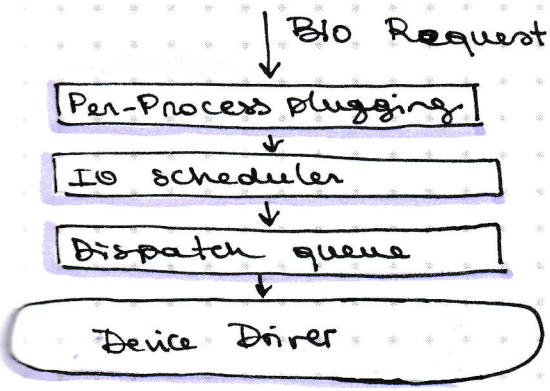
Impact du cache important : presque tout est un fichier donc tout passe par lui.

Metadata fjs en direct, seulem^{t} les datas dans le cache.

Cache très utile pr mmap!
Au moment du deref, page fault → il voit que mapping de fichier
→ vm_operations_struct : *fault
2 types de mapping : • private read → page cache write → duplique
• shared changem^{t} propagé → ecrit ds page cache

Block layer:
→ requetes IO : • start, r/w, taille
• hints (SYNC), flags (FUA, FLUSH)

Vie d'une requête
• créée ds block layer quand IO
• peut ê delayed, merged (IO scheduler, multi queue handling)
• dispatch dans le bon driver
• driver signale qd IO finit

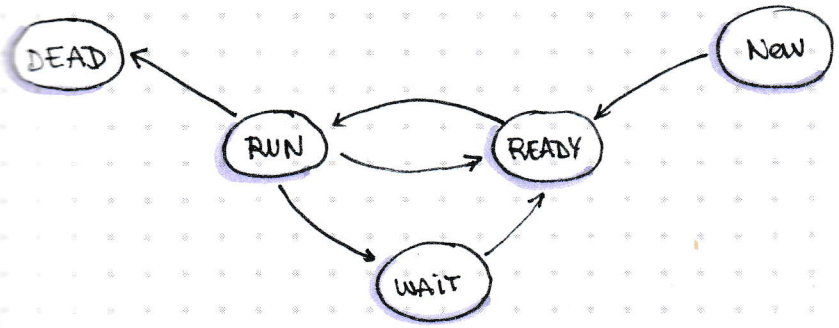


• NOOP : just pass requests into dispatch queue

• CFQ : • Completely fair queueing
• actif par défaut
• sync > async
• support IO priorities, cgrps, sync request idling...

• Deadline : • read > write
• reduce seeking : sort
• dispatch request at least after deadline expired.
→ le - tard possible

SCHEDULING



Contraintes:
• No starvat°
• Fair
• Fast ⇒ O(1)
(de -, en amorti)

Ce qui nous intéresse : ## le monde tourne en ê tps, de manière équitable, ## le tps cpu qui tourne.

Choisir qui tourne → prend du tps. → - de tps pr process

Temps CPU : tps d'exec sur processeur
calcul en global ou par process
- tps user → max
- tps $ / Kernel → min

Tps d'exec : abs unité de tps d'exec d'un process.
process qui finit : global
process qui finit pas : tps d'exec / tps → ratio
(tps passé en running)

Tps turn around : tps entre new et dead ⇒ tps que le process met à finir.

Débit (throughtput) : diff entre exec et turn around.

But : tjs qqn qui tourne
Pb : on ne sait pas qd le process va faire un syscall
→ prédire le futur ?
On peut faire des stats sur les anciennes exec pour essayer de prédire un scheduling idéal.

Tps de réponse : tps qu'une appli met à démarrer
→ tps entre new et 1° running
on aimerai que ce tps de réponse soit le + petit possible.

Tps d'attente : tps passé en ready : tps attendu alors qu'elle était prête.
on cherche à le minimiser → il faut faire des choix.

ex : shell. Passe son tps en waiting. Mais on a envie qu'elle soit très rapide → peu de tps en ready !

Appli interactive : interagit ac qqch (user, $, réseau ...)
passe son tps à faire des syscall
Il y a des applis qui font peu de syscall (peu ou pas d'interact°)
→ + calculatoire.

Goulot d'étranglemt : limite le débit. Ralentis l'appli et l'empêche d'aller aussi vite qu'elle voudrait

ex : vitesse du disk pour accès au fichier, tps CPU pr compil, qt de mémoire, jeu limité par CPU & GPU, ...

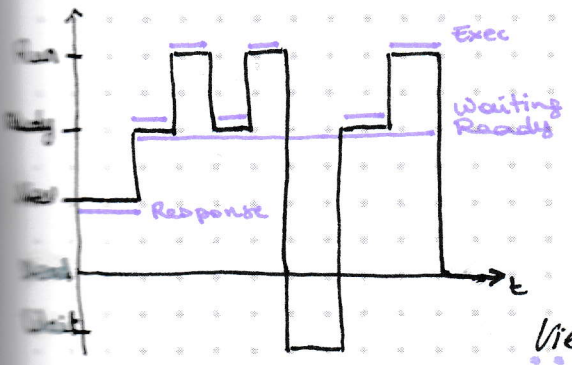
Identifier les goulots d'étranglemt permet de savoir où opti.

Goulot d'étranglemt du scheduler : IO & tps CPU.

→ Tâches IO Bound (syscall bloquant) → interactif
→ Tâches CPU Bound → calculatoire

→ Tâches IO Bound : min waiting time
→ Tâches CPU Bound : exec time ou turnaround time

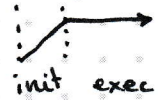
Rq: en pratique le bound change pendant l'exec



Vie d'un process

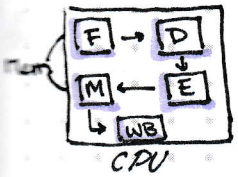
On veut min exec time
→ mais le nbr d'instr ne change pas?
L'exec time ne se resume pas à ça
→ access mem, gest° cache ...

À chaque rescheduling → tps d'init (not. TLB (cache pagination°))
↳ flush à chaque changement de process.



On perd bcp de tps en exec à remplir ts les caches.

Processeur : • Fetch • Decode • Exec • Mem • Writeback



Pls calculs peuvent être fait en mm tps → parallèle.
À chaque tick on avance (+ vite, ↗ trgce)

1 seule instr qui nous intéresse à la fois.
Pr aller + vite :

Fetch (1)	Decode (2)	Exec (1)	Mem (4)	Writeback (1)
1				
2	1			
3	2	1		
4	3	2	1	
5	4	3	2	1

⇒ Pipeline

- Sans pipeline : 1 instr 9 tick, 1 instr tous les 9 ticks
- Avec pipeline : 1 instr 9 tick, 1 instr tous les 4 ticks
  → C'est l'étage le + lent qui ralentit.

Pr aller + vite, on peut split (Mem en 4 part, Decode en 2)
→ on monte artificiellement en trgce.

Intel allait jusqu'à 35 étages sur Pentium 4
Pb : • ça chauffe.
• jmp → solut° ? • on vire tout (on perd potentiellement bcp)
• on exec ∀ cas (c'est au code d'ê bien fait)
• prédire le futur ? → predictor de branchement
on change les instr qu'on a le + de chance d'exe. → likely / unlikely

ex : cmp ——
nop
je ——
mov 0, a
↳ mov 1, a

Au final, augmenter autant la trgce n'est pas la solut° → archi + petite, coeurs + petits, mais pls coeurs. (Archi Core)

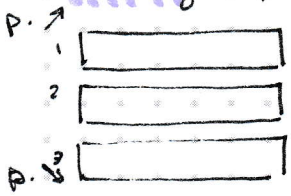
Rq : Basic block → bloc entre instr et jmp.

Pour pas perdre tous les caches, predictor de branch[e] ...
→ exec CPUBound + lgtps, - svt
→ exec IOBound + svt, - lgtps
} équitable

Au final :
IO Bound ⇒ waiting ↘ Quantum ↘
CPU Bound ⇒ waiting ↗ Quantum ↗

Ct detecter quel bound ?
IOBound → nb syscall / s, ne finit pas ses quantum.
CPUBound → finit le quantum svt

Ready : pls queues par priorité



On prend d'abord ds la queue haute priorité
puis on descend.
• Quand on devient IOBound, priorité ↗
• " " " CPUBound, priorité ↘

(Ça marche pas vrmt car on prioris trop les CPUBound au final)
→ Windows (en vrai, il prend pas que IO / CPU Bound en compte.
graphisme in Kernel → priorité qui change en fct)
((affichée priorité ↗)) ((trucs chelous avec installateurs))