Programmazione di Sistema
9 luglio 2018 (teoria)

NOTA: le risposte sono fornite in forma compatta, mettendone in evidenza gli aspetti essenziali

Si prega di rispondere in maniera leggibile, descrivendo i passaggi e i risultati intermedi. Non è possibile consultare alcun materiale. Durata della prova: 70 minuti. Sufficienza con punteggio >= 8. Le due parti possono essere sostenute in appelli diversi. La presenza a una delle due parti annulla automaticamente un'eventuale sufficienza già ottenuta (per la stessa parte): viene intesa come rifiuto del voto precedente.

1. *(4 punti)* Siano dati i frammenti di programma (A e B) rappresentati, che calcolano nel vettore V, rispettivamente, le medie per riga/colonna dei dati in una matrice triangolare inferiore/superiore M *(si ricorda che, in modo informale, una matrice si dice triangolare superiore/inferiore se tutti gli elementi sotto/sopra la diagonale sono nulli):*

A	B

`float M[512][512],V[512];`
`...`
`for (j=0; j<512; j++) V[j]=0;`
`for (k=1; k<512*512; k=k*2) {`
`  i = k/512;`
`  j = k%512;`
`  V[i] += M[i][j];`
`}`
`...`
	`float M[512][512],V[512];`
	`...`
	`for (i=511; i>=0; i--) V[i]=0;`
	`for (k=512*512-1; k>0; k=k/2) {`
	`  j = k/512;`
	`  i = k%512;`
	`  V[j] += M[j][i];`
	`}`
	`...`

NOTA: Si sono corretti gli errori contenuti nel testo originale (dimensioni di matrice e vettore portati da 1024 a 512, test di fine iterazione da k>=0 a k>0, entrambi per il caso B). Nella correzione del compito si sono considerate corrette tutte le interpretazioni. Si fa notare che, a parte la dimensione di vettore e matrice (che varia di un fattore 4), la dimensione 1024 o 512 non ha impatto sul resto del problema (se non, in forma molto limitata, nell'accedere o meno alla pagina comuna al vettore V e alla matrice M).

I programmi eseguibili generati da tali sorgenti sono eseguiti in un sistema con memoria virtuale gestita mediante paginazione, con **pagine di 1Kbyte**, utilizzando come politica di sostituzione pagine la **FIFO**. Si sa che un `float` ha dimensione **32 bit** e che le istruzioni in codice macchina, corrispondenti al frammento di programma visualizzato, sono contenute in una sola pagina. Si supponga i che `M` e `V` siano allocati ad indirizzi logici contigui (prima `M`, poi `V`), a partire dall'indirizzo logico 0x7474CB00. La matrice `M` è allocata secondo la strategia "row major", cioè per righe (prima riga, seguita da seconda riga, …).

- Quante pagine (e frame) sono necessarie per contenere la matrice e il vettore ?

R1	|V| = 512*sizeof(float) = 2KB = 2KB/1KB pagine = 2 pagine |M| = 512*512*sizeof(float) = 1MB = 1MB/1KB pagine = 1K pagine = 1024 pagine L'indirizzo di partenza 0x7474CB00 non è multiplo di pagina, quindi si utilizza una pagina in più (con una pagina condivisa tra V e M): 1027 pagine

- Ipotizzando che le variabili i, j siano allocate in registri della CPU, quanti accessi in memoria (in lettura e scrittura) fanno il programma proposto, per accedere a dati (non vanno conteggiati gli accessi a istruzioni) ?

R2	Nota: si ipotizza che anche k sia in un registro (in caso contrario vanno conteggiati anche gli accessi a k)
	A:
	`for (j=0; j<512; j++)  // 512 iterazioni`
	`V[j]=0;  // 1 Write`
	`for (k=1; k<512*512; k=k*2)  // 18 iterazioni (log(512*512))`
	`V[i] += M[i][j];  // 1 Read si V, 1 Write su V, 1 Read su M`
	
	In totale: 512W + 18*(2R+1W) = 566 operazioni
	B: risultati identici. La sequenza per i valori di k è leggermente diversa ma il numero di iterazioni coincide.

- Detto N_T il numero totale di accessi a dati in memoria, ed N_L il numero di accessi a dati nella stessa pagina di uno dei precedenti 10 accessi, si definisca come **località** del programma per i dati il numero $L = N_L/N_T$. Si calcoli la località del programma proposto.

R3	A: Accessi a V: 3 sui primi 512 sono NON locali (2 + 1 per l'allineamento degli indirizzi). Nelle 18 iterazioni su k, 2 volte si accede a V in modo NON locale. Accessi a M: fino a k==8 tutti gli accessi sono nella prima pagina, da k==9 in poi si cambia sempre pagina (anche tenendo conto dell'allineamento in memoria e della pagina condivisa tra V e M, il computo totale non cambia). In totale 1 (prima, o seconda per l'allineamento, pagina di M) + 9 accessi NON locali a M $L = N_L/N_T = (566 – (3 + 2 + 1 + 9))/566 = 551/566 = 9,7$ (si sono considerati corretti tutti i risultati vicini a questo, purché basati su un ragionamento corretto. B: il tipo di accessi è molto simile. Viste le sequenze discendenti possono variare in modo trascurabile i conteggi degli accessi non locali.

- Calcolare il numero di page fault generati dal programma proposto, supponendo che siano allocati per esso **10 frame**, di cui uno utilizzato (già all'inizio dell'esecuzione) per le istruzioni. (motivare la risposta)

R4	Dato il tipo di politica e il numero di frame disponibili, i page fault coincidono in questo caso con gli accessi non locali, quindi 15 PF (sia caso A che B). Sono possibili variazioni di 1 o 2 a seconda di come si affettuano i conteggi e di come si tiene conto dell'allineamento della pagine .

2. *(3 punti)* Si consideri il problema della gestione dei blocchi liberi su un volume con file system, Si confrontino la soluzione mediante FAT e quella con bitmap (vettore di bit) per un sistema basato su iNode:
    - Si descrivano brevemente entrambe le soluzioni

R1	Una Bitmap è un vettore di bit (realizzato come vettore di word o byte, in quanto le RAM non forniscono indirizzamento del bit) che associa a ogni frame in RAM un bit 0/1 (libero/occupato). La Bitmap, inizialmente su disco, viene copiata in RAM (memoria kernel). Una FAT è un vettore di indici di blocchi, in cui si realizzano liste concatenate, sia per i blocchi allocati a file, che per la "free list".

    - Ne si confronti l'efficienza nel caso di ricerca di un blocco libero, ricerca di N blocchi liberi contigui, allocazione di un N blocchi contigui

R2	La Bitmap fornisce accesso diretto, dato il blocco, ma questo non serve nelle operazioni di ricerca. Ricerca di uno o di N frame contigui liberi si effettua con un algoritmo lineare (O(NF), con NF numero di frame). Le ricerche su bitmap sono tuttavia agevolate da istruzioni efficienti di manipolazione e localizzazione di 0/1 e intervalli di 0/1 all'interno di una word. In una FAT la ricerca di un frame libero è O(1) (primo frame in lista). Per la ricerca di N frame contigui si possono considerare 2 casi: • Liste non ordinate: la ricerca sarà $O(NF^2)$, in quanto per ogni frame libero devo cercare (con una scansione lineare, altri eventuali frame contigui • Liste ordinate: la ricerca sarà O(NF), ma per gestire la lista ordinata diventerà O(NF) (anziché O(1)) la liberazione di un frame, con inserimento in lista ordinata • Sono possibili altre soluzioni o verianti, che richiedono tuttavia strutture dati aggiuntive. L'allocazione di un intervallo di N blocchi contigui viene fatta, dopo la ricerca, modificando (con accesso diretto) i bit corrispondenti (0 o 1) nella Bitmap, oppure rimuovendo (per allocarli ai file) elementi nella free list, nel caso della FAT. Se la FAT è ordinata, si rimuove un intero intervallo (più semplice ed O(N), l'operazione è meno semplice con lista non ordinata (di fatto occorrerà fare una nuova ricerca, con rimozione di blocchi selezionati, e costo O(N*NF)).

    - Supponendo che ad un certo istante, su un volume da 200 GB, con blocchi di 2KB, ci siano 20M blocchi liberi, si calcoli la percentuale di FAT dedicata alla free list e la dimensione totale della bitmap.

R3	N blocchi totali = 200GB/2KB = 100M |Bitmap| = 100M * 1bit = 100Mbit = 12,5 MB La free list occupa 20M blocchi su 100M totali, quindi il 20% della FAT

3. *(3 punti) (le riposte si/no vanno motivate)* Si considerino i due tipi di sincronizzazione, relativi ad operazioni di I/O: sincrono e asincrono. Si spieghino le principali differenze tra gli I/O dei due tipi. Si definisca poi I/O bloccante e non bloccante: si tratta di sinonimi di asincrono e sincrono? Ci sono differenze (tra bloccante/non-bloccante e sincrono/asincrono)? (se no, perché, se si, quali?).

R1	In modo molto sintetico, si può affermare che • IO bloccante e sincrono siano equivalenti: il processo che effettua IO ne attende il completamento, in stato di wait. • IO non bloccante permette al processo di proseguire, IO asincrono è di fatto NON bloccante, con l'aggiunta di tecniche che permettano di gestire (successivamente) il completamento dell'IO. Tali tecniche sono: funzioni di `wait` (dipendono dal sistema operativo) che consentano di attendere il completamento dell'IO, oppure funzioni di tipo *"callback",* richiamate in modo automatico dal sistema, al completamento dell'IO (attenzione: sono funzioni che vanno scritte dallo user)

Un I/O sincrono può essere effettuato in polling oppure è necessario effettuarlo in interrupt? (si risponda alla stessa domanda per in caso dell'I/O asincrono).

R2	Polling e interrupt sono due modalità diverse per gestire un dispositivo di IO. Ovviamente il polling risulta meno efficiente, con rare eccezioni. Si tratta tuttavia di un problema interno ai driver (moduli del Kernel) e quindi indipendenti dal processo user. In sostanza, realizzare una system call read/write in modo sincrono o asincrono, può esser fatto con driver che lavorino sia in modo polling che interrupt,

In quale modo può un processo beneficiare di un I/O asincrono ? E' possibile, nel caso di I/O asincrono, scrivere un programma con istruzioni, successive all'I/O, che dipendano dai dati coinvolti (ad es. letti) durante l'I/O ?

R3	Il processo può beneficiare in quanto può eseguire altre istruzioni mentre l'IO è in corso. Si tratta quindi di una possibile forma di concorrenza. Il processo non può, durante l'IO, utilizzare i dati coinvolti. Se deve farlo, occorre sincronizzarsi (e aspettare) sulla relativa (dell'IO asincrono) operazione di wait.

4. *(4 punti)* Sia dato un sistema operativo OS161.

4.a) Dato il seguente frammento di codice relativo alla funzione `as_define_region` (si riportano per completezza due definizioni dal file `vm.h`).

```
#define PAGE_SIZE 4096 /* size of VM page */
#define PAGE_FRAME 0xfffff000 /* mask for getting page number from addr */
...
as_define_region(struct addrspace *as, vaddr_t vaddr, size_t sz,
          int readable, int writeable, int executable) {
  size_t npages;
  ...
  /* Align the region. First, the base... */
  sz += vaddr & ~(vaddr_t)PAGE_FRAME;
  vaddr &= PAGE_FRAME;
  /* ...and now the length. */
  sz = (sz + PAGE_SIZE - 1) & PAGE_FRAME;
  npages = sz / PAGE_SIZE;
  ...
  if (as->as_vbase1 == 0) {
     as->as_vbase1 = vaddr;
     as->as_npages1 = npages;
     return 0;
  }
  ...
}
```

Si spieghi brevemente che cosa svolgono le istruzioni riportate.

R1	La `as_define_region` assegna a un address space gli intervalli di indirizzi logici per i due segmenti (codice e dato, 1 e 2). Le istruzioni riportate servono ad allineare i segmenti alle pagine (frame). `vaddr` viene riportato (retrocesso) all'inizio della pagina, azzerandone i bit meno significativi, `sz` viene incrementato in modo da essere multiplo di pagina.

Perché si controlla il valore di `as->as_vbase1` per decidere se assegnare o meno `as->as_vbase1` e `as->as_npages1`?

R2	Perché nella versione attuale `as_define_region` viene chiamata due volte, una per il segmento di codice (1) e una per il segmento di dati (2, parte non rappresentata). Internamente, la scelta se assegnare `as->as_vbase1` e `as->as_npages1`, oppure `as->as_vbase2` e `as->as_npages2`, viene fatta su pura base cronologica (si assegna il primo ancora nullo, partendo da as->as_vbase1.

	Ovviamente si tratta anche di un controllo di errore, ma normalmente, quando si trova `as->as_vbase1` non nullo, non si tratta di errore, ma di assegnazione al secondo segmento.

Supponendo che `vaddr` contenga il valore `0x612480` e che `sz` contenga il valore `8200` (decimale), si dica quali saranno i valori di `vaddr`, `sz` e `npages` al termine delle istruzioni riportate. Supponendo inoltre che `as->as_vbase1` contenga `0` e la successiva `as_prepare_load` ottenga per `as->as_pbase1` il valore `0x52000`, si dica a quale indirizzo fisico corrisponderà l'indirizzo logico `0x617500`.

R3	Si converte tutto in esadecimale: sz = 8200 = 8192+8 = 0x2008, PAGE_SIZE = 4096 = 0x1000
	`sz += vaddr & ~(vaddr_t)PAGE_FRAME; // 0x2008+0x480 -> 0x2488`
	`vaddr` `&= PAGE_FRAME; //` **`0x612000`**
	`sz` `= (sz + PAGE_SIZE - 1) & PAGE_FRAME; //(0x2488+0xFFF)&0xFFFFF000 ->` **`0x3000`**
	`npages` `= sz / PAGE_SIZE; // 0x3000/0x1000 ->` **`3`**
	
	Se `as->as_vbase1` contiene `0` si sta assegnando il segmento di codice (1). Ad `as->as_vbase1` verrà quindi assegnato `vaddr = 0x612000`, cui corrisponderà successivamente `as->as_pbase1 = 0x52000`. L'indirizzo (user!) proposto è fuori dal segmento 1 in quanto dista più di 3 pagine da `0x612000`. Non è quindi possibile tradurlo coi dati a disposizione. Se si suppone che l'indirizzo sia nel segmento 2 (dati) e che tale segmento sia adiacente al primo, sia nello spazio logico che in quello fisico, allora l'indirizzo fisico sarà `0x57500`.

4.b) Dato il seguente frammento di codice relativo alla funzione `cv_signal` (il codice potrebbe contenere errori), si spieghi brevemente il funzionamento della funzione.

R4	La funzione `cv_signal` serve a segnalare che la condizione a cui è associata una condition variable è vera. Questo sblocca uno (uno solo) dei thread eventualmente in attesa su una `cv_wait`.

Si spieghi il ruolo del lock ricevuto come parametro dalle funzioni `cv_wait` e `cv_signal`.

R5	Il lock serve a garantire la mutua esclusione sulla condizione logica associata alla condition variable. Viene passato alla `cv_wait` affinchè possa essere rilasciato durante l'attesa, per essere restituito al thead non appena risvegliato. Non è invece necessario all'interno della `cv_signal`, che in OS161 lo riceve semplicemente per verificarne il possesso.

Correggere eventuali errori presentando, se necessario, l' implementazione corretta (giustificandola).

```
cv_signal(struct cv *cv, struct lock *lock) {
   …
   lock_release(lock);
   spinlock_acquire(&cv->cv_lock);
   wchan_wakeone(cv->cv_wchan,&cv->cv_lock);
   lock_acquire(Lock);
   spinlock_release(&cv->cv_lock);
   …
}
```

R6	L'errore principale di questa funzione è l'uso improprio del lock. Il lock, come detto in precedenza, NON VA RILASCIATO E RI-ACQUISITO! La `cv_signal`, che chiama `wchan_wakeone`, è operazione veloce, che non manda il thread in attesa. Non ha senso rilasciare il lock per permettere al thread svegliato di acquisirlo subito (si farebbero più context switch). Il lock sarà rilasciato dal thread che chiama `cv_signal` (dopo l'uscita da questa). A questo punto il thread in cv_wait (svegliato) potrà acquisire il lock e andare avanti.
	Versione corretta:
	`cv_signal(struct cv *cv, struct lock *lock) {`
	`…`
	`KASSERT(lock_do_i_hold(lock));`
	`spinlock_acquire(&cv->cv_lock);`
	`wchan_wakeone(cv->cv_wchan,&cv->cv_lock);`
	`spinlock_release(&cv->cv_lock);`
	`…`
	`}`
	
	Il codice proposto contiene anche una errata alternanza tra gestione di lock e di spinlock, ma questo problema sarebbe significativo nella `cv_wait`, nella `cv_signal` prevale quanto detto sopra.