

PROGETTO DI SISTEMI OPERATIVI

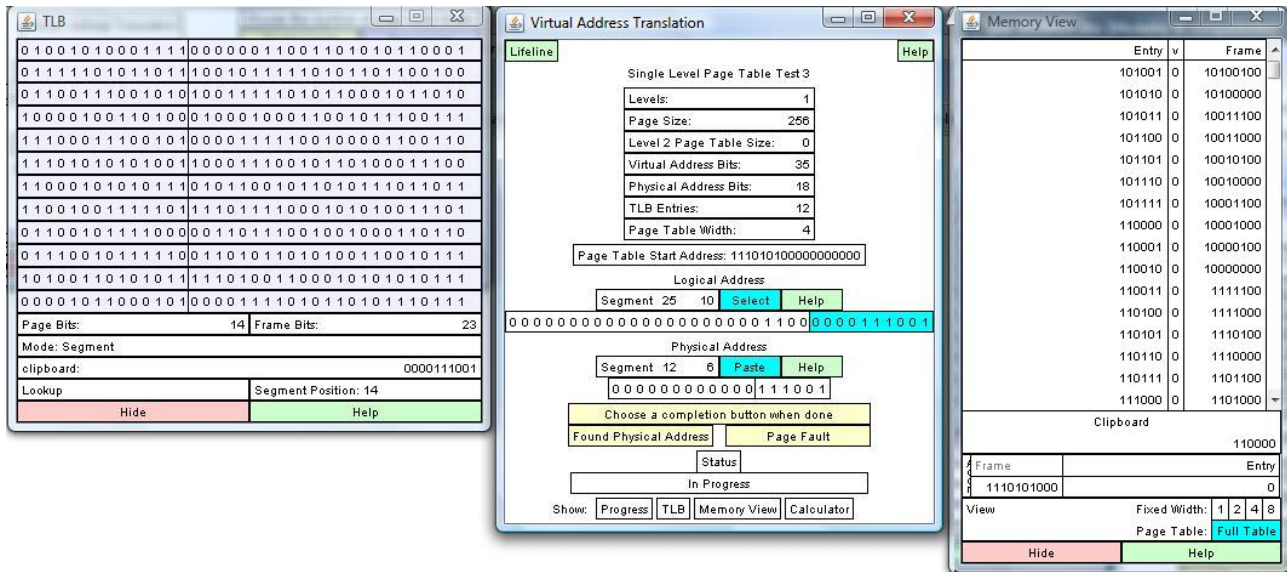
Ingegneria Informatica

21 aprile 2010

(teoria con risposte)

(si prega di rispondere descrivendo i passaggi e i risultati intermedi)

1. (6 punti) Sia data la figura che segue, nella quale si rappresenta una visualizzazione intermedia del simulatore di traduzione da indirizzi logici a fisici di UTSA.



- Nell'esecuzione proposta sono stati commessi due errori, uno nella segmentazione dell'indirizzo logico, uno nella configurazione della TLB. Si segnalino tali errori e si dica come vanno corretti.
- Si determinino le dimensioni degli spazi di indirizzamento logico e fisico.
- Si dica se l'indirizzo logico proposto genera TLB hit, oppure miss. Perché?
- Si determini se per l'indirizzo logico proposto si genererà page-fault oppure si troverà l'indirizzo fisico corrispondente (in tal caso si dica quale indirizzo fisico verrà generato)

Risposta

- Si ricorda che il simulatore proposto usa il termine "segmentazione" per indicare il partizionamento dell'indirizzo in pagina/offset. La dimensione delle pagine è 256. L'indirizzo logico va quindi scomposto in 27/8 anziché 25/10 (anche l'indirizzo fisico andrà segmentato coerentemente in 10/8). La TLB va segmentata in 27/10 bit, anziché 14/23.
- Gli spazi di indirizzamento logico e fisico sono di 2^{35} Byte (32 GByte) e 2^{18} Byte (256 KByte).
- TLB miss, in quanto, delle 12 righe, nessuna indica pagina logica (primi 27 bit) = 000...0110000.
- Page-fault, in quanto la Page Table (memory view) contiene la pagina logica cercata (110000), ma con bit di validità falso (v=0).

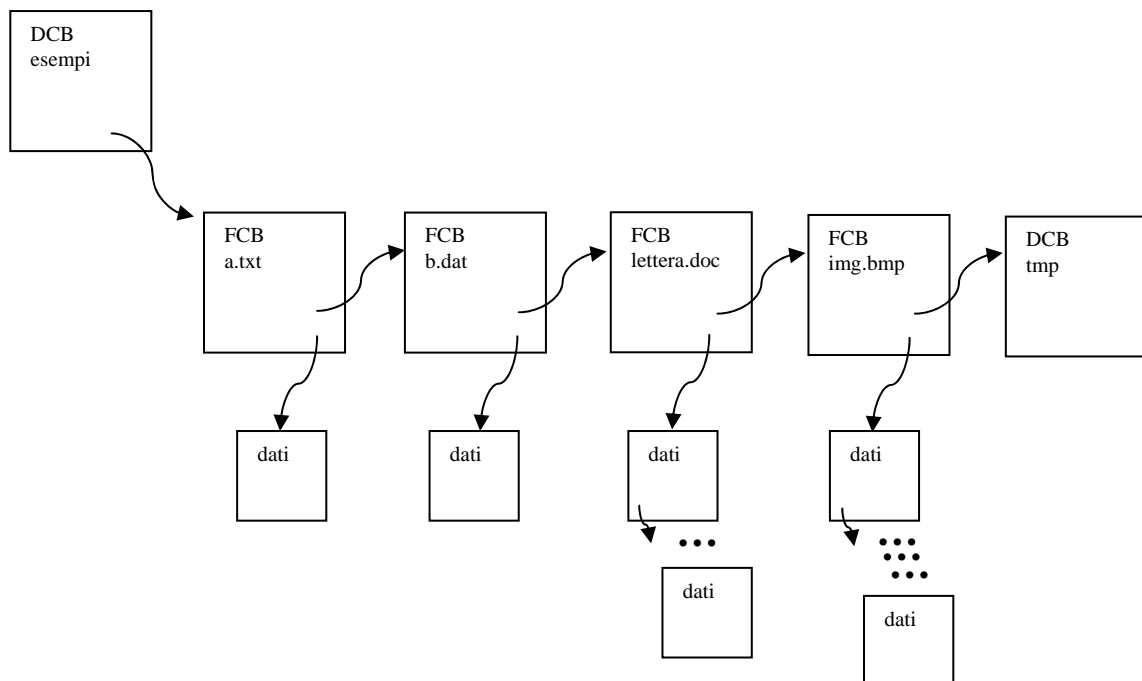
2. (6 punti) Si descrivano brevemente, nell'ambito di un file system, differenze e/o similitudini tra allocazione contigua, allocazione a blocchi e allocazione basata su extent.

Si supponga poi che un file system allochi sia i direttori che i file mediante blocchi di dimensione 2 Kbyte: Un file viene realizzato mediante un blocco di controllo (FCB: File Control Block) contenente varie informazioni, tra cui nome file, informazioni di protezione, puntatore alla lista dei blocchi di dato. Un direttorio è realizzato mediante un blocco principale (DCB: Directory Control Block), contenente, tra le altre informazioni, il nome del direttorio, le informazioni di protezione, il puntatore a una lista di DCB/FCB, uno per sotto-direttorio/file contenuto.

Si rappresenti graficamente lo schema di allocazione e si calcoli il numero totale di blocchi complessivamente allocati (per direttori e file) per un direttorio /esempi, contenente 4 file, di nome "a.txt", "b.dat", "lettera.doc", "img.bmp" (e di dimensione 150, 1045, 12000 e 324056 Byte, rispettivamente) e un sottodirettorio (vuoto) di nome "tmp".

Risposta

- Allocazione contigua: il file viene allocato in blocchi contigui su disco. Basta immagazzinare puntatore (o indice) all'inizio del file e dimensione del file. Va bene per accesso diretto (e anche per il sequenziale, in quanto si garantisce località di accessi a disco). Problema: frammentazione esterna.
- Allocazione a blocchi. File partizionato in blocchi di dimensione fissa, gestiti in liste linkate (bene accesso sequenziale, lento l'accesso diretto) o mediante indici (OK per accesso sia diretto che sequenziale).
- Allocazione a extent. Strategia intermedia. Un extent è un insieme di blocchi contigui. Un file è composto da più extent (non necessariamente contigui) serve a migliorare la località degli accessi.
- Nell'esempio proposto si usano liste linkate sia per gestire i contenuti dei direttori che per i blocchi di dato di un file.
 - Rappresentazione grafica:



- In totale sono allocati 2 DCB (ignorando la root /) e 4 FCB. Per i dati dei 4 file sono allocati, rispettivamente, 1, 1, 6 ($12000/(2048-4)$), 159 ($324056/(2048-4)$) blocchi (si sono considerati puntatori su 4 byte, ma il risultato non cambierebbe con altre dimensioni). Il numero totale di blocchi allocati è 173.
3. (6 punti) Cinque job (processi), identificati dalle lettere A-E, arrivano all'elaboratore agli istanti di tempo 5, 3, 2, 1, 4, rispettivamente. I processi hanno tempi di esecuzione di 6, 16, 15, 8 e 20 unità di tempo, rispettivamente. I processi B ed E, effettuano una richiesta di I/O ogni 6 unità di tempo di esecuzione. Si supponga che tali richieste di I/O siano soddisfatte in 5 unità di tempo. Descrivere (mediante diagramma di Gantt) la sequenza di esecuzione dei job su un sistema dotato di 2 CPU e calcolare i tempi di turnaround individuale (per ognuno dei processi) e globale, trascurando i tempi dovuti allo scambio di contesto, per una schedulazione di tipo multi-level feedback queue, con 2 code, gestite, rispettivamente, con criterio round-robin (coda A: preemption con quanto di tempo 4) e FCFS (coda B: gestione FIFO senza preemption). La coda A ha priorità sulla B. Ogni processo interrotto per preemption passa alla coda B, mentre al completamento di I/O passa alla coda A.

Risposta

Si considerano il caso di symmetric multiprocessing, in cui ogni CPU burst può essere eseguito in modo indifferente su una qualunque delle due CPU. Tutti i job fanno un burst di 4 istanti, terminato con preemption (time-out), dopo di che andranno in Q_B , da cui saranno inviati in esecuzione senza preemption. Tuttavia B ed E faranno richieste di I/O che, una volta terminati, riportano i job (tramite Q_A) in esecuzione con preemption. La soluzione viene analizzata con dettaglio superiore a quello richiesto all'esame (un carattere sottolineato indica termine del job).

- All'inizio tutti i job passano (secondo l'ordine di arrivo: D,C,B,E,A) dalla coda Q_A ed eseguono un CPU burst di durata 4, terminato il quale, con preemption, si mettono in coda in Q_B

```

-----1--
0123456789012
CPU0  DDDDBBBBAAAA
CPU1  CCCCCEEE

```

- La coda Q_B ordina i job con lo stesso ordine di arrivo (D, C e A eseguono burst di durata 4, 11 e 2, sino al termine), B ed E un burst di durata 2, sino alla prima richiesta di IO.

```

-----1-----2---
012345678901234567890123
CPU0  DDDDBBBBAAAACCCCCCCCCC
CPU1  CCCCCEEEEDDDDBBEEAA

```

- B ed E iniziano IO agli istanti 16 e 18, terminano IO (e vanno in coda Q_A) agli istanti 21 e 23. B viene immediatamente eseguito in quanto CPU1 è libera (a 21).

```

-----1-----2----
0123456789012345678901234
CPU0  DDDDBBBBAAAACCCCCCCCCC
CPU1  CCCCCEEEEDDDDBBEEAA-BBBB

```

- All'istante 24 C termina ed E (da Q_A) va in esecuzione su CPU0. A 25, B viene preempted, va in Q_B , da cui torna immediatamente in esecuzione (CPU1 è libera). Analogamente E all'istante 28.

```

-----1-----2-----
012345678901234567890123456789
CPU0  DDDDBBBBAAAACCCCCCCCCCEEEEEEE
CPU1  CCCCCEEEEDDDDBBEEAA-BBBBBB

```

- A 27 B esegue IO. Per tutti gli IO di qui in avanti, sia B che E attenderanno 5 istanti (esecuzione IO) poi passeranno in Q_A da cui andranno immediatamente in esecuzione su una CPU libera. Nei casi di preemption, passano da Q_B e rientrano immediatamente in esecuzione. Per semplicità di soluzione, nei casi in cui la scelta di CPU sia arbitraria, E resta su CPU0 e B su CPU1.

```

-----1-----2-----3-----4-----
012345678901234567890123456789012345678
CPU0  DDDDBBBBAAAACCCCCCCCCCEEEEEEE-----EEEEEE-----EE
CPU1  CCCCCEEEEDDDDBBEEAA-BBBBBB-----BBBB

```

La tabella mostra i tempi di arrivo-completamento dei job, e turnaround individuale:

A: 5-20	15
B: 3-36	33
C: 2-24	22
D: 1-14	13
E: 4-48	44

Il turnaround globale è 127.

4. (6 punti) Sia data la figura che segue, nella quale si visualizza una videata di debug di os161, relativa all'esecuzione del comando "p testbin/palin". Nello specifico, si tratta di un breakpoint sulla funzione syscall, con visualizzazione del trapframe tf, e del dato puntato da tf->tf_a0.

```

DDD: /home/pso/os161/os161-base-1.99.05/kern/arch/mips/syscall/syscall.c
File Edit View Program Commands Status Source Data Help
(): main
Lookup Find Break Watch Print Display Plot Show Rotate Set Undisp
syscall(struct trapframe *tf)
{
    int callno;
    int32_t retval;
    int err;

    KASSERT(curthread != NULL);
    KASSERT(curthread->t_curspl == 0);
    KASSERT(curthread->t_ip1high_count == 0);

    callno = tf->tf_v0;

Breakpoint 1, syscall (tf=0x80036f68) at ../../arch/mips/syscall/syscall.c:1
Current language: auto; currently c
(gdb) p *tf
$1 = {tf_vaddr = 0, tf_status = 65292, tf_cause = 32, tf_lo = 0, tf_hi = 0,
tf_ra = 4194996, tf_at = 4456448, tf_v0 = 55, tf_v1 = 0, tf_a0 = 1, tf_a1 =
2147483528, tf_a2 = 1, tf_a3 = 0, tf_t0 = 4294967288, tf_t1 = 0, tf_t2 = 0,
tf_t3 = 0, tf_t4 = 0, tf_t5 = 0, tf_t6 = 0, tf_t7 = 0, tf_s0 = 87, tf_s1 = 0,
tf_s2 = 0, tf_s3 = 0, tf_s4 = 0, tf_s5 = 0, tf_s6 = 0, tf_s7 = 0, tf_t8 = 0,
tf_t9 = 0, tf_k0 = 2147680256, tf_k1 = 2147483512, tf_gp = 4505408, tf_sp =
2147483512, tf_s8 = 0, tf_epc = 4195232}
(gdb) p *(char *) (tf->tf_a1)
$2 = 87 'W'
(gdb)
$2 = 87 'W'

```

Si risponda, in relazione a tale esempio, alle domande seguenti:

- Il trapframe si trova in memoria user oppure kernel ? Perché ?
- La funzione syscall viene eseguita in un programma utente, oppure è parte del kernel ?
- A quale system call (indicata da un numero intero) fa riferimento l'esempio ?
- Si spieghi brevemente che cosa contiene la parte successiva (non visualizzata) della funzione syscall, nella versione base di os161, e (facendo riferimento al laboratorio n.2) come va completata per gestire opportunamente i dati nel trapframe proposto.
- Quale dovrebbe essere l'azione svolta da syscall (e dalle funzioni eventualmente chiamate da questa) una volta completata, come descritto al punto precedente ?

Risposta

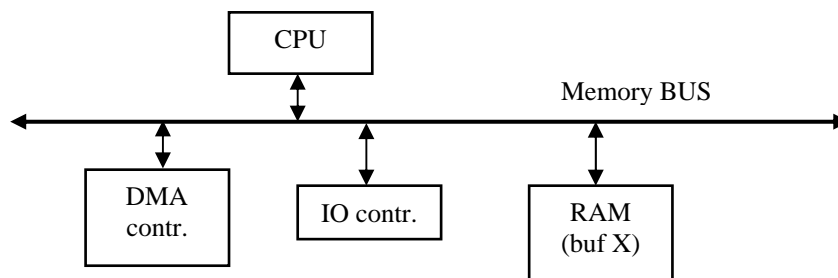
- Il trapframe si trova in memoria kernel (nel kernel stack), più in particolare nel segmento kseg0, che in OS161 va dall'indirizzo logico 0x80000000 a 0x9ffffff. Il trapframe si trova infatti all'indirizzo 0x80036f68 (visualizzato dal debugger).
- La funzione syscall è parte del kernel, in quanto appartiene alla parte di chiamata a sistema attivata da trap (interrupt software). Nella figura, questo è confermato dal direttorio in cui si trova syscall.c (in "kern"), dal fatto che syscall richiama asserzioni in kernel (KASSERT), nonché dal fatto che riceve come parametro un indirizzo logico in kseg0 (tf).
- Alla system call 55 (SYS_write), contenuta in tf->v0, poi assegnata a callno.
- La parte successiva di syscall contiene un costrutto switch/case, che seleziona in base al valore di callno (nel caso proposto 55), in modo da effettuare il compito relativo alla system call. I parametri della system call possono essere reperiti nel trapframe (tf_a0, tf_a1, tf_a2, ...).

- Nel caso proposto (SYS_write) occorre chiamare una funzione in grado di stampare su file una stringa (file e stringa sono identificati dai parametri in tf_a0, ...). La versione semplificata di lab.2 stampa a video mediante kprintf. Al termine syscall controlla eventuali errori e ritorna.

5. (6 punti) Si descriva, nel contesto della gestione dei dispositivi di I/O, la differenza tra I/O programmato (programmed) e in DMA. Facendo riferimento ad un I/O a blocchi in DMA, si descrivano le azioni svolte da CPU, DMA controller, memoria RAM e controller del dispositivo di I/O (es. disco). Si rappresenti, in relazione a tale tipo di trasferimento, uno schema architetturale, indicando la sequenza delle azioni svolte dai vari dispositivi. Perché il trasferimento in DMA risulta vantaggioso rispetto al quello di tipo programmato? Mentre viene eseguito il trasferimento in DMA, in che stato si trova il processo in attesa di I/O? E la CPU che cosa esegue? Il processo in esecuzione in CPU viene rallentato dal trasferimento in DMA, rispetto a un'eventuale esecuzione senza tale trasferimento in concorrenza? Perché?

Risposta

- Con "programmed IO" (PIO, o IO programmato), si intende un IO gestito direttamente dalla CPU, nel quale un flusso di dati tra memoria principale (RAM) e dispositivo di IO (es. disco, porta di comunicazione, ...) transita nella CPU. Ad esempio, per inviare 1000 caratteri a un dispositivo di IO, la CPU effettua un ciclo, in cui ad ogni iterazione legge un carattere (istruzione di tipo load) da RAM a un registro interno, poi lo invia al dispositivo (istruzione di tipo store o out).
- Il DMA controller consente di effettuare un trasferimento RAM-IO senza il controllo diretto della CPU (e senza far passare i dati nei registri della CPU, con 2 passaggi per ogni dato!). Il DMA controller effettua questo abilitando contemporaneamente RAM e dispositivo di IO ad utilizzare i bus (es. RAM in lettura e dispositivo in scrittura, o viceversa). Ogni dato, a differenza di PIO, transita una sola volta sul bus dati.
- Schema architetturale. Lo schema è semplificato, in quanto non si considera un secondo bus per i dispositivi di I/O (es. IDE)



- Sequenza di azioni: (1) il device driver riceve richiesta di trasferire C dati tra buf X (in RAM) e IO. (2) il device driver comanda il dispositivo di IO (trasferire C dati). (3) Il disp. Di IO inizia il trasferimento. (4) il DMA controller controlla il trasferimento dei dati in RAM a partire da X, tenendo un conteggio dei dati. (5) a trasferimento terminato il DMA controller interrompe la CPU
NB. Nel caso di un secondo bus per IO, il DMA controller può essere integrato con il bus controller. Ha accesso (da due parti) al memory-bus e all'IO-bus. I dati "passano quindi fisicamente nel DMA/BUS-controller.
- Il trasferimento in DMA è vantaggioso rispetto a PIO perché non fa transitare i dati nei registri della CPU (quindi dimezza i dati in transito sul memory-bus) e perché lascia la CPU libera di fare altro. Mentre si effettua trasferimento in DMA, il processo richiedente P0 va in wait (IO sincrono/bloccante) oppure continua (IO asincrono/non-bloccante). In ogni caso la CPU può lavorare (in concorrenza) su un processo P (P0 se non in wait, o un altro eventualmente ready). Il processo in esecuzione viene leggermente rallentato, in quanto deve accedere alla RAM (per istruzioni ed eventuali dati), in concorrenza con il trasferimento in DMA: pur essendo tutto gestito via HW, vi è un rallentamento rispetto al caso in cui P venga eseguito senza DMA in concorrenza.

PROGETTO DI SISTEMI OPERATIVI
Ingegneria Informatica (a.a. 2008/2009 e precedenti)
21 aprile 2010

(teoria)

(si prega di rispondere descrivendo i passaggi e i risultati intermedi)

1. (6 punti) Sia dato un sistema di memoria virtuale con paginazione a richiesta (*demand paging*). Sia data la seguente sequenza di riferimenti a pagine: 2, 4, 3, 5, 3, 5, 3, 4, 5, 2, 2, 5, 1, 2, 3, 2, 1, 4. Sapendo che il programma dispone di 3 frame, e che si utilizza come algoritmo di rimpiazzamento LFU (Least Frequently Used), nel quale, per ogni pagina nel resident set, si mantiene un contatore di riferimenti a tale pagina.
 - rappresentare il resident set (e i contatori di accessi) dopo ogni riferimento a pagina virtuale
 - determinare quanti page fault si verificheranno.
2. (6 punti) Cinque processi, identificati dalle lettere A-E, arrivano all'elaboratore agli istanti di tempo 5, 3, 2, 1, 4, rispettivamente. I processi hanno tempi di esecuzione di 6, 16, 15, 8 e 20 unità di tempo, rispettivamente. I processi A e B, effettuano una richiesta di I/O dopo le prime 4 unità di tempo di esecuzione. Si supponga che tali richieste di I/O siano soddisfatte in 5 unità di tempo. Descrivere (mediante diagramma di Gantt) la sequenza di esecuzione dei job su un sistema dotato di 2 CPU e calcolare i tempi di turnaround individuale (per ognuno dei processi) e globale, trascurando i tempi dovuti allo scambio di contesto, per una schedulazione di tipo multi-level feedback queue, con 2 code, gestite, rispettivamente, con criterio round-robin (coda A: preemption con quanto di tempo 4) e FCFS (coda B: gestione FIFO senza preemption). La coda A ha priorità sulla B. Ogni processo interrotto per preemption passa alla coda B, mentre al completamento di I/O passa alla coda A.
3. (6 punti) Si descrivano brevemente, nell'ambito di un file system, differenze e/o similitudini tra allocazione contigua e allocazione a blocchi.

Si supponga poi che un file system allochi sia i direttori che i file mediante blocchi di dimensione 2 Kbyte: Un file viene realizzato mediante un blocco di controllo (FCB: File Control Block) contenente varie informazioni, tra cui nome file, informazioni di protezione, puntatore alla lista dei blocchi di dato. Un direttorio è realizzato mediante un blocco principale (DCB: Directory Control Block), contenente, tra le altre informazioni, il nome del direttorio, le informazioni di protezione, il puntatore a una lista di DCB/FCB, uno per sotto-direttorio/file contenuto.

Si rappresenti graficamente lo schema di allocazione e si calcoli il numero totale di blocchi complessivamente allocati (per direttori e file) per un direttorio /esempi, contenente 4 file, di nome "a.txt", "b.dat", "lettera.doc", "img.bmp" (e di dimensione 150, 1045, 12000 e 324056 Byte, rispettivamente) e un sottodirettorio (vuoto) di nome "tmp".
4. (6 punti) Si descriva il compito della funzione getblk (di cui si riporta il prototipo nel seguito) nell'ambito della gestione del buffer cache unix

```
struct buf *getblk (dev_t dev, daddr_t blkno);
```

Si descrivano, in particolare, i parametri e il valore ritornato. Si descriva infine l'algoritmo implementato, mettendo in evidenza i 5 sotto-casi gestiti dalla funzione.
5. (6 punti) Si risponda, a scelta, a una delle due domande seguenti (A oppure B)
 - (A) Si confrontino i meccanismi di gestione della memoria implementati nei sistemi Windows e Linux, descrivendo, in particolare:
 - La suddivisione dello spazio di indirizzamento logico tra user e kernel (system) space, nel caso di indirizzi a 32 bit.
 - Le caratteristiche della paginazione on-demand
 - Gli algoritmi di rimpiazzamento (in caso di page-fault)
 - La possibilità di fare swap in/out di interi processi.
 - (B) Si descriva, nel contesto Unix, la struttura di un driver a caratteri, basato su "line discipline", in modalità sia raw che canonica. Si dica, in tale contesto, che cosa sono i c-block e le c-list.