

Modello della memoria in Unix

- Ogni processo UNIX ha uno spazio indirizzi separato. Non vede le zone di memoria dedicate agli altri processi.
- Come detto più volte un processo UNIX ha tre *segmenti*:
 - **Stack** Stack di attivazione delle subroutine. Cambia dinamicamente.
 - **Data** Contiene i dati inizializzati al caricamento del programma e lo *heap* (memoria allocata dinamicamente). Cambia dinamicamente su richiesta esplicita del programma (es., con la **malloc**).
 - **Text** codice eseguibile. In generale: Non modificabile, protetto in scrittura.
- I processi lavorano sullo spazio di indirizzamento *virtuale* (es. $0, \dots, 2^{32} - 1$ su indirizzi a 32bit)

- testo e dati: crescono a partire dalla bassa degli indirizzi *virtuali* ($0 \rightarrow$)
- stack: cresce a partire dalla parte alta degli indirizzi *virtuali* ($\rightarrow 2^{32}$)
- Unix supporta inoltre
 - Condivisione dei segmenti di testo e dati (es dopo una `fork`) con copy-on-write
 - Hardware con spazio indirizzi separato per codice e dati
 - File mappati in memoria principale (ad es. per librerie condivise)

Gestione della memoria in UNIX

- Fino a 3BSD (1978): solo segmentazione con swapping;
- Da 3BSD: segmentazione + paginazione;
- Da 4BSD (1983): segmentazione + paginazione on demand
- Idea principale
 - Un processo in memoria (ready-to-run) ha solo bisogno della u-area e della tabella delle pagine;
 - le pagine di codice, dati e stack sono portate in memoria dinamicamente

Quando si alloca la memoria

- Ulteriore memoria può essere richiesta da un processo in corrispondenza ad uno dei seguenti eventi:
 1. Una fork, che crea un nuovo processo (allocazione di memoria per i segmenti data e stack);
 2. Una brk (e.g., in una malloc) che estende un segmento data;
 3. Uno stack che cresce oltre le dimensioni prefissate.
 4. Un accesso in scrittura ad una pagina condivisa tra due processi (*copy-on-write*)

Oppure, per un processo che era swapped da troppo tempo e che deve essere caricato in memoria.

Gestione della memoria in UNIX Moderni (4BSD, ...)

- Il sistema operativo mantiene residente in memoria principale una tabella con informazioni sul contenuto dei frame della memoria chiamata *core map*
- Attraverso la core map si può gestire una lista di frame liberi (free list)
- Per ogni frame la corrispondente entry della core map mantiene informazioni quali:
 - se il frame è libero: puntatori a prec/succ. frame libero (cella della free list)
 - se il frame è occupato:
 - * indirizzo su disco della pagina
 - * indice per la tabella dei processi (processo che usa la pagina fisica)
 - * puntatore all'inizio del segmento testo/dati/stack e relativo offset
 - * informazioni per la paginazione

Gestione della memoria in UNIX

- Le pagine vengono allocate dalla lista libera dal kernel, su base libera, con strategia *copy-on-write* (ad es. per la fork)
- La free list viene mantenuta “piena” entro un certo livello dal demone *pagedaemon* (PID=2) che applica varianti dell’algoritmo dell’orologio
- Lo swapping rimane come soluzione contro il thrashing: uno *scheduler a medio termine* decide il grado di multiprogrammazione.
- Lo swapper (PID=0) si attiva automaticamente, in situazioni estreme

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 messe sulla free list
- Il sistema usa allocazione libera per gestire la richiesta di pagine da parte di un processo
- Tuttavia se la free list scende sotto un certo livello *minfree* fissato dal sistema, il kernel si rifiuta di allocare nuove pagine di memoria. Tipicamente, *minfree* è un valore tra 100K e 5M.
- 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

Pagedaemon

- *Pagedaemon* viene lanciato al boot e si attiva ad intervalli regolari (tip. 2–4 volte al secondo) o su richiesta del kernel
- Il pagedaemon interviene quando il numero di frame liberi (NFL) è minore di *lotsfree* un parametro che denota il numero minimo ammissibile di frame liberi (solitamente 1/4 dei frame totali), altrimenti torna inattivo
- Nel primo caso ($NFL < lotsfree$) inizia il trasferimento di pagine fino a che non ci sono *lotsfree* pagine libere
- Pagedaemon utilizza una politica di rimpiazzamento *globale* (non si guarda il processo a cui appartiene la pagina) e una variante dell'algoritmo dell'orologio: l'orologio (CLOCK) a 2 lancette

CLOCK a due lancette

- L'algoritmo usa due puntatori (lancette) per scorrere la lista delle pagine allocate nella *core map*.
- Fino a che $NFL < lotsfree$ si eseguono i seguenti passi:
 - la prima lancetta azzerava il reference bit della pagina a cui punta correntemente
 - la seconda sceglie la pagina vittima:
 - * se trova $r = 0$ (i.e. la pagina non è stata usata nel periodo trascorso tra il passaggio delle due lancette):
 - se dirty-bit=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.

Swapping di processi

- Interi processi possono essere sospesi (*disattivati*) temporaneamente, per abbassare la multiprogrammazione
- Il processo *swapper* o *sched* (PID=0, lanciato al boot) decide quale processo deve essere swappato su disco
- Parametro: *desfree*, impostato al boot. 100K–10M.
 $minfree < desfree < lotsfree$
- Lo *swapper* si sveglia ogni 1–2 secondi, e interviene solo se
 $NFL < minfree$ e
 $NFL < desfree$ nella storia recente

Swapping: chi esce...

- Regole di scelta del processo vittima:
 - si cerca tra i processi in “wait”, 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 l'area-u (*easy swap*) oppure no (*hard swap*);
3. se è un hard swap, libera memoria swappando qualche altro processo;
4. carica le page table e l'area-u in memoria e mette il processo in "Ready-to-run, in memory"

Si ripete finché non ci sono processi da caricare.

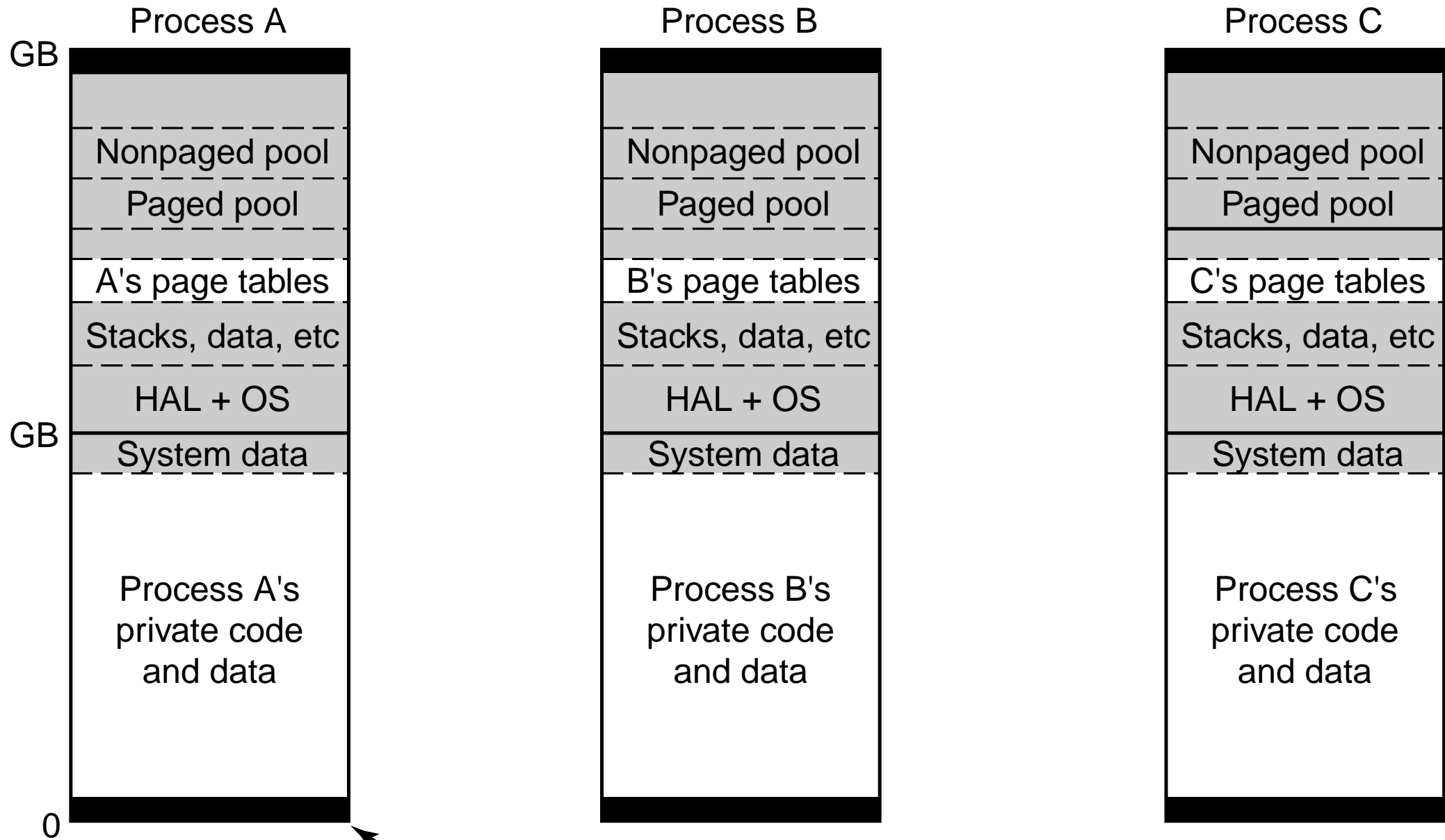
Considerazioni

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

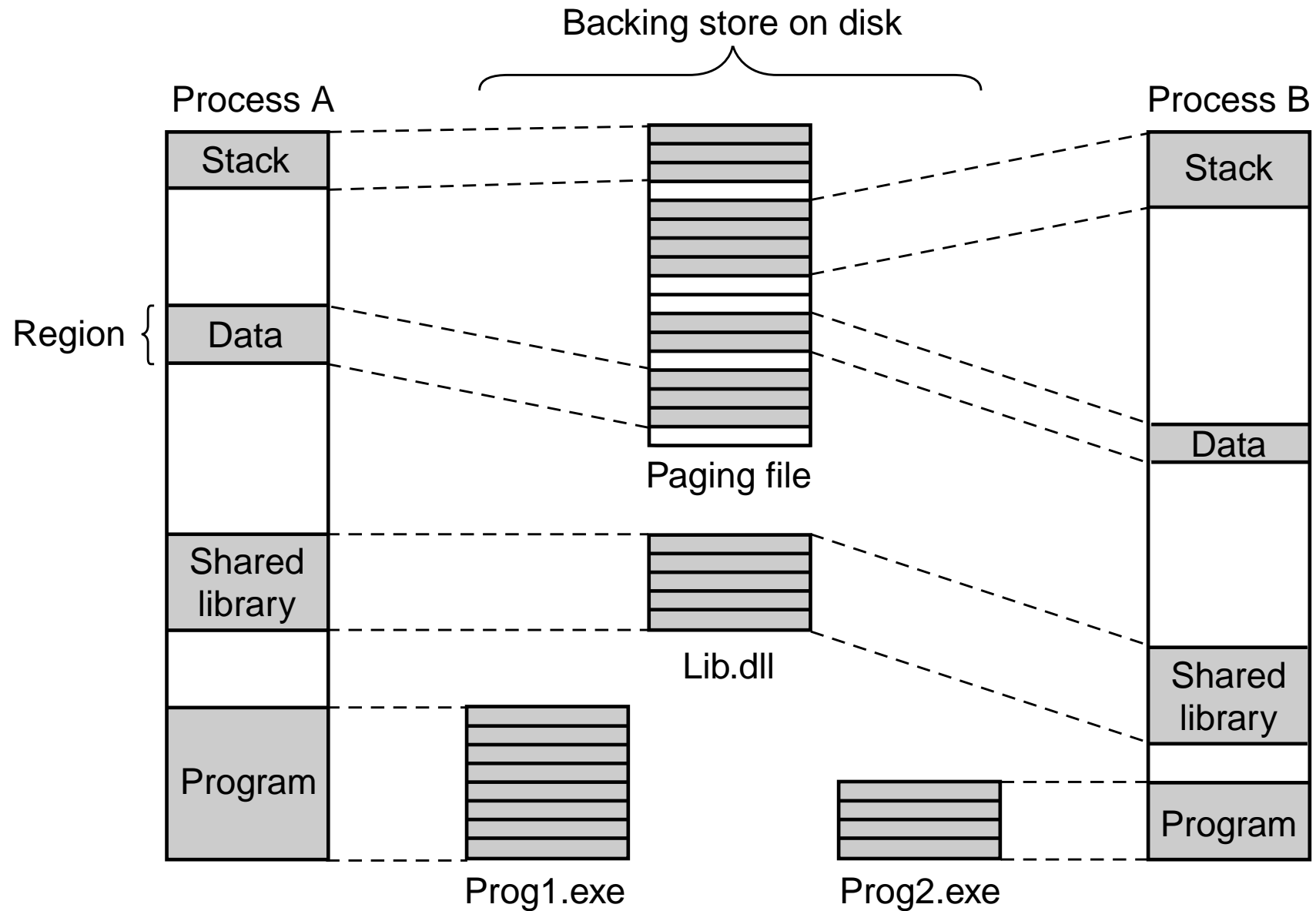
Modello della memoria in Windows 2000

- Ogni processo utente di Windows riceve uno spazio indirizzi virtuali di 4G, diviso in due parti
 - codice e dati, nella parte bassa (< 2G) Liberamente accessibile.
 - kernel, HAL (strato di astrazione hardware) e strutture di sistema (comprese le page tables) nella parte alta La maggior parte di questo spazio non è accessibile, neanche in lettura.
 - i primi ed ultimi 64kb sono invalidi per individuare rapidamente errori di programmazione (puntatori a 0 e -1).
- La memoria è puramente paginata (senza prepaging), con copy-on-write.
- Il caricamento dei segmenti è basato fortemente sul memory mapping.
- Mappare il kernel nello spazio dei processi utenti aumenta l'efficienza delle chiamate di sistema: un thread che passa in modo kernel non deve cambiare tabelle per gestire la rilocazione



Bottom and top
64 KB are invalid

Mappatura dei segmenti in memoria reale e su file



Gestione della paginazione

Nessuna forma di prepaging: tutte le pagine vengono caricate su page fault.

Le pagine possono essere in diversi *stati* dai quali dipendono le sorti di un page fault

Windows utilizza la strategia del working set per gestire la paginazione

Stati di una pagina

- *Available*: pagina non usata da nessun processo. Si divide in tre possibilità:
 - Free: riusabile
 - Standby: rimossa da un ws ma richiamabile (buffering)
 - Zeroed: riusabile e in più tutta azzerata (si cancellano i dati per ragioni di sicurezza)
- *Reserved*: riservata da un processo ma non ancora usata (ad es. spazio per lo stack di un thread). Non fa parte del ws fino a che non viene veramente usata.
- *Committed*: usata da un processo e associata ad un blocco su disco (*mappata*): la pagina fisica potrebbe non essere in memoria

Casi di page fault

- La pagina riferita non è committed
⇒ Terminazione del processo
- Violazione di protezione
⇒ Terminazione del processo
- Scrittura su una pagina condivisa
⇒ Copy-on-write su una pagina reserved
- Crescita dello stack
⇒ Allocazione di una pagina azzerata
- La pagina riferita è committed (ha un indirizzo fisico associato) ma non attualmente caricata in memoria
⇒ il vero page fault: pagein della pagina mancante

Algoritmo di rimpiazzamento di pagina

La paginazione è basata sul modello del Working Set per i *processi*

- ogni processo ha un working set (insieme delle pagine mappate in memoria) e una dim. minima *min* e massima *max* (si può scendere sotto *min* e salire sopra *max*)
- Tutti i processi iniziano con lo stesso *min* e *max* (risp. 20-50 e 45-345, in proporzione alla RAM; modificabile dall'amministratore del sistema)
- Ad un page fault ($ws = \text{dimensione del working set}$)
 - se $ws < max$, la pagina viene allocata ed aggiunta al working set.
 - se $ws > max$, una pagina vittima viene scelta nel working set (politica di rimpiazzamento *locale*)
- I limiti possono cambiare nel tempo: se un processo sta paginando troppo (thrashing locale), il suo *max* viene aumentato
- vengono mantenute sempre libere almeno 512 pagine

Le pagine allocate vengono prelevate dalla *free list*:

- ogni secondo parte un thread del kernel (*balance set manager*)
- se la *free list* è troppo corta, parte il *working set manager* che esamina i *working sets* per liberare pagine
 - prima i processi più grandi e idle da più tempo; il processo in foreground è considerato per ultimo
 - se un processo ha $ws < min$ o ha avuto molti page fault recentemente, viene saltato
 - altrimenti una o più pagine vengono rimosse
- si ripete sempre più aggressivamente finché la *free list* ritorna accettabile
- anche parte del kernel può essere paginata
- Eventualmente un *ws* può scendere sotto il *min*
- non esiste completo swapout di processi

Gestione della memoria fisica

- I frame liberi sono organizzati in diverse liste mantenute dal sistema operativo
- Windows 2000 utilizza strategie ed euristiche complesse (che non vedremo) per gestire la scelta della pagina fisica da allocare per una richiesta
- Le liste sono 5:
 1. Pagine modificate: pagine eliminate da un ws, associate ad un processo, ma che devono essere copiate su disco
 2. Pagine in attesa (standby): eliminate da un ws, assoc. ad un processo, consistenti con immagine su disco
 3. Pagine libere: non sono più associate ad alcun processo

4. Azzerate (zeroed): libere e con contenuto azzerato

- le pagine nelle liste 1 e 2 si possono recuperare quando il processo corrispondente le richiede
- la lista 3 contiene pagine di processi che hanno terminato l'esecuzione
- Le pagine possono cambiare lista:
 - dei demoni di scrittura di pagine mappate/modificate spostano pagine da 1 a 2
 - come effetto di una deallocazione una pagina può andare da 2 a 3
 - il demone *zero page thread* (gira a priorità più bassa) sposta pagine da 3 a 4: se la CPU è inattiva vengono azzerate delle pagine (più utili di pagine sporche)