Gestione della memoria in Linux

Index

Introduction

In Linux viene fatta una distinzione netta tra richieste di memoria da parte del kernel e di processi utente.
L'idea è quella che il kernel si fida di sé stesso, dunque se una parte dice che ha bisogno di memoria, gli viene concessa con pochi o nessun controllo. Per quanto riguarda i processi utente vengono eseguiti controlli di protezione e di rispetto dei limiti assegnati (`SIGSEGV` segmentation fault)

Gestione della memoria kernel (cenni)

Il kernel potrebbe aver necessità di richiedere tanta RAM tutta insieme, quindi in Linux le richieste di memoria del kernel sono ottimizzate sia per le richieste piccole che per le grandi.
Il kernel può usare sia la memoria a lui riservata nella parte alta della memoria, che quella usata dai processi utente

Se la richiesta è piccola (pochi bytes), fa in modo di avere alcune pagine già printe da cui può prendere pochi bytes richiesti (*slab allocator*)
Se la richiesta è grande (fino a 4MB), fa in modo di allocare più pagine contigue in frame contigui. Avere pagine contigue in frame contigui è importante per il DMA (meccanismo che permette ad un dispositivo di I/O di scrivere direttamente in RAM) che ignora il fatto che ci sia la paginazione e va a scrivere direttamente in RAM.

Dunque quando il kernel assegna una memoria ad un DMA questa deve essere contigua (usa essenzialmente il buddy system)

Gestione della memoria utente

- Fetch policy → paging on demand
- Placement policy → il primo frame libero
- Replacement policy → nel seguito
- Gestione del resident set → politica dinamica con replacement scope globale
  - molto globale: vi rientrano anche la cache del disco (page cache, ci torneremo)
  - anche qui, c'è un buddy system per richieste grandi
- Politica di pulitura → ritardata il più possibile
  - scrittura quando la page cache è troppo piena con molte richieste pending
  - troppe pagine sporche, o pagina sporca (*dirty page*, pagina modificata da tanto tempo ma non su disco) da troppo tempo
  - richiesta esplicita di un processo: flush, sync o simili
- Controllo del carico → assente

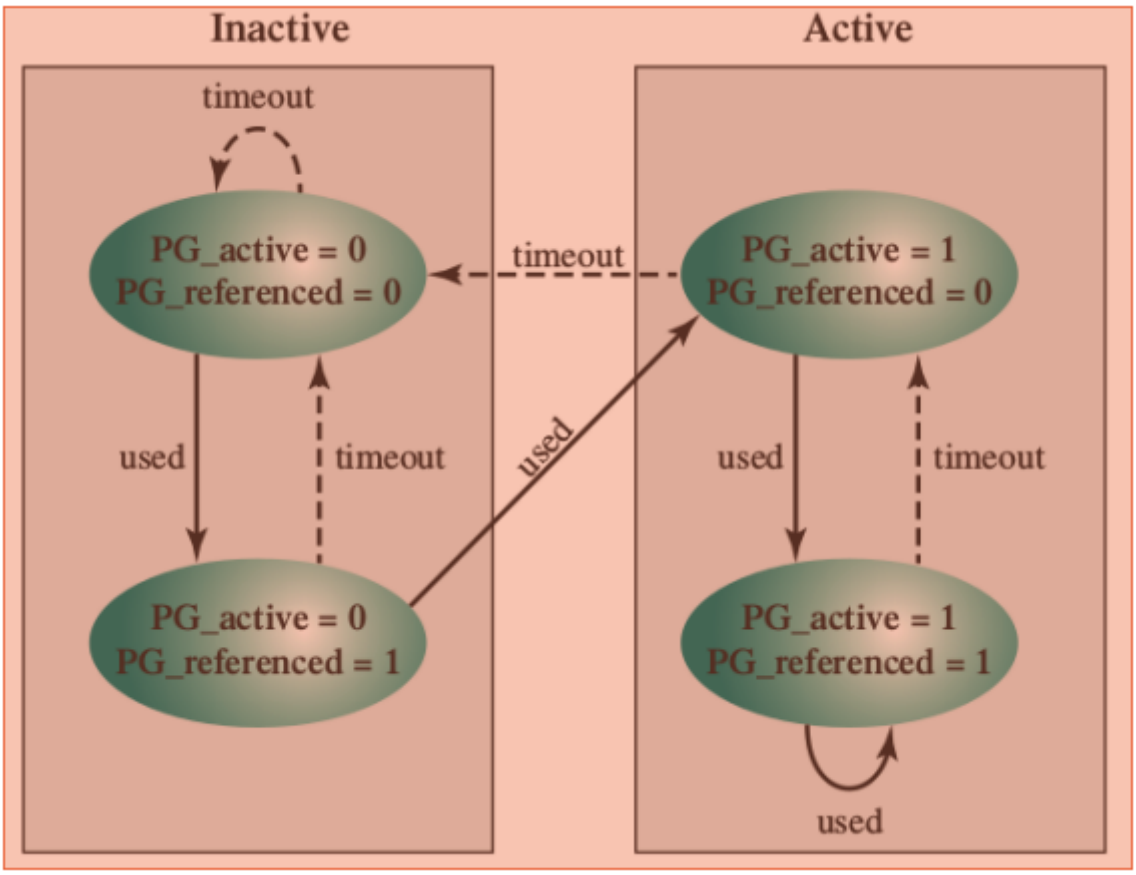
Replacement Policy

Si basa sull'algoritmo dell'orologio ma con una modifica (in particolare utilizza un LRU "corretto", il precedente algoritmo dell'orologio era molto efficace ma poco efficiente soprattutto con memorie molto grandi).

In Linux si hanno due flag per ogni entry delle page table: `PG_referenced` (pagine a cui sono stati fatti riferimenti) e `PG_active` (pagine effettivamente attive). Sulla base di `PG_active` sono mantenute due liste di pagine: le attive e le inattive.
`kswapd` è un kernel thread in esecuzione periodica che scorre le pagine inattive.
`PG_referenced` viene settato ogni qual volta che la pagina viene richiesta; quindi si hanno due possibilità:

- o arriva prima `kswapd`, in questo caso `PG_references` viene settato a zero
- o arriva un atro riferimento, in questo caso la pagina diventa attiva

Solo le pagine inattive possono essere rimpiazzate (tra di esse si fa una specie di LRU con contatore a 8 bit)



il `timeout` è da intendere come `kswapd` che viene eseguito