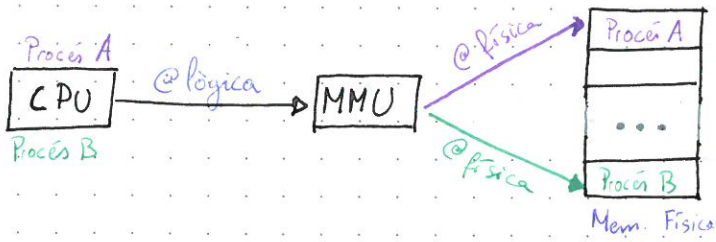
MEMÒRIA

Taula de Pàgines: Estructura de dades que guarda relació @lògica ↔ @física.
Permet que @lògiques iguals apuntin a @físiques diferents.
Cada procés té una taula de pàg. diferents.
La TLB és una optimització.

MMU: Chip gestionat pel SO que conté la @Base de la TP del procés actual.
Quan hi ha un canvi de context, el SO canvia el REG de la MMU.
Ofereix mecanismes per detectar accessos il·legals i traducció.



⚠ El contingut de la MMU també pot variar si:
- Afegeix / Elimina mem. dinàmica.
- Muta el programa

Protecció davant:

⚠⚠ NO guarda la TP.

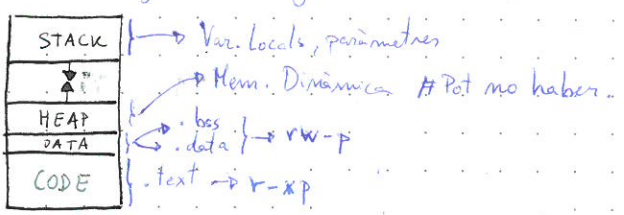
- @lògiques invàlides
- @lògiques correctes, però amés invàlid. // Escriptura quan lectura.
- @lògiques vàlides, però que SO ha marcat com invàlides. // COW

El Kernel sempre sap tota la info així que pot verificar que ha passat.

Recorda que @lògiques depenen dels bits del bus direccions.

Serveis del SO

1. Llegir i interpretar executable. ELF és el més comú en POSIX.
   .text = Codi **.data** = Global Var Inicialitzat **.bss** = Global Var Sense Inicialitzar.
   Les var en .bss no ocupen espai en l'executable resultant pq no hem guardar valor.
2. Preparar esquema del procés en mem. lògica i assignar mem. física.
   Inicialitzar la MMU.
3. Carregar seccions programa a mem.
4. Carregar Program Counter. ← @lògica.



Optimitzacions

Càrrega Sota Demanda

Rutina no es carrega fins que no es crida. #Evitar tindre funcions que no farem ús

1. SO Registra més @lògiques per cada rutina (No totes es carreguen en la RAM)
   A la MMU no li assoscia traducció.
2. Quan per primera vegada accedim MMU llança excepció que no sap traduir.
3. SO comprova que @ sigui vàlida, va a disc i escriu en RAM la funció.
4. Actualitza la MMU amb la nova relació.
5. Reexecuta la instrucció.

Llibreries Compartides

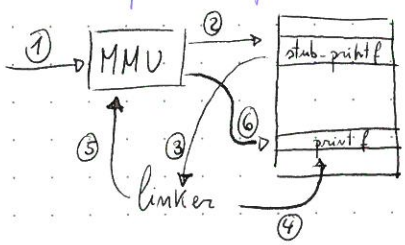
"Tindre només 1 còpia en disc i 1 còpia en RAM".

Hi ha 2 tipus de llibreries → Estàtiques: On "*.a" inclouen diversos "*.o" i només "linkem" aquells necessaris.

→ Dinàmiques: Binàri conté el codi d'una rutina enllaç (stub).
Que no és lo de compilar

Aquesta stub fa intermediari Codi ↔ Llibreria Dinàmica. Interactua amb procés "Linker" del SO.

Quan aplicació fa crida a una funció, realment ho fa al stub i aquesta gestiona si està en mem o no.



① Programa executa "printf()"

② MMU passa de @lògica → física

③ Executem stub_printf i aquesta gestiona amb el linker si està en memòria o no. (Procés del SO)

④ Si no està càrrega en RAM la funció "printf()"

⑤ Posteriorment modifiquem la MMU pq apunti a la que toca i reexecutem.

⑥ Ara MMU sí que sap on està "printf()." i tots programes van a la mateixa.

NOTA: Això permet no recompilar aplicació quan canvia alguna funció.

NOTA: stub_printf() només s'executa la primera vegada.

Reservar / Alliberar Memòria Dinàmica

Fer ús de la memòria Heap per a reservar regions de memòria.

*brk apunta al límit del Heap.

→ brk(*addr) Actualitza *brk pq apunti a aquesta @ lògica.

→ sbrk(mida) Augmenta el *brk i retorna el nou límit.

Si mida = 0, retorna límit actual.

▽▽ Per alliberar mem. hem de passar mida negativa.

Això implica que només podem alliberar en ordre. (No accedir al mig).

malloc() / free()

Funcions (NO crides a sistema) de C que permeten "solucionar" el problema de que la memòria sigui consecutiva.

malloc(mida): Reserva més espai del passat com a paràmetre i retorna una @ lògica base que assegura que hi ha mida consecutiva suficient.

Aquella regió que retorna passa a ser marcada com "reservat".

Si no hi ha suficient mem. consecutiva ⇒ Augmenta Heap.

free(*ptr): "Allibera" la regió on apunta el punter.

Internament hi ha una estructura de dades que guarda les relacions:

@ lògica ↔ reservada ? si, no ?.

▽▽ IMPO ▽▽: Pot ser que un punter apunti a una zona del Heap que realment hem reservat amb el malloc, però aquella zona no està marcada com a reservada fent que punter apunti a zona "vàlida" però amb contingut desconegut ⇒ RISC seguretat.

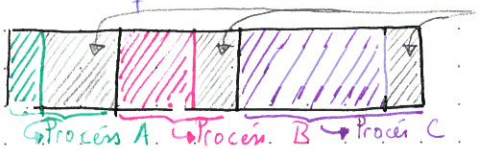
NOTA: Diverses crides a malloc() no asseguren que @ lògiques siguin cons.

Assignació de Memòria

Fragmentació (Problema)

Problema que succeeix al no poder fer ús de zona memòria per a un procés.

- **Interna**: La mida de mem. reservada per a un procés és més de la que necessita. Està reservada, però no ocupada.



"Hi ha aire dins la bosc patata"

- **Externa**: Hi ha mem. lliure, però no assignada degut a que no té la mida necessària. "Hi ha forats en la memòria"

Paginació (Mètode)

Subdividir la memòria en pàgines → Lògica = "Pàgina" / Física = "Frame"

Per a cada pag. del procés, ⇒ Buscar frame disponible.

Per a cada sessió del programa hem de fer ús de X pàgines.

⚠ NO mesclar pàgines de diferents sessions.

Procediment

1. CPU genera @ lògica que necessita traduir-se a física.
2. MMU mira si hi ha una entrada en la TLB.
   - → Encert: Retorna @ física i s'accedeix a memòria ▬ # Només 1 accés mem.
   - → Fallada: Fem ús del REG de la MMU per saber la @Base de la TP del procés actual.
3. Accedim a la TP corresponent (Que està en memòria) per saber @ física.
4. Actualitzem la TLB i accedim a memòria ▬ # Total de 2 accessos a mem.

⚠ Quan hi ha canvi context hem d'invalidar la TLB.

PROBLEMES

- Mida de la TP: Pg. tenim $\frac{2^{bus}-1}{T^{a}pag}$ entrades de 4B ⇒ $\frac{2^{32}}{2^{12}} = 2^{20}$ entrades de 2^2B → 2^{22}B = 4MB
- Fragmentació Interna: Pot ser que secció codi no necessiti tota la mida pàgina.

Segmentació

@lògica = S + offset
↳ segments[S] = {mida, @física, base}

Espai @lògic es divideix en seccions del procés amb la mida que necessita.

La unitat del SO són segments.

Polítiques d'assignació:

- First Fit: Quant trobo suficient espai consecutiu després l'assigno. (Ni que sobri).
- Best Fit: Primer que s'ajusti millor a la mida del segment.
- Worst Fit: Pitjor mida és la que assigno (La mida més gran).
  Això permet que l'espai que sobri es pugui reaprofitar.

Procediment

1. CPU genera @lògica que necessita traduir-se a física.
2. MMU mira en la TS (que està en mem.) quina és la mida del segment.
3. Si el offset és menor a la mida ⇒ Tot OK ⇒ @física = @física_base + offset
   ⇒ NO OK ⇒ Excepció.

OBS: Aquí no hi ha TLB.

PROBLEMES

- Fragmentació Externa: Pq. pot ser que entre segments hi hagi espai de mida més petita i no es pugui fer ús. #SEMPRE ho de ser contigu.

Segmentació Paginada

Els segments han de ser de mida múltiple al tamany de pàgina.

La memòria continua estant dividida en segments, però unitat treball SO és pàgina.

#A ulls de l'assignatura, no hi ha diferència amb "Paginació".

Compartició de Memòria

Es pot especificar a nivell ↳ Pàgina / Segment.

→ Llibreries Dinàmiques: Permeten que diferents programes accedeixin mateix codi.

→ Memòria Compartida: Regió de memòria específica del procés on altres processos poden accedir.. Serveix tmb per comunicar-se.

Optimització Ús de Memòria

COW (Copy On Write)

Quan procés fa un fork() les zones de memòria són en "read-only" fins que el pare/fill fa modificació ⇒ Es fa la còpia del ↳ Segment ↳ Pàgina.

Endarrerir al màxim el reservar espai "duplicat".

Es pot aplicar tmb al reservar mem. dinàmica.

IMPO:
- "execlp()" no es veu afectat pq tot l'espai de mem. (@lògic) canvia.
- "just després del fork()" el fill ha modificat 0 pàgines, però quem s'assigna la variable de retorn del fork() ja és.

Memòria Virtual

Resumit: Tenir en la RAM allò que el procés està fent servir. Allò que no, portar-ho al disc.

Un procés realment només necessita mem. per la inst. actual i dades que fa ref.

Permet augmentar el grau de multiprogramació.

Swapping ▽ Parla de Processos

Tenir en memòria només procés actual. Si aquest necessita @mem ocupada per un altre que no està actual ⇒ Portar antic a disc (Swap).

Permet no perdre els canvis que havia fet el procés anterior (Donat que executable no mod)

Espai Lògic dividit ↳ Pàgines en RAM (Pàgines Residents).
↳ Pàgines en SWAP (Pàgines NO Residents).

Reemplaçament de Memòria ▽ Parla de pàgines

Algorisme del SO que determina quina pàgina ha de ser alliberada en cas de que necessitem frame. # Recorda que hi ha relació frame ↔ pàgina.

Contingut és guarda en PSABU per poder-se recuperar.

1. MMU no pot accedir a pàg. de la swap ⇒ Genera excepció.
2. Comprova que accés sigui vàlid (PCB→TP del procés)
3. Assigna frame per guardar la pàgina.
4. Busca la pàgina en swap (disc) i escriu en frame. ▽ Això Bloqueja Procés
5. Actualitza MMU.

Problemes:

- Suma de @lògiques de tots els processos en execució pot superar a l'espai físic.
- L'espai lògic d'un sol procés també pot superar l'espai físic.
- Accessos a pàg. no residents són molt lents → Hem de llegir de disc
  - → Fallada Pàgina
  - → Bloquejar Procés.

Thrashing

Invertir més temps intercanviant pàgines de memòria amb SWAP que executant.
Es a dir que hi ha moltes fallades de pàgina.

Prefetch

Carregar pàgines que suposem que necessitarem en futur pròxim.
Minimitzant així nº fallades pàgina.

- Distància Prefetch: Quantes hem de carregar per adelantat.

// Volem aprofitar "Localitat Espacial".

Si ens quedem sense espai en la RAM, les pàg. que hem "portat" per adelantat seràn les primeres en ser remplaçades.

Linux Sobre Pentium

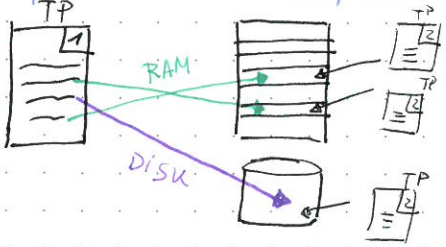
- Segmentació Paginada
- COW, Llibreries Dinàmiques, Càrrega Sota Demanda, Prefetch (Seqüencial)

Taula Pàgines Multinivell

Problemes: → Taula Pàgines pot ocupar molt
→ No podem enviar TP a disc si no recordem on l'hem guardat.

Solució: Tindre 1 TP principal (4kB) que guardarà les @ físiques → RAM / DISK
d'on hem guardat la TP de cada procés.

D'aquesta manera sempre hem de guardar la TP primer nivell per no perdre les de segon.



Mínim 2 TPs → TP nivell 1 / TP del procés.