

Cheat Sheet

Gabr1313

January 10, 2024

Programmazione concorrente

- Conviene guardare un contesto alla volta e spesso nel metre si trovano anche i deadlock.
- nel caso di una deadlock, se entrambi i thread sono bloccati si registrano i valori di global nel caso dei thread fermi a tali istruzioni. Se uno invece potrebbe essere terminato, si considerano i valori che global può assumere da quando esegue l'istruzione che blocca l'altro in poi.

Gestione dei processi

- Il padre prosegue l'esecuzione dopo la creazione del figlio
- Il task che da attesa passa a pronto viene messo in esecuzione
- Alla exit di un processo, anche i thread figli vengono terminati
- Quando un processo termina, anche se non scritta esplicitamente, bisogna riportare la `exit`

Moduli del kernel

- Segui lo schema della del cheat sheet
- prima di ogni ritorno in modalità U causato da `R_int()`, vai a `schedule` e `pick_next_task`. Al ritorno da `pick_next_task` salvi sulla pila l'USP e poi avviene il context switch.
- Scrivi sulla pila tutte le chiamate, e tira una riga sopra a quella da cui ritorni
- segui le frecce finchè bisogna (in base al contesto, non sempre fino alla fine)
 - `resched(set TIF_NEED_RESCHED)` avviene solo se effettivamente è necessario un rescheduling: se vieni da `check_preempt_curr` ci devono essere processi in stato di pronto, mentre se viene da `task_tick`, devi assicurarti che il timer sia scaduto per il quanto di tempo.
 - `TIF_NEED_RESCHED` è la condizione che determina la chiamate a `schedule` appena prima del ritorno in modalità U.
 - `schedule` setta sempre `TIF_NEED_RESCHED` a 0.
- si usa la notazione a `*1*` da `*2*` per indicare che la funzione 1 ha chiamato la funzione 2, e quindi sullo stack ci si salva l'indirizzo a cui ritornare nella funzione 1.
- `R_int(evento)`: chiamata prima `task_tick()` → ... e poi `controlla_timer()` →
L' `s_stack` è del tipo:
 - USP
 - ...
 - rientro a `R_int(CK)` da `schedule`
 - ...
 - PSR U
 - rientro a CU da `R_Int(evento)` (codice utente)

Mentre in un interrupt annidato

- ...
 - PSR S
 - rientro a `R_Int_1(evento)` da `R_Int_2(evento)`
 - ...
- `syscall`: dipende da `syscall` a `syscall`.
La chiamata a è del tipo:
 - > `*funzione_di_libreria*`
 - > `***`
 - > `syscall`
 - > `System_Call: SYSCALL`
 - > `sys_***`
 - ...

```

- > schedule
- > pick_next_task <
- ...

```

Il ritorno è del tipo:

```

- ...
- schedule <
- ...
- sys_*** <
- System_Call:  SYSRET <
- syscall <
- *** <
- *funzione_libreria* <

```

L' u_stack è del tipo:

```

- rientro a *** da syscall
- rientro a CU da ***
- ...

```

L' s_stack è del tipo:

```

- USP
- ...
- rientro a System_Call da sys_***
- PSR U
- rientro a syscall da System_Call

```

- quando un thread termina l'esecuzione::

```

- fn <
- clone (?)
- > syscall
- ...

```

- le strutture dati HW di un processo contengono:

```

- registro PC
- registro SP: cima dell'ultima pila (sia U o S) utilizzata
- SSP: base della pila S
- USP: cima della pila U
- PSR: process status register
- descrittore.stato: pronto, attesa(di cosa?)
- descrittore.sp: sp corrente (o nel momento della sospensione)
- descrittore.sp0: SSP

```

Memoria virtuale e file system

- **VMA:** `area NPV_iniziale | dim | R/W | P/S | M/A | <nome_file, offset>` (il nome del file è -1 se la pagina è anonima)
- **PT:** `process dim : n_phys_page -/D R/W`
- **MEMORIA FISICA:** `n : processo_area_n -/D`
- **SWAP FILE:** `s_n : processo_area_n`
- **TLB:** `processo_area_n : n - D A`
- **Dirty bit**
 - Finchè non avviene un `context_switch` questo viene salvato nel TLB.
 - Appena avviene il context switch il bit viene riportato sulla memoria fisica.
 - Essendo che i file non sono contenuti nel TLB, il bit di dirty viene sempre salvato sulla memoria fisica.
 - nella page table il bit di dirty si mette se e solo se è presente anche nella memoria fisica
- **pagine di pila (P)**
 - crescono verso il basso, quindi la pagina successiva ha un NPV minore di 1.
 - Nella pila serve sempre avere la prima pagina libera (pagina di growdown).
- **pagine condivise (S)**
 - le pagine **shared** possono essere abilitate alla scrittura anche se più processi le condividono.
 - le pagine non codivise (P), anche se mappate su file, quando vengono scritte fanno scattare COW.
- **kswapd**
 - gestisce tutte le pagine nelle LRU: la **active list** dalla coda, la **inactive list** dalla testa
 - se viene chiamato più volte, i bit di accesso delle pagine precedenti rimangono a 1 (come se venissero acceduti anche loro ripetutamente)
 - Il bit di **A** è salvato nel TLB, quindi pagine non presenti nel TLB è come se avessero bit di **A** a 0.
 - bisogna quindi porre attenzione allo stato del TLB prima di eseguire `kswapd`: le pagine con bit di **A** settato saranno in testa alla LRU corrispondente, mentre le altre in coda.
 - Scansione dalla coda della active list:

```
if (A) {
    A = 0;
    if (ref) move Page to the head of active list;
    else ref = 1;
} else {
```

```

        if (ref) ref = 0;
        else {
    ref = 1;
    move Page to the head of inactive list;
}
    }
}

```

- Scansione dalla testa della inactive list:

```

    if (A) {
        A = 0;
        if (ref) {
            ref = 0;
            move Page to the tail of active list;
        } else ref = 1;
    } else {
        if (ref) ref = 0;
        else move Page to the tail of inactive list;
    }
}

```

- **swap file**

- ci vanno solo le pagine anonime e dirty

- **read/write**

- le pagine vengono accedute una alla volta (il **pfra** non sa a prescindere quante pagine saranno accedute).
- **R** indica che la pagina ha permessi di sola lettura.
- **W** indica che la pagina ha permessi di lettura e scrittura.
- Le pagine **W**, dopo una **fork**, vengono marcate come **R**.

- **swap_in**

- finchè una pagina non viene scritta rimane duplicata sulla swap
- se una pagina in swap era "condivisa" tra 2 processi, e uno di questi fa una **write**, entrambe le pagine sono messe in memoria: **COW** (nel momento in cui avviene la **COW**, solo la pagina non scritta rimane in swap). Inoltre la pagina che viene scritta è messa in cima alla **LRU active** mentre quella non scritta in coda alla **LRU inactive**.

- **sbrk**

- alloca pagine **D** appena dopo le precedenti, o se non erano già presenti appena dopo **S**.
- Essendo le nuove pagina anonime, inizialmente non hanno indirizzo, ma si comportano come **COW** sulla **<ZP>** (una volta lette, sono mappate nella memoria fisica sulla **<ZP>**).

- **mmap(add, n_pag, W/R, P/S, M/A, file, file_page_offset)**

- alloca la pagina specificata, facendo attenzione che i 12 bit minori di **address** sono di offset nella pagina (non fanno quindi parte della NPV).

- `context switch`
 - svuota il TLB: se le pagine sono dirty marca le stesse pagine in memoria fisica come dirty.
 - carica prima la pagina del codice e poi quella della pila del nuovo processo in memoria fisica e quindi anche sul TLB. (attenzione, sulla pila ci scrive anche, quindi nel caso va tolta dalla swap)
- `fork`
 - le pagine vengono tutte "condivise" tra i processi (COW, R).
 - tutte le pagine vengono aggiunte in testa alla stessa LRU di quelle del padre, con lo stesso bit di ref.
 - la pagina della pila viene "regalata" al figlio, e clonata (COW) dal padre.
 - la pagina del figlio viene marcata D nella memoria fisica (scrive il PID).
 - il puntatore al descrittore dei file punta alla stessa area di memoria.
- `clone`
 - crea un nuovo thread: tutte le pagine vengono condivise con quella del padre (NON COW).
 - viene creata una nuova NPV (pila) di tipo T, di dimensione 2, W P A. Questa viene marcata D nella memoria fisica (come la pila di un nuovo processo in una `fork`).
 - dopo un `context switch` il TLB viene aggiornato (malgrado contenga pagine condivise).
- `pfra`
 - prima libera le pagine dei file, e poi quelle dei processi.
 - quando interviene libera fino ad avere `MAX_FREE` pagine.
 - non sa a prescindere quali pagine verranno usate: potrebbe mettere in swap la stessa pagina condivisa per poi ricaricarla in memoria fisica.
- `FILE`
 - `READ` indica che la pagina è stata caricata in memoria
 - `WRITE` indica che la pagina è stata salvata sul disco (avviene solo se la pagina è dirty)
 - se `f_count == 0` allora le pagine dirty vengono scritte in memoria (e quindi sparisce il bit D) e la `f_pos` si "resetta"
- `fopen`
 - se un file viene aperto, allora viene creato un nuovo descrittore, e quindi `f_pos = 0`.
- `fopen`
 - se un descrittore ha `f_count == 0` allora TUTTE le pagine del file associato vengono scritte su disco (anche se altri descrittori hanno alcune pagine dello stesso file aperte)

Struttura del filesystem

- prima di aprire un nuovo file, libera una posizione nel `P->files.fd_array`

Tabella della pagine

- l'indirizzo di una pagina è di 48 bit: 9 di offset e 36 di VMA. La VMA è divisa in:
 - PGD: 9 bit
 - PUD: 9 bit
 - PMD: 9 bit
 - PT: 9 bit
- $\#PGD = 1$
- $\#PUD = \#of_different_PGD$
- $\#PMD = \#of_different_PUD$
- $\#PT = \#of_different_PMD$
- $rapporto_di_occupazione = (\#PGD + \#PUD + \#PMD + \#PT) / numero_di_NPV$
- $dimensione_massima_del_processo_in_pagine_virtuali = 512 * \#PT$