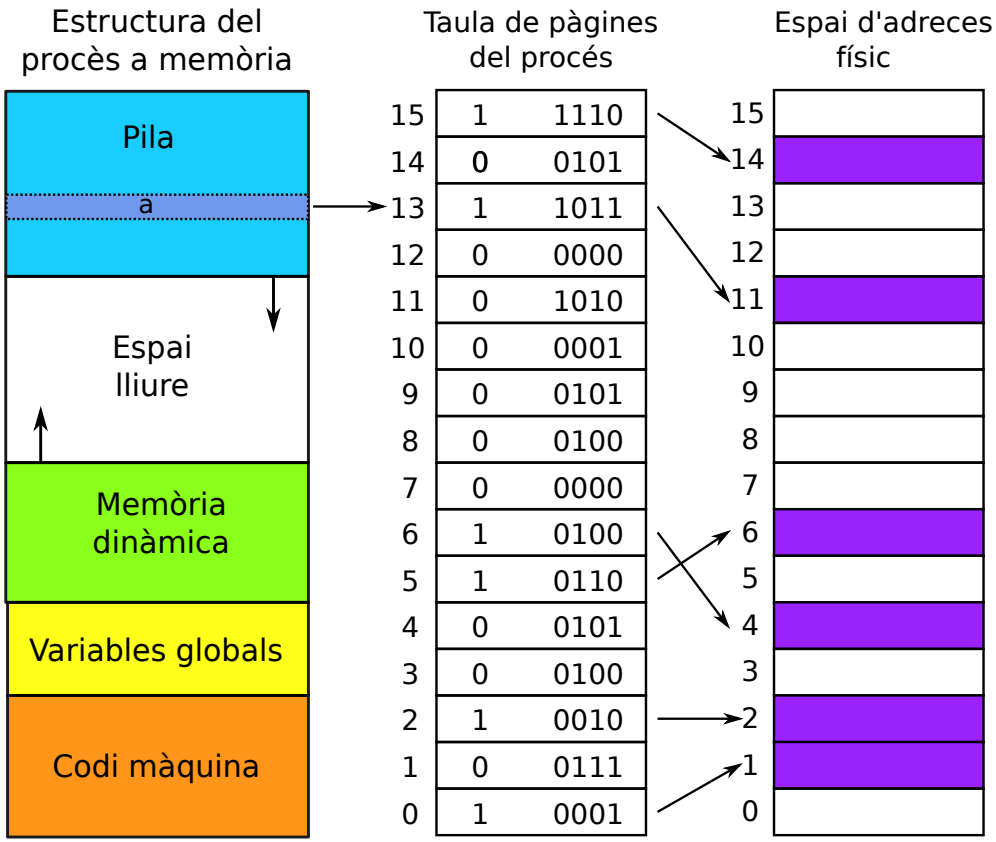
```
Direccio d'a abans fork:  140723702953032
Fill valor d'a:  2
Fill direccio d'a:  140723702953032
Pare valor d'a:  1
Pare direccio d'a:  140723702953032
```

Tot i que es canviï de valor la variable "a" al pare i/o al fill, l'adreça de memòria no canvia. Això és perquè s'estan imprimint per pantalla les adreces virtuals!

Pel cas del `fork` es fa servir una tècnica anomenada Copy-On-Write (COW) per optimitzar el `fork`.

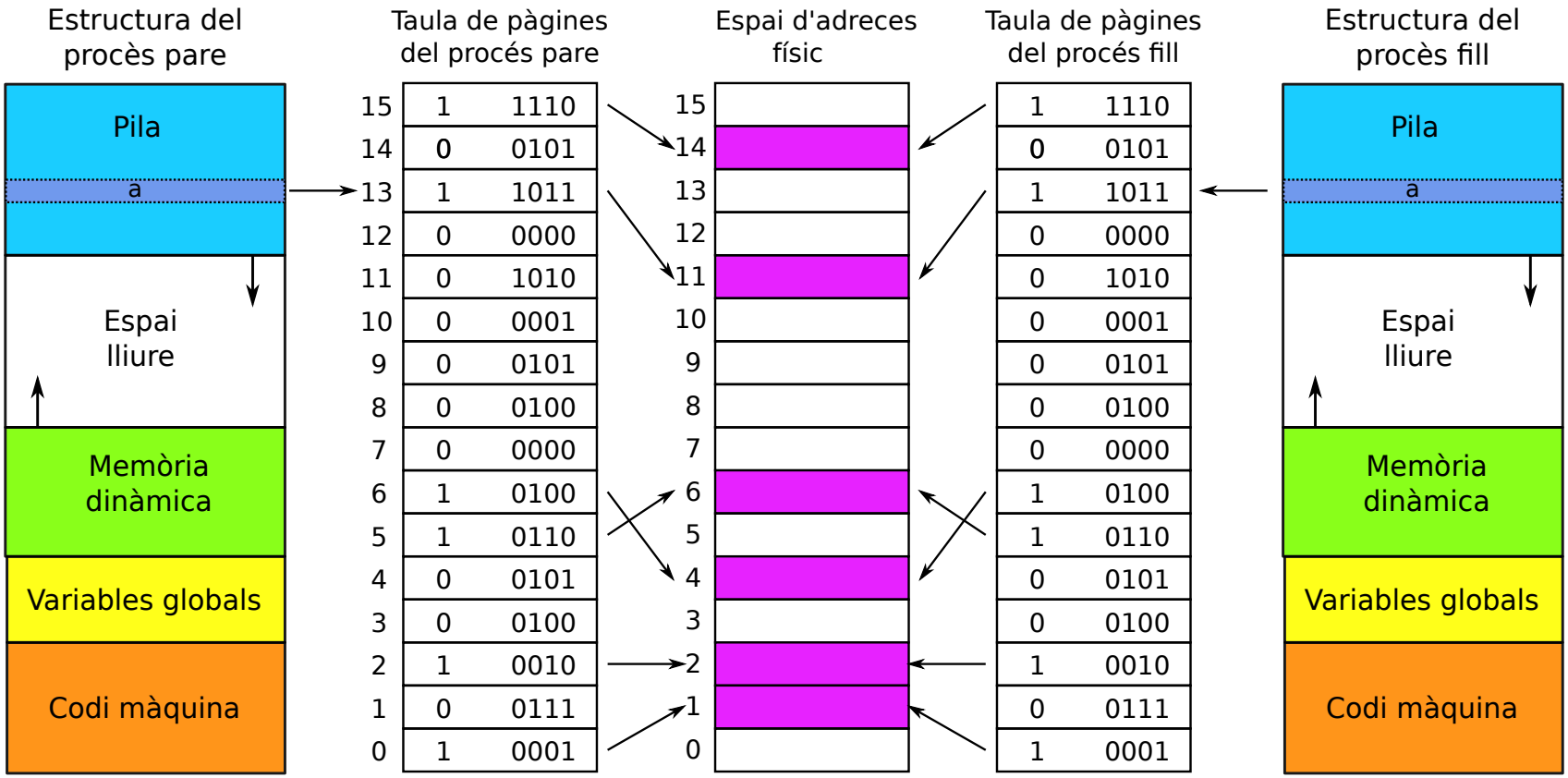
Traducció d'adreça: memòria paginada

Mapa de memòria abans de realitzar el fork



Traducció d'adreça: memòria paginada

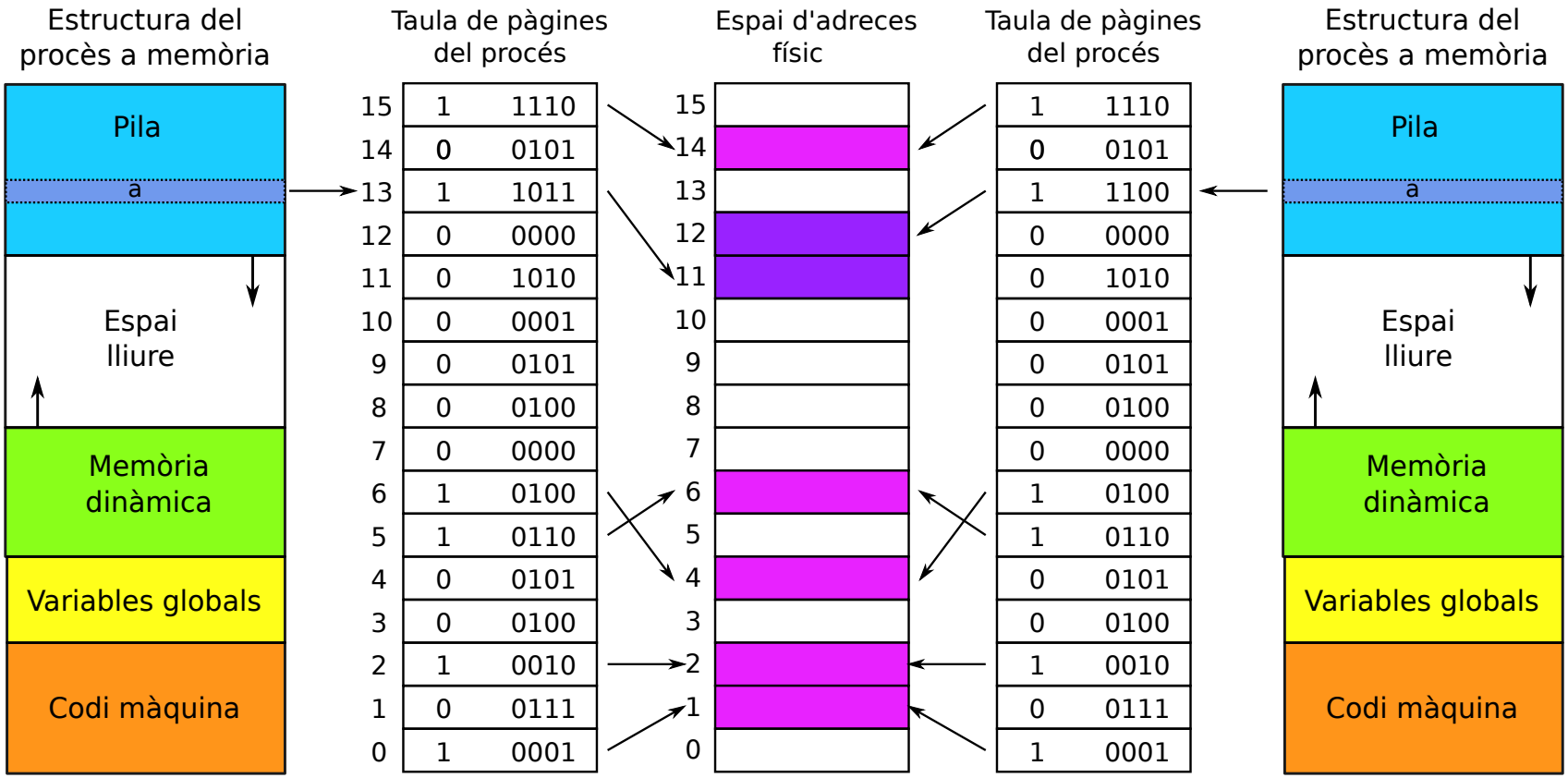
Mapa de memòria després de realitzar el fork



1. El sistema operatiu duplica la taula del procés però no pas les dades físiques associades.
2. A la taula es marquen les pàgines físiques com a només de lectura.

Traducció d'adreça: memòria paginada

Mapa de memòria després de realitzar l'assignació a = 2 al fill.



1. En realitzar el fill a = 2 es produeix una excepció (fallada de pàgina). El SO la captura i duplica les pàgines físiques.
2. Es retorna de la fallada i el procés continua l'execució com si res hagué passat...

Traducció d'adreça: memòria paginada

Tècnica Copy-On-Write (COW)

- Les pàgines es marquen com a de lectura i es copien només quan hi accedim per escriptura. Només es copien aquelles pàgines que es modifiquen, la resta són compartides entre pare i fill.

Compartició de memòria

- Observar que el codi màquina es comparteix entre pare i fill (ja que no es modifica).
- De forma genèrica, compartir zones de memòria (llibreries, codi executable, ...) és senzill a memòria paginada.
- Si després del fork es fa un exec, la taula de pàgines s'adapta al nou executable.

Traducció d'adreça: memòria paginada

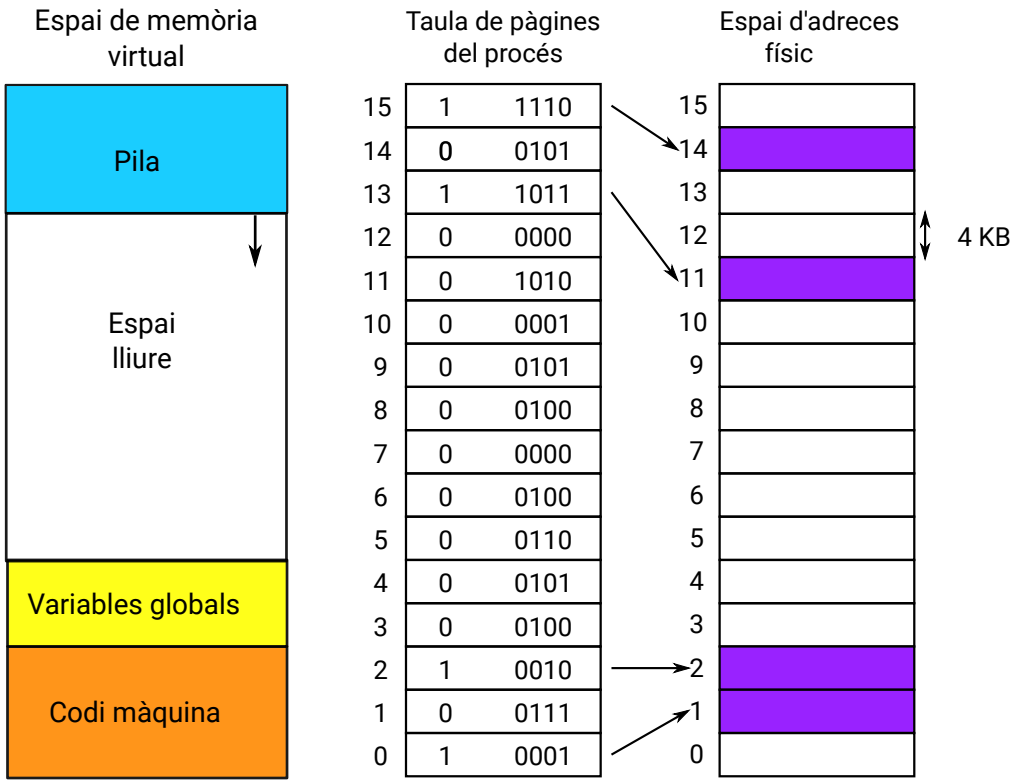
Què passa en fer un malloc?

- Aquesta funció només inicialitza el mapa de memòria virtual.
- L'assignació de marcs de pàgina físics es fa a mesura que hi accedim realment.

Veure codi `exemple_malloc.c`.

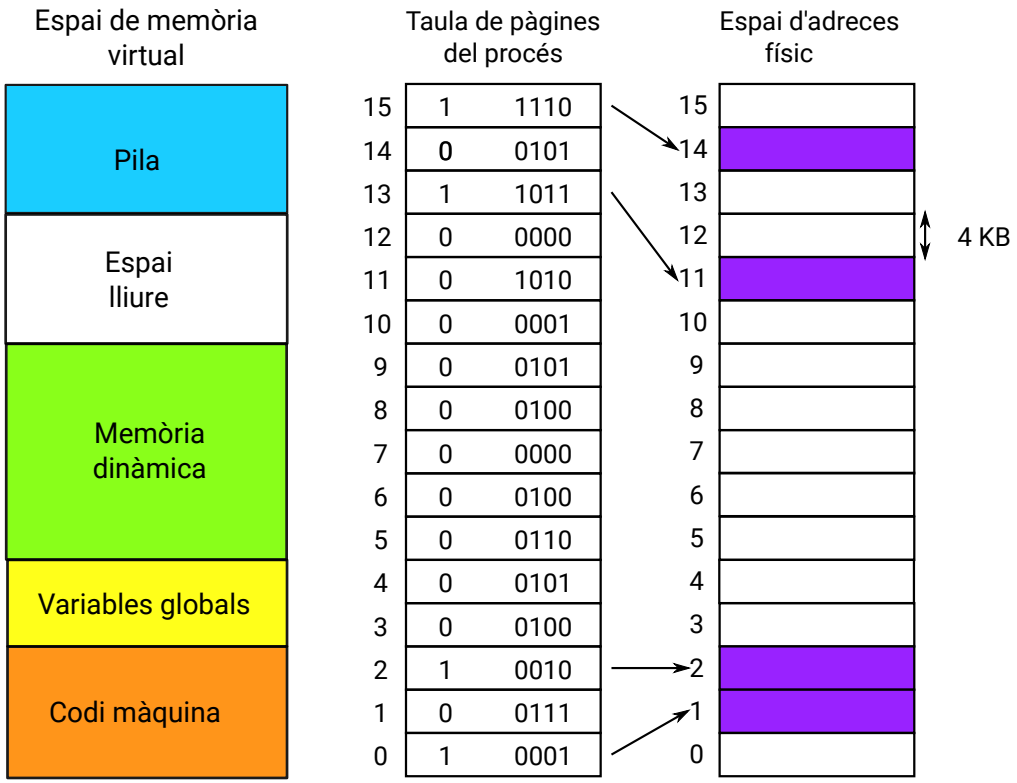
Traducció d'adreça: memòria paginada

Mapa de memòria abans del malloc (amb arquitectura de 16 bits)



Traducció d'adreça: memòria paginada

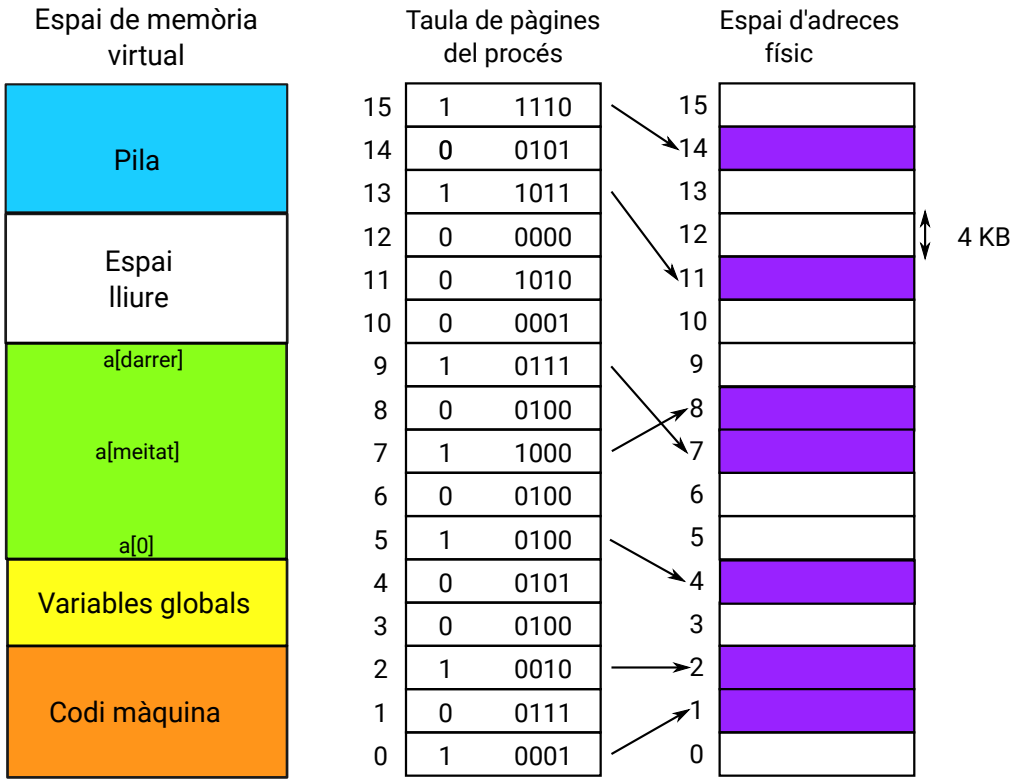
Mapa de memòria després del malloc (amb arquitectura de 16 bits)



Un cop fet el malloc només s'ha fet la reserva a l'espai virtual, no pas a l'espai físic.

Traducció d'adreça: memòria paginada

Mapa de memòria després dels accessos (arquitectura de 16 bits)



1. Si l'adreça no existeix a l'espai físic es produeix una fallada de pàgina. El SO "cerca" una pàgina disponible a l'espai físic.
2. Es retorna de la fallada i es realitza l'accés...

Traducció d'adreça: memòria paginada

Per al codi `exemple_malloc.c`.

- En cridar `malloc` només es fa la reserva a l'espai virtual, no pas a l'espai físic. El sistema operatiu assigna espai físic en accedir a una determinada posició del vector.
- Per a l'exemple donat hem reservat molts GBytes però físicament només s'assignen 3×4 Kbytes de memòria! El sistema operatiu ha de decidir quin marc de pàgina físic utilitzar.

Traducció d'adreça: memòria paginada

La memòria paginada resol els principal problema de la memòria segmentada: trobar zones de memòria lliures.

- El sistema operatiu ha de mantenir una taula amb els marcs de pàgina lliures.
- La memòria física és limitada: pot arribar un moment en què tots els marcs de pàgines físiques estiguin ocupats. Si un procés en requereix nous marcs, el sistema operatiu ha de decidir quins marcs "descarta" de l'espai de memòria física.
- El sistema operatiu fa servir un algorisme per decidir quins marcs descartar. Aquests marcs poden pertànyer a un procés diferent del que s'està executant. Veurem els principis a la part de "fitxers mapats a memòria".

Traducció d'adreça: memòria paginada

Altres punts a tractar

- En un sistema x86 de 32 bits els marcs físics tenen mida de $2^{12} = 4096$ bytes. Això vol dir que la taula té $2^{20} = 1048576$ entrades. Si cada entrada ocupa 4 bytes significa que una taula ocupa 4Mbytes, i hi ha una per a cada procés!
- Què passa en un sistema de 64 bits? Cada taula ocuparia un munt a memòria (molts Gigaytes)! Podríem augmentar la mida del marc físic (que a l'actualitat continua sent de 4KB), però malbarataríem memòria RAM. Aleshores, què?

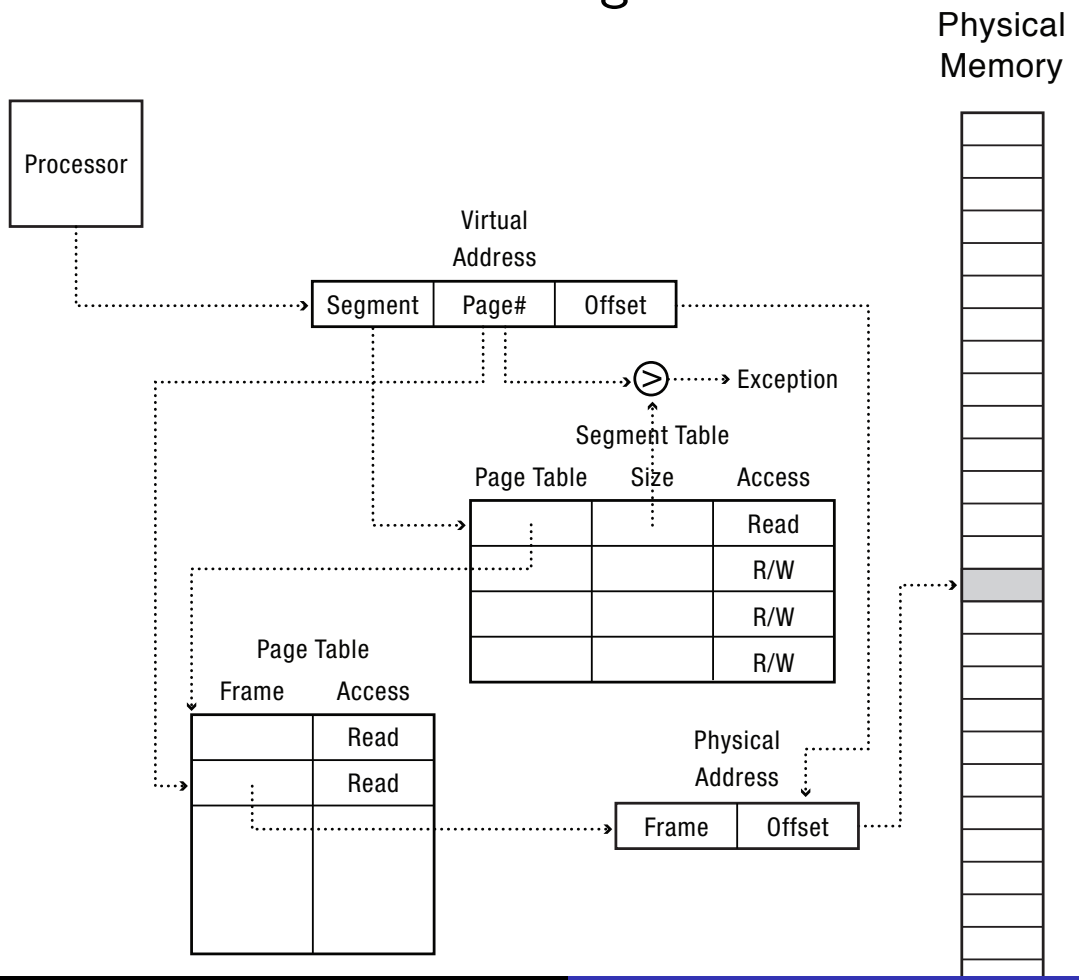
Traducció d'adreça: memòria paginada

La solució en sistemes de 64 bits és no fer servir una taula! Es pot fer servir un arbre o una taula de hash... tots dos s'utilitzen en sistemes reals. De forma genèrica són esquemes de traducció multinivell.

- Tots els sistemes multinivell utilitzen l'esquema de paginació al nivell més fi (a les fulles). La diferència està en com arribar, a traves dels nivells, al nivell més fi.
- Anem a veure aquests esquemes d'arbre: paginació segmentada, paginació multinivell i paginació segmentada multinivell.

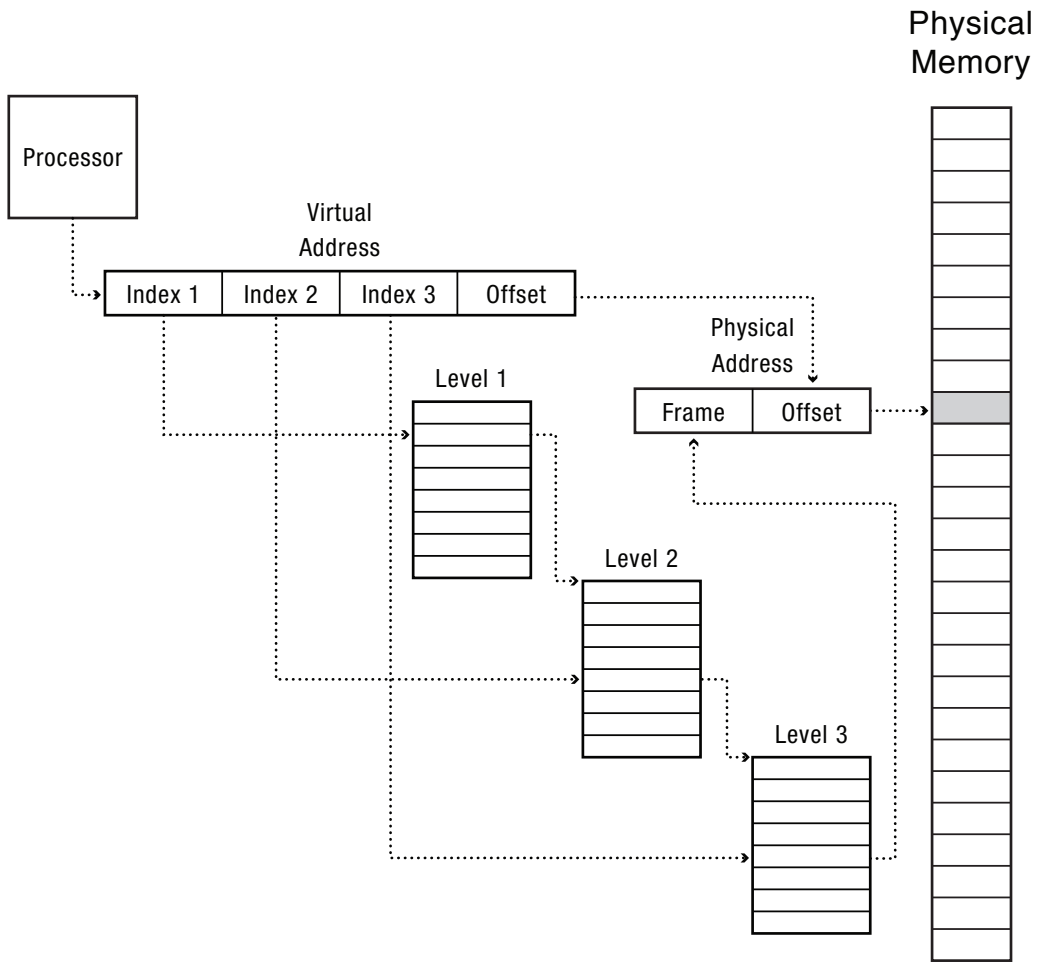
Traducció d'adreça: memòria paginada

Amb paginació segmentada l'arbre té dos nivells: el 1r nivell, el segment, apunta a una taula, del 2n nivell, que apunta als marcs de pàgina físics. Les propietats d'accés (lectura, escriptura, execució, ...) es poden establir al nivell de segment.



Traducció d'adreça: memòria paginada

Amb paginació multinivell l'arbre té múltiples nivells. Només s'ubiquen les parts de l'arbre que realment són utilitzades pel procés. Penseu-hi...



Traducció d'adreça: memòria paginada

Es poden combinar els dos esquemes anteriors per obtenir un esquema de paginació multinivell segmentada: cada segment conté una taula multinivell. És l'esquema utilitzat pels sistemes x86, tant per 32 com per 64 bits.

- Per qüestions històriques, el número de "segment" s'emmagatzema en un registre de CPU separat (que no es fa servir!).
- En sistemes de 32 bits, s'utilitza un esquema multinivell de dos nivells: 10 bits al primer nivell, 10 al segon i 12 per l'offset.
- En sistemes de 64 bits, actualment, només s'utilitzen 48 bits de l'espai d'adreces virtual (Terabytes). S'utilitzen 4 nivells d'arbre ($48 = 9+9+9+9+12$). Només s'ubiquen les parts de l'arbre que realment són utilitzades pel procés.

Traducció d'adreça: memòria paginada

Tot això funciona molt bé, però... les taules (multinivell) que fan la traducció es guarden a memòria RAM.

- Suposem que volem accedir al valor d'una variable "a". Quants accessos a memòria fan falta per accedir al valor de la variable "a"? Penseu-hi...
- Suposem que volem accedir al valor d'un element d'un vector, "a[10]". Quants accessos a memòria fan falta pera accedir al valor de "a[10]"? Penseu-hi...

Com arreglem aquesta "ineficiència" a l'hora d'accedir a adreces de memòria?

- Es fa servir maquinari específic que implementa memòria *caché* a diferents parts per aconseguir-ho! No hi entrarem en detalls, aquí...

Contingut

El contingut d'aquestes transparències és

1. Concepte de traducció d'adreça. Els conceptes bàsics associats
2. Traducció d'adreça. Com es pot dissenyar el maquinari per proveir de la màxima flexibilitat al sistema operatiu.
3. Fitxers mapats a memòria i la memòria virtual.
4. Protecció d'adreces a nivell de programari.

Centrem-nos en els dos darrers punts...

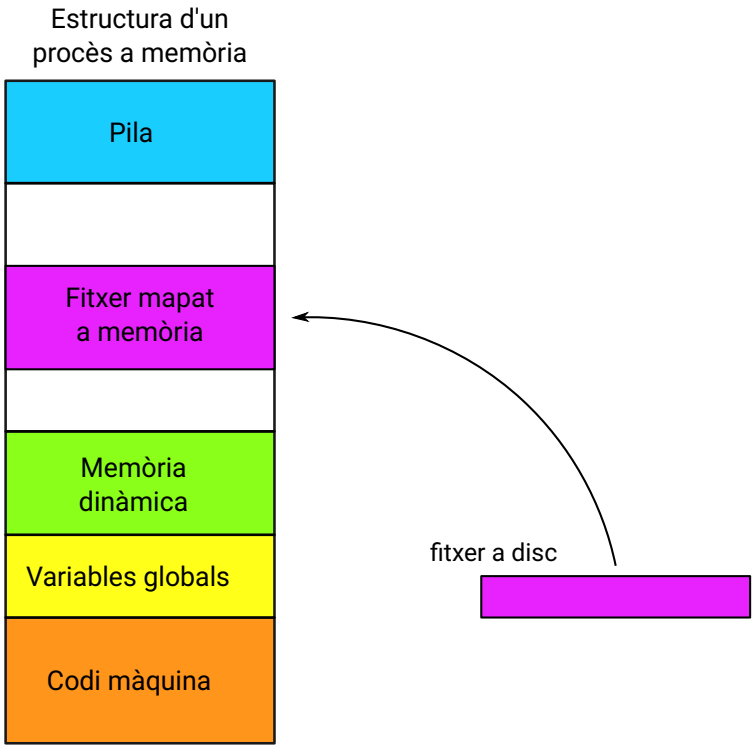
Fitxers mapats a memòria

Anem a veure dos casos d'ús de la memòria virtual

1. Fitxers mapats a memòria
2. La memòria virtual pels processos

Fitxers mapats a memòria

El sistema operatiu permet manipular fitxers mapant-los directament a memòria. Teniu un exemple al codi `mmap.c`.



Fitxers mapats a memòria

Algunes característiques dels fitxers mapats a memòria:

- Qualsevol fitxer de disc pot ser manipulat fent servir l'espai d'adreces virtual.
- Els blocs es porten a memòria si són referenciats i són guardats a disc automàticament si són modificats per l'aplicació (són guardats en sortir de l'aplicació o si la pàgina es descartada pel sistema operatiu).

De tot això se n'encarrega el sistema operatiu.

Fitxers mapats a memòria

Això porta al concepte de paginació sota demanda

- Amb paginació sota demanda, una aplicació pot accedir a més memòria de la que físicament és present. Les pàgines de memòria física es poden interpretar com una "caché" dels blocs de disc.
- Si una aplicació accedeix a una pàgina no disponible a memòria RAM, el sistema operatiu la porta de forma transparent de disc a memòria.

Anem a veure un exemple amb el codi `mmap.c`.

Fitxers mapats a memòria

Com funciona?

1. En intentar accedir a una posició vàlida però que no està carregada a memòria es produeix una excepció de pàgina.
2. El sistema operatiu captura l'excepció de pàgina; detecta que es correspon a una direcció virtual vàlida i converteix l'adreça virtual a una posició de bloc al disc.
3. Es llegeix el bloc de disc. La lectura es bloquejant i el sistema operatiu fa mentrestant altres coses com, per exemple, executar un altre procés.

Fitxers mapats a memòria

continua...

1. En haver-se llegit el bloc el disc es produeix una interrupció (de disc) i es fa una crida al sistema operatiu.
2. El sistema operatiu actualitza la taula multinivell amb la nova pàgina carregada. Es possible que el sistema hagi hagut d'expulsar una altre pàgina que hi havia carregada anteriorment.
3. El sistema operatiu recupera l'execució del procés en el punt en què es va produir l'excepció de pàgina.

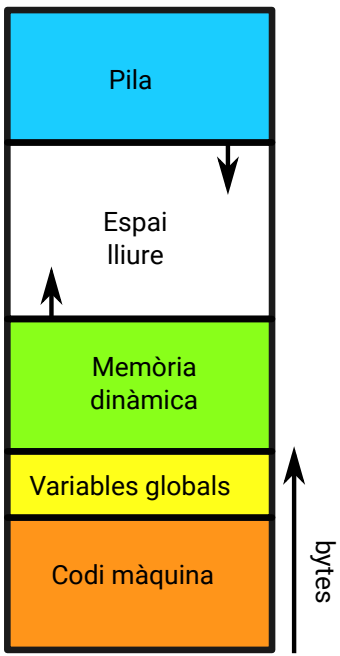
Fitxers mapats a memòria

Alguns detalls més...

- En expulsar una pàgina de memòria el sistema operatiu pot haver de guardar la pàgina a disc. Com sap si ho ha de fer? Això ho sap perquè el maquinari guarda a la taula un bit, anomenat *dirty bit*, que indica si la pàgina ha estat modificada.
- El sistema operatiu decideix quina pàgina expulsar (descartar) fent servir un algorisme que utilitza el *use bit*, un bit de la taula de pàgines on el maquinari indica si s'ha accedit a la pàgina. El sistema operatiu recorre periòdicament les taules per saber quines pàgines han estat accedides i fer-ne "reset de la taula". El sistema operatiu pot descartar una pàgina que no necessàriament ha de pertànyer al procés que s'està executant en aquell moment.

Memòria virtual

La memòria virtual és una generalització del concepte de fitxers mapats a memòria: tots els segments (o regions) estan mapats a un fitxer de disc.



Estructura d'un procés a memòria

- Cada segment del procés es mapa a disc: l'executable, les llibreries, les variables globals, la pila, la memòria dinàmica.
- Exemple: en carregar-se un programa de disc, es pot començar a executar sense que aquest estigui completament a memòria. El sistema operatiu carrega a memòria les pàgines de disc a memòria a mesura que es va executant el programa.

Memòria virtual

Alguns punts a tenir en compte

- El sistema de memòria virtual ens ofereix flexibilitat: podem tenir més processos executant dels que caben realment a memòria. El sistema de memòria virtual s'encarrega de gestionar les pàgines associades a cada procés.
- El "balanceig" de quantes pàgines s'assignen a cada procés és delicat. Cal reduir al màxima les fallades de pàgina: un ordinador modern pot gestionar unes 100 fallades de pàgina per segon i el processador pot executar milers de milions d'instruccions per segon.
- En produir-se una falla de pàgina, els sistema pot haver de descartar una pàgina. Ho fa basant-se en el *use bit*: és descarten les pàgines que no han estat accedides recentment.

Memòria virtual

Alguns punts a tenir en compte

- L'executable o les llibreries dinàmiques es mapen amb el fitxer original en mode de només lectura (i execució).
- Les zones de la pila o de memòria dinàmica es mapen a la zona (partició) de *swap* perquè s'hi puguin emmagatzemar aquelles pàgines que no caben a memòria. S'hi guarden les modificacions de les pàgines però en sortir del procés es "perden" aquestes dades.

Veure codi `exemple1.c`. Executar l'aplicació i des d'un altre terminal executar:

```
$ cat /proc/<process_id>/maps
```

Algunes conclusions

El sistema de memòria pagina permet

- Traduir una direcció virtual a una física mitjançant una taula. Hi ha una taula per a cada procés, gestionada pel sistema operatiu.
- La taula permet protegir les pàgines dels processos entre sí.
- La taula permet que diferents processos comparteixin codi: un clar exemple són les llibreries, que acostumen a estar compartides entre processos.
- El sistema de memòria virtual de memòria gestiona quines pàgines es carreguen a memòria física. Per exemple, d'una llibreria molt gran només hi ha a memòria física aquelles parts que realment fan falta.

Contingut

El contingut d'aquestes transparències és

1. Concepte de traducció d'adreça. Els conceptes bàsics associats
2. Traducció d'adreça. Com es pot dissenyar el maquinari per proveir de la màxima flexibilitat al sistema operatiu.
3. Fitxers mapats a memòria i la memòria virtual.
4. Protecció d'adreces a nivell de programari.

Protecció de memòria a nivell de programari

Cada dia apareixen més sistemes en què el programari complementa els mecanismes de maquinari

- Idealment podríem utilitzar un intèrpret de codi ensamblador: agafa la instrucció de memòria, la interpreta, comprovar si les direccions són vàlides, ... això seria molt lent.
- Quines tècniques podem fer servir aleshores per executar codi en un entorn restringit?
- A l'actualitat la protecció es realitza pràcticament sempre a nivell de maquinari. Quin sentit té fer-ho a nivell a programari?

Protecció de memòria a nivell de programari

Avantatges d'implementar la protecció a nivell de programari

- Simplificació del maquinari: amb una implementació a nivell de programari disposaríem de molta més flexibilitat.
- Protecció a nivell d'aplicació: una aplicació es podria protegir a l'hora d'executar codi de tercers (navegadors web, ...)
- Protecció a nivell de nucli: a l'actualitat també s'executa codi no fiable al nucli del sistema operatiu (gestors o "*driver*" de dispositius).
- Portabilitat: es permet que l'aplicació executi a múltiples dispositius diferents (smartphone, tablet, netbook, ...) amb les mesures de seguretat necessàries.

El terme *sandbox* es fa servir per denotar la protecció a nivell de programari necessària per executar codi no fiable.

Protecció de memòria a nivell de programari

Una solució és fer que el llenguatge implementi la protecció

- Diversos dels primers ordinadors personals només es podien programar en un llenguatge: Xerox va desenvolupar el llenguatge Mesa, inspirador del Java; altres ordinadors només executaven Lisp, i d'altres Smalltalk, precursor del Python.
- A l'actualitat moltes aplicacions utilitzen llenguatges propis per controlar la seguretat: Java, Javascript, Python, ...
- Cal assegurar però que l'intèrpret i les llibreries que utilitzen són fiables, cosa que no és fàcil. Una errada en l'intèrpret o la llibreria pot ser aprofitada per realitzar un atac.
- Interpretar codi és lent: per això moltes llibreries es compilen directament en codi ensamblador. Amb la qual cosa tornem al nostre problema inicial... com protegim?

Protecció de memòria a nivell de programari

A l'actualitat existeixen múltiples llenguatges de programació... com podem protegir aïllar una aplicació fent servir programari (sense maquinari), de forma independent al llenguatge de programació?

- Seria molt interessant poder-ho: els navegadors, el sistema operatiu, aplicacions de bases de dades, ... acostumen a utilitzar codi de tercers.
- Google i Microsoft tenen productes, Native Client i AppDomain respectivament que, a partir d'un codi ensamblador qualsevol, el modifiquen per assegurar que no accedeixen a posicions de memòria fora de l'espai que tenen assignat. Tècnicament estem executant el codi dins d'un *sandbox*. Segons Google, la "degradació" per executar el codi modificat al processador és menys d'un 10%.

Protecció de memòria a nivell de programari

També s'està treballant en implementar *sandboxes* a partir de codi intermedi en comptes de codi ensamblador

- El codi intermedi és un llenguatge associat a una màquina abstracte que permet analitzar programes d'ordinador. El codi itermedi es portable entre diverses arquitectures.
- El sandbox genera el codi segur a partir d'aquest codi intermedi.
- La màquina virtual de Java es un tipus de sandbox: el codi Java es tradueix a codi intermedi. En temps d'execució es pot comprovar que les instruccions estan contingudes dins del sandbox.

Un resum...

El sistema de gestió de memòria es caracteritza per

- Els processos s'executen a l'espai de memòria virtual i la traducció a l'adreça física es realitza mitjançant un dispositiu maquinari. El sistema operatiu és l'encarregat de gestionar aquest maquinari.
- Un procés es pot executar i pot manipular dades sense tenir totes les dades carregades a memòria física. El sistema operatiu gestiona la memòria física per carregar de disc tot allò que el procés necessita per executar.
- Cadascun dels segments d'un procés (el codi, la pila, una llibreria, la memòria dinàmica, ...) té associat un fitxer mapat a disc. És la base del sistema de memòria virtual.
- El sistema operatiu és l'encarregat de gestionar quines pàgines d'un procés han d'estar a memòria física i quines es poden descartar. En cas necessari, el sistema operatiu les desa a disc perquè es puguin tornar a carregar més endavant.