Unix: Gestione della Memoria

Modello della memoria in Unix

- I processi Unix lavorano su uno spazio di indirizzamento *virtuale*
  Es. $0, \ldots, 2^{32} - 1$ su indirizzi a 32bit
- Ogni processo ha uno spazio indirizzi separato per i segmenti text, data e stack.
- Le dimensioni di stack e data possono cambiare durante l'esecuzione
- Gli indirizzi di Text e Data crescono a partire dalla parte bassa degli indirizzi *virtuali* ($0 \rightarrow$)
- Gli indirizzi dello stack crescono a partire dalla parte alta degli indirizzi *virtuali* ($\rightarrow 2^{32}$)
- Unix supporta inoltre
  - La condivisione dei segmenti di testo e dati con copy-on-write
  - I file mappati in memoria principale (ad es. per librerie condivise)

Gestione della memoria in UNIX - storia

- Fino a 3BSD (1978): solo segmentazione con swapping;
- Da 3BSD: segmentazione e paginazione;
- Da 4BSD (1983): segmentazione e paginazione su richiesta

Gestione della memoria in UNIX - fondamenti

- Un processo in memoria (ready-to-run) necessita solo delle seguenti strutture dati:
  - user structure (id e pid, tabella file aperti, ecc)
  - tabella delle pagine (allocazione pagine logiche in memoria principale/secondaria)
- le pagine data, stack, text sono portate in memoria dinamicamente

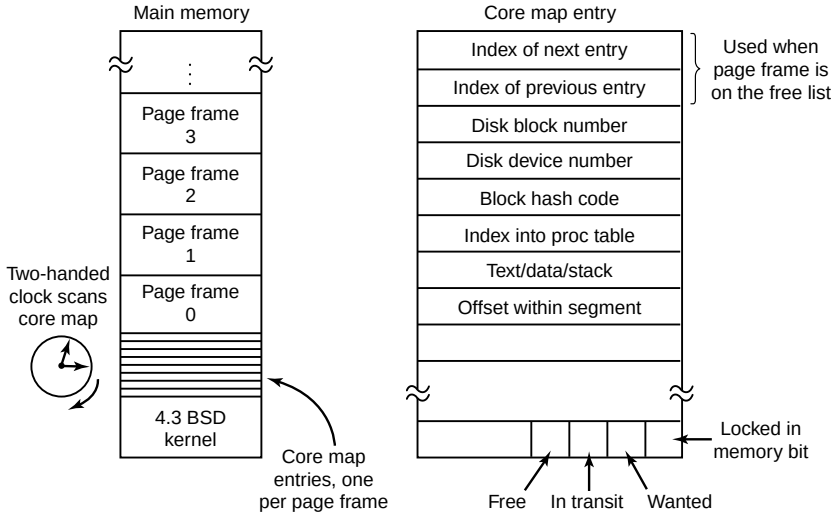
Quando si alloca la memoria

- Un processo può richiedere memoria nei seguenti casi:
  1. Al momento della creazione tramite fork (allocazione di memoria per i segmenti data e stack);
  2. Chiamata della funzione di sistema `malloc` (estende il segmento data);
  3. Lo stack cresce oltre le dimensioni prefissate del segmento stack;
  4. Si accede in scrittura ad una pagina condivisa tra due processi e gestita il metodo copy-on-write
- Inoltre potrebbe essere necessario caricare memoria ad un processo che era swapped da troppo tempo

Gestione della memoria in UNIX

- Il sistema operativo mantiene residente in memoria principale una tabella con informazioni sul contenuto dei frame della memoria (*core map*)
- Ogni frame ha uno spazio disco dove si copia una pagina da rimpiazzare
- I frame liberi sono organizzati in una lista concatenata (free list) all'interno della core map
- Una entry della core map ha la seguente struttura:
  - se il frame è libero mantiene i puntatori al precedente e al successivo frame libero (fa parte della free list)

  - se il frame è occupato mantiene
    - l'indirizzo su disco per il frame
    - l'indice (entry) nella tabella dei processi relativo al processo che usa la pagina fisica
    - il puntatore all'inizio del segmento testo/dati/stack e relativo offset per il processo in questione
    - flag utilizzati dall'algoritmo di paginazione



Nota: se pagina è di 1K e descrittore della core map è di 16byte allora la core map (1 descrittore per pagina) occupa meno del 2% della memoria

Allocazione di memoria

- Quando un processo viene lanciato, molte pagine vengono precaricate e poste sulla free list (prepaging)
- Quando un processo termina, le sue pagine vengono rimesse sulla free list
- Il sistema usa allocazione libera (quindi anche in celle non consecutive) per gestire la richiesta di pagine da parte di un processo
- Tuttavia se la free list scende sotto una certa soglia (parametro del kernel) il kernel si rifiuta di allocare nuove pagine di memoria
- La free list viene anche utilizzata come *memoria cache*: le pagine richieste da un processo che risultano *invalide* vengono cercate sulla free list, prima di essere caricate da disco
- Quando un processo termina, le sue pagine vengono messe sulla free list

Paginazione e swapping in Unix

- In Unix la memoria principale viene gestita da due demoni:
  - Lo swapper (processo 1) che si occupa di rimuovere *processi* dalla memoria
  - Il *demone delle pagine* o *pagedaemon* (processo 2) che si occupa del rimpiazzamento delle pagine

- I due demoni lavorano sulla base di alcuni parametri di sistema
  - *Lotsfree*: indica il numero minimo di frame per evitare l'intervento del demone delle pagine
  - *Minfree*: indica il numero minimo di frame per evitare lo swapping di processi
  - *Desfree*: numero di frame liberi desiderabile per un buon funzionamento del sistema
  - La relazione tra i parametri è: $Minfree < Desfree < Lotsfree$

Esempio Tuning del Kernel di Unix Solaris

```
* The values specified are in pages (pages * pagesize (8192) / 1024 = Kbytes /
* The values listed here are the defaults for a system with 4GB of memory.
*
* physmem:        511929  Reports the physical amount of system memory
*
* lotsfree:       7932  when free memory drops below this, the page
*                       daemon starts scanning for memory to add to it
*                       at the slowscan rate.
* minfree:        1983  as free memory drops below lotsfree and nears
*                       minfree, it starts scanning faster.  If it reaches
*                       it is then scanning at the fastscan rate.
* desfree:        3966  this is the level at which swapping begins.
```

Demone delle pagine

- Le pagine vengono allocate dalla lista libera dal kernel con strategia *copy-on-write*
- La free list viene mantenuta "piena" entro un certo livello dal demone delle pagine
- Il demone delle pagine viene lanciato al tempo di boot e si attiva ad intervalli regolari (tipicamente 2–4 volte al secondo) o su richiesta del kernel
- Quando viene risvegliato
  - Se il numero di frame liberi (NFL) è $\geq$ di *lotsfree* torna inattivo
  - Se $NFL < lostfree$, inizia a scorrere le pagine cercando di spostare pagine su disco fino a che non ci sono *lotsfree* pagine libere

  - La velocità di scansione cresce al diminuire di NFL
- Il demone utilizza una politica di rimpiazzamento *globale* (non si guarda il processo a cui appartiene la pagina)
- Come algoritmo di sostituzione delle pagine utilizza una variante dell'algoritmo dell'orologio (con due lancette invece che una)

L'algoritmo dell'orologio a due lancette

- L'algoritmo usa due puntatori (lancette) per scorrere la lista delle pagine allocate nella *core map*.
- Sia NFL=numero di frame liberi
- Fino a che $NFL <$ *lotsfree* si eseguono i seguenti passi:
  - la prima lancetta azzera il reference bit R della pagina a cui punta correntemente
  - la seconda sceglie la pagina vittima:
    - se trova $R = 0$ (cioè la pagina non è stata usata nel periodo trascorso tra il passaggio delle due lancette):
      - se il dirty-bit è a 1 il frame viene salvato su disco
      - il frame viene aggiunto alla free-list
    - poi si fanno avanzare i due puntatori

CLOCK a due lancette (cont.)

- La distanza tra i puntatori (*handspread*) viene decisa al boot, per liberare frame abbastanza rapidamente
- Se le lancette sono vicine: solo le pagine realmente usate rimarranno in memoria
- Se le lancette sono distanti 359° = algoritmo dell'orologio (la seconda passa dopo un giro)
- Possibile variante:
  - ulteriore parametro *maxfree* $>$ *lotsfree*
  - quando il livello di pagine scende sotto *lotsfree*, il pagedaemon libera pagine fino a raggiungere *maxfree*

  Permette di evitare una potenziale instabilità del CLOCK a due lancette.

Swapper

- Il processo *swapper* o *sched* (PID=0, lanciato anch'esso al tempo di boot) decide quale processo deve essere swappato su disco
- Lo swapper si sveglia ogni 1–2 secondi, e interviene solo se il sistema di paginazione è sovraccarico:
  - il numero di frame liberi è sotto la soglia minima ammissibile *minfree*
  - Il numero medio di frame liberi nell'unità di tempo è minore di *desfree*

Swapping: chi esce...

- Regole di scelta del processo vittima:
  - si cerca tra i processi in attesa, senza considerare quelli in memoria da meno di 2 secondi
  - se ce ne sono, si prende quello con il valore *priorità+tempo di residenza in memoria* più alto (nota: tempo CPU$\neq$ tempo in memoria)
  - Altrimenti, si cerca tra quelli in "ready", con lo stesso criterio
- Per il processo selezionato:
  - i suoi segmenti data e stack (non il text) vengono scaricati sul device di swap; i frame vengono aggiunti alla lista dei frame liberi
  - Nel PCB, viene messo lo stato "swapped" e agganciato alla lista dei processi swappati
- Si ripete fino a che sufficiente memoria viene liberata.

Swapping: ...e chi entra

Quando *swapper* si sveglia da sé:

1. cerca nella lista dei PCB dei processi swappati e ready, il processo swappato da più tempo, ma almeno 2 secondi (per evitare thrashing);
2. se lo trova, determina se c'è sufficiente memoria libera per la page table e la u-structure (*easy swap*) oppure no (*hard swap*);
3. se è un hard swap, libera memoria swappando qualche altro processo;
4. carica le page table e la u-structure in memoria e mette il processo in "Ready-to-run, in memory"

Si ripete finché non ci sono processi da caricare.

Considerazioni

Interazione tra scheduling a breve termine, a medio termine e paginazione

- minore è la priorità, maggiore è la probabilità che il processo venga swappato
- per ogni processo in esecuzione, la paginazione tende a mantenere in memoria il suo working set
- quindi, processi che non sono idle tendono a stare in memoria, mentre si tende a swappare solo processi idle da molto tempo
- nel complesso, il sistema massimizza l'utilizzo della memoria e la multiprogrammazione, limitando il thrashing e garantendo l'assenza di starvation per i processi swappati