Domande preparazione

Domanda 1 (processi)
Una sola delle seguenti affermazioni sui processi è **falsa**. Quale?

1. Un processo è nello stato waiting se è in attesa di un evento come la terminazione di un'operazione di I/O.
2. Un processo è nello stato zombie se è terminato, ma il genitore non ha ancora effettuato una wait per recuperarne lo stato di terminazione.
3. Un processo è nello stato ready se non è in esecuzione solo perché tutti i core della CPU sono impegnati ad eseguire altri processi.
4. Un processo è nello stato running se sta correntemente impegnando un core della CPU.
5. I sistemi operativi reali hanno un numero maggiore di stati per descrivere situazioni come ready in modalità supervisore (ad esempio se avviene preemption mentre un processo sta eseguendo una system call)
6. Nessuna delle precedenti…

Domanda 2 (memorie cache)
Si consideri la seguente sequenza di indirizzi acceduti da un processo: `980, 408, 984, 612, 410, 412, 500`. Quali blocchi verrebbero a trovarsi alla fine della sequenza in una piccola cache completamente associativa con 3 linee da 32 byte ciascuna e politica di rimpiazzo LRU? (l'ordine degli indici di blocco non conta)

1. 30, 12, 19
2. 15, 12, 19
3. 15, 19, 20
4. 15, 30, 19
5. 12, 30, 15
6. Nessuna delle precedenti

Domanda 3 (pipelining)
Si consideri la seguente sequenza di istruzioni:

```
movl $12, %eax
addl %ecx, %edx
movl %eax, %ebx
```

Quanti cicli di clock vengono richiesti da una semplice pipeline a 5 stadi (Fetch, Decode, Execute, Memory, Write-Back) per completare tutte le istruzioni assumendo che gli hazard vengano risolti con stalli?

1. 0
2. 5
3. 9
4. 7
5. 3
6. 11

Domanda 4 (memoria virtuale)
Si consideri un sistema di memoria virtuale con pagine da 2 KB e spazio logico a 24 bit. Quanti byte occuperebbe la tabella delle pagine assumendo di avere indici dei frame a 32 bit?

1. 65536 byte
2. 32768 byte
3. 262144 byte
4. 16384 byte
5. 131072 byte
6. Nessuna delle precedenti

Domanda 5 (allineamento e padding)
Si consideri la seguente `struct C`:

```
struct A {
      char a;
      double b;
      char c;
      double d;
      char e;
      };
```

Come riordineresti la struttura per minimizzarne la dimensione assumendo che il compilatore inserisca padding per garantire l'allineamento dei campi e quale dimensione otterresti?

1. `char, char, char, double, double` => 19 byte
2. `char, double, double, char, char` => 32 byte
3. `char, char, char, double, double` => 24 byte
4. `char, char, double, char, double` => 32 byte
5. `char, char, double, char, double` => 24 byte
6. Nessuna delle precedenti

Domanda 6 (memory layout)
Si consideri il seguente frammento di programma:

```
int x;
int main(int argc, char** argv){
      int* p = malloc(10*sizeof(int));
      char* s1 = "hello";
      char s2[] = "hello";
       ...
      return 0;
}
```

Il un moderno sistema Linux, una sola fra le seguenti affermazioni è **vera**. Quale?

1. x denota un oggetto allocato in `bss`, p denota un oggetto allocato in `heap`, `s1` denota un oggetto allocato in `rodata` e `*s2` denota un oggetto allocato in `data`.
2. x denota un oggetto allocato in `bss`, *p denota un oggetto allocato in `heap`, `*s1` denota un oggetto allocato in `rodata` e `*s2` denota un oggetto allocato in `stack`.
3. x denota un oggetto allocato in `data`, p denota un oggetto allocato in `stack`, `s1` denota un oggetto allocato in `stack` e `*s2` denota un oggetto allocato in `heap`.
4. `argc` denota un oggetto allocato in `stack`, `*argv` denota un oggetto allocato in `heap`.
5. p denota un oggetto allocato in `stack`, `argv` denota un oggetto allocato in `stack`, `**argv` denota un oggetto allocato in `heap`.
6. Nessuna delle precedenti

Domanda 7 (schedulazione dei processi)
Una sola fra le seguenti affermazioni è **falsa**. Quale?

1. La schedulazione dei processi serve a selezionare un processo in stato ready per metterlo in esecuzione.
2. In un sistema multiprogrammato time-sharing come Linux, la schedulazione avviene contestualmente a un context switch innescato dallo scadere di una time slice.
3. Uno degli algoritmi di schedulazione più usati è il Round-Robin, che assegna la CPU (un core) a un processo alla volta, a rotazione.
4. Un sistema operativo fa preemption se è il processo che spontaneamente rilascia la CPU per consentire allo scheduler di mandare in esecuzione un altro processo.
5. Lo scadere di una time slice viene segnalata al sistema da un interrupt generato da un timer.
6. Nessuna delle precedenti.

Domanda 8 (manipolazioni numeriche)
Si consideri il seguente frammento di programma C:

```
int x = 0xDEADBEEF & ...
```

Cosa bisogna inserire al posto di ... affinché x prenda il valore 0xD0A0?

1. `(252645135 << 16)`
2. `(4042322160 >> 16)`
3. `(3735928559 >> 8)`
4. `(256736178 | 0xFF)`
5. `(3462378762 >> 16)`
6. Nessuna delle precedenti

Domanda 9 (Ottimizzazioni dei programmi)
Si consideri questo frammento di programma:

```
int f(int x) {
      return 2*x;
}

int g() {
      return f(10) + 5;
}
```

A seguire la sua versione ottimizzata:

```
int f(int x) {
      return 2*x;
}

int g() {
      return 25;
}
```

Quali ottimizzazioni sono state applicate? Una sola delle seguenti affermazioni è **vera**.

1. common subexpression elimination + constant folding
2. function inlining + dead code elimination
3. loop-invariant code motion + constant folding
4. register allocation + constant folding
5. function inlining + constant folding
6. Nessuna delle precedenti

Domanda 10 (interrupt)
Una sola delle seguenti affermazioni è **falsa**. Quale?

1. Un'interruzione hardware alla CPU può essere generata da un dispositivo esterno per segnalare l'occorrenza di un evento come la pressione di un tasto sulla tastiera, il click del mouse, o il completamento di un'operazione di I/O.
2. È possibile avere interruzioni software, chiamate trap, generate da un programma, ad esempio eseguendo l'istruzione IA32 `int`.
3. Le interruzioni possono essere sia recuperabili che non recuperabili.
4. Le interruzioni possono essere sia sincrone che asincrone.
5. Un'interruzione provoca l'esecuzione di un frammento di codice di gestione del sistema operativo puntato dal vettore delle interruzioni (interrupt vector) indicizzato dal numero dell'interruzione.
6. Nessuna delle precedenti

Domanda 11 (ottimizzazioni)
Che speedup otterremmo per un programma dimezzando il tempo di esecuzione di una sua porzione che richiede il 10% del tempo complessivo?

1. 1.56x
2. 1.32x
3. 1.05x
4. 2.31x
5. 1.15x
6. 1.27x

Domanda 12 (segnali)
Una sola fra le seguenti affermazioni è **falsa**. Quale?

1. I segnali sono un semplice meccanismo di comunicazione fra processi che serve per notificare eventi.
2. Un segnale può essere innescato da un interrupt.
3. Alcuni segnali possono essere catturati o ignorati, mentre altri no.
4. Ogni segnale ha un gestore di default sul lato del ricevente che può essere ripristinato in ogni momento.
5. Diversamente dalle interruzioni, i segnali sono interamente gestiti dal sistema operativo e dal programmatore.
6. Un segnale è accompagnato da un puntatore a un dato che può essere comunicato al destinatario insieme al segnale, o `NULL` se non è necessario inviare alcun dato.

Domanda 13 (ottimizzazioni)
Supponiamo di essere in grado di dimezzare il tempo di esecuzione di una porzione di un programma mediante delle ottimizzazioni. Che percentuale del tempo complessivo deve prendere quella porzione per ottenere uno speedup di 1.5x?

1. 0.77777
2. 0.5
3. 0.66667
4. 0.13333
5. 0.22223
6. 0.812

Domanda 14 (memorie cache)
Si consideri una cache associativa a 2 vie con 4 linee da 8 byte ciascuna e politica di rimpiazzo LRU.
Potendo scegliere durante un cold miss, selezionare sempre la linea con indice più basso.
Data la seguente sequenza di accessi a indirizzi di memoria, qual è il contenuto delle linee di cache alla fine della sequenza: `96, 98, 124, 86, 104, 120, 68, 138`?

1. 12, 10, 15, 17
2. 12, 10, 15, 8
3. 17, 10, 15, 8
4. 8, 10, 15, 17
5. 12, 10, 15, 11
6. 10, 12, 13, 15

Domanda 15 (allocazione dinamica della memoria)
Si consideri un semplice allocatore di memoria malloc/free dove, potendo scegliere, la malloc riusa il blocco libero con l'indirizzo più basso. Assumere per semplicità che non vi sia alcuna header e che i blocchi allocati vengano arrotondati a una dimensione multiplo di 4 byte.
Qual è il contenuto dell'heap alla fine della seguente sequenza di operazioni (dove X denota 4 byte allocati e . denota 4 byte liberi)?

```
p1=malloc(2);
p2=malloc(10;
p3=malloc(4);
free(p1);
p4=malloc(4);
p5=malloc(8);
free(p4);
free(p3);
p6=malloc(6);
```

1. X | XXX | . | XX
2. XX | XXX | . | XX
3. . | XXX | . | XX | XX
4. XXX | . | XX | . | XX
5. XX | . | XXX | . | XX
6. … | . | XXX | . | XX

Domanda 16 (memoria virtuale)
Una sola fra le seguenti affermazioni è **vera**. Quale?
1. Un sistema di memoria paginato soffre di frammentazione esterna.
2. La memoria virtuale è delegata a tenere traccia dei blocchi che vengono allocati con malloc e free.
3. La memoria virtuale consente di evitare che un processo possa accedere allo spazio di indirizzi di un altro processo.
4. La memoria virtuale è un ingrediente indispensabile in qualsiasi sistema di calcolo moderno.
5. La dimensione di una pagina deve essere un multiplo della dimensione di un frame.
6. È possibile gestire uno spazio di indirizzi logico a 64 bit con una singola tabella delle pagine.

Domanda 17 (spazio utente e spazio kernel)
Una sola fra le seguenti affermazioni è **vera**. Quale?
1. Una chiamata a `strlen` finisce per invocare una system call.
2. Una chiamata a `printf` finisce per invocare una system call.
3. In un ambiente Linux/IA32, una chiamata a `write` richiama una porzione di codice che gira in spazio utente che a sua volta invoca l'istruzione `int $80`.
4. La libreria C (libc) è indispensabile per ottenere un eseguibile Linux usando gcc
5. La funzione main è la prima ad essere invocata all'avvio di un processo.
6. Nessuna delle precedenti

Domanda 18 (context switch)
Una sola fra le seguenti affermazioni sul context switch è **falsa**. Quale?

1. Un context switch avviene in situazioni di flusso del controllo eccezionale.
2. Durante un context switch un'attività viene interrotta per passare ad un'altra.
3. Durante un context switch viene salvato il contesto di esecuzione in modo da potere eventualmente riprendere in seguito un'attività interrotta.
4. Si ha un context switch quando un processo in un sistema operativo con time sharing viene interrotto per schedulare un altro processo.
5. Il PCB associato a un processo viene utilizzato per memorizzarne lo stato durante un context switch in modo da poterlo ripristinare in seguito.
6. I context switch non hanno alcun impatto sulle prestazioni.

Domanda 19 (scala degli eventi di un sistema di calcolo)
Approssimativamente, ignorando il parallelismo introdotto dalla pipeline della CPU, quante istruzioni macchina che coinvolgono solo registri potrebbero essere eseguite nel tempo richiesto da un'istruzione che accede a una DRAM?

1. 1
2. 10
3. 100
4. 1000
5. 10000
6. 100000

Domanda 20 (gerarchie di memoria)
Una sola fra le seguenti affermazione sulle gerarchie di memoria è **vera**. Quale? Per livelli più alti si intende quelli più vicini alla CPU.

1. I livelli più bassi hanno il costo maggiore per byte.
2. I livelli più alti hanno i tempi di accesso più bassi.
3. I livelli più bassi hanno la capienza minore.
4. Il trasferimento di dati tra livelli consecutivi avviene alla granularità del byte.
5. I vari livelli includono solo memorie SRAM e DRAM.
6. L'efficacia della gerarchia di memoria è indipendente dal livello di località che viene esposto da un programma.

Domanda 21 (potenze di due)
Si considerino le seguenti grandezze, nell'ordine: 1TB, 2KB, 8MB, 4GB, 16PB.
A quali potenze di due corrispondono?

1. 2^42, 2^10, 2^24, 2^31, 2^54
2. 2^50, 2^12, 2^22, 2^42, 2^34
3. 2^50, 2^11, 2^23, 2^32, 2^44
4. 2^40, 2^11, 2^23, 2^32, 2^54
5. 2^39, 2^12, 2^22, 2^31, 2^53
6. Nessuna delle precedenti

Domanda 22 (paginazione)
Si consideri un sistema di memoria virtuale con uno spazio di indirizzi a 16 bit, pagine da 4 KB, e la seguente tabella delle pagine: `{0x1, 0x6, 0xB, 0x3, 0xA, 0x7, 0xC, 0x5, 0x9, 0x2, 0xD, 0xF, 0x0, 0x4, 0xE, 0x8 }`.
A quali indirizzi fisici corrispondono i seguenti indirizzi logici: `0xF1D6, 0x4F02, 0x6000`?

1. `0x90C1, 0xAF02, 0xE12B`
2. `0xAFD4, 0xD071, 0xD011`
3. `0x81D6, 0xAF02, 0xC000`
4. `0xA0D1, 0xFF00, 0x9117`
5. `0x10CE, 0x1E94, 0xA7D9`
6. Nessuna delle precedenti

Domanda 23 (paginazione e manipolazioni numeriche)
Dato un indirizzo x a 32 bit, come definiresti il numero di pagina p(x) e l'offset o(x) assumendo pagine di 8 KB?

1. `p(x) = x / 2^13, o(x) = x & 0xFFFF`
2. `p(x) = x >> 13, o(x) = x & 0x1FFF`
3. `p(x) = x / 13, o(x) = x & 0xFFF`
4. `p(x) = x >> 2/13, o(x) = x & 0x8FF`
5. `p(x) = x >> 13, o(x) = x & 0x18FF`
6. Nessuno dei precedenti

Domanda 24 (IA32) Assumendo che il registro `eax` contenga inizialmente il valore `0xDEADBEEF`, quale sarebbe il suo contenuto dopo l'esecuzione dell'istruzione `movsbw %al, %ax`?

1. 0xDEADFFEF
2. 0xFFFFFFEF
3. 0x0000FFEF
4. 0xDEAD00EF
5. 0xFFFF00EF
6. Nessuna delle precedenti

Domanda 25 (processi)
Si consideri il seguente frammento di programma che si origina da un singolo processo:

```
fork();
fork();
fork();
```

Quanti processi sono attivi alla fine del frammento?

1. 3
2. 4
3. 6
4. 8
5. 10
6. Nessuno dei precedenti

Domanda 26 (cache)
Una sola delle seguenti affermazioni è **falsa**. Quale?

1. Scorrere sequenzialmente una lista collegata provoca in generale un numero di cache miss non inferiore che scorrere sequenzialmente un array.
2. Assumendo una cache con linee di dimensione L, scorrere sequenzialmente un array di N `short` genera `sizeof(short)*N/L` cache miss.
3. Scorrere sequenzialmente una lista collegata con N nodi genera nel caso peggiore N cache hit
4. Assumendo una cache con linee di dimensione L, scorrere sequenzialmente un array di N `short` genera `N-sizeof(short)*N/L` cache hit.
5. Indipendentemente dai vincoli di associatività, scorrere sequenzialmente un array genera lo stesso numero di cache miss.
6. Indipendentemente dai vincoli di associatività, scorrere sequenzialmente una lista collegata genera nel caso peggiore lo stesso numero di cache miss.

Domanda 27 (interrupt)
Una sola delle seguenti affermazioni è **vera**. Quale?

1. Un interrupt può essere generato da un'istruzione della forma `movl %eax, %ecx`
2. L'interrupt vector non può essere modificato in alcun modo da un programma utente.
3. L'interrupt vector cambia in genere nel tempo durante l'esecuzione dei processi.
4. Un interrupt segnala esclusivamente eventi generati da dispositivi esterni come il completamento di un'operazione di I/O.
5. Gli interrupt sono un meccanismo che altera il normale flusso del controllo dei programmi con un tempismo generalmente prevedibile.
6. Nessuna delle precedenti

Domanda 28 (variabili di ambiente)
Una sola delle seguenti affermazioni è **falsa**. Quale?

1. Una variabile d'ambiente è caratterizzata da una coppia di stringhe della forma (NOME, VALORE)
2. È possibile passare le variabili d'ambiente a un programma eseguibile al momento del suo lancio mediante le system call `execve` o `execvpe`.
3. Il comando `env` permette di elencare le variabili di ambiente correntemente definite nella shell
4. Un programma lanciato da una shell può aggiungere all'ambiente della shell stessa nuove variabili mediante la chiamata `setenv`.
5. La variabile d'ambiente PATH contiene i percorsi delle directory dove vengono cercati i programmi eseguibili i cui nomi sono digitati sulla shell o invocati con `execvp`.
6. Un programma può accedere alle variabili d'ambiente che gli vengono passate dal processo genitore mediante un terzo parametro del main.

Domanda 29 (permessi)
Che permessi dovrebbe avere un file per essere accessibile in lettura e scrittura dall'utente proprietario, in sola lettura dal gruppo proprietario, e nessun permesso per tutti gli altri utenti?

1. `0750`
2. `0530`
3. `0642`
4. `0641`
5. `0640`
6. Nessuno dei precedenti

Domanda 30 (memoria virtuale)
Una sola delle seguenti affermazione è **vera**. Quale?

1. Due processi possono condividere un gruppo di pagine in modo che gli spazi logici degli indirizzi abbiano un'intersezione comune
2. La tabella delle pagine di un processo è accessibile dal processo stesso nel proprio spazio utente (spazio logico degli indirizzi)
3. Le entry di una tabella delle pagine contengono gli indirizzi fisici del primo byte dei frame
4. La dimensione tipica di una pagina di un sistema Linux è di 64 byte
5. Due processi A e B possono condividere dei frame fisici impostando opportunamente le rispettive tabelle delle pagine con il tramite del sistema operativo. Questo consente ad A di scrivere nel proprio spazio logico degli indirizzi in modo che B veda le modifiche effettuate accedendo al proprio, generalmente a un indirizzo logico diverso rispetto ad A.
6. Nessuna delle precedenti

Domanda 31 (ottimizzazioni)
Di quanto bisognerebbe velocizzare una funzione f che consuma il 10% del tempo di esecuzione di un programma P in modo da ottenere uno speedup complessivo per P pari a 2x?

1. 20x
2. 50x
3. 100x
4. 1000x
5. 10000x
6. È impossibile

Prima esercitazione

Domanda 1

Quale dei seguenti frammenti di codice potrebbe essere scritto in linguaggio macchina?

- 55 23 C3 D3 00 00 00 C3
- `movl $2, %eax`
- `while(i<y) i++;`
- nessuno dei precedenti

Domanda 2

Quanti bit servono per rappresentare il numero esadecimale 0xDEADBEEF?

- 8
- 32
- 24
- 16

Domanda 3

Il comando gcc hello.s -o hello attiva i seguenti stadi della toolchain di compilazione:

- assemblatore
- linker
- assemblatore e linker
- preprocessore, compilatore e assemblatore

Domanda 4

Il comando gcc hello.o -o hello attiva i seguenti stadi della toolchain di compilazione:

- preprocessore
- linker
- assemblatore e linker
- preprocessore, compilatore e assemblatore

Domanda 5

Fra i seguenti, qual è il tipo primitivo C con la sizeof minore che consente di rappresentare il numero 256?

- short
- char
- unsigned char
- nessuno dei precedenti

Domanda 6

Per quale operatore bit a bit OP si ha che 0x13 OP 0x21 == 0x32?

- ^ (XOR)
- ~ (NOT)
- & (AND)
- | (OR)

Domanda 7

Dati: char s[]="hello"; int a=sizeof(s), b=strlen(s), c=sizeof("hello"); quale delle seguenti affermazioni è vera? Assumere puntatori a 64 bit.

- a=6, b=5, c=5
- a=5, b=5, c=5
- a=6, b=5, c=8
- a=6, b=5, c=6

Seconda esercitazione

Domanda 1
L'operando (%eax, %ecx, 5) è valido?

- Sì
- No

Domanda 2
Si consideri la variabile int* p e si assuma che venga tenuta nel registro %eax. A quale istruzione assembly corrisponde l'istruzione C p++?

- incl %eax
- addl $4,%eax
- addl $2,%eax
- incw %ax

Domanda 3
Quali delle seguenti operazioni IA32 permette di azzerare il registro %eax?

- movl $0, %eax
- andl $0, %eax
- xorl %eax, %eax
- qualunque delle precedenti

Domanda 4
Quali dei seguenti predicati C permette di verificare se la variabile int x contiene un valore pari?

- x & 1 == 0
- x | 1 == 1
- x % 2 == 1
- nessuna delle precedenti

Domanda 5
Come tradurresti in IA32 l'assegnamento v[5] = 7, assumendo che v sia int* e sia tenuto nel registro %eax?

- movl $7, 5(%eax)
- movl $7, 20(%eax)
- movl $7, 10(%eax)
- nessuna delle precedenti

Domanda 6
Quale operazione useresti per scambiare i 16 bit meno significativi di int x con i 16 più significativi?

- x = (x << 16) + (x >> 16);
- x = (x << 16) + ((unsigned)x >> 16);
- nessuna delle precedenti

Domanda 7
Si consideri la riga di comando gcc main.c prova.s -o prova su una piattaforma a 64 bit. Dove prova.s è un codice IA32. Che esito probabile ti aspetteresti?

- Link error
- Segmentation Fault
- Il programma, se corretto, funziona normalmente.

Terza esercitazione

Domanda 1
Se una funzione foo ha un prologo in cui vengono salvati due registri callee-save e vengono riservati 12 byte per ospitare variabili locali, argomenti ed eventuale padding, quale di questi operandi permette di accedere al secondo argomento di foo?

- (%esp)
- 4(%esp)
- 8(%esp)
- 12(%esp)
- 20(%esp)
- 24(%esp)
- 28(%esp)
- 32(%esp)

Domanda 2
Assumendo %al = 5, eseguire "movsbl %al, %eax" porta allo stesso risultato in %eax rispetto eseguire "movzbl %al, %eax"?

- Sì
- No

Domanda 3
Assumendo di avere una funzione foo che chiama una funzione baz.
Quale tra le seguenti affermazioni risulta essere **vera**?

- foo non può utilizzare il registro %eax
- foo non può utilizzare il registro %ebx
- baz non può utilizzare il registro %eax
- baz non può utilizzare il registro %ebx
- Nessuna delle precedenti affermazioni è vera

Domanda 4
Se una funzione baz viene chiamata da una funzione foo, quale delle seguenti affermazioni risulta essere **falsa**?

- baz prima di poter utilizzare %edi deve salvare il suo contenuto e ripristinarlo prima di effettuare la ret
- baz può utilizzare %edx senza dover preservare il suo contenuto iniziale
- foo deve salvare il contenuto di %ecx se vuole preservarne il contenuto prima di chiamare baz
- foo deve salvare il contenuto di %esi se vuole preseverne il valore prima di chiamare baz

Domanda 5
Quale fra le seguenti istruzioni risulta essere valida:

- setl %eax
- setba %al
- leal (%eax, %edx, 6), %ecx
- movl (%eax), 4(%esp)
- leal -1(%ecx), %eax
- addl %eax, $4

Domanda 6

Assumendo che %eax=0x0000BEEF, quanto vale %ecx dopo aver eseguito l'istruzione movsbw %al, %cx?

- 0x000000EF
- 0xFFFFFFEF
- 0x0000FFEF
- 0x0000EFEF

Domanda 7

Se %ecx=0, qual è il valore di %al dopo le istruzioni testl %ecx, %ecx e setge %al

- 0
- 1
- nessuna delle precedenti

Quarta esercitazione

Domanda 1

Sia s un puntatore a stringa C costante memorizzata all'indirizzo (valido) 0xAABBCCDD, cosa stampa l'istruzione "printf("%x", *s++);" ?

- AABBCCDD
- AABBCCDE
- il valore esadecimale del carattere presente all'indirizzo AABBCCDD
- il valore esadecimale del carattere presente all'indirizzo AABBCCDE (ossia 0xAABBCCDD+1)
- non viene stampato nulla perchè si genera un segmentation fault o non compila
- il valore esadecimale incrementato di 1 del carattere presente all'indirizzo AABBCCDD
- il valore esadecimale incrementato di 1 del carattere presente all'indirizzo AABBCCDE

Domanda 2

Sia s un puntatore a stringa C memorizzata all'indirizzo (valido) 0xAABBCCDD, cosa stampa l'istruzione "printf("%x", (*s)++);"?

- AABBCCDD
- AABBCCDE
- il valore esadecimale del carattere presente all'indirizzo 0xAABBCCDD
- il valore esadecimale del carattere presente all'indirizzo 0xAABBCCDE (ossia 0xAABBCCDD+1)
- Se la stringa è constante non viene stampato nulla perchè si genera un segmentation fault o non compila
- il valore esadecimale incrementato di 1 del carattere presente all'indirizzo AABBCCDD
- il valore esadecimale incrementato di 1 del carattere presente all'indirizzo AABBCCDE

Domanda 3

Sia s un puntatore a stringa C costante memorizzata all'indirizzo (valido) 0xAABBCCDD, cosa stampa l'istruzione "printf("%x", ++*s);"?

- AABBCCDD
- AABBCCDE
- il valore esadecimale del carattere presente all'indirizzo 0xAABBCCDD
- il valore esadecimale del carattere presente all'indirizzo 0xAABBCCDE (ossia 0xAABBCCDD+1)
- non viene stampato nulla perchè si genera un segmentation fault o non compila
- il valore esadecimale incrementato di 1 del carattere presente all'indirizzo AABBCCDD
- il valore esadecimale incrementato di 1 del carattere presente all'indirizzo AABBCCDE

Domanda 4

Sia s un puntatore a stringa C costante memorizzata all'indirizzo (valido) 0xAABBCCDD, cosa stampa l'istruzione "printf("%x", ++(*s));"?

- AABBCCDD
- AABBCCDE
- il valore esadecimale del carattere presente all'indirizzo 0xAABBCCDD
- il valore esadecimale del carattere presente all'indirizzo 0xAABBCCDE (ossia 0xAABBCCDD+1)
- non viene stampato nulla perchè si genera un segmentation fault o non compila
- il valore esadecimale incrementato di 1 del carattere presente all'indirizzo AABBCCDD
- il valore esadecimale incrementato di 1 del carattere presente all'indirizzo AABBCCDE

Domanda 5
Data la struct C "struct s { char x, short * y; int z;}", quanti byte (sizeof) sono necessari per memorizzarla in stack?

- 7
- 8
- 10
- 12
- 14
- 16

Domanda 6
Data la struct C "struct s { char x, short y; char z}", quanti byte di padding sono presenti nella struct?

- 0
- 1
- 2
- 3
- 4

Domanda 7
Data la struct C "struct s { char x, int y; char * z; short q; char p; }", qual è l'offset del campo p?

- 8
- 10
- 12
- 14
- 16

Domanda 8
Considerata la costante -72, quale tra le seguenti affermazioni risulta essere **vera**?

- La sua rappresentazione binaria (8bit) è 0100 1000
- La sua rappresentazione binaria (16 bit) è 1111 1111 1011 0110
- La sua rappresentazione binaria (16 bit) è 0000 0000 1011 0110
- La sua rappresentazione binaria (16bit) è 1111 1111 0100 1000
- Se gli viene sommato 1 la sua rappresentazione binaria (16 bit) è 1111 1111 1011 1000
- Nessuna delle precedenti

Quinta esercitazione

Domanda 1
Assumendo che il registro %ecx contenga il valore 0xF0F0F0F0, dopo aver eseguito l'istruzione "shll $4, %ecx" quale delle seguenti affermazioni risulta **vera**?

- Il bit più significativo di %ecx è 1
- Il bit meno significativo di %ecx è 1
- %cl contiene il valore 0
- %ch contiene il valore 0
- Se si esegue l'istruzione "sarl $4, %ecx" si riporta il valore di %ecx a 0xF0F0F0F0

Domanda 2
Assumendo che il registro %ecx contenga il valore 0x00FF00FF, quale delle seguenti affermazioni risulta **vera**?

- Se eseguiamo "sarl $3, %ecx" otteniamo in %ecx un numero più grande rispetto a prima
- Se eseguiamo "sarl $8, %ecx" oppure "shrl $8, %ecx" otteniamo lo stesso valore in %ecx
- Se eseguiamo "sarl $31, %ecx" otteniamo in %ecx un numero diverso da 0
- Se eseguiamo "sarb $1, %cl" il valore contenuto in %ecx cambia
- Nessuna delle precedenti

Domanda 3
Quale delle seguenti istruzioni *NON* calcola lo stesso risultato di "shll $1, %eax":

- imul $2, %eax
- addl %eax, %eax
- leal (%eax, %eax), %eax
- leal 2(%eax), %eax
- sall $1, %eax

Domanda 4
Siano %eax=33 (decimale), %edx=0 (decimale) e %ecx=10 (decimale), quale delle seguenti affermazioni risulta **vera**?

- Per calcolare %eax diviso 10 posso eseguire "idivl %eax" oppure "sarl $10, %eax"
- "idivl $3" è un'istruzione valida e calcola "%edx:%eax" diviso 3
- Non posso usare "sarl" per calcolare %eax diviso 2, perché %eax contiene un valore che non è una potenza di 2
- "idivl %ecx" scrive il valore 3 nel registro %eax e nel registro %edx
- Nessuna delle precedenti

Domanda 5
Siano %eax=20 (decimale), %edx=0 (decimale) e %ecx=8 (decimale). Con riguardo alle due istruzioni "idivl %ecx" e "sarl $3, %eax", quale delle seguenti affermazioni risulta **vera**?

- Le due istruzioni scrivono lo stesso valore in %eax, ma "idivl" è più efficiente di "sarl"
- Le due istruzioni scrivono lo stesso valore in %eax, ma "sarl" è più efficiente di "idivl"
- Le due istruzioni scrivono valori diversi in %eax
- In questo caso la "idivl" non modifica il valore del registro %edx
- Nessuna delle precedenti

Domanda 6
Assumendo %eax=0xFF000000, %ecx=1 (decimale) e %edx=10 (decimale), dopo aver eseguito l'istruzione "testl %eax, %eax" quale delle seguenti affermazioni risulta **vera**?

- Se eseguiamo "cmovnzl %edx, %ecx" viene scritto il valore 10 nel registro %ecx
- Se eseguiamo "cmovzl %edx, %ecx" viene scritto il valore 0 nel registro %ecx
- L'istruzione "cmovnzw %dx, %cx" non modifica il valore del registro %ecx
- L'istruzione "cmovel %edx, %ecx" è equivalente all'istruzione "cmovnzl $10, %ecx"
- Nessuna delle precedenti

Domanda 7

Assumendo %eax=10 (decimale), %ecx=7 (decimale) e %edx=2 (decimale).
Quale delle seguenti affermazioni risulta **vera**?

- Se eseguiamo "cmpl %ecx, %eax" e "cmovlel %edx, %eax" viene scritto il valore 2 nel registro %eax
- Se eseguiamo "cmpl %ecx, %eax" e "cmovbl %edx, %eax" viene scritto il valore 2 nel registro %eax
- Se eseguiamo "cmpb $0, %al" e "cmovel %edx, %eax" viene scritto il valore 2 nel registro %eax
- Se eseguiamo "subl %ecx, %eax" e "cmovgl %edx, %eax" viene scritto il valore 2 nel registro %eax
- Nessuna delle precedenti

Domanda 8

Quale delle seguenti è un'istruzione valida:

- cmovzb %al, %cl
- cmovzl $3, %eax
- cmovgew %ax, (%ecx)
- cmpl %ecx, $10
- Nessuna delle precedenti

Sesta esercitazione

Domanda 1
Dato il seguente codice in linguaggio C e la sua traduzione in linguaggio assembly, dire quale tra le seguenti tecniche di ottimizzazione è stata applicata:

```c
#define FACTOR 3

void f(int *v, int n) {
    int i, x = 10;
    for (i = 0; i < n; i++) {
        v[i] += x * FACTOR;
    }
}
```

```
f:
    movl    4(%esp), %ecx
    movl    8(%esp), %edx
    testl   %edx, %edx
    jle L1
    movl    %ecx, %eax
    leal    (%ecx,%edx,4), %edx
L3:
    addl    $30, (%eax)
    addl    $4, %eax
    cmpl    %edx, %eax
    jne L3
L1:
    ret
```

- Constant propagation e dead code elimination
- Constant folding e common subexpression elimination
- Dead code elimination, constant folding e loop unrolling
- Constant folding e constant propagation
- Nessuna delle precedenti

Domanda 2
Quale tra le seguenti affermazioni è **falsa**?

- La tecnica dell'augmentation mira a migliorare le prestazioni in termini di tempo, ma può causare un peggioramento delle prestazioni in termini di spazio
- Il compilatore può applicare la tecnica della register allocation su una variabile anche se la variabile non viene dichiarata con la parola chiave register
- La tecnica del loop invariant code motion non è mai applicata dai compilatori
- La tecnica dell'augmentation non è mai applicata dai compilatori
- Le tecniche di cortocircuitazione sfruttano la semantica degli operatori booleani && e II del C

Domanda 3
Data la seguente funzione f in linguaggio C e la sua traduzione in linguaggio assembly, dire quale tra le seguenti tecniche di ottimizzazione è stata applicata

```c
int f(int x) {
    int s = 0;
    while (x > 0) s += x--;
    return s;
}
```

```
.globl  f
f:
    subl    $16, %esp
    movl    $0, 12(%esp)
    jmp L2
L3:
    movl    20(%esp), %eax
    leal    -1(%eax), %edx
    movl    %edx, 20(%esp)
    addl    %eax, 12(%esp)
L2:
    cmpl    $0, 20(%esp)
    jg  L3
    movl    12(%esp), %eax
    addl    $16, %esp
    ret
```

- Loop unrolling
- Register allocation
- Common subexpression elimination
- Loop invariant code motion
- Nessuna delle precedenti

Domanda 4
Dato il seguente codice quale delle seguenti affermazioni risulta **vera**?

```c
int c = 0;

int g(int x) {
    c++;
    return x*x;
}

int f(int *v, int n, int x) {
    int i, a = 0;
    for (i = 0; i < n; i++)
        a += i + g(x);
    return a;
}
```

- gcc non è in grado di applicare la tecnica del loop invariant code motion in questo codice
- gcc è in grado di applicare la tecnica del loop invariant code motion, ma solo se g è definita nello stesso file sorgente di f
- gcc è in grado di applicare la tecnica del loop invariant code motion, ma solo se si attiva almeno il livello di ottimizzazione 3 (gcc -O3)
- gcc è in grado di applicare la tecnica del loop invariant code motion, ma solo se g viene dichiarata "static"

Domanda 5
Dato il seguente codice in linguaggio C e la sua traduzione in linguaggio assembly, dire quale tra le seguenti tecniche di ottimizzazione è stata applicata:

```c
int hash(int x, int y) {
    return 31*(31 + x) + y;
}
```

```
hash:
    movl    4(%esp), %eax
    leal    31(%eax), %edx
    movl    %edx, %eax
    sall    $5, %eax
    subl    %edx, %eax
    addl    8(%esp), %eax
    ret
```

- Common subexpression elimination
- Constant propagation
- Constant folding
- Algebraic identities
- Dead code elimination
- Strength reduction
- Nessuna delle precedenti

Domanda 6
Dato un programma P, supponendo di ottimizzare una porzione di codice che impegna il 40% del tempo totale di esecuzione di P, ottenendo su tale porzione di codice uno speedup di 2x, qual è lo speedup complessivo su P?

- circa 1.67x
- 1.25x
- 0.8x
- 2x
- Nessuna delle precedenti

Domanda 7
Dato un programma P, supponendo di ottimizzare una porzione di codice che impegna il 75% del tempo totale di P, qual è lo speedup *massimo* che possiamo ottenere su P?

- Non è possibile determinare alcun limite allo speedup ottenibile su P, senza conoscere lo speedup ottenuto sulla porzione di codice ottimizzata
- 4/3 x
- 4x
- 0.75x
- Nessuna delle precedenti

Domanda 8
Data la seguente porzione del report di gprof per un dato programma, quale funzione rappresenta il maggior collo di bottiglia prestazionale (bottleneck)?

```
            Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.20% of 4.89 seconds

index % time    self  children    called     name
                                               <spontaneous>
[1]    100.0    0.00    4.89                 main [1]
                0.00    4.56    1000/1000        A [2]
                0.02    0.31    1000/1000        B [5]
                0.00    0.00       1/1           init [6]
-----------------------------------------------
                0.00    4.56    1000/1000        main [1]
[2]     93.3    0.00    4.56    1000         A [2]
                4.26    0.31    1000/1000        C [3]
-----------------------------------------------
                4.26    0.31    1000/1000        A [2]
[3]     93.3    4.26    0.31    1000         C [3]
                0.31    0.00   10000/20000       D [4]
-----------------------------------------------
                0.31    0.00   10000/20000       B [5]
                0.31    0.00   10000/20000       C [3]
[4]     12.6    0.61    0.00   20000         D [4]
-----------------------------------------------
                0.02    0.31    1000/1000        main [1]
[5]      6.7    0.02    0.31    1000         B [5]
                0.31    0.00   10000/20000       D [4]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[6]      0.0    0.00    0.00       1         init [6]
-----------------------------------------------
```

- main
- A
- B
- C
- D
- init

Settima esercitazione

Domanda 1
Quale tra le seguenti affermazioni è **falsa**?

- Alcune istruzioni x86 non possono essere eseguite quando la CPU è in “user mode”
- All’interno del kernel le system call sono identificate da un numero
- L’istruzione int genera un interrupt sincrono che passa il controllo al kernel
- Funzioni wrapper di system call come read, write, fork e _exit vengono eseguite interamente in “kernel mode”
- Le system call eseguono codice del kernel

Domanda 2
Quale tra le seguenti affermazioni è **vera**?

- POSIX è un sistema operativo
- La X di POSIX è un riferimento a Linux.
- La funzione printf non fa parte dello standard POSIX, ma della libreria standard C (libc)
- POSIX definisce solo l’interfaccia tra programmi utente e kernel basata su system call
- Nessuna delle precedenti

Domanda 3
Quale tra le seguenti affermazioni è **vera**?

- Una “trap” è un tipo di interruzione asincrona
- Una “trap” viene generata, ad esempio, quando si muove il mouse
- L’istruzione int può essere usata esclusivamente per invocare le system call
- L’interrupt generata dal timer è asincrona
- Nessuna delle precedenti

Domanda 4
Quale tra le seguenti affermazioni è **vera**?

- Una system call si invoca con l’istruzione int seguita da un numero che identifica la system call da eseguire
- Il valore di ritorno di una system call viene scritto nel registro EAX.
- Un programma utente può passare i parametri ad una system call anche copiandoli sullo stack del kernel
- Se una system call viene invocata da un programma utente, viene eseguita interamente in user mode
- Nessuna delle precedenti

Domanda 5
Quale tra le seguenti affermazioni è **vera**?

- perror non è una system call, ma un wrapper che invoca una system call
- perror stampa un messaggio di errore che dipende dal valore contenuto nella variabile errno
- errno è una variabile di tipo char* che contiene la stringa identificativa dell’errore
- La variabile errno viene restituita come valore di ritorno dalle funzioni della libreria standard C in caso di errore
- Nessuna delle precedenti

Domanda 6
Relativamente al seguente codice C (supponendo che la fork non generi un errore), quale tra le seguenti affermazioni è **vera**?

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>

int main() {
    int x = 1;
    pid_t pid = fork();
    if (pid == -1) {
        perror("Errore nella fork");
        exit(1);
    }
    if (pid == 0) {
        printf("Figlio: %d\n", x);
        x = 2;
        _exit(0);
    }
    wait(NULL);
    printf("Padre: %d\n", x);
    return 0;
}
```

- Viene stampato “Padre: 1” e poi “Figlio: 2”
- Viene stampato “Figlio: 2” e poi “Padre: 1”
- Viene stampato “Figlio: 1” e poi “Padre: 1”
- Viene stampato “Figlio: 1” e poi “Padre: 2”
- Viene stampato “Figlio: 2” e poi “Padre: 2”
- L’ordine delle stampe è impredicibile, poiché dipende dallo scheduler
- Nessuna delle precedenti

Domanda 7
Relativamente al seguente codice C (supponendo che la fork non generi un errore), qual è l'output atteso?

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>

#define N 3

int main() {
    int i, x = 10;
    for (i = 1; i <= N; i++) {
        pid_t pid = fork();
        if (pid == -1) {
            perror("Errore nella fork");
            exit(1);
        }
        if (pid == 0) { // figlio
            printf("Figlio %d: x=%d\n", i, x);
            exit(0);
        }
        wait(NULL);
        x += 10;
    }
    printf("Padre: x=%d\n", x);
    return 0;
}
```

- Nell'ordine (una per linea): "Figlio 1: x=10", "Figlio 2: x=20", "Figlio 3: x=30", "Padre: x=40"
- Nell'ordine (una per linea): "Figlio 1: x=10", "Figlio 2: x=10", "Figlio 3: x=10", "Padre: x=30"
- Nell'ordine (una per linea): "Figlio 1: x=10", "Figlio 2: x=10", "Figlio 3: x=10", "Padre: x=40"
- Nell'ordine (una per linea): "Figlio 1: x=10", "Figlio 2: x=20", "Figlio 3: x=30", "Padre: x=30"
- Nell'ordine (una per linea): "Figlio 1: x=10", "Figlio 2: x=30", "Figlio 3: x=60", "Padre: x=100"
- L'ordine delle stampe è impredicibile, poiché dipende dallo scheduler
- Nessuna delle precedentiù

Domanda 8
Relativamente al seguente codice C (supponendo che la execv non generi un errore), qual è l'output atteso?

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

int main() {
    printf("--- started ---\n");
    char *args[] = {"/bin/echo", "Hello", NULL};
    execv(args[0], args);
    printf("--- finished ---\n");
    return 0;
}
```

- Nell'ordine (una per linea): "--- started ---", "Hello", "NULL, "--- finished ---"
- Nell'ordine (una per linea): "--- started ---", "Hello", "--- finished ---"
- Nell'ordine (una per linea): "--- started ---", "Hello"
- Nell'ordine (una per linea): "--- started ---", "--- finished ---"
- L'ordine delle stampe è impredicibile, poiché dipende dallo scheduler
- Nessuna delle precedenti

Ottava esercitazione

Domanda 1
Con riferimento ai permessi sui file, la notazione ottale 0640 a quale configurazione di permessi corrisponde?

- -wx--x---
- rw-r-----
- r-xr-----
- ---rw-r--
- rw-r--r--
- rwxrw-r--
- Nessuna delle precedenti

Domanda 2
Dato il seguente codice, quale tra le seguenti affermazioni è **vera**?

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

void check_error(const char *msg, int res) {
    if (res == -1) {
        perror(msg);
        exit(EXIT_FAILURE);
    }
}

int main() {
    int fd = open("test.txt", O_RDWR | O_CREAT | O_TRUNC, 0600);
    check_error("open", fd);

    int r = write(fd, "hello", 5);
    check_error("write", r);

    r = close(fd);
    check_error("close", r);

    return EXIT_SUCCESS;
}
```

• Il file test.txt viene aperto in sola scrittura
• Se il file test.txt esiste già, la open restituisce -1
• Il file test.txt viene creato con i permessi di lettura e scrittura per "user", "group" e "others"
• Se il programma termina correttamente, dopo l'esecuzione, il file test.txt contiene esclusivamente i caratteri hello
• La write fallisce sempre perché non è possibile scrivere sul file a causa di come viene creato dalla open
• Nessuna delle precedenti

Domanda 3
Quale tra le seguenti affermazioni è **falsa**?
• La directory /proc/<pid> , dove <pid> è un numero intero, contiene informazioni sul processo con PID=<pid>
• Nello spazio di indirizzamento logico di un processo l'area riservata allo stack cresce verso indirizzi bassi, mentre l'area riservata allo head cresce verso indirizzi alti
• La tabella delle pagine di un processo P permette di tradurre gli indirizzi logici di P in indirizzi fisici
• Il fenomeno del trashing si verifica quando la cardinalità (in termini di numero di pagine) dell'unione dei working set dei processi in esecuzione supera il numero di frame fisici a disposizione
• In ogni istante in cui un processo è nello stato "running", tutte le sue pagine di memoria devono essere mappate su frame della memoria fisica

Domanda 4
Data la seguente porzione di codice, selezionare la risposta **vera**:

```
int main() {
    char s[] = "Test";
    // ...
}
```

• s è una variabile puntatore che punta ad un array di char allocato nella sezione "data" del processo
• s è una variabile puntatore che punta ad un array di char allocato nello stack frame della funzione main
• s è un array di char allocato nella sezione "data" del processo
• s è un array di char allocato nello stack frame della funzione main
• Nessuna delle precedenti

Domanda 5
Data la seguente porzione di codice, selezionare la risposta **vera**:

```
int main() {
    char *s = "Test";
    // ...
}
```

- s è una variabile puntatore che punta ad un array di char allocato nella sezione "data" del processo
- s è una variabile puntatore che punta ad un array di char allocato nello stack frame della funzione main
- s è un array di char allocato nella sezione "data" del processo
- s è un array di char allocato nello stack frame della funzione main
- Nessuna delle precedenti

Domanda 6

Supponendo di essere su una architettura con indirizzi di 32 bit, se la dimensione di una pagina di memoria è 4 kB (4096 byte), lo spazio di indirizzamento logico di un processo quante pagine (allocate o meno) contiene? (con la notazione 2^n si indica 2 elevato al numero n)

- 2^32
- 2^12
- 2^20
- 2^30
- Dipende dal numero di frame contenuti nella memoria fisica
- Nessuna delle precedenti

Domanda 7

Sia A = 0x08F444D8 un indirizzo nello spazio di indirizzamento logico di un processo P, su un'architettura a 32 bit. Si assuma che la memoria virtuale utilizzi pagine di 4 kB (4096 byte).
Indicare il numero di pagina e l'offset di A (in decimale):

- Pagina = 586820, offset = 216
- Pagina = 36676, offset = 1240
- Pagina = 143, offset = 216
- Pagina = 2292, offset = 17624
- Dipende dalla tabella delle pagine
- Nessuno dei precedenti

Domanda 8

Sia A = 0xFF9257B0 un indirizzo nello spazio di indirizzamento logico di un processo P, su un'architettura a 32 bit. Si assuma che la memoria virtuale utilizzi pagine di 4 kB (4096 byte) e che la pagina dell'indirizzo A sia stata mappata nel frame numero 128 (decimale). Indicare l'indirizzo fisico corrispondente all'indirizzo logico A:

- 0xFF925128
- 0xFF925080
- 0xFF1287B0
- 0x008057B0
- 0x000807B0
- 0x809257B0
- Nessuno dei precedenti

Nona esercitazione

Domanda 1
Quale delle seguenti affermazioni è **vera**?

- Un processo è nello stato zombie se non ha ancora terminato la sua esecuzione, ma il processo padre è già terminato.
- Un processo non può fare waitpid(pid,&status,0) se pid è il PID di un processo zombie
- Quando nel processo padre la wait(&status) ritorna, dopo che il processo figlio è terminato con exit(n), _exit(n) o return n, la variabile status è uguale a n
- WEXITSTATUS(status) estrae il byte meno significativo dalla variabile int status
- Nessuna delle precedenti

Domanda 2
Dato il seguente codice, quale tra le seguenti affermazione è **vera**?

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/wait.h>

#define NUM_CHILDREN 3

int child_code(int n) {
    printf("I'm child %d!\n", n);
    exit(EXIT_SUCCESS);
}

int main(int argc, char *argv[]) {
    pid_t pids[NUM_CHILDREN];

    int i = 0;
    for (i = 0; i < NUM_CHILDREN; i++) {
        pids[i] = fork();
        if (pids[i] == 0) {
            child_code(i);
        }
    }

    while (--i >= 0) {
        waitpid(pids[i], NULL, 0);
    }

    return 0;
}
```

- Il programma stampa "I'm child 0", "I'm child 1","I'm child 2" ma l'ordine delle stampe non è predicibile perché dipende dallo scheduler
- Il programma stampa nell'ordine: "I'm child 0", "I'm child 1", "I'm child 2"
- Il programma stampa nell'ordine: "I'm child 2", "I'm child 1", "I'm child 0"
- Il programma causa un numero infinito di fork
- Il processo padre termina prima dei figli
- Nessuna delle precedenti

Domanda 3
Data la seguente porzione di codice, quale delle seguenti affermazioni è **falsa**?

```c
#include<stdio.h>
#include<unistd.h>

int main(int argc, char *argv[]) {
    int s = getpagesize();
    int m = s - 1;
    int t = s, k = 0;
    while (t >>= 1) k++;

    int x = 100;
    void *p = &x;

    unsigned A = (unsigned)p / s;
    unsigned B = (unsigned)p % s;
    unsigned C = (unsigned)p >> k;
    unsigned D = (unsigned)p & m;
    unsigned E = (unsigned)p & ~m;
    unsigned F = (unsigned)p % (1 << k);

    return 0;
}
```

- A è il numero di pagina dell'indirizzo di x
- B è l'offset (all'interno della pagina) dell'indirizzo di x
- C è il numero di pagina dell'indirizzo di x
- D è l'offset (all'interno della pagina) dell'indirizzo di x
- E è il numero di pagina dell'indirizzo di x
- F è l'offset (all'interno della pagina) dell'indirizzo di x

Domanda 4
Dato il seguente codice, selezionare la risposta **vera**:

```c
#include<stdio.h>
#include<stdlib.h>
#include<signal.h>

void handler(int signo) {
    printf("Catturato segnale\n");
}

int main(int argc, char *argv[]) {
    struct sigaction act = { 0 };
    act.sa_handler = handler;
    if (sigaction(SIGFPE, &act, NULL) == -1){
        perror("sigaction");
        exit(EXIT_FAILURE);
    }

    printf("%d\n", 3/0);

    return 0;
}
```

- Il programma non termina (o termina dopo molto)
- Il programma termina con un core dump
- Il programma stampa "Catturato segnale" e poi termina
- Il programma stampa "inf" e termina
- Nessuna delle precedenti

Domanda 5
Dato il seguente codice, selezionare la risposta **vera**:

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

int main(int argc, char *argv[]) {
    alarm(3);
    pause();
    printf("bye!\n");
    return 0;
}
```

- Il programma non termina
- Il programma termina dopo circa 3 secondi stampando "bye!"
- Il programma termina dopo circa 3 secondi ma non stampa "bye!"
- Il programma stampa "bye!" ogni 3 secondi
- Nessuna delle precedenti

Domanda 6

Quale delle seguenti affermazioni è **falsa**?

- Il segnale SIGKILL non può essere ignorato né catturato
- Il segnale SIGSEGV non può essere catturato
- Il segnale SIGCHLD viene ignorato per default
- Il segnale SIGINT, se non catturato o ignorato, causa la terminazione del processo che lo riceve
- Il segnale SIGQUIT , se non catturato o ignorato, causa la terminazione del processo che lo riceve

Domanda 7

Quale delle seguenti affermazioni è **vera**?

- Il comando kill -9 <pid> invia il segnale SIGINT al processo con pid <pid>
- Il segnale inviato col comando kill -SIGINT <pid> non può essere catturato dal processo con pid <pid>
- alarm(s) invia il segnale SIGALRM ogni s secondi
- ualarm(t, n) invia un segnale dopo t microsecondi e successivamente ogni n microsecondi
- Nessuna delle precedenti

Domanda 8

Quale delle seguenti affermazioni è **falsa**?

- mmap può essere usata per allocare al processo intere pagine di memoria
- mmap permette di specificare i permessi di accesso sulle pagine di memoria che alloca
- mprotect permette di specificare i permessi di accesso sulle pagine di memoria, ma solo se sono state allocate con mmap
- Il primo parametro di mprotect deve essere un indirizzo fisico (non logico)
- getpagesize() restituisce la dimensione delle pagine di memoria

Decima esercitazione

Domanda 1
Quale delle seguenti affermazioni è **vera**?

- malloc è una system call
- malloc alloca tutti blocchi della stessa dimensione
- malloc restituisce indirizzi fisici
- L'indirizzo passato alla free punta al payload del blocco allocato da malloc
- Nessuna delle precedenti

Domanda 2
Dato il seguente codice, quale tra le seguenti affermazione è **vera**?

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {
    int status;
    pid_t pid = fork();
    if (pid == -1) return EXIT_FAILURE;

    if (pid == 0) {
        printf("bye!\n");
        _exit(0);
    }

    sleep(10);
    wait(&status);
    return EXIT_SUCCESS;
}
```

- Il processo figlio non entra mai nello stato zombie poiché il padre esegue la wait
- Il processo figlio diventa orfano non appena il processo padre termina, e viene quindi adottato (cioè diventa figlio di un altro processo) da un processo antenato (nell'albero dei processi) del processo padre
- Il processo figlio entra nello stato zombie dopo aver terminato la propria esecuzione e fino al momento in cui il processo padre raccoglie il suo exit status
- Il processo padre termina sicuramente prima del figlio in virtù della chiamata a wait
- Nessuna delle precedenti

Domanda 3
Dato il seguente output di valgrind, quale delle seguenti affermazioni è **vera**?

```
==11403== Memcheck, a memory error detector
==11403== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==11403== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==11403== Command: ./a.out
==11403==
==11403== Invalid write of size 4
==11403==    at 0x400621: crea_array_int (main3.c:11)
==11403==    by 0x400651: duplica (main3.c:18)
==11403==    by 0x4006FC: main (main3.c:31)
==11403==  Address 0x5204054 is 0 bytes after a block of size 20 alloc'd
==11403==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==11403==    by 0x4005F1: crea_array_int (main3.c:6)
==11403==    by 0x400651: duplica (main3.c:18)
==11403==    by 0x4006FC: main (main3.c:31)
==11403==
==11403==
==11403== HEAP SUMMARY:
==11403==     in use at exit: 0 bytes in 0 blocks
==11403==   total heap usage: 1 allocs, 1 frees, 20 bytes allocated
==11403==
==11403== All heap blocks were freed -- no leaks are possible
==11403==
==11403== For counts of detected and suppressed errors, rerun with: -v
==11403== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

- Il programma profilato da valgrind soffre di memory leak
- Il programma accede in scrittura oltre la fine di un blocco allocato con malloc
- Non si è verificato nessun errore durante l'esecuzione del programma
- Nel programma non viene mai invocata la funzione malloc
- Al termine dell'esecuzione del programma 20 byte sono stati allocati, ma non sono stati liberati tramite free()
- Nessuna delle precedenti

Domanda 4
Dato il seguente output di valgrind, quale delle seguenti affermazioni è **falsa**?

```
==17605== Memcheck, a memory error detector
==17605== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==17605== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==17605== Command: ./a.out
==17605==
==17605== Conditional jump or move depends on uninitialised value(s)
==17605==    at 0x400507: evaluate (main.c:11)
==17605==    by 0x400520: main (main.c:17)
==17605==
==17605==
==17605== HEAP SUMMARY:
==17605==     in use at exit: 0 bytes in 0 blocks
==17605==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==17605==
==17605== All heap blocks were freed -- no leaks are possible
==17605==
==17605== For counts of detected and suppressed errors, rerun with: -v
==17605== Use --track-origins=yes to see where uninitialised values come from
==17605== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

- Valgrind segnala un errore che riguarda lo heap
- Valgrind segnala un errore nella funzione evaluate alla riga 11 del file sorgente main.c
- In tutto il programma non viene effettuata nessuna malloc/free
- L'errore segnalato da valgrind è causato da un'area di memoria non inizializzata
- la funzione evaluate è invocata direttamente dalla funzione main

Domanda 5
Dato il seguente output di valgrind, quale delle seguenti affermazioni è **falsa**?

```
==18770== Memcheck, a memory error detector
==18770== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==18770== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==18770== Command: ./a.out
==18770==
Hello, world
==18770==
==18770== HEAP SUMMARY:
==18770==     in use at exit: 13 bytes in 1 blocks
==18770==   total heap usage: 2 allocs, 1 frees, 1,037 bytes allocated
==18770==
==18770== 13 bytes in 1 blocks are definitely lost in loss record 1 of 1
==18770==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==18770==    by 0x4006A0: clone (main.c:9)
==18770==    by 0x4006DF: capitalized (main.c:15)
==18770==    by 0x400721: say_hello (main.c:21)
==18770==    by 0x400757: main (main.c:29)
==18770==
==18770== LEAK SUMMARY:
==18770==    definitely lost: 13 bytes in 1 blocks
==18770==    indirectly lost: 0 bytes in 0 blocks
==18770==      possibly lost: 0 bytes in 0 blocks
==18770==    still reachable: 0 bytes in 0 blocks
==18770==         suppressed: 0 bytes in 0 blocks
==18770==
==18770== For counts of detected and suppressed errors, rerun with: -v
==18770== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

- Valgrind segnala un errore di tipo memory leak
- Il programma profilato da valgrind esegue una sola malloc
- Valgrind segnala che nella funzione clone (alla riga 9 del sorgente main.c) viene allocato un blocco di 13 byte tramite malloc, ma non viene mai liberato tramite free
- Il programma profilato da valgrind ha "Hello, world" come suo unico output sul terminale
- Il nome del programma profilato da valgrind è a.out

Domanda 6
Quale delle seguenti affermazioni è **vera**?

- La frammentazione esterna si verifica a causa del padding
- L'allocatore della mamoria dinamica (malloc/free) non soffre di frammentazione esterna
- Il meccanismo di paginazione della memoria virtuale non soffre di frammentazione interna, poiché tutte le pagine hanno la stessa dimensione
- L'area di swap è necessaria per contrastare la frammentazione interna
- Nessuna delle precedenti

Domanda 7
Quale delle seguenti affermazioni è **falsa**?

- Il comando env stampa le variabili d'ambiente della shell
- echo $HOME stampa il valore della variabile d'ambiente HOME
- Quando si lancia un comando o un programma attraverso la shell specificando solo il suo nome, questo viene cercato nei percorsi specificati nella varibile d'ambiente PATH
- Le variabili d'ambiente definite attraverso la shell con la parola chiave export vengono ereditate dai processi figli della shell
- Non è possibile modificare il contenuto di una variabile d'ambiente esistente

Domanda 8

Quale delle seguenti affermazioni è **falsa**?

- La funzione main può essere definita con un terzo parametro che punta a un array di puntatori a char la cui fine è marcata dal valore NULL.
- getenv("HOME") restituisce un puntatore ad un array di char che contiene il valore della variabile d'ambiente HOME
- setenv("GREET", "HELLO", 0) assegna il valore HELLO alla variabile d'ambiente GREET, ma solo se questa non esiste
- setenv("ABC", "123", 1) assegna il valore "123" alla variabile "ABC" ma solo se "ABC" esiste già
- Le variabili d'ambiente non hanno un "tipo" associato (es. int, float, double, ecc.)

Undicesima esercitazione

Domanda 1
Si consideri una cache completamente associativa con 4 linee da 16 byte ciascuna e politica di rimpiazzo LRU. Potendo scegliere durante un cold cache miss, selezionare sempre la linea con indice più basso.
Data la seguente sequenza di accessi a indirizzi di memoria, qual è il contenuto delle linee di cache alla fine della sequenza: 116, 71, 142, 162, 70, 244, 273, 132, 112 ? (l'ordine è importante)

- 10, 4, 8, 7
- 4, 15, 17, 7
- 8, 7, 17, 15
- 17, 7, 8, 15
- Nessuno dei precedenti

Domanda 2
Si consideri una cache associativa a 2 vie con 4 linee da 16 byte ciascuna e politica di rimpiazzo LRU. Potendo scegliere durante un cold cache miss, selezionare sempre la linea con indice più basso.
Data la seguente sequenza di accessi a indirizzi di memoria, qual è il contenuto delle linee di cache alla fine della sequenza: 72, 199, 131, 77, 138, 115, 40, 250, 208?

- 7, 2, 15, 13
- 8, 2, 13, 15
- 15, 2, 13, 7
- 2, 4, 13, 15
- Nessuno dei precedenti

Domanda 3
Quale delle seguenti affermazioni è **vera**?

- Una cache completamente associativa è una cache a k vie con k = 1
- Un cache miss di tipo "cold" prevede il "rimpiazzo" di una linea di cache
- Con una cache completamente associativa possono verificarsi cache miss di tipo "conflict"
- Con una cache ad accesso diretto non è possibile usare la politica di rimpiazzo LRU
- Nessuna delle precedenti

Domanda 4
Quale delle seguenti affermazioni è **falsa**?

- Le memorie cache sfruttano i principi di località spaziale e temporale per ridurre i tempi di accesso ai dati.
- I principi di località spaziale e temporale emergono spesso spontaneamente nel codice dei programmi, ma in alcuni casi il programmatore può riscrivere il codice in modo da favorirli
- La politica di rimpiazzo LRU prevede di selezionare per il rimpiazzo il blocco nella linea di cache che è stato acceduto meno recentemente
- In alcune architetture è presente una cache L1 dedicata alle istruzioni e una dedicata ai dati
- La dimensione di una linea di cache è pari alla dimensione di una pagina della memoria virtuale

Domanda 5
Data la seguente porzione di codice. Quanti cicli di clock vengono richiesti da una semplice pipeline a 5 stadi (Fetch, Decode, Execute, Memory, Write-Back) per completare tutte le istruzioni assumendo che gli hazard vengano risolti con stalli?

```
movl $1, %eax
addl %eax, %ecx
addl $2, %edi
incl %esi
addl %esi, %edx
movl $3, %ebx
```

- 16
- 10
- 13
- 18
- 25
- Nessuno dei precedenti

Domanda 6
Con riferimento alla domanda 5, è possibile ottimizzare il codice riordinando le istruzioni (senza cambiare la semantica) in modo da ridurre il numero di cicli di clock richiesti?

- No, non è possibile
- Sì, è possibile ridurre il numero di cicli di clock a 12, ma non a meno
- Sì, è possibile ridurre il numero di cicli di clock a 10, ma non a meno
- Sì, è possibile ridurre il numero di cicli di clock a 16, ma non a meno
- Sì, è possibile, ma le risposte B, C e D non sono corrette

Domanda 7
Quale delle seguenti affermazioni è **vera**?

- La tecnica di ottimizzazione nota come "instruction scheduling" permette di ridurre gli stalli, ma può essere applicata solo dal programmatore
- Il pipelining permette di ridurre il "throughput"
- Il pipelining permette di ridurre la "latenza" delle singole istruzioni
- Il pipelining è possibile soltanto nei sistemi multicore
- Nessuna delle precedenti

Domanda 8
Quale delle seguenti affermazioni è **falsa**?

- L'uso di istruzioni branchless permette di evitare hazard sul controllo
- Le architetture moderne fanno "branch prediction" quando viene eseguita una istruzione di salto condizionato (invece di inserire stalli per attendere che venga determinato il ramo da eseguire)
- CMOV è un'istruzione "branchless"
- La tecnica del pipelining permette di eseguire più processi in parallelo
- Nessuna delle precedenti

Dodicesima esercitazione

Domanda 1
Si consideri il frammento di programma sottostante e la sua versione ottimizzata. Quali ottimizzazioni sono state applicate? Una sola delle seguenti risposta è **vera**.

VERSIONE ORIGINALE:

```
#define N 10
int f(int x, int y) {
    int z = (x*x) + log(y * x * x);
    int w = 3 * N;
    while (!isprime(z)) z++;
    return w + z;
}
```

VERSIONE OTTIMIZZATA:

```
#define N 10
int f(int x, int y) {
    int x2 = (x*x);
    int z = x2 + log(y * x2);
    while (!isprime(z)) z++;
    return 30 + z;
}
```

- loop unrolling + constant folding + constant propagation
- register allocation + function inlining + common subexpression elimination
- constant folding + loop-invariant code motion + constant propagation
- common subexpression elimination + constant folding + constant propagation
- Nessuno dei precedenti

Domanda 2
Si consideri una cache associativa a 2 vie con 4 linee da 32 byte ciascuna e politica di rimpiazzo LRU. Potendo scegliere durante un cold cache miss, selezionare sempre la linea con indice più basso.
Data la seguente sequenza di accessi a indirizzi di memoria, qual è il contenuto delle linee di cache alla fine della sequenza: 362, 422, 261, 228, 45, 200, 258, 78? (l'ordine è importante)

- 2, 7, 1, 6
- 8, 2, 7, 1
- 8, 2, 1, 6
- 1, 6, 8, 2
- Nessuno dei precedenti

Domanda 3
Quale delle seguenti affermazioni è **falsa**?

- Le system call vengono eseguite in modalità kernel
- Le system call sono identificate da un numero
- Nel kernel Linux i parametri delle system call vengono passati attraverso i registri della CPU
- L'istruzione int $0x80 viene usata per invocare una system call, dove $0x80 è il numero della system call
- Nessuna delle precedenti

Domanda 4
Si consideri un sistema di memoria virtuale con uno spazio di indirizzi a 20 bit, pagine da 48 KB, e la seguente tabella delle pagine: {0x1, 0x7, 0x0, 0xF, 0xD, 0x8, 0xE, 0xC, 0x6, 0x2, 0xB, 0x3, 0x5, 0x9, 0x4, 0xA}.
A quali indirizzi fisici corrispondono i seguenti indirizzi logici: 0xFF5AB, 0xC1A01, 0xDB328?

- 0xFF5A3, 0xC1A07, 0xDB326
- 0xAA5AB, 0x57A01, 0x93328
- 0xAF5AB, 0x51A01, 0x9B328
- 0xAA8AB, 0x57B01, 0x93F28
- La tabella delle pagine non è valida
- Nessuna delle precedenti

Domanda 5
Si consideri un sistema di memoria virtuale con uno spazio di indirizzi a 24 bit, pagine da 4 KB, e la seguente tabella delle pagine: {0x123, 0xA5F, 0xBE2, 0xF02, 0xA13, 0x79F, 0xA0C, 0xB5B, 0xC23, 0x4AB, 0xCD1, 0xAF3, 0x04C, 0x6BA, 0xDAC, 0x992}.
A quali indirizzi fisici corrispondono i seguenti indirizzi logici: 0xFFF0A1, 0x12A3F2, 0x002C00?

- 0x9920A1, 0x1233F2, 0xBE2C00
- 0x9920A1, 0xDAC3F2, 0xBE2C00
- 0xFFF4BA, 0x12AF02, 0x002BE2
- 0x9FF0A1, 0xA2A3F2, 0x102C00
- La tabella delle pagine non è valida
- Nessuna delle precedenti

Domanda 6
Si consideri un semplice allocatore di memoria malloc/free dove, potendo scegliere, la malloc riusa il blocco libero con l'indirizzo più basso. Assumere per semplicità che non vi sia alcuna header e che i blocchi allocati vengano arrotondati a una dimensione multiplo di 4 byte. Inoltre, l'allocatore non divide blocchi liberi in più blocchi di dimensione inferiore, né fonde eventuali blocchi liberi adiacenti in un blocco di dimensioni superiore. Qual è il contenuto dell'heap alla fine della seguente sequenza di operazioni (dove X denota 4 byte allocati e O denota 4 byte liberi)?

```
p1=malloc(7)
p2=malloc(3)
p3=malloc(18)
p4=malloc(1)
free(p2)
free(p1)
p5=malloc(2)
p6=malloc(6)
free(p4)
free(p3)
p7=malloc(3)
```

- XX | X | OOOOO | X
- XX | O | XXXXX | O | XX
- XX | X | OOOOO | X | OO
- XX | O | OOOOO | X | XX
- XX | X | XXXXX | O | XX
- Nessuna delle precedenti

Domanda 7

Di quanto bisognerebbe velocizzare una funzione f che consuma il 40% del tempo di esecuzione di un programma P in modo da ottenere uno speedup complessivo per P pari a 1.5x?

- 0.5x
- 1.5x
- 3x
- 6x
- È impossibile
- Nessuna delle precedenti

Domanda 8

Quale delle seguenti affermazioni è **vera**?

- I segnali possono essere inviati esclusivamente dal kernel, ma possono essere catturati da programmi utente
- L'interrupt vector è una tabella che contiene i puntatori ai gestori dei segnali
- Tutti i segnali possono essere catturati e gestiti
- Il comando kill può essere usato solo per inviare il segnale SIGKILL
- Nessuna delle precedenti