Programmazione di sistema

Esame dell'8/9/2022 (C2 intermedio - Calabrodi) - PER ESAME IN AULA

Tempo impiegato

...ne. La valutazione finale è saturata a 13. Eventuali voti > 13 sono pertanto portati automaticamente a 13.

Domanda 1

Completo

Punteggio ottenuto 2,50 su 3,00

TUTTE LE RISPOSTE SÌ / NO DEVONO ESSERE MOTIVATE. QUANDO I RISULTATI SONO NUMERI, SONO RICHIESTI SIA IL RISULTATO FINALE SIA I RELATIVI PASSI INTERMEDI (O FORMULE)

Sia dato il frammento di programma rappresentato:

```
...

#define N 128
#define M 32

int i,j, k;

float *A[N],B[N*M]

for(i=0; i<N*M; i++) {
  B[i] = i;
}
for(i=0; i<N; i++) {
  A[i] = calloc((i+1)*sizeof(float));
}

for(k=0; k<N*M; k++) {ì
  i = rand()%N;
  j = rand()%i;
  A[i][j] += B[i+j*M];
}
...
```

Il codice macchina generato da tali istruzioni viene eseguito in un sistema con memoria virtuale gestita mediante paginazione, con **pagine di 4Kbyte**, utilizzando come politica di sostituzione pagine la **LRU (Least Recently Used)**. Si sa che i puntatori occupano 64 bit, mentre sia int che float hanno dimensione **32 bit** e che le istruzioni in codice macchina, corrispondenti al frammento di programma visualizzato, sono contenute in una sola pagina. Si supponga che A e B siano allocati ad indirizzi logici contigui (prima A, poi B), a partire dall'indirizzo logico 0x74240000.

A) Quante pagine (e frame) sono necessarie per contenere le strutture dati A e B, più la

memoria allocata dinamicamente, supponendo (pur trattandosi di una semplificazione) che le chiamate alla calloc allochino in modo progressivo e contiguo nello heap, a partire dall'inizio di una pagina.

B) Ipotizzando che le variabili i, j siano allocate in registri della CPU, quanti accessi in memoria (in lettura e scrittura) fa il programma proposto, per accedere a dati (non vanno conteggiati gli accessi a istruzioni, nè gli azzeramenti fatti dalla calloc, per la quale si suppone di poter ottenere pagine già azzerate; si trascurino inoltre le chiamate alla rand())?

C) Calcolare il numero di page fault generati dal programma proposto, supponendo che siano allocati per esso **16 frame**, di cui uno utilizzato (già all'inizio dell'esecuzione) per le istruzioni.

TUTTE LE RISPOSTE SÌ / NO DEVONO ESSERE MOTIVATE. QUANDO I RISULTATI SONO NUMERI, SONO RICHIESTI SIA IL RISULTATO FINALE SIA I RELATIVI PASSI INTERMEDI (O FORMULE)

A) Quante pagine (e frame) sono necessarie per contenere le strutture dati A e B, più la memoria allocata dinamicamente, supponendo (pur trattandosi di una semplificazione) che le chiamate alla calloc allochino in modo progressivo e contiguo nello heap, a partire dall'inizio di una pagina.

L'indirizzo 0x74240000 finisce con più di 12 zeri, dunque è multiplo di 4K(2^12), multiplo di pagina.

Ogni pagina può contenere 4K/8 puntatori = 512 puntatori e 4KB/4B int o float quindi 1K.

A è un vettore di N puntatori. Sullo stack occupa 1/4 di pagina. Ogni entry contiene un vettore di float pari all'indice + 1, quindi in totale conterrà un numero di float pari a sum_from=i=1 to 128 i = (128 * 129)/2 = 129*64 float = 7936. questo equivale a 7936/1024 pagine = 7.75 -> 8 pagine

|A| = 1(non usata tutta) + 8 (di cui l'ultima non usata tutta)

B invece è allocata nello stack, e contiene N*M float = 128 * 32 float, che equivalgono a 128 * 32 / 1K pagine = 4 pagine. Poichè B inizia a 1/4 di pagina occuperà 5 pagine in totale.

|B| = 5 pagine (di cui una condivisa con A)

B) Ipotizzando che le variabili i, j siano allocate in registri della CPU, quanti accessi in memoria (in lettura e scrittura) fa il programma proposto, per accedere a dati (non vanno conteggiati gli accessi a istruzioni, nè gli azzeramenti fatti dalla calloc, per la quale si suppone di poter ottenere pagine già azzerate; si trascurino inoltre le chiamate alla rand())?

Il primo ciclo for effettua N*M scritture.

Il secondo ciclo for effettua N scritture(scrive i puntatori)

Il terzo ciclo for effettua N*M letture, N*M scritture e N*M letture.

In totale n_accessi = (N*M) W + N W + 2(N*M)R + N*M W) (N + 2*(N*M))W + N*M R) = 8320 W + 4096 R = 12416

C) Calcolare il numero di page fault generati dal programma proposto, supponendo che siano allocati per esso **16 frame**, di cui uno utilizzato (già all'inizio dell'esecuzione) per le istruzioni.

Considerando un frame è usato per il programma, rimangono 15 frame. Il primo ciclo for porta dentro tutto il vettore B -> 5 pagine e genera un pf per pagina -> 5 pf

Il secondo for alloca A, e non fa pf in quanto i puntatori occupano il primo quarto della prima pagina di B. In questo momento ci sono 6/16 pagine occupate.

Successivamente vengono portate in memoria le pagine di A allocate sullo heap, e fanno un pf ciascuno. -> 8 pf.

In totale il programma fa 13 page fault.

A) Quante pagine (e frame) sono necessarie per contenere le strutture dati A e B, più la memoria allocata dinamicamente, supponendo (pur trattandosi di una semplificazione) che le chiamate alla calloc allochino in modo progressivo e contiguo nello heap, a partire dall'inizio di una pagina.

|A| = N*sizeof(float *) = N*8 B = 1024 B = 1 KB

|B| = N*M*sizeof(float) = 4*N*M=128*128 B = 16 KB

|Allocated memory| = N*(N+1)/2 * sizeof(float) = 2N(N+1) B = 33024 KB = 32KB + 256B

|page| = 4KB

A -> 1 page

B -> 4 pages

fragmentation of 4KB-1KB = 3KB

allocated -> 9 pages (with frag of 4096-256=3840B)

B) Ipotizzando che le variabili i, j siano allocate in registri della CPU, quanti accessi in memoria (in lettura e scrittura) fa il programma proposto, per accedere a dati (non vanno conteggiati gli accessi a istruzioni, nè gli azzeramenti fatti dalla calloc, per la quale si suppone di poter ottenere pagine già azzerate; si trascurino inoltre le chiamate alla rand())?

First for: N*M iterations, one write for each iteration (B[i] = …): N*M write

Second for: N iterations, one write for each iteration (A[i] = …): N write

Third for: N*M iterations, with 2 reads and 1 write for each iteration (assignments to i and j ignored)

read: 2*N*M

write: (2+1)*N*M + N

tot: 4*N*M + N = 4*32*128+128 = 128*129 = 16512

C) Calcolare il numero di page fault generati dal programma proposto, supponendo che siano allocati per esso **16 frame**, di cui uno utilizzato (già all'inizio dell'esecuzione) per le istruzioni.

As the number of frames (16) is greater than the number of pages, no replacement will occur. So just 1 PF per page will occur (first access) in the worst case:

PF = 1 + 4 + 9 = 14

Commento:
A) ok ma fa un errore di calcolo: scrive 129*64 ma poi calcola 124*64 -> 0,75

B) formule corrette ma perde 4k read nei calcoli. -> 0,75

C) ok -> 1

Domanda 2

Completo

Punteggio ottenuto 3,00 su 3,00

TUTTE LE RISPOSTE SÌ / NO DEVONO ESSERE MOTIVATE. QUANDO I RISULTATI SONO NUMERI, SONO NECESSARI IL RISULTATO FINALE E I RELATIVI PASSI INTERMEDI (O FORMULE)

Considerare un file system basato su allocazione indicizzata nel quale, per gestire i file di grandi dimensioni, si mettono il lista i blocchi indice. I puntatori/indici hanno una dimensione di 32 bit e i blocchi del disco hanno una dimensione di 4KB. Il file system risiede su una partizione del disco di 600 GB, che **include sia blocchi di dato che blocchi di indice.**

A) Dato un file binario di dimensione 10381 KB, calcolare esattamente quanti blocchi indice e blocchi di dato occupa il file. Calcolare anche la frammentazione interna, sia per i blocchi di dato che per quelli di indice.

B) Si sa che il file contiene una sequenza di numeri reali in formato double (64 bit per numero). Si calcoli fino a quale blocco di indice e a quale riga (casella) di questo occorre accedere per leggere i primi 1M (un Mega) numeri del file. Si supponga che il puntatore al blocco indice successivo in lista sia posizionato all'inizio di un blocco indice. (verificare che la dimensione sia consistente)

A) Dato un file binario di dimensione 10381 KB, calcolare esattamente quanti blocchi indice e blocchi di dato occupa il file. Calcolare anche la frammentazione interna, sia per i blocchi di dato che per quelli di indice.

Calcolo prima quanti blocchi dato richiede il file. Il file richiede 10381KB/4KB blocchi = 2595,25 blocchi -> 2596. Da questo calcolo che la frammentazione interna è di 3/4*4KB = 3KB.

Successivamente calcolo il numero di puntatori per blocco = 4KB/4B = 1K puntatori per blocco, di cui uno per il next block(ho fatto questa assunzione, altrimenti si potrebbe avere una struttura nel kernel).

Il file, richiede 2596/1023 blocchi indice = 2,54, quindi 3 blocchi indice. In tutto, occupa quindi 2596 + 3 blocchi = 2599. calcolo la frammentazione interna per i blocchi indice, considerando che il terzo è riempito per poco più della metà = 3*1023 - 2596 = 473 puntatori quindi 1892B, 1.84K

B) Si sa che il file contiene una sequenza di numeri reali in formato double (64 bit per numero). Si calcoli fino a quale blocco di indice e a quale riga (casella) di questo occorre accedere per leggere i primi 1M (un Mega) numeri del file. Si supponga che il puntatore al blocco indice successivo in lista sia posizionato all'inizio di un blocco indice. (verificare che la dimensione sia consistente)

Il file è grosso circa 10M. Ogni blocco contiene 4KB / 8B = 512 double. Per arrivare a 1M devo leggere il blocco numero 2048. Il blocco numero 2048 è contenuto nel blocco numero 2048/1023 = 2. Per calcolare la riga basta fare 2048 - 2*1023 + 1 = 3. Considerando che la prima riga è occupata dall'eventuale puntatore al next block, il blocco dato si trova alla riga 4.

A) Dato un file binario di dimensione 10381 KB, calcolare esattamente quanti blocchi indice e blocchi di dato occupa il file. Calcolare anche la frammentazione interna, sia per i blocchi di dato che per quelli di indice.

Blocchi dato: ceil(10381KB/4KB) = 2596
Framm. Int. = 1-0.25 blocks = 0.75 blocks = 3KB

Blocchi indice
n. indici/blocco: 4KB/4B = 1K
1 indice/puntatore usato per la lista, 1K-1 = 1023 usati per blocchi dato

Blocchi indice necessari: ceil(2596/1023) = 3
Framm. Int (spazio non utilizzato nel'ultimo blocco indice)
indici non usati: 1023 – (2596%1023) = 1023 – 550 = 473
Framm: 473 * 4B = 1892B

B) Si sa che il file contiene una sequenza di numeri reali in formato double (64 bit per numero). Si calcoli fino a quale blocco di indice e a quale riga (casella) di questo occorre accedere per leggere i primi 1M (un Mega) numeri del file. Si supponga che il puntatore al blocco indice successivo in lista sia posizionato all'inizio di un blocco indice. (verificare che la dimensione sia consistente)

Calcolo blocchi dato (ND) che vanno letti

ND = 1M*8B/4KB = 8MB/4KB = 2K
Il numero di blocchi dato coincide con il numero di indici necessari

Per 2K indici servono quindi 2 blocchi indice (ognuno contenente 1023 = 1K-1 indici) più un terzo blocco indice da cui si leggono 2 indici

Si leggono quindi 3 blocchi indice, due completi, il terzo sino alla casella 2 cpompresa (la 0 è occupata dal puntatore al blocco indice successivo)

Commento:
ok

Domanda 3

Completo

Punteggio ottenuto 3,00 su 3,00

SE I RISULTATI SONO NUMERI, RIPORTARE PASSAGGI INTERMEDI RILEVANTI E/O FORMULE USATE
LE RISPOSTE SI/NO VANNO MOTIVATE

Si consideri un contesto di paginazione a richiesta con sostituzione di pagine. Si risponda alle domande seguenti.

A) Spiegare brevemente la differenza tra “equal allocation” e “proportional allocation”.

B) Con quale delle due tecniche (ci si riferisce alla domanda A) si puo’ realizzare una politica di sostituzione di ripo “working set”? (motivare la risposta)

- Equal allocation
- Proportional allocation
- Entrambe
- Nessuna delle due

C) Considerando politiche di sostituzione di tipo “local” e “global”,

- Quale delle due permette a un processo di non dipendere (o dipendere in modo minore) dagli altri processi in esecuzione (per quanto riguarda la sequenza di allocazioni/sostituzioni)? (motivare)
- Perché una politica di tipo “global” può ottenere un miglior “throughput”

SE I RISULTATI SONO NUMERI, RIPORTARE PASSAGGI INTERMEDI RILEVANTI E/O FORMULE USATE
LE RISPOSTE SI/NO VANNO MOTIVATE

A) Spiegare brevemente la differenza tra “equal allocation” e “proportional allocation”.

Per equal allocation e proportional allocation si parla di tecniche per assegnare frame in RAM a processi. Con la tecnica equal allocation si assegna ad ogni processo lo stesso numero di frame(pari a (n frame - qualcosa per il kernel)/numero di processi) mentre per proportional allocation si assengnano frame al processo in maniera proporzionale alla dimensione relativa del processo rispetto agli altri, cioe (dimensione del processo/dimensione di tutti i processi)*n_frame

B) Con quale delle due tecniche (ci si riferisce alla domanda A) si puo’ realizzare una politica di sostituzione di ripo “working set”? (motivare la risposta)

- Nessuna delle due

In quanto una politica working set consiste nel portare in memoria i frame che a cui il processo ha fatto accesso recentemente, il numero di frame di un processo varia nel tempo, mentre le politiche precedenti allocano un numero fisso nel tempo di frame per processo.

C) Considerando politiche di sostituzione di tipo “local” e “global”,

- Quale delle due permette a un processo di non dipendere (o dipendere in modo minore) dagli altri processi in esecuzione (per quanto riguarda la sequenza di allocazioni/sostituzioni)? (motivare)

Un politica di tipo global consente ad un processo che ha bisogno di frame di "rubare" un frame ad un altro processo. In tal senso, un processo che fa molti accessi non locali, richiede molte pagine, rubandole ad altri processi, che in questo senso dipendono dal processo "vorace". La risposta è dunque politica di tipo local, che invece non permette a un processo di rubare frame da altri.

- ◦ Perché una politica di tipo “global” può ottenere un miglior “throughput”

Per quello detto prima, in questo modo un processo che ha bisogno di molte pagine potrebbe rubarle a processi che fanno accessi molto più locali e quindi hanno bisogno di meno pagine.

LE RISPOSTE SONO FORMULATE IN INGLESE IN QUANTO COMUNI AL CORSO IN INGLESE DI SYSTEM AND DEVICE PROGRAMMING

A) Spiegare brevemente la differenza tra “equal allocation” e “proportional allocation”.

Si tratta di due schemi di allocazione fissa di frame a processi:
- con equal allocation si intende che i processi hanno lo stesso numero di frame
- con proportional allocation si intende che ogni processo ha un numero di frame proporzionale alla sua dimensione (il numero di pagine)

B) Con quale delle due tecniche (ci si riferisce alla domanda A) si puo’ realizzare una politica di sostituzione di ripo “working set”? (motivare la risposta)

- Equal allocation (NO)
- Proportional allocation (NO)
- Entrambe (NO)
- Nessuna delle due: SI - La politice working set NON è uno schema di allocazione fissa, ma variabile, quindi nessuno di questi due schemi è compatibile con essa.

C) Considerando politiche di sostituzione di tipo “local” e “global”,

- Quale delle due permette a un processo di non dipendere (o dipendere in modo minore) dagli altri processi in esecuzione (per quanto riguarda la sequenza di allocazioni/sostituzioni)? (motivare)

Con l'allocazione di tipo "local" un processo dipende MENO dagli altri, perchè la politica di sostituzione e di ricerca vittime viene gestita completamente tra i frame allocati al processo. Con politiche di allocazine "global", invece, un processo puo' trovare "vittime" tra le pagine di altri processi, influenzandoli quindi maggiormente..

- Perché una politica di tipo “global” può ottenere un miglior “throughput”

Perchè, pur penalizzando eventualmente un processo a causa di altri processi attivi, bilancia e ripartisce meglio i frame a seconda di quanto i processi siano attivi o meno. Si permette ad esempio che frame non utilizzati da parte di un processo vengano assegnati (nella ricerca di vittime) ad altri processi.

Commento:
A) ok -> 1

B) FINALMENTE UNA GIUSTA: la hanno sbagliata tutti eccetto te -> 1

C) ok -> 1

Domanda 4

Completo

Punteggio ottenuto 3,00 su 3,00

TUTTE LE RISPOSTE SÌ / NO DEVONO ESSERE MOTIVATE. QUANDO I RISULTATI SONO NUMERI, SONO NECESSARI IL RISULTATO FINALE E I RELATIVI PASSI INTERMEDI (O FORMULE)

Si confrontino le due implementazioni (parziali) di spinlock_acquire, in OS161, fornite di seguito (spinlock_acquire_v1 e spinlock_acquire_v2) e si risponda alle seguenti domande:
A) Le due implementazioni sono (per la parte visibile) corrette, il che significa che implementano un comportamento di spinlock corretto? (In caso di errori, elencare gli errori, se sono corretti spiegarne il motivo).
B) Quali sono le principali differenze in termini di comportamento tra le due implementazioni? (le diverse istruzioni sono chiaramente visibili, la domanda è cosa implementano)
C) Sulla base del codice visto e del funzionamento richiesto a uno spinlock, fornire una possibile definizione (linguaggio C) di struct spinlock (spiegare/motivare brevemente la propria definizione).

```
void spinlock_acquire_v1(struct spinlock *splk) {
  int again = 1;
  ...
  while (again) {
    if (spinlock_data_get(&splk->splk_lock) == 0) {
      if (spinlock_data_testandset(&splk->splk_lock) == 0) {
        again=0;
      }
    }
  }
  ...
}


void spinlock_acquire_v2(struct spinlock *splk) {
  ...
  while (spinlock_data_testandset(&splk->splk_lock) != 0);
  ...
}
```

A) Le due implementazioni sono (per la parte visibile) corrette, il che significa che implementano un comportamento di spinlock corretto? (In caso di errori, elencare gli errori, se sono corretti spiegarne il motivo).

Il comportamento delle due implementazioni è corretto. L'accesso in mutua esclusione offerto dallo spinlock è realizzato tramite busy waiting. Quando una varibile interna diventerà 0, allora si potrà accedere. La test_and_set è molto semplice, mette in quanto testa se il valore è zero ed in quel caso ci scrive 1, e dopo ritorna il valore precedente(e quindi 0) . L'atomicità dell'operazione è garantita dal processore.

Sia la prima che la seconda funzionano in quanto aspettano, ciclando su test_and_set(...) == 0 che la seziona critica sia libera e che il valore interno valga 0.

B) Quali sono le principali differenze in termini di comportamento tra le due implementazioni? (le diverse istruzioni sono chiaramente visibili, la domanda è cosa implementano)

La differenza tra le due implementazioni è che la prima presenta una ottimizzazione, in quanto esegue la test e set solo se il valore è già 0. Questo è importante in quanto la test and set è una operazione costosa per il processore, e in questo modo si limita il numero di volte che viene eseguita.

C) Sulla base del codice visto e del funzionamento richiesto a uno spinlock, fornire una possibile definizione (linguaggio C) di struct spinlock (spiegare/motivare brevemente la propria definizione).

```
struct spinlock {
    int lock; // campo lock  contiene un intero da ritornare dalle funzioni test_and_set
      struct thread *owner; //l'owner attuale del thread, serve a garantire che chi acquisisce lo spinlock lo rilasci
}
```

A) Le due implementazioni sono (per la parte visibile) corrette, il che significa che implementano un

comportamento di spinlock corretto? (In caso di errori, elencare gli errori, se sono corretti spiegarne il motivo).

Entrambe le implementazioni sono corrette, in quanto utilizzano uno schema di tipo test-and-set per verificare la disponibilità e ottenere eventualmente lo spinlock.
Infatti per ottenere uno spinlock occorre, in modo atomico, aspettare che questo sia libero e occuparlo non appena lo diventa.
La versione _v1 è leggermente più efficiente, in quanto itera con una semplice lettura, chiamando la test-and-set solo quando si sa che lo spinlock è disponibile.

B) Quali sono le principali differenze in termini di comportamento tra le due implementazioni? (le diverse istruzioni sono chiaramente visibili, la domanda è cosa implementano)

Come detto nella risposta precedente, la principale differenza sta nel fatto che la versione _v1 è leggermente più efficiente, in quanto itera con una semplice lettura, chiamando la test-and-set solo quando si sa che lo spinlock è disponibile: questa versione viene detta test-and-test-and-set

C) Sulla base del codice visto e del funzionamento richiesto a uno spinlock, fornire una possibile definizione (linguaggio C) di struct spinlock (spiegare/motivare brevemente la propria definizione).

```
struct spinlock {
   unsigned splk_lock; // or other number type, to be used for test-and-set
   struct cpu *owner;  // because spinlocks have ownership - other ownerships
                       // (e.g. process/thread) can be considered correct
};
```

Commento:
tutto ok

Domanda 5

Completo

Punteggio ottenuto 2,25 su 3,00

SE I RISULTATI SONO NUMERI, RIPORTARE PASSAGGI INTERMEDI RILEVANTI E/O FORMULE USATE
LE RISPOSTE SI/NO VANNO MOTIVATE

Si consideri, in OS161, la realizzazione della system call waitpid. Si risponda alle seguenti domande

A) Perché il processo che termina (con sys_exit) non può chiamare direttamente la proc_destroy, dopo aver segnalato la fine del processo stesso (con semaforo o condition variable) e prima della thread_exit?

B) Si vogliono realizzare due funzioni (di cui si forniscono i prototipi) in grado di passare da puntatore a processo a pid e viceversa:

pid_t proc_to_pid(struct proc *p);

struct proc * proc_from_pid(pid_t pid);

Si realizzino le due funzioni. NON è necessario realizzare inizializzazioni di strutture dati o altre funzioni. Ma è necessario spiegare a parole cosa siano (e come andrebbero inizializzate) eventuali strutture dati necessarie per effettuare le due conversioni.

Si consideri, in OS161, la realizzazione della system call waitpid. Si risponda alle seguanti domande

A) Perché il processo che termina (con sys_exit) non può chiamare direttamente la proc_destroy, dopo aver segnalato la fine del processo stesso (con semaforo o condition variable) e prima della thread_exit?

Non si può chiamare proc_destroy prima di thread_exit in quanto la thread_exit usa il puntatore al processo per accedere all'addresspace e rilasciarlo. Sarà compito della thread_exit rilasciare la struct proc una volta rilasciato l'as e quando si sta distruggendo la struct thread.

B) Si vogliono realizzare due funzioni (di cui si forniscono i prototipi) in grado di passare da puntatore a processo a pid e viceversa:

pid_t proc_to_pid(struct proc *p);

struct proc * proc_from_pid(pid_t pid);

Si realizzino le due funzioni. NON è necessario realizzare inizializzazioni di strutture dati o altre funzioni. Ma è necessario spiegare a parole cosa siano (e come andrebbero inizializzate) eventuali strutture dati necessarie per effettuare le due conversioni.

Per fare questo, occorre dotare lo struct proc di un campo interno di tipo pid_t(che non è altro che un alias per un intero), mentre si potrebbe creare un vettore globale di puntatori a struct proc, in cui l'indice nel vettore funge da pid.

```
struct proc*processes[MAX_PID]

pid_t proc_to_pid(struct proc *p) {
    // questa funzione non ha bisogno di controlli particolari
    return p -> pid;
}

struct proc * proc_from_pid(pid_t pid) {
    KASSERT(pid > 0);
    KASSERT(pid < MAX_PID);
    return processes[(int) pid];
}
```

A) Perché il processo che termina (con sys_exit) non può chiamare direttamente la proc_destroy, dopo aver segnalato la fine del processo stesso (con semaforo o condition variable) e prima della thread_exit?

Perchè la semantica della exit (e sys_exit) prevede che un aptro processo, che fa waitpid, debba poter aspettare la fine del processo. Per fare questo è necessario che una struct proc non venga distrutta (il processo diventa zompie) finchè il processo in attesa riceve la segnalazione. La chiamata a proc_destroy varrà fatta invece all'inteno della sys_waitpid, quindi da lato in cui si "riceve" la segnalazione.

B) Si vogliono realizzare due funzioni (di cui si forniscono i prototipi) in grado di passare da puntatore a processo a pid e viceversa:

```
pid_t proc_to_pid(struct proc *p);
struct proc * proc_from_pid(pid_t pid);
```

Si realizzino le due funzioni. NON è necessario realizzare inizializzazioni di strutture dati o altre funzioni. Ma è necessario spiegare a parole cosa siano (e come andrebbero inizializzate) eventuali strutture dati necessarie per effettuare le due conversioni.

Si suppone che esista una variabile globale processTable, vettore di puntatori a struct proc. In tale vettore un processo viene collocato all'indice corrispondente al pid. Sono possibili anche altri schemi, ed esempio un vettore di coppie (puntatore a processo, pid)
Si suppone inoltre che ogni struct proc al suo interno abbia un campo pid.

```
pid_t proc_to_pid(struct proc *p) {
   return p->pid;
}

struct proc * proc_from_pid(pid_t pid) {
     return processTable[pid];
}
```

Commento:
A) risposta solo in parte corretta. Non è vero che sarà compito della thread_exit a rilasciare il processo: al limite si poteva pensare a questo nella versione semplificata della sys_exit in lab 2 -> 0,25

B) ok -> 2

Domanda 6

Completo

Non valutata

Per ritirarti, seleziona "Mi ritiro".

In caso contrario, è equivalente non rispondere oppure rispondere " Desidero che Il mio esame sia valutato".

Potrai ancora comunicare l'intenzione di ritirarti a esame chiuso, una volta vista la soluzione proposta.

- (a) Mi ritiro (il mio esame non verrà valutato)
- (b) Desidero che Il mio esame sia valutato

Risposta errata.

La risposta corretta è: Mi ritiro (il mio esame non verrà valutato)