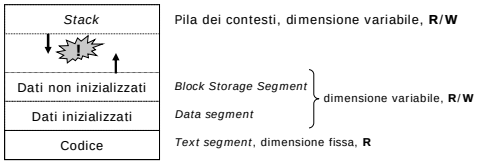
Gestione della memoria – 1

- Massima **semplicità** per massima **portabilità** su architetture fisiche diverse
- Ogni processo possiede un proprio spazio di indirizzamento privato (**memoria virtuale**)
  - Suddiviso in 4 sezioni



Gestione della memoria – 2

- Il segmento dati varia in dimensione a seconda delle attività del programma (p.es. `malloc()` in C)
  - POSIX **non definisce** le chiamate di sistema relative alle gestione della memoria
  - Una parte del segmento dati può ospitare *file* mappati in memoria
- Il segmento *stack* contiene l'ambiente d'esecuzione corrente (***record* di attivazione**) e cresce nel verso **opposto** alla crescita del segmento dati
- Il segmento codice può essere condiviso tra più processi
  - Ma **non gli altri** segmenti
  - Tranne che per processi duplicati da `fork()`

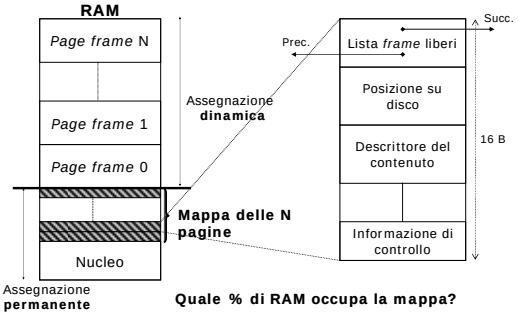
Gestione della memoria (UNIX) – 3

- In origine l'allocazione di memoria principale avveniva mediante ***swap*** di processi
  - Rimpiazzo di interi processi quando una particolare esecuzione rilevava mancanza di memoria
    - A seguito di `fork()`
    - A causa di allocazione esplicita richiesta dal programma
    - Per allocazione implicita conseguente a chiamata di procedura
      - Che richiede l'allocazione di un nuovo *record* di attivazione
  - Il gestore (***swapper***) creava lo spazio necessario salvando su disco i processi sospesi con più tempo d'esecuzione recente e minor priorità
    - Bastava utilizzare una lista dei blocchi liberi su disco

Gestione della memoria (UNIX) – 4

- In seguito fu introdotta paginazione con modalità **a richiesta** (***paging on demand***)
  - Un processo è eseguibile se il suo descrittore e la sua tabella delle pagine si trovano in RAM
    - Il suo spazio di indirizzamento è caricato da disco per ogni riferimento che richiede dati non presenti in RAM
    - Nessun caricamento anticipato di pagine
      - Nessun uso di *working set*
  - Il processo "**2**" gestisce lo stato dei *page frame* in RAM
    - *Page daemon*
    - Tenendo nucleo di S/O e "mappa delle pagine" (*core map*) **sempre in RAM**
    - Il resto è paginato e ciascuna *page frame* indica il proprio uso
      - Codice, dati, *stack*; tabella delle pagine
      - Altrimenti in lista pagine libere

Mappa delle pagine (*core map*)

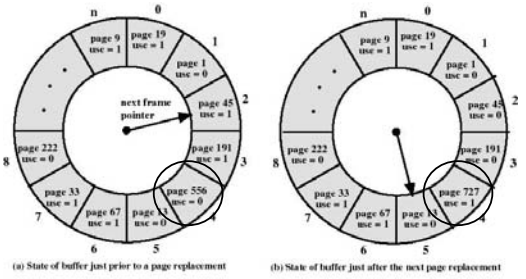


Quale % di RAM occupa la mappa?

Gestione della memoria (UNIX) – 5

- *Page daemon* verifica con periodo 1/4 s che in RAM vi siano ≥ `lotsfree` pagine libere
  - Se ne mancano ne libera quante ne servono salvandone il contenuto corrente su un'area di disco specifica per pagina
  - La selezione delle pagine in uscita usa un algoritmo "**a doppia passata**"
    - ***Two-handed clock algorithm***
    - Lista circolare delle pagine
    - La 1ª passata pone a 0 il *bit* di riferimento
    - La 2ª passata, a distanza **programmabile**, rimuove le pagine nel frattempo non riferite (*bit* vale 1 altrimenti)

Orologio a una passata



Intervallo troppo lungo tra 2 passate successive!

Gestione della memoria (UNIX) – 6

- Il *page daemon* limita la frequenza di paginazione spostando processi su disco
  - Quelli che non abbiano eseguito negli ultimi 20 s
  - Tra i 4 più grandi quello da più tempo in memoria
- Se vi è spazio libero il *page daemon* riporta in RAM processi pronti selezionati con una euristica di "valore"
  - Caricando solo il descrittore di processo e la sua tabella delle pagine
    - Lasciando che il resto sia caricato via *paging on demand*

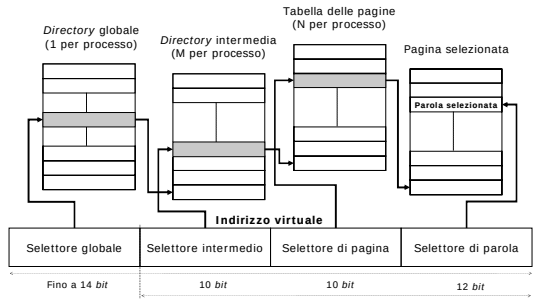
Gestione della memoria (GNU/Linux) – 6

- Per architetture a 32 *bit* la memoria virtuale di processo è ampia 4 GB
  - 1 GB **riservato e invisibile** al modo operativo normale per la tabella delle pagine del processo e per altri dati di controllo **a uso del nucleo**
- Spazio suddiviso in **regioni** = **sequenze contigue** di pagine
  - Le regioni non sono necessariamente **consecutive** tra loro
- Ogni regione ha un descrittore noto al nucleo
  - Lo spazio di indirizzamento virtuale di un processo è visto come una **lista di descrittori** di regione

Gestione della memoria (GNU/Linux) – 7

- La `fork()` di GNU/Linux replica per il figlio l'intera lista di descrittori del padre
- Le pagine del figlio sono fisicamente duplicate **solo** in caso di modifica (***copy on write***)
  - La regione è marcata **R/W**
  - Le sue pagine dati sono inizialmente marcate **R**
  - Ogni richiesta di scrittura causa eccezione così il nucleo duplica la pagina richiesta e marca la copia come **R/W**

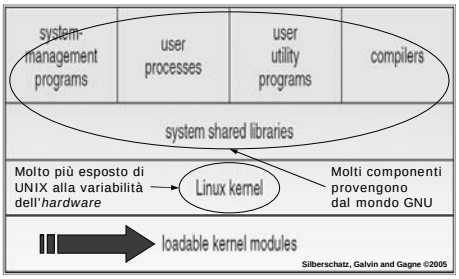
Gestione della memoria (GNU/Linux) – 8



Gestione della memoria (GNU/Linux) – 9

- Il nucleo rimane **sempre** in RAM
  - Ha dimensione **variabile** a causa del caricamento **dinamico** di moduli di gestione dispositivi
- La RAM rimanente viene usata per
  - **[P1]** Le pagine attive dei processi utente
  - **[P2]** Una *cache* di blocchi di *file* usata dal FS
    - Dimensione **variabile** organizzata per pagine
  - **[P3]** Un insieme di pagine utente inattive ma presenti
- La RAM viene assegnata in frazioni (*slab*) di dimensione variabile e arbitraria

Architettura di S/O GNU/Linux



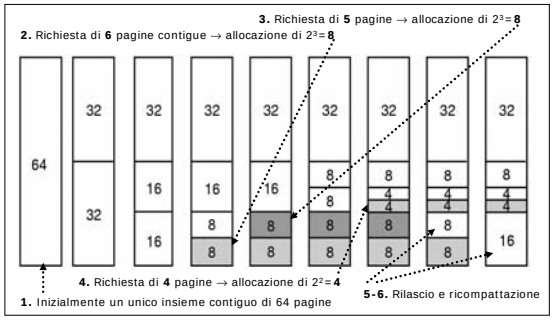
Moduli di nucleo caricabili

- Il nucleo GNU/Linux di base **non include** gestori di dispositivi
  - Caricati in fase di installazione oppure anche successivamente
    - E allo stesso modo possono essere anche rimossi
- **[1]** Trattamento del modulo in ingresso da parte del nucleo
  - Caricamento del codice oggetto del modulo nella memoria virtuale del nucleo
  - Collegamento dei simboli usati dal modulo alla tabella dei simboli del nucleo
- **[2]** Registrazione del modulo gestore
  - Presso tabelle mantenute dal nucleo
    - Servizi e meccanismi distinti per dispositivi distinti
      - Per esempio: *mouse*, disco, FS, connessione di rete, ...
  - Inizializzazione e poi eventualmente rimozione
- **[3]** Regolazione dei possibili conflitti tra gestori
  - Tramite "prenotazione" del gestore al nucleo del proprio accesso a dispositivo

Gestione della memoria (GNU/Linux) – 10

- **Algoritmo di allocazione primario** (*buddy*)
  - Ogni richiesta di ampiezza N è arrotondata a $2^n \geq N$
  - La memoria disponibile viene frazionata in metà successive fino a frazioni di ampiezza 2^n
    - Una singola frazione viene assegnata al richiedente
    - Una struttura ausiliaria contiene la testa di liste predefinite di frazioni di ampiezza 2^i (i=0, ..., n) per velocizzare la ricerca
  - La memoria disponibile usata per l'allocazione è sempre la frazione libera di minore dimensione
  - Al rilascio ogni frazione tornata libera si unisce con la frazione vicina se libera (il suo *buddy*)
  - Due algoritmi sussidiari cercano di ridurre la frammentazione causata dall'algoritmo primario

Funzionamento del *buddy algorithm*



Gestione della memoria (GNU/Linux) – 11

- I segmenti codice e i *file* mappati in memoria hanno un corrispondente *file* su disco
- Al resto (aree di lavoro dei processi) si assegna una **partizione paginata** (***paging partition***) vista come *byte stream* oppure un *file* di pagine
  - La partizione paginata è d'uso più semplice e veloce
    - Nessun limite fissato di scrittura
    - Possibilità di scrittura contigua
- Le pagine libere sulla partizione e/o sul *file* sono individuate mediante *bitmap*
  - Lo spazio disponibile viene assegnato alle pagine di lavoro rimosse **temporaneamente** dalla RAM

Gestione della memoria (GNU/Linux) – 12

- `kswapd` è il *page daemon* e ha periodo 1 s
  - Per ogni attivazione esegue fino a 6 passate di un ciclo di lavoro cercando pagine da spostare su disco
    - Tra quelle in **[P2]** e **[P3]** con una variante del *clock algorithm*
    - Tra quelle condivise da più processi ma poco usate
    - Infine tra quelle in **[P1]**
      - Cominciando dal processo con più pagine scandendo l'intera lista dei suoi descrittori di regione (per indirizzo virtuale) con *clock algorithm* ma **solo** sulle pagine attive
  - Ogni pagina selezionata modificata
    - Posta in attesa di riscrittura se ha corrispondente su disco
    - Altrimenti salvata su *paging partition* o *paging file*
- Il *daemon* `bdflush` gestisce la riscrittura

Gestione dell'I/O – 1

- UNIX tratta i dispositivi di I/O come *file* di tipo speciale, ciascun con posizione specifica nel FS
  - Per esempio `/dev/…`
  - *File* orientati a carattere (p.es. tastiera, rete, ...)
  - *File* orientati a blocco (p.es. disco)
- Un gestore (*device driver*) è associato in modo esclusivo a ciascun dispositivo o a famiglia di dispositivi dello stesso tipo
  - Una coppia di indici <`maggiore, minore`> identifica precisamente ciascun dispositivo di I/O

Gestione dell'I/O – 2

- GNU/Linux consente invece caricamento **dinamico** dei moduli di gestione dei dispositivi
  - Soluzione molto preferibile alla configurazione statica che richiede ogni volta una nuova compilazione dell'intero nucleo
    - Inevitabile a fronte della grande varietà di *hardware* attuale
- Il caricamento dinamico richiede al nucleo di effettuare diverse azioni di configurazione
  - **[1]** Rilocazione dello spazio di indirizzamento del modulo
  - **[1-2]** Allocazione delle risorse necessarie
    - P.es. interruzione assegnata al dispositivo
  - **[2]** Configurazione del vettore delle interruzioni
  - **[2]** Attivazione e inizializzazione del gestore

Gestione dell'I/O – 3

- Un *file* speciale (detto *socket*) viene utilizzato per la connessione di rete e i relativi protocolli
  - Può essere creato e distrutto dinamicamente
  - Un *socket* è associato a uno specifico indirizzo di rete
- Tre tipi di connessione con scelta alla creazione
  - Connessione affidabile a flusso di caratteri (~ **TCP**)
    - Il gestore garantisce la correttezza della trasmissione
    - Invio e ricezione per blocchi di dimensione variabile
  - Connessione affidabile a flusso di pacchetti (**TCP**)
    - Come sopra, ma con invio e ricezione solo per pacchetti
  - Trasmissione inaffidabile di pacchetti (**UDP**)
    - L'utente deve occuparsi di trattare gli eventuali errori

Gestione dell'I/O – 4

- Una zona di RAM è usata come *cache* dedicata per velocizzare R/W di dati su dispositivi a blocchi
  - Ogni blocco richiesto in lettura viene prima cercato in *cache*
  - Ogni blocco scritto viene trattenuto in *cache* il più a lungo possibile
    - Fin quando la *cache* è piena e serve spazio
    - Fino all'attivazione del *daemon* di scrittura su disco