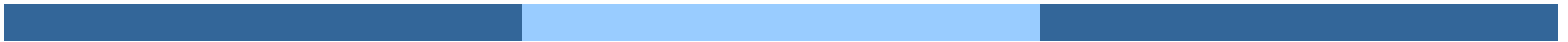


T3-Memòria



Índex

- Conceptes relacionats amb la gestió de memòria
- Serveis bàsics per a la gestió de memòria
 - Càrrega de programes en memòria
 - Memòria dinàmica
 - Suport HW a la gestió de memòria
 - ▶ A l'assignació de memòria
 - ▶ A la traducció d'adreces
 - Serveis per a l'optimització de l'ús de memòria física
 - COW
 - Memòria virtual
 - Prefetch
- Linux sobre Pentium



Memòria física vs. Memòria lògica

Espai d'adreces d'un procés

Assignació de adreces a un procés

Tasques del Sistema operatiu en la gestió de memòria

Suport del maquinari a la gestió de memòria

CONCEPTES

Memòria física vs. Memòria lògica

- CPU només pot accedir directament a memòria i registres
 - Instruccions i dades s' han de carregar en memòria per poder referenciar-se
 - Càrrega: reservar memòria, escriure-hi el programa i passar l'execució al punt d'entrada del programa
- Tipus d' adreces:
 - Referència emesa per la CPU: @ lògica
 - Posició ocupada en memòria: @ física
 - No tenen per què coincidir si el SO i el HW ofereixen suport per a la **traducció**
 - ▶ Els sistemes de propòsit general actuals l'ofereixen

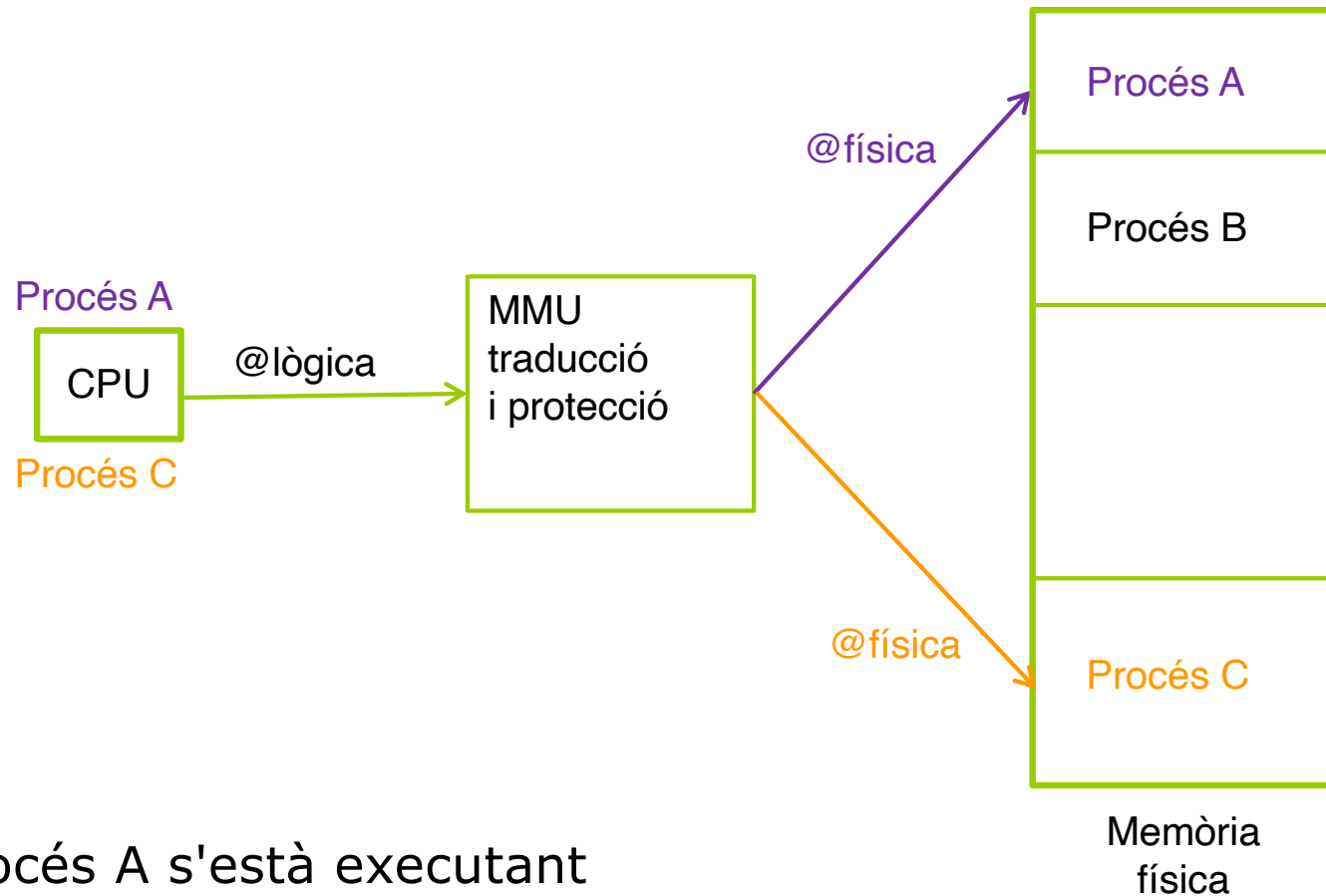
Espai d' @ d'un procés

- Espai d' adreces del procesador
 - Conjunt d' @ que el processador pot emetre, (depèn del bus d'adreces)
- Espai de direccions lògiques d'un procés
 - Conjunt d' @ lògiques que un procés pot referenciar (que el kernel decideix que són vàlides per a aquest procés)
- Espai d'adreces físiques d'un procés
 - Conjunt d' @ físiques associades a l'espai d'adreces lògiques del procés (decidit també pel kernel)
- Correspondència entre @ lògiques i @ físiques
 - Fixa: Espai de @ lògiques == Espai de @ físiques
 - Traducció:
 - ▶ En carregar el programa en memòria: el kernel decideix on posar el procés i es tradueixen les adreces en copiar-les a memòria
 - ▶ En executar: es tradueix cada direcció que es genera
 - **Col·laboració entre HW i SO**
 - » HW ofereix el mecanisme de traducció
 - » Memory Management Unit (MMU)
 - » **El Kernel el configura**

Sistemes multiprogramats

- Sistemes multiprogramats
 - Diversos programes carregats en memòria física simultàniament
 - Facilita l'execució concurrent i simplifiquen el canvi de context
 - ▶ **1 procés a la CPU però N processos en memòria física**
 - ▶ En fer canvi de context no cal carregar de nou en memòria el procés que ocupa la cpu
 - SO ha de garantir protecció de la memòria física
 - ▶ Cada procés només ha d'accedir a la memòria física que té assignada
 - ▶ Col·laboració entre SO i HW
 - MMU ofereix el mecanisme per detectar accessos il·legals
 - SO configura la MMU
 - El kernel ha de modificar la MMU per reflectir qualsevol canvi:
 - ▶ En fer canvi de context el SO ha d'actualitzar la MMU amb la informació del nou procés
 - ▶ Si s'augmenta l'espai d'adreces
 - ▶ etc.

Sistemes multiprogramats



- 1-Procés A s'està executant
- 2-Canvi de context a C

Assignació d' @ a un programa

- **Hi ha altres alternatives però... sistemes actuals l' assignació d' @ a instruccions i dades es realitza en temps d' execució**
 - @ físiques != @ lògiques → requereix traducció en temps d'execució
 - Processos poden canviar de posició en memòria sense modificar el seu espai lògic d'@
 - ▶ Exemple: Paginació (Vist en EC)

Suport HW: MMU

- MMU(Memory Management Unit). Component HW que ofereix la traducció d' adreces i la protecció de l'accés a memòria. Com a mínim ofereix **suport a la traducció i a la protecció** però pot ser necessari per a altres tasques de gestió
- **SO és el responsable de configurar la MMU amb els valors de la traducció d' adreces corresponents al procés en execució**
 - Quines @ lògiques són vàlides i amb quines @ físiques es corresponen
 - Assegura que cada procés només té associades les seves @ físiques
- **Suport HW a la traducció i a la protecció entre processos**
 - MMU rep @ lògica i fa servir les seves estructures de dades per traduir-la a l'@ física corresponent
 - ▶ Si l'@ lògica no està marcada com a vàlida o no té una @ física associada genera una excepció per avisar el SO
- **SO gestiona l'excepció en funció del cas**
 - Per exemple, si l'@ lògica no és vàlida pot eliminar el procés (SIGSEGV)

Suport HW: Traducció

- Quan el SO ha de modificar la traducció d' adreces???
- En assignar memòria
 - Inicialització en **assignar nova memòria**. (en la mutació, execlp)
 - **Canvis en l'espai d'adreces**: augmenta/disminueix. En demanar/alliberar memòria dinàmica.
- En el canvi de context
 - Per al procés que abandona la CPU: si encara no ha acabat l'execució, salvar a les estructures de dades del procés (PCB) la informació necessària per reconfigurar la MMU quan torni a ocupar la CPU
 - Per al procés que passa a ocupar la CPU: configurar la MMU

Suport HW : Protecció

- Es realitza en els mateixos casos que l'assignació
- També permet implementar protecció contra accessos/tipus d'accessos no desitjats
 - Adreces lògiques invàlides
 - Adreces lògiques vàlides amb accés incorrecte (escriure en zona de lectura)
 - Adreces lògiques vàlides i accés "incorrecte" però que el SO ha marcat com a incorrecte per implementar alguna optimització
 - ▶ Per exemple COW que veurem més endavant
 - En qualsevol cas → excepció capturada per la CPU i gestió per part del SO
 - El kernel sempre té la informació correcta sobre l'espai d'adreces, per la qual cosa pot comprovar si és realment una fallada o no

Tasques del SO en la gestió de memòria

- Càrrega de programes en memòria
- Reservar/Alliberar memòria dinàmicament (mitjançant crides a sistema)
- Oferir compartició de memòria entre processos
 - Amb COW hi haurà compartició de forma transparent als processos en zones de només lectura
 - Hi ha compartició explícita de memòria (mitjançant crides a sistema) però no el treballarem aquest curs
- Serveis per a l'optimització de l'ús de memòria
 - COW
 - Memòria virtual
 - Prefetch



Càrrega d'un programa

Memòria dinàmica

Assignació de memòria

Compartició de memòria entre processos

SERVEIS BÀSICS DEL SO

Serveis bàsics: càrrega de programes

- L'executable ha d'estar a memòria per ser executat, però els executables estan a "disc"
- SO ha de
 1. Llegir i Interpretar l'executable (els executables tenen un format)
 2. Preparar l' esquema del procés en memòria lògica i assignar memòria física
 1. **Inicialitzar estructures de dades** del procés
 1. Descripció de l'espai lògic
 1. Quines @ lògiques són vàlides
 2. Quin tipus d'accés és vàlid
 2. Informació necessària per configurar la MMU cada vegada que el procés passa a ocupar la CPU
 2. **Inicialitzar MMU**
 3. **Llegir seccions del programa** del disc i escriure memòria
 4. Carregar registre **program counter** amb l'adreça de la instrucció definida a l'executable com a **punt d'entrada**

Serveis bàsics: càrrega de programes

- Optimitzacions aplicades a la càrrega de programes
 - Càrrega sota demanda
 - Llibreries compartides i enllaç dinàmic
- A Linux es provoca quan un procés muta (**exec**)

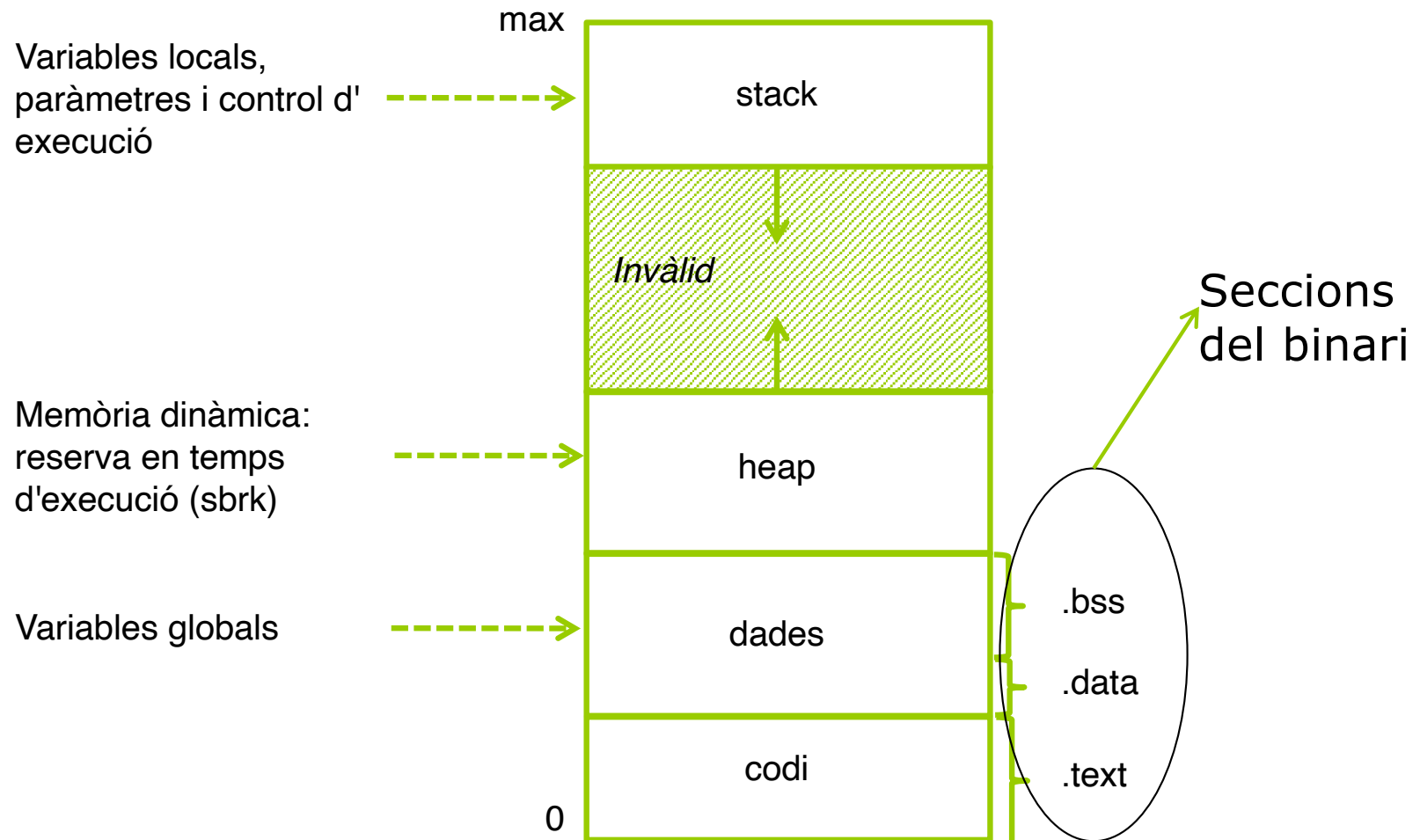
Càrrega: format de l' executable

- PAS 1: Interpretar el format de l' executable en disc
 - Si la traducció es fa en temps d'execució...¿quin tipus d'adreces contenen els binaris? Lògiques o Físiques?
- Capçalera de l'executable defineix les seccions: tipus, mida i posició dins del binari (podeu provar `objdump -h programa`)
- Existeixen diferents formats d' executable
 - ELF (*Executable and Linkable Format*): és el més estès en sistemes POSIX

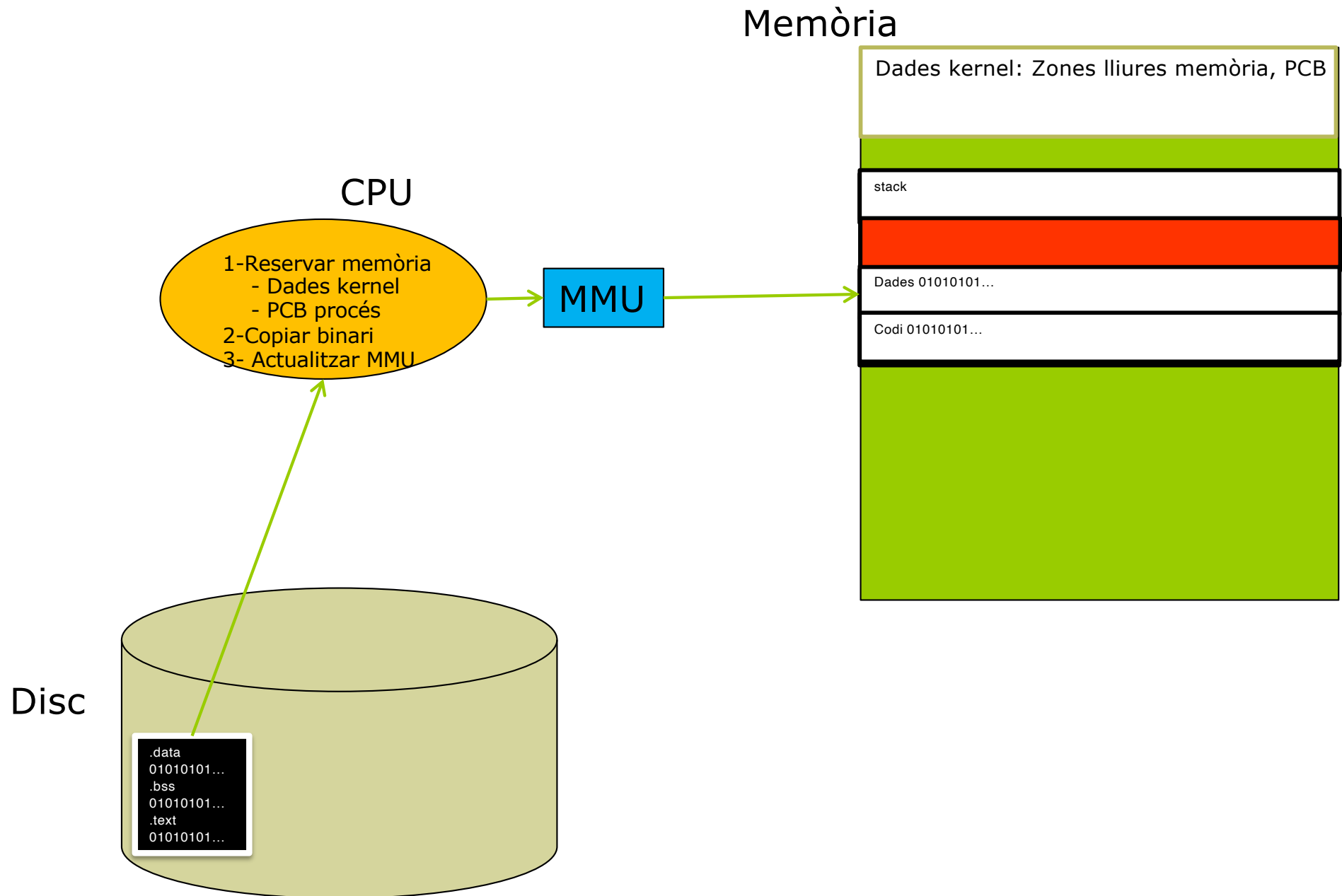
Algunes seccions per defecte d' un executable ELF	
.text	codi
.data	Dades globals inicialitzades
.bss	dades globals sense valor inicial
.debug	informació de debug
.comment	informació de control
.dynamic	informació per a enllaç dinàmic
.init	codi d'inicialització del procés (conté la @ de la 1a instrucció)

Càrrega: Esquema del procés en memòria

- PAS 2: Preparar l' esquema del procés en memòria lògica
 - Esquema habitual



Càrrega executables



Càrrega: Optimització càrrega sota demanda

- Optimitzacions: càrrega sota demanda
 - **Una rutina no es carrega fins que es crida**
 - S'aprofita millor la memòria ja que no es carreguen funcions que no es criden mai (per exemple, rutines de gestió d'errors)
 - S'accelera el procés de càrrega (tot i que es pot notar durant l'execució)
 - Cal un mecanisme que detecti si les rutines no estan carregades. Per exemple:
 - ▶ SO:
 - Registra en les seves estructures de dades que aquesta zona de memòria és vàlida i d'on llegir el seu contingut
 - A la MMU no li associa una traducció
 - ▶ Quan el procés accedeix a l'@, la MMU genera una excepció per avisar el SO d'un accés a una @ que no sap traduir
 - ▶ SO comprova en les seves estructures que l'accés és vàlid, provoca la càrrega i reprèn l'execució de la instrucció que ha provocat l'excepció

Càrrega: Optimització de llibreries compartides

- Els binaris (en disc) no contenen el codi de les llibreries dinàmiques, només un enllaç
→ Estalvia molt espai en disc
 - ▶ **Es retarda l'enllaç fins al moment d'execució**
 - ▶ Penseu en quants programes utilitzen la libC, com a espai necessitem si cadascú té una còpia (idèntica) de la llibreria
- Els processos (en memòria) poden compartir la zona en memòria que conté el codi (que és només de lectura) de les llibreries comunes → Estalvia espai en memòria
- Facilita l'actualització dels programes perquè usin les noves versions de les llibreries de sistema
 - No cal recompilar, en executar el programa s'enllaçarà amb la nova versió
- Mecanisme
 - Binari conté el codi d'una rutina d'enllaç (stub), és un tipus de rutina que fa de pont a la que conté el codi realment
 - ▶ Comprova si algun procés ja ha carregat la rutina de la llibreria compartida i la càrrega si no és així
 - ▶ Substitueix la crida a si mateixa per la crida a la rutina de la llibreria compartida

Servei: Reservar/Alliberar memòria dinàmica

- Hi ha variables la mida de les quals depèn de paràmetres de l'execució
 - Fixar la mida en temps de compilació no és adequat
 - ▶ O es desaprofita memòria o es té error d'execució per no haver reservat suficient
- Els SO ofereixen trucades a sistema per reservar noves regions de memòria en temps d'execució: memòria dinàmica
 - S'emmagatzema a la **zona heap** de l'espai lògic de @
- Implementació
 - Pot retardar el moment d'assignar @ físiques fins que s'intenti escriure a la regió
 - ▶ S'assigna temporalment una zona inicialitzada amb 0 per resoldre lectures. L'interfície pot definir que la regió està inicialitzada amb 0 o no
 - Actualitza la MMU en funció de la política d'assignació de memòria que es segueixi

Servei: Reservar/Alliberar memòria dinàmica

- Linux sobre Pentium
 - Interfície tradicional d'Unix poc amigable
 - ▶ `brk` y `sbrk` (usarem aquesta)
 - ▶ Permeten modificar el límit del heap . El SO no té consciència què variables hi ha ubicades en què zones, simplement augmenta o redueix la mida del heap
 - ▶ Programador és responsable de controlar posició de cada variable al heap → La gestió és complexa

```
int sbrk(int mida_variacio_heap);
```

- `>0` augmenta el heap
- `<0` redueix el heap
- `==0` no es modifica

Sbrk:exemple

```
int main(int argc, char *argv[])
{
    int num_procs = atoi(argv[1]);
    int *pids;
    pids = sbrk(num_procs * sizeof(int));
    for(i=0; i<10; i++){
        pids[i] = fork();
        if (pids[i] == 0){
            ....
        }
    }
    sbrk(-1 * num_procs * sizeof(int));
}
```

max

PILA (STACK)

HEAP

DADES

CODI

0

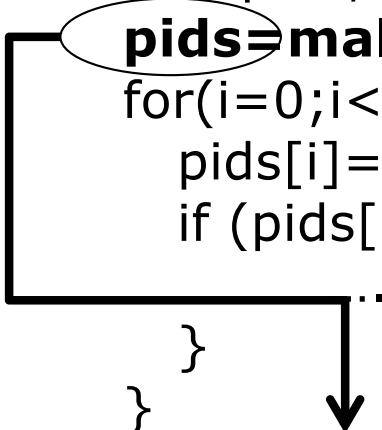
Senzill si tenim una variable, que passaria si en tenim diverses i volem "alliberar" una del mig del heap??
NO ES POT!

Servei: Reservar/Alliberar memòria dinàmica

- La llibreria de C afegeix la gestió que vincula les direccions amb les variables.
 - És una gestió transparent al kernel
- **Llibreria de C.** Demanar memòria: `malloc(tamaño_en_bytes)`
 - ▶ Si hi ha espai **consecutiu** suficient, el marca com a reservat i retorna la direcció d' inici
 - ▶ Si no hi ha espai **consecutiu** suficient, augmenta la mida del heap
 - ▶ La llibreria de C gestiona el heap, sap que zones estan lliures i que zones usades. Intentar satisfer peticions sense recórrer al sistema
 - En augmentar el heap, es reserva més del necessari amb l'objectiu de reduir el nombre de crides a sistema i estalviar temps. La propera petició de l'usuari trobarà espai lliure
- **Librería de C.** alliberar memòria: `free(zona_a_alliberar)`
 - ▶ Quan el programador allibera una zona es decideix si simplement passa a formar part de la llista de zones lliures o si és adequat reduir la mida del heap
 - ▶ La llibreria ja sap que tenia la zona ja que se suposa que correspon amb una zona demanada anteriorment amb `malloc`

Com seria amb malloc/free

```
int main(int argc, char *argv[])
{
    int num_procs=atoi(argv[1]);
    int *pids;
    pids=malloc(num_procs*sizeof(int));
    for(i=0;i<10;i++){
        pids[i]=fork();
        if (pids[i]==0){
            ..
        }
    }
    free(pids);
```



A l'hora de demanar és igual, però en alliberar hem de passar un punter concret, no una mida

Memòria dinàmica(IV): exemples

- Quines diferències a nivell de heap observeu en els següents exemples?

- Exemple 1:

```
...  
new = sbrk(1000);  
...
```



- Exemple 2:

```
...  
new = malloc(1000);  
...
```



- Canvia la mida del heap en els dos casos?

Memòria dinàmica (V): exemples

- Quines diferències a nivell de heap observeu en els següents exemples?

- Exemple 1:

```
...  
ptr = malloc(1000);  
...
```



- Exemple 2:

```
...  
for (i = 0; i < 10; i++)  
    ptr[i] = malloc(100);  
...
```



- Es reserven les mateixes posicions de memòria lògica?
 - Exemple1: necessitem 1000 bytes consecutius
 - Exemple2: Necessitem 10 regions de 100 bytes

Memòria dinàmica (VI): exemples

- Quins errors contenen els següents fragments de codi?

- Codi 1:

```
...  
for (i = 0; i < 10; i++)  
    ptr = malloc(SIZE);  
  
// uso de la memoria  
// ...  
  
for (i = 0; i < 10; i++)  
    free(ptr);  
...
```



- Codi 2:

```
int *x, *ptr;  
  
...  
ptr = malloc(SIZE);  
...  
x = ptr;  
...  
free(ptr);  
  
sprintf(buffer, "...%d",  
*x);
```



- Codi 1: Que passarà en la segona iteració del segon bucle?
- Codi 2: ¿Produeix error sempre l'accés a "*x"?

Serveis bàsics: assignació de memòria

- S'executa cada vegada que un procés necessita memòria física:
 - En linux: creació (fork), mutació de l'executable (exec)==càrrega, ús de memòria dinàmica, implementació d'alguna optimització (càrrega sota demanda, memòria virtual, COW...).
- Passos
 - **Seleccionar memòria física lliure** i marcar-la com a ocupada en les estructures de dades del SO
 - Actualitzar MMU amb el mapatge @ lògiques → @ físiques
 - ▶ Necessari per implementar la **traducció d'adreces**
- Quan tenim un problema d'assignar una quantitat X (en aquest cas memòria) en una zona més gran, depenent de la solució apareixen problemes de **FRAGMENTACIÓ**
 - També apareix en la gestió del disc

Assignació: Problema fragmentació

- Fragmentació de memòria: memòria que està lliure però no es pot fer servir per a un procés
 - **Fragmentació interna:** memòria assignada a un procés encara que no la necessita. Està reservada però no ocupada.
 - **Fragmentació externa:** memòria lliure i no assignada però no es pot assignar per no estar contiguous. No està reservada però no serveix.
 - ▶ Es pot evitar compactant la memòria lliure si el sistema implementa assignació de @ en temps d'execució
 - Costós en temps

Serveis bàsics: assignació de memòria

- Primera aproximació: **assignació contigua**
 - Espai de @ físiques contigu
 - ▶ Tot el procés ocupa una partició que se selecciona en el moment de la càrrega
 - Poc flexible i dificulta aplicar optimitzacions (com càrrega sota demanda)
- **Assignació no contigua**
 - Espai de @ físiques no contigu
 - Augmenta flexibilitat
 - Augmenta la granularitat de la gestió de memòria d'un procés
 - Augmenta complexitat del SO i de la MMU
- Basada en
 - **Paginació** (particions fixes)
 - **Segmentació** (particions variables)
 - Esquemes combinats
 - ▶ Per exemple, segmentació paginada

Vist en EC

Assignació: Paginació

■ Esquema basat en paginació

- Espai de @ lògiques dividit en particions de mida fixa: pàgines
- Memòria física dividida en particions de la mateixa mida: marcs
- Assignació
 - ▶ Per a cada pàgina del procés buscar un marc lliure
 - Llista de marcs lliures
 - ▶ Hi pot haver fragmentació interna
- Quan un procés acaba l'execució retornar els marcs assignats a la llista de lliures
- **Pàgina: unitat de treball del SO**
 - ▶ Facilita la càrrega sota demanda
 - ▶ Permet especificar protecció a nivell de pàgina
 - ▶ Facilita la compartició de memòria entre processos
 - ▶ Normalment, per temes de permisos, una pàgina pertany a una regió de memòria (codi/dades/heap/pila)

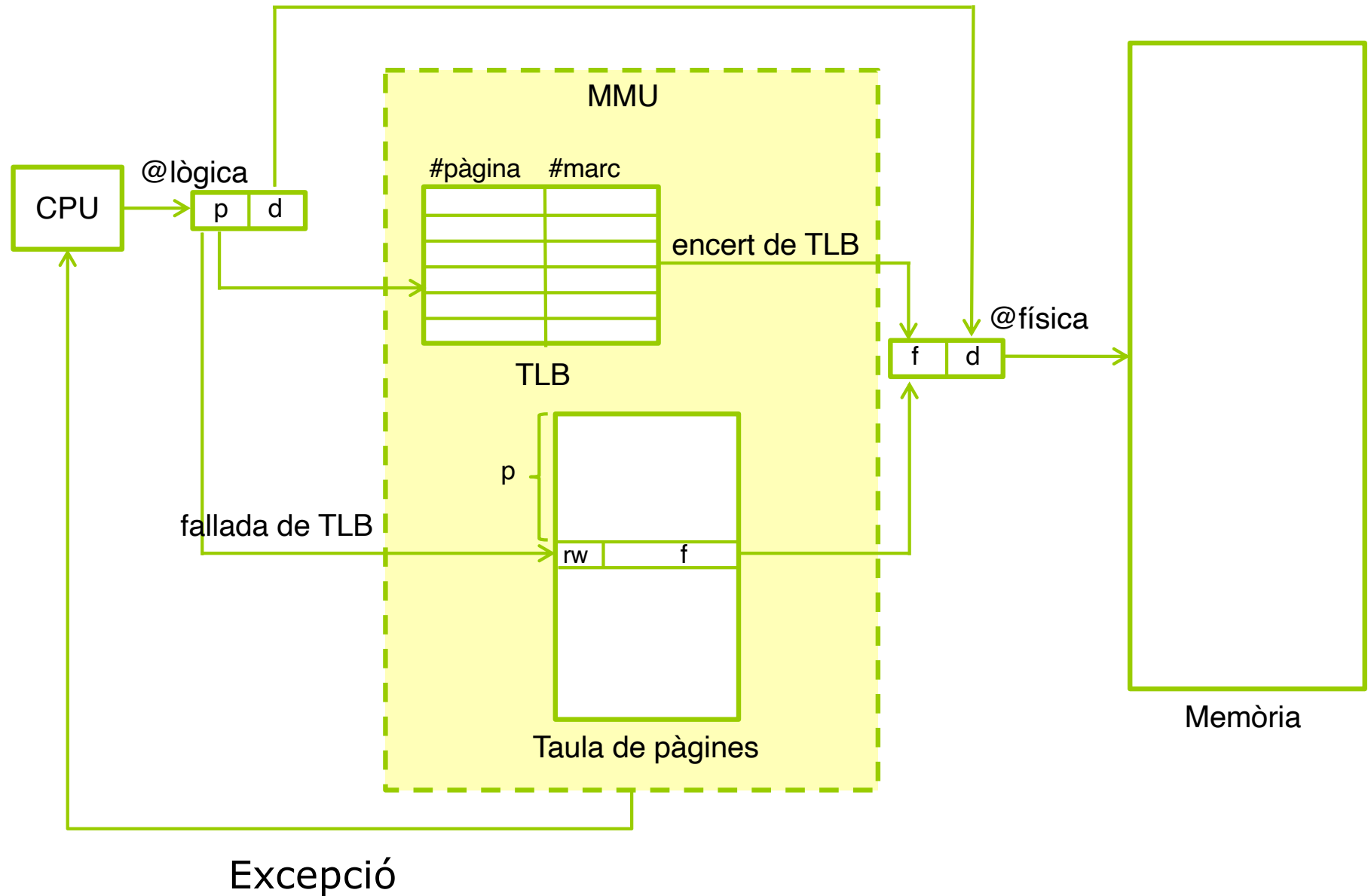
Assignació: Paginació

■ MMU

● Taula de pàgines

- ▶ Per mantenir informació a nivell de pàgina: validesa, permisos d' accés, marc associat, etc....
 - ▶ Una entrada per a cada pàgina
 - ▶ Una taula per procés
- Sol guardar-se en memòria i SO ha de conèixer l'@ base de la taula de cada procés (per exemple, guardant-la al PCB)
 - Processadors actuals també disposen de TLB (Translation Lookaside Buffer)
 - ▶ Memòria associativa (cache) d'accés més ràpid en la qual s'emmagatzema la informació de traducció per a les pàgines actives
 - ▶ Cal actualitzar/invalidar la TLB quan hi ha un canvi a la MMU
 - Gestió HW del TLB/Gestió Programari (SO) del TLB
 - Molt dependent de l'arquitectura

Assignació: Paginació



Assignació: Paginació

- PROBLEMA: Mida de les taules de pàgina (que estan guardades en memòria)
- Mida de pàgina potència de 2
 - Mida molt usat 4Kb (2^{12})
 - Influeix en
 - Fragmentació interna i granularitat de gestió
 - Mida de la taula de pàgines
- Esquemes per reduir l' espai ocupat per les TP: TP multinivel
 - TP dividida en seccions i s'afegeixen seccions a mesura que creix l'espai lògic d'adreces

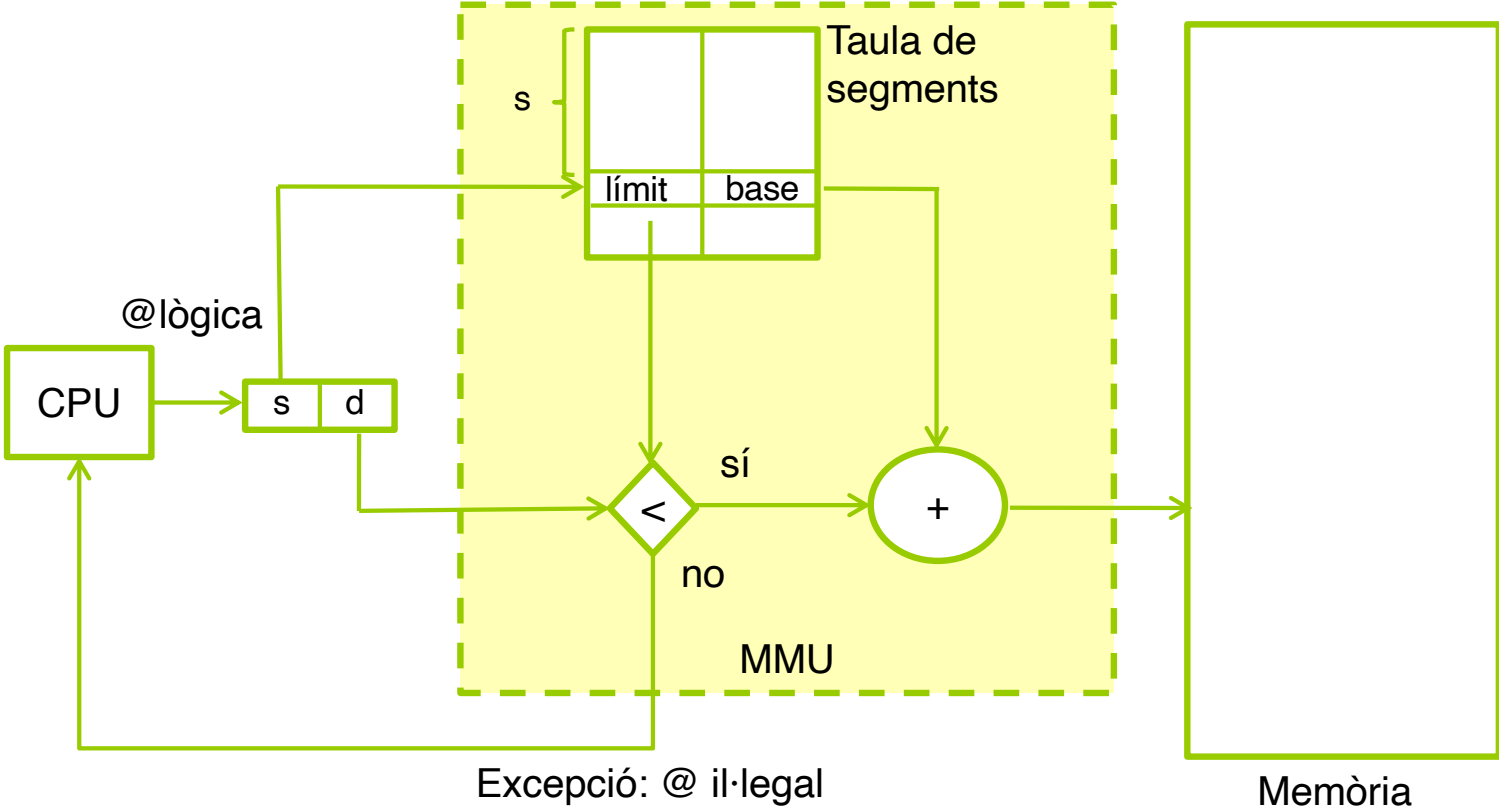
	Espai lògic de processador	Nombre de pàgines	Mida TP
Bus de 32 bits	2^{32}	2^{20}	4MB
Bus de 64 bits	2^{64}	2^{52}	4PB

Assignació: Segmentació

■ Esquema basado en segmentación

- Es divideix l'espai lògic del procés tenint en compte el tipus de contingut (codi, dates., etc)
 - ▶ Aproxima la gestió de memòria a la visió d'usuari
- Espai d'@ lògiques dividit en particions de mida variable(**segments**), **ajustat al que es necessita**
 - ▶ Com a mínim un segment per al codi i un altre per a la pila i les dades
 - ▶ Les referències a memòria que fa el programa estan formades per un segment i el desplaçament dins del segment
- Memòria física lliure contiguous forma una partició disponible
- Assignació: Per a cada segment del procés
 - Busca una partició en la qual hi càpiga el segment
 - Possibles polítiques: first fit, best fit, worst fit
 - Selecciona la quantitat de memòria necessària per al segment i la resta continua en la llista de particions lliures
 - ▶ Hi pot haver fragmentació externa
 - ▶ No tots els "trossos" lliures són igual de bons.

- ▶ Per a cada segment: @ base i mida
- ▶ Una taula per procés



Assignació: Esquema mixt

■ Esquemes combinats: segmentació paginada



- Espai lògic del procés dividit en segments
- Segments dividits en pàgines
 - ▶ Mida de segment múltiple de la mida de pàgina
 - ▶ Unitat de treball del SO és la pàgina

Serveis bàsics: compartició

- Compartició de memòria entre processos
 - Es pot especificar a nivell de pàgina o de segment
 - Per a processos que executen el mateix codi no és necessari diverses còpies en memòria física (accés de lectura)
 - ▶ **Llibreries compartides (implícit)**
 - Els SO proporcionen crides a sistema perquè un procés creï zones de memòria en el seu espai lògic que siguin compartibles i perquè un altre procés la pugui mapejar en el seu espai de memòria
 - ▶ **Memòria compartida** com a mecanisme de **comunicació entre processos (explícit)**
 - La resta de memòria és **privada** per a un procés i ningú la pot accedir



COW

Memòria virtual

Prefetch

SERVEIS PER A L' OPTIMITZACIÓ DE L'ÚS DE MEMÒRIA

Optimitzacions: COW (Copy on Write)

- Objectiu: reduir la reserva/inicialització de memòria física fins que sigui necessari
 - Si no s'accedeix a una zona nova → no necessitem reservar-la realment
 - Si no modifiquem una zona que és una còpia → no necessitem duplicar-la
 - Estalvia temps i espai de memòria
- En el fork:
 - **Endarrerir el moment de la còpia de codi, dades, etc mentre només s'accedeixi en mode lectura**
 - Es pot evitar la còpia física si els processos només han de fer servir la regió per llegir, per exemple el codi
 - Se sol gestionar a nivell de pàgina lògica: es van reservant/copiant pàgines a mesura que es necessita
- Es pot aplicar
 - Dins d'un procés: en demanar memòria dinàmica
 - Entre processos (per exemple, fork de Linux)
 - En general sempre que s'augmenta/modifica l'espai de direccions.

COW: Implementació

- **La idea és: el kernel assumeix que es podrà estalviar la reserva de la memòria física, però necessita un mecanisme per detectar que no és així i realitzar la reserva si realment SÍ era necessària**
- En el moment que caldria fer l'assignació:
 - En l'estructura de dades que descriu l'espai lògic del procés (al PCB) el SO marca la regió destinació amb els permisos d'accés reals
 - A la MMU el SO marca la regió destinació i la regió font amb permís només de lectura
 - A la MMU el SO associa a la regió les direccions físiques associades:
 - ▶ A les regions del pare si era un fork (mateixa traducció, memòria compartida)
 - ▶ A una pàgines que actuen de comodí en el cas de memòria dinàmica
- Si un procés intenta escriure a la zona nova, la MMU genera excepció i SO la gestiona fent la reserva real i reiniciant l'accés

COW: exemple

- Procés A ocupa:
 - Codi: 3 pàgines, Dades 2 pàgines, Pila: 1 pàgina, Heap: 1 pàgina
- Si procés A ejecuta fork, just després del fork:
 - Total memòria física:
 - ▶ Sense COW: procés A = 7 pàgines + fill A = 7 pàgines = 14 pàgines
 - ▶ Amb COW: procés A = 7 pàgines + fill A = 0 pàgines = 7 pàgines
- Al cap d'una estona... depèn del que facin els processos, per exemple:
 - Si fill A muta (i el nou espai del fill ocupa 10 pàgines):
 - ▶ Sense COW: procés A = 7 pàgines + fill A = 10 pàgines = 17 pàgines
 - ▶ Amb COW: procés A = 7 pàgines + fill A = 10 pàgines = 17 pàgines
 - Si fill A no muta, depèn del que faci, però el codi almenys pot ser compartit, suposant que la resta no ho sigui:
 - ▶ Sense COW: procés A = 7 pàgines + fill A = 7 pàgines = 14 pàgines
 - ▶ Amb COW: procés A = 7 pàgines + fill A = 4 pàgines = 11 pàgines
- **En qualsevol cas cal veure quines pàgines es modifiquen (per tant no es poden compartir) i quines pàgines sí es poden compartir**

Optimitzacions: Memòria Virtual(I)

■ Memòria virtual

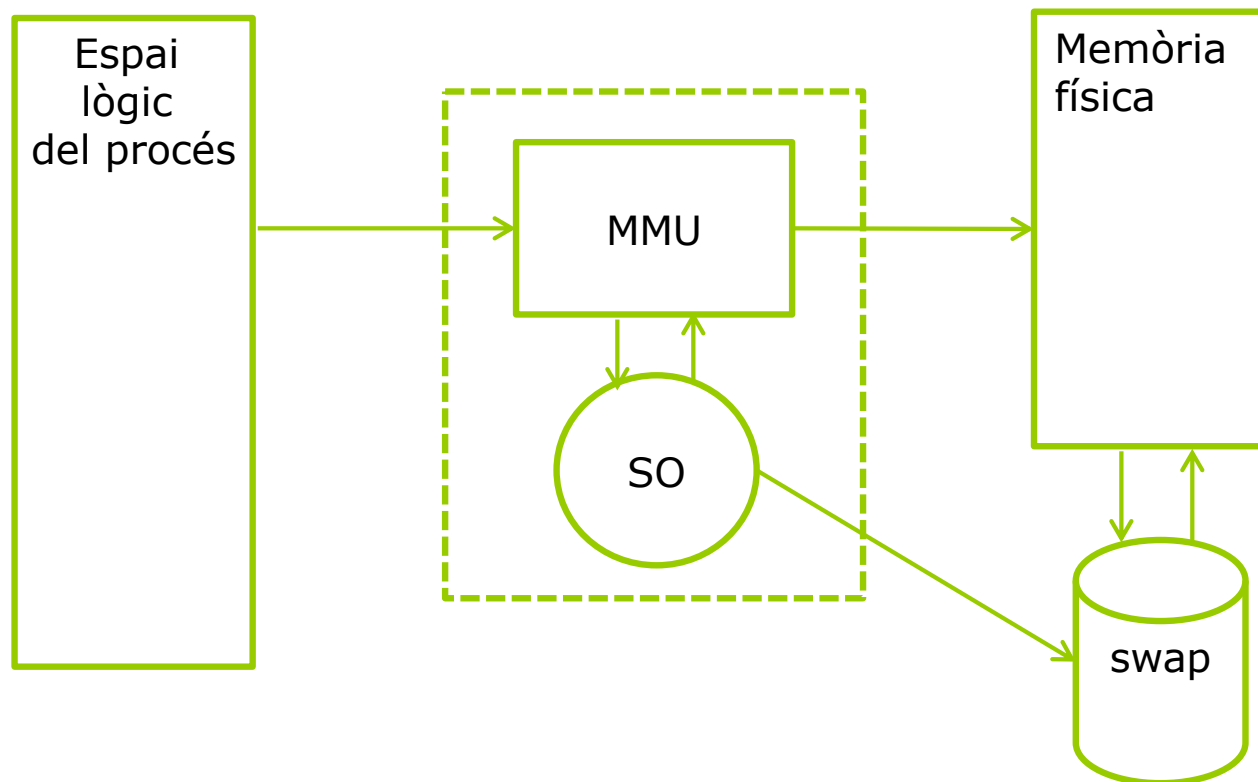
- Estén la idea de la càrrega sota demanda
- A més de "portar" coses a memòria sota demanda, permet "treure" coses sota demanda
- Objectiu
 - ▶ Reduir la quantitat de memòria física assignada a un procés en execució
 - **Un procés realment només necessita memòria física per a la instrucció actual i les dades que aquesta instrucció referencia**
 - ▶ Augmentar el grau de multiprogramació
 - Quantitat de processos en execució simultàniament

Optimitzacions: Memòria Virtual(II)

- Primera aproximació: intercanvi(*swapping*)
 - Idea: només cal tenir en memòria el procés actiu (el que tenia la CPU assignada)
 - ▶ Si el procés actiu necessitava més memòria física que la disponible en el sistema es pot expulsar temporalment de memòria algun dels altres processos carregats (swap out)
 - ▶ Magatzem secundari o de suport (backing storage):
 - Dispositiu d'emmagatzematge en el qual es guarda l'espai lògic dels processos a l'espera de tornar a ocupar la CPU
 - Més capacitat que la que ofereix la memòria física
 - Típicament una zona de disc: espai d'intercanvi (Swap area)
 - ▶ Estat dels processos: no residents (swapped out)
 - ▶ En assignar la cpu a un procés no resident cal carregar-lo en memòria de nou abans de permetre que reprengui l'execució
 - Ralentitza l'execució
 - Evolució de la idea
 - ▶ Evitar expulsar de memòria processos sencers per minimitzar la penalització en temps de l'execució
 - ▶ Es pot aprofitar la granularitat que ofereix la paginació

Optimitzacions: Memòria Virtual(III)

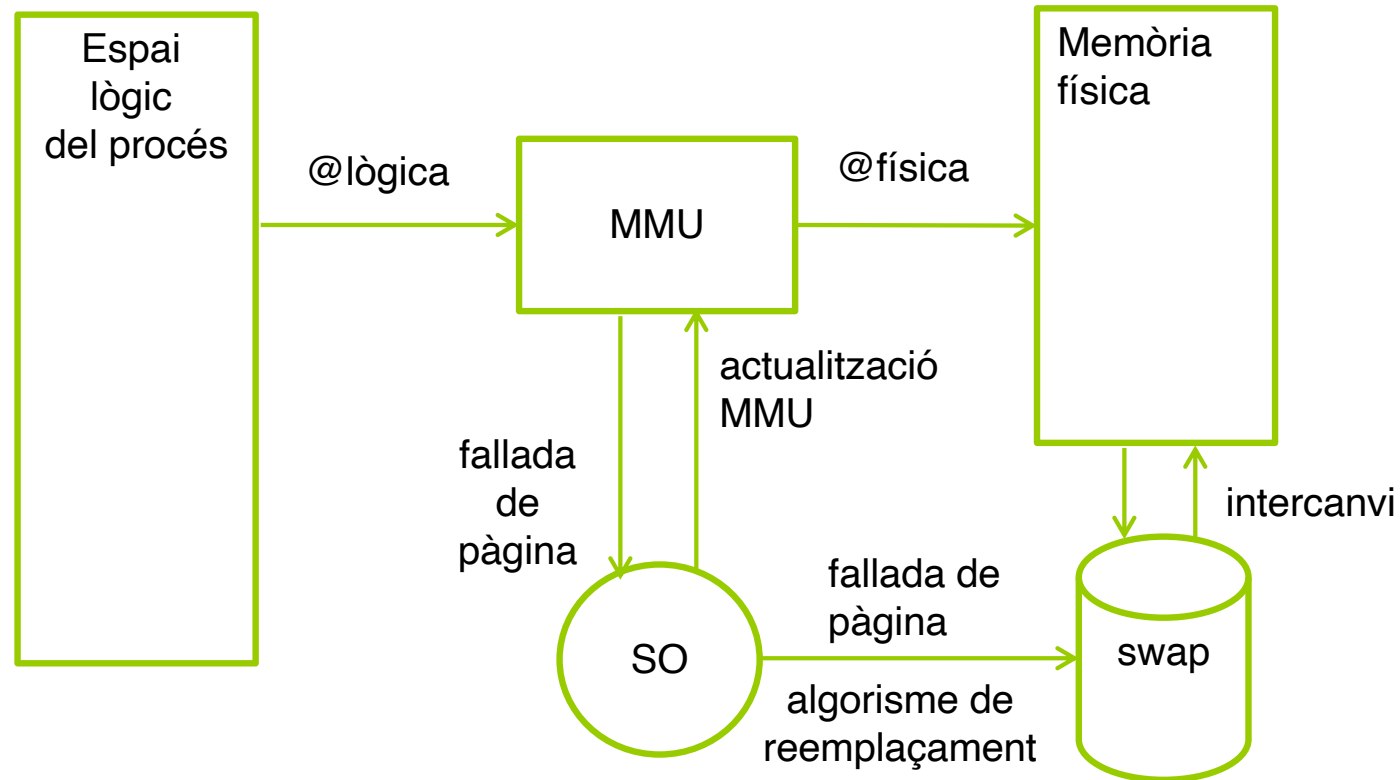
- Memòria virtual basada en paginació
 - **Espai lògic d'un procés està distribuït entre memòria física (pàgines residents) i àrea de swap (pàgines no residents)**



Optimitzacions: Memòria Virtual(IV)

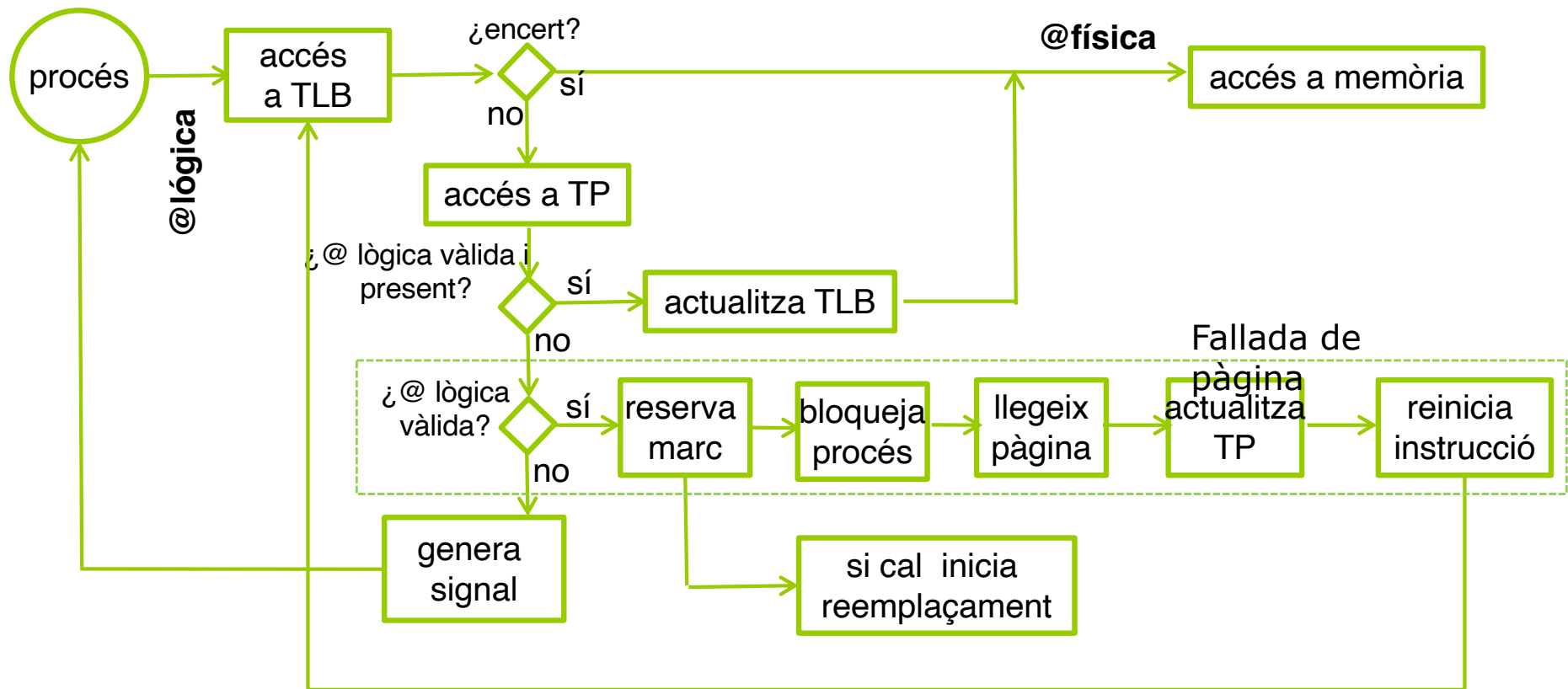
- **Reemplaçament de memòria: quan SO necessita alliberar marcs**
 - Selecciona una pàgina víctima i actualitza la MMU eliminant la seva traducció
 - Guarda el seu contingut a l'àrea de PSARU perquè es pugui recuperar
 - Assigna el marc ocupat a la pàgina que es necessita en memòria
- Quan s' accedeix a una pàgina guardada a l' àrea de swap
 - MMU no pot fer la traducció: genera excepció
 - ▶ **Fallada de pàgina**
 - SO
 - ▶ Comprova en les estructures del procés que l'accés és vàlid
 - ▶ Assigna un marc lliure per a la pàgina (llança el reemplaçament de memòria si cal)
 - ▶ Localitza a l'àrea de swap el contingut i l'escriu en el marc
 - ▶ Actualitza la MMU amb l'@ física assignada

Optimitzacions: Memòria Virtual (V)



Optimitzacions: Memòria Virtual(VI)

■ Passos en l' accés a memòria



Optimitzacions: Memòria Virtual(VII)

- Efectes de l'ús de la memòria virtual
 - La suma dels espais lògics dels processos en execució pot ser més gran que la quantitat de memòria física de la màquina
 - L'espai lògic d'un procés també pot ser més gran que la memòria física disponible
 - Accedir a una pàgina no resident és més lent que accedir a una pàgina resident
 - ▶ Excepció + càrrega de la pàgina
 - ▶ Important minimitzar el nombre de fallades de pàgina

Optimitzacions: Memòria Virtual(VIII)

- Modificacions al SO
 - Afegir les estructures de dades i els algoritmes per gestionar l'àrea de swap
 - ▶ Assignació, alliberament i accés
 - **Algorisme de reemplaçament**
 - ▶ Quan s'executa? Com se seleccionen les pàgines víctimes? Quantes pàgines víctimes en cada execució de l'algorisme?
 - ▶ Objectiu: minimitzar el nombre de fallades de pàgina i accelerar la seva gestió
 - **Intentar seleccionar les víctimes entre les pàgines que ja no es necessiten o que es trigarà més temps a necessitar**
 - » Exemple: Least Recently Used (LRU) o aproximacions
 - Intentar que sempre que es doni una fallada de pàgina hi hagi un marc disponible
- Modificacions en la MMU: depèn dels algorismes de gestió de memòria virtual.
 - Per exemple, algorisme de reemplaçament pot necessitar un bit de referència per pàgina

Optimitzacions: Memòria Virtual(IX)

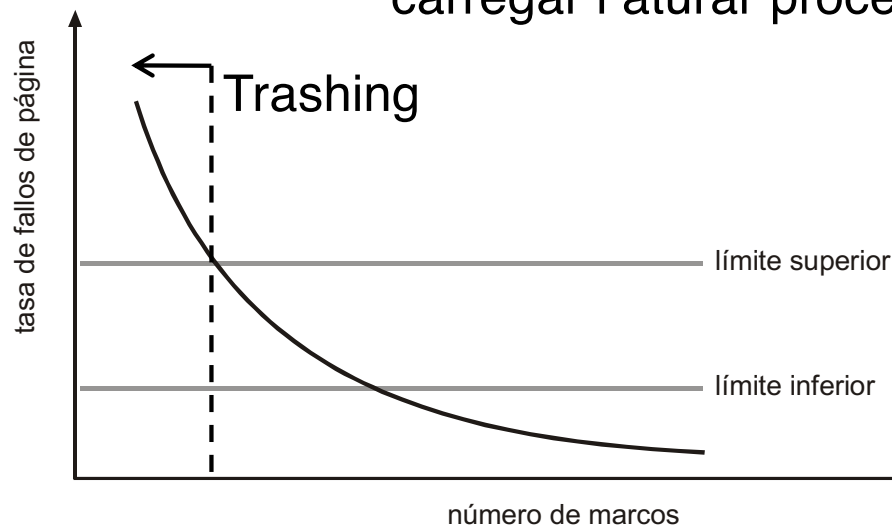
■ Sobrepaginació (*thrashing*)

● Procés enthrashing

- ▶ **Inverteix més temps en l'intercanvi de memòria que avançant la seva execució**
- ▶ No aconsegueix mantenir simultàniament en memòria el conjunt mínim de pàgines que necessita per avançar

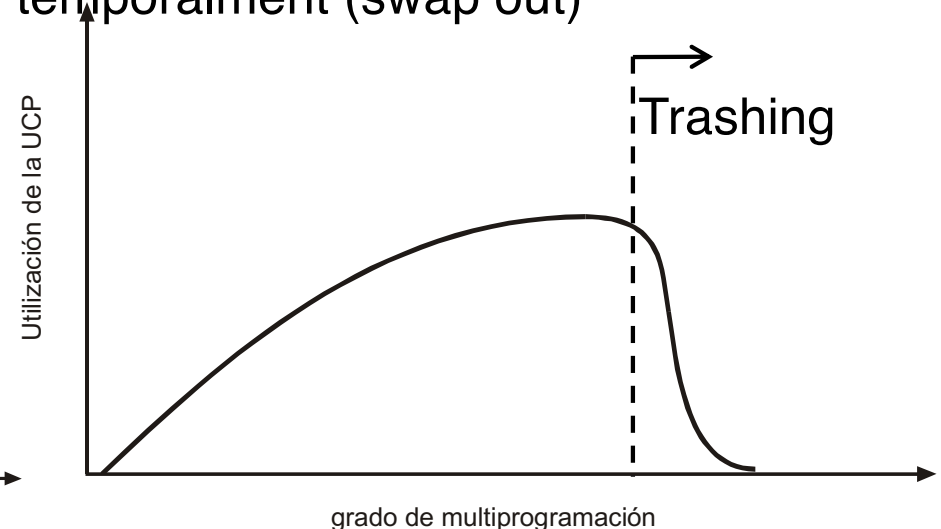
● Es deu al fet que s'ha sobrecarregat el sistema de memòria

- ▶ Detecció: controlar taxa de fallades de pàgina per procés
- ▶ Tractament: controlar el nombre de processos que es permeten carregar i aturar processos temporalment (swap out)



número de marcos

3.53



grado de multiprogramación

Optimitzacions: Memòriaprefetch

- Objectiu: minimitzar nombre de fallades de pàgina
- Idea: anticipar quines pàgines necessitarà el procés en el futur immediat i carregar-les amb anticipació
- Paràmetres que cal tenir en compte:
 - Distància de prefetch: amb quina antelació cal carregar les pàgines
 - Nombre de pàgines a carregar
- Algorismes senzills de predicció de pàgines
 - Seqüencial
 - Strided

Resumen: Linux sobre Pentium

- Crida a sistema exec: provoca la càrrega d' un nou programa
 - Inicialització del PCB amb la descripció del nou espai d' adreces, assignació de memòria, ...
- Creació de processos(**fork**):
 - Inicialització del PCB amb la descripció del seu espai d'adreces (còpia del pare)
 - S'utilitza COW: fill comparteix marcs amb pare fins que algun procés els modifica
 - Creació i inicialització de la TP del nou procés
 - ▶ Es guarda al seu PCB la '@ base de la seva TP
- Planificació de processos
 - En el **canvi de context** s'actualitza a la MMU l'@ base de la TP actual i s'invalida la TLB
- Crida a Sistema **exit**:
 - Elimina la TP del procés i allibera els marcs que el procés tenia assignats (si ningú més els estava fent servir)

Resumen: Linux sobre Pentium

- Memòria virtual basada en segmentació paginada
 - Taula de pàgines multinivell (2 nivells)
 - ▶ Una per procés
 - ▶ Guardades en memòria
 - ▶ Registre de la cpu conté l'@ base de la TP del procés actual
 - Algorisme de reemplaçament: aproximació de LRU
 - ▶ S'executa cada cert temps i quan el nombre de marcs lliures és menor que un llindar
- Implementa COW a nivell de pàgina
- Càrrega sota demanda
- Suport per a llibreries compartides
- Prefetch simple (seqüencial)

Jerarquia d'emmagatzematge

