Ogni risposta corretta viene contata 2 punti, ogni risposta sbagliata viene contata -1 punti, ogni risposta non data viene contata 0 punti

ESERCIZIO 1a
Quale delle seguenti affermazioni è vera?

- Linux è multiutente, perché definisce più utenti, di cui però uno solo può essere loggato su una data macchina
- **Linux è multiutente, perché permette a più utenti contemporaneamente di essere loggati sulla stessa macchina**
- Linux è multiprocesso, perché permette a più utenti contemporaneamente di essere loggati sulla stessa macchina
- Linux è multiprocesso, perché può essere usato su una macchina con più processori

ESERCIZIO 1b
Quale delle seguenti affermazioni è vera?

- È possibile stampare esclusivamente le ultime N righe di un file di testo con il comando `cat -N -6`
- **Nessuna delle altre opzioni è vera**
- Il comando `tail -c /var/log/maillog` stampa in maniera continua le ultime righe del file `/var/log/maillog`
- Il comando `cut` consente di tagliare un file in gruppi di righe contigue in base ad un carattere di spaziatura

ESERCIZIO 1c
Quale delle seguenti affermazioni è vera?

- `cat -N -6` filename stampa le ultime 6 righe del file filename
- `tail -c` n filename stampa le ultime n righe del file filename
- `cut` consente di tagliare un file in gruppi di righe contigue in base a un carattere di spaziatura `/var/log/maillog`
- **Nessuna delle altre risposte è vera**

ESERCIZIO 1d
Quale delle seguenti affermazioni è vera?

- Nel file system di Linux, ogni nodo interno è un file o una directory, mentre le foglie sono sempre directory
- Il file system di Linux è una foresta, ovvero un albero con più radici (una per ogni dispositivo fisico di memoria di massa collegato alla macchina, per esempio hard disk, chiave USB, CD, DVD, ...)
- Il file system di Linux è gerarchico, perchè per accedere ad un file occorre effettuare una chiamata di sistema al kernel
- **Nel file system di Linux esistono anche directory (denominate virtuali) che non si trovano su nessun dispositivo fisico di memoria di massa**

ESERCIZIO 1e
Quale delle seguenti affermazioni è vera?

- Linus Torvalds ha scritto il primo kernel di Linux all'inizio degli anni '80
- **Richard Stallman ha descritto per primo la licenza GPL**
- Linus Torvalds ha riscritto i pacchetti di Unix, creando i pacchietti GNU
- Tutte le opzioni sono false

ESERCIZIO 1f
Quale delle seguenti affermazioni è vera?

- Ogni risorsa di un sistema Unix, ad eccezione dei processi e periferiche hardware, è rappresentato da un file
- Ogni risorsa di un sistema Unix, ad eccezione delle connessioni di rete (socket), è rappresentato da un file
- **Ogni risorsa di un sistema Unix, ad eccezione dei processi, è rappresentato da un file**
- In un sistema Unix tutte le risorse sono rappresentati da un file

ESERCIZIO 1g
Quale delle seguenti affermazioni è vera?

- Non è possibile definire una basic REGEX che abbia un match con una stringa formata da 3 o più occorrenze del carattere `*`
- Tutte le risposte sono vere
- **Il metacarattere `^` consente di dichiarare un match ad inizio stringa solo se compare come primo carattere della REGEX**
- `@` è un metacarattere delle REGEX *estese*

ESERCIZIO 2
Quale delle seguenti affermazioni è falsa?

- La concatenzione di due o più regex è una regex
- . è un metacarattere che ha un match con una occorrenza di qualsiasi carattere
- **Il carattere \ ha un match con la regex '\'**
- I metacaratteri `[]` consentono di definire il match con un range di caratteri

ESERCIZIO 3
Quale delle seguenti affermazioni ~~sul controllo di accesso~~ è vera?

- Nessuna delle altre opzioni è vera
- Ad ogni filesystem corrisponde un disco fisico o parte di esso (partizione)
- È possibile montare un filesystem solo se è dichiarato nel file `/etc/fstab`
- È possibile montare un filesystem solo se è dichiarato nel file `/etc/mtab`

ESERCIZIO 4a
Quale delle seguenti affermazioni sui processi Linux è vera?

- Ogni processo può conoscere il proprio PID, ma non quello del processo che l'ha creato
- Un processo diventa zombie se termina prima di almeno uno dei processi che abbia eventualmente creato
- **Con l'eccezione del primo processo, tutti i processi sono creati con una `fork` effettuata da un altro processo in esecuzione**
- Normalmente, il processo figlio, una volta terminata la sua computazione, attende, con una chiamata alla *syscall* `wait`, che il padre termini e gli restituisca il suo exit status

ESERCIZIO 4b
Quale delle seguenti affermazioni sui processi Linux è vera?

- Per ogni terminale aperto, ci può essere al massimo un job in background
- Per lanciare un processo in modo tale che non scriva su stdout, lasciando cosiÌ€ modo di scrivere altri comandi sulla bash, è sufficiente lanciarlo in background
- Se si vuole dare input da stdin senza redirezioni ad un processo, è necessario lanciarlo in foreground
- Nessuna delle altre opzioni è vera

ESERCIZIO 4c
Quale delle seguenti affermazioni sui processi Linux è vera?

- Nessuna delle altre opzioni è vera
- Il text segment contiene le istruzioni da eseguire, e non può essere condiviso con altri processi
- Lo stack contiene i dati statici inizializzati ed alcune costanti d'ambiente
- Il process control block (PCB) mantiene le informazioni essenziali di ogni processo, e uno stesso PCB può essere condiviso tra processi diversi

ESERCIZIO 4d
Quale delle seguenti affermazioni sui processi Linux è vera?

- Il process control block mantiene le informazioni essenziali di ogni processo, è mantenuto su disco e viene swappato in RAM quando il processo va in esecuzione
- Lo heap contiene i dati statici inizializzati ed alcune costanti d'ambiente
- Il text segment contiene le istruzioni da eseguire, e viene sempre mantenuto interamente in RAM
- Nessuna delle altre opzioni è vera

ESERCIZIO 4e
Quale delle seguenti affermazioni sui processi Linux è vera?

- I comandi builtin della bash generano sempre nuovi processi
- Per capire se un comando della bash è o no builtin, è sufficiente usare il comando type
- Il comando cd è builtin della bash
- Un comando builtin della bash non corrisponde ad alcun file eseguibile dedicato

ESERCIZIO 4f
Quale delle seguenti affermazioni sui processi Linux è vera?

- Un processo è sempre un'istanza di uno script bash
- Per poter lanciare un file eseguibile, è prima necessario aspettare che il comando precedente sia terminato
- Eseguendo (con successo) *k* volte un file eseguibile, si generano *k* diversi processi
- Tutti i processi sono sempre in stato di `RUNNING`

ESERCIZIO 4g
Quale delle seguenti affermazioni sui processi Linux è vera?

- Per visualizzare i processi attualmente in esecuzione su una determinata bash, è sufficiente usare il comando `jobs`
- Tutti i processi in background sono in stato Stopped
- Per riportare in foreground un determinato job in background, è sufficiente dare il comando fg sulla bash dove il job è in background, ma solo se tale job è l'attuale "current job" in background
- Per stoppare un processo in foreground, su può sia mandare un segnale SIGTSTP che premere CTRL+Z in una qualsiasi shell

ESERCIZIO 4h
Quale delle seguenti affermazioni sui processi Linux è vera?

- Il text segment contiene le istruzioni da eseguire, e viene sempre mantenuto interamente in RAM
- Il process control block mantiene le informazioni essenziali di ogni processo, è mantenuto su disco e viene swappato in RAM quando il processo va in esecuzione
- Lo heap contiene i dati statici inizializzati ed alcune costanti d'ambiente
- Nessuna delle altre opzioni è vera

ESERCIZIO 4i
Quale delle seguenti affermazioni sui processi Linux è vera?

- Per vedere i jobs in foreground, è sufficiente usare il comando `jobs`
- Nessuna delle altre opzioni è vera
- Ciascun job è composto al massimo da un processo
- Quando un processo in foreground termina, la bash stampa il job id del processo e la ragione della terminazione

ESERCIZIO 4j
Quale delle seguenti affermazioni sui processi Linux è vera?

- Ogni processo è identificato da un PID e da un job id, che devono coincidere
- Nessuna delle altre opzioni è vera
- Un job è un comando della bash che prende sempre il controllo dello stdin
- Fissato un istante nell'esecuzione del sistema operativo, ci sarà sempre al massimo un job in foreground, mentre quelli in background possono essere più d'uno

se il contesto e' un unica shell, senno l'altro

ESERCIZIO 5a
Quale delle seguenti affermazioni sui processi Linux è falsa?

- Qualsiasi computazione eseguita dal sistema operativo è contenuta dentro un qualche processo
- Digitare un comando sulla shell genera sempre un nuovo processo
- Affinché un file possa diventare un processo è necessario che abbia i permessi di esecuzione
- Esistono file che non possono essere eseguiti per diventare processi

ESERCIZIO 5b
Quale delle seguenti affermazioni sui processi Linux è falsa?

Il comando cd è built-in della bash
Un comando built-in della bash non corrisponde ad alcun file eseguibile dedicato
● I comandi built-in della bash generano sempre nuovi processi
Per capire se un comando della bash è o no built-in, è sufficiente usare il comando `type`

ESERCIZIO 5c
Quale delle seguenti affermazioni sui processi Linux è falsa?

In istanti diversi, possono esserci 2 processi distinti con lo stesso PID
● Per creare i PID dei processi si usano dei numeri interi che crescono sempre
Ogni processo può conoscere il suo PID
In un determinato istante, non possono esserci 2 processi distinti con lo stesso PID

ESERCIZIO 6
Quale delle seguenti affermazioni sulle *syscall* dei processi in Linux è vera?

Se un processo viene lanciato nel seguente modo: `var=valore ./a.out`, allora esso può ottenere `var` con la chiamata a `getenv("valore");`
Se un processo viene lanciato nel seguente modo: `var=valore ./a.out`, allora esso può ottenere `valore` con la chiamata a `setenv("var", "valore", 1);`
● Per qualsiasi processo è possibile conoscere il suo ambiente di esecuzione senza effettuare alcuna *syscall*
Se un processo viene lanciato nel seguente modo: `var=valore ./a.out`, allora esso può ottenere valore con la chiamata a `putenv("var=valore");`

ESERCIZIO 7
Quale delle seguenti affermazioni sulle *syscall* dei processi in Linux è falsa?

● La *syscall* `setuid()` permette a qualsiasi processo di cambiare il suo real user ID
La *syscall* `getppid()` ritorna il PID del processo che ha generato quello chiamante (o che lo ha adottato)
La *syscall* `getpid()` ritorna il PID del processo chiamante
La *syscall* `getuid()` permette a qualsiasi processo di conoscere il suo real user ID

ESERCIZIO 8a
Quale delle seguenti affermazioni sulle *syscall* di Linux che riguardano i files è falsa?

La *syscall* `chdir(path)` ha lo stesso effetto del comando bash `cd path` lanciato in una sottoshell
● La *syscall* `rename(oldpath, newpath)` ha lo stesso effetto del comando `bash cp oldpath newpath`
La *syscall* `chroot` ha l'effetto di cambiare l'esecuzione delle sole chiamate ad `open` che usano path *assoluti* come primo argomento
La *syscall* `chdir` ha l'effetto di cambiare l'esecuzione delle sole chiamate ad `open` che usano path *relativi* come primo argomento

ESERCIZIO 8b
Quale delle seguenti affermazioni sulle *syscall* di Linux che riguardano i files è falsa?

● La *syscall* `stat(nomefile, buf)` ha lo stesso effetto del comando bash `stat nomefile`
La *syscall* `dup2(2, 1)` ha l'effetto di ridirigere lo stdout nello stderr
La *syscall* `chown(nomefile, -1, gid)` ha lo stesso effetto del comando bash `chgrp gid nomefile`
La *syscall* `mkdir(nomedir, mode)` ha lo stesso effetto del comando bash `mkdir -m mode nomedir`

ESERCIZIO 8c
Quale delle seguenti affermazioni sulle *syscall* di Linux che riguardano i files è falsa?

Chiamando la *syscall* `select` , è possibile monitorare un insieme di file descriptor, ed essere notificati non appena ce n'è uno che è diventato disponibile per un'operazione di lettura o scrittura
● Per richiedere un lock su un file (o su una porzione di esso), occorre chiamare la *syscall* `ioctl`
È possibile usare la *syscall* `select` sia in modo bloccante che in modo non bloccante
Le *syscall* `ioctl` e `fcntl` ammettono 2 o 3 argomenti, a seconda dell'operazione richiesta

ESERCIZIO 8d
Quale delle seguenti affermazioni sulle *syscall* di Linux che riguardano i files è falsa?

● La *syscall* `unlink(nomefile)` rimuove sempre il contenuto di `nomefile` dal disco, se `nomefile` è un file regolare
La *syscall* `symlink(oldpath, newpath)` ha lo stesso effetto del comando bash `ln -s oldpath newpath`
La *syscall* `link(oldpath, newpath)` ha lo stesso effetto del comando bash `ln oldpath newpath`
La *syscall* `unlink(nomefile)` ha lo stesso effetto del comando bash `rm nomefile`

ESERCIZIO 8e
Quale delle seguenti affermazioni sulle *syscall* di Linux che riguardano i files e che si trovano nella sezione 2 del manuale è falsa?

Mentre le funzioni della libreria standard possono solo agire solo su file regolari, le syscall di Linux possono agire su tutti i tipi di file (regolari, directory, pipe, ...)
Le funzioni della libreria standard agiscono su una struttura di tipo FILE * , mentre le syscall agiscono su un file descriptor intero
Nessuna delle syscall di Linux accetta come argomento input o output formattato stile printf
● Le syscall Linux permettono solamente le seguenti operazioni: apertura, chiusura, scrittura, lettura, posizionamento

ESERCIZIO 9
Quale delle seguenti affermazioni sulle *syscall* di Linux che riguardano le directory è falsa?

Chiamare la *syscall* `open` su una directory può avere successo
Per poter leggere il contenuto di una directory è possibile aprirla con la *syscall* `opendir`
● Per poter cambiare il contenuto di una directory occorre aprirla con la *syscall* `opendir`
La *syscall* `readdir`, con argomento uguale a quanto ritornato da una precedente `opendir` avvenuta con successo, ritorna un puntatore ad una struttura `struct dirent`, che contiene il nome di un file o di una directory contenuta nella directory passata a `opendir`

ESERCIZIO 10
Quale delle seguenti affermazioni sulle directory di un filesystem è vera?

- Non può mai contenere degli hard link
- Può contenere solo file regolari e altre directory
- Nessuna delle altre opzioni è vera
- **Ha sempre una directory padre, eventualmente corrispondente a se stessa**

ESERCIZIO 11
Quale delle seguenti affermazioni sul filesystem Linux è vera?

- Tutte le opzioni sono false
- **È logicamente unico ed ha come punto di inizio la root (radice) rappresentata dal simbolo `/`**
- Tutte le opzioni sono vere
- È una struttura ad albero in cui le directory sono nodi intermedi e non possono mai essere nodi terminali

ESERCIZIO 12
Quale delle seguenti affermazioni sulle *syscall* `wait` e `waitpid` è falsa?

- **Se una chiamata `wait(&status)`; ha successo, il valore di `status` coincide con l'exit status del processo figlio appena terminato**
- Ogni chiamata `wait(&status)`; è equivalente alla chiamata `waitpid(-1, &status, 0)`
- Le chiamate alla `wait` sono sempre bloccanti
- Le chiamate alla `waitpid` possono non essere bloccanti

ESERCIZIO 13
Quale delle seguenti affermazioni sulla *syscall* `fork` è falsa?

- Genera una copia esatta del processo chiamante, con alcune eccezioni; tra queste ultime vi è il PID
- **Genera una copia esatta del processo chiamante, con alcune eccezioni; tra queste ultime vi è lo stack delle chiamate**
- Genera una copia esatta del processo chiamante, con alcune eccezioni; tra queste ultime vi è il PPID
- Ritorna 2 valori diversi a seconda che si tratti del processo padre o del processo figlio

ESERCIZIO 14
Quale delle seguenti affermazioni sulla *syscall* `sigaction` è vera?

- **Nessuna delle altre opzioni è vera**
- Permette, tramite il campo `sa_mask` della struttura `struct sigaction`, di definire quali segnali vanno ignorati finché l'handler del segnale è in esecuzione
- Permette di definire una funzione che viene eseguita qualsiasi segnale arrivi al processo
- Chiamandola ripetutamente, e passandole al primo argomento via via tutti i segnali disponibili, è possibile definire una funzione che viene eseguita qualsiasi segnale arrivi al processo
- Permette di definire quali segnali vanno ignorati finché l'handler del segnale è in esecuzione

ESERCIZIO 15
Quale delle seguenti affermazioni sugli errori delle *syscall* di Linux è vera?

- Per stampare su stdout la spiegazione di un errore verificatosi in una *syscall* è sufficiente chiamare `perror`
- Per stampare su stdout la spiegazione di un errore verificatosi in una *syscall* è necessario scrivere uno `switch` sulla variabile globale `errno`
- **Per stampare su stdout la spiegazione di un errore verificatosi in una *syscall* si può effettuare la seguente chiamata: `printf("%s\n", strerror(errno));`**
- Per stampare su stderr la spiegazione di un errore verificatosi in una *syscall*, il cui nome sia contenuto nella variabile `syscall_name` (di tipo `char *`), si può effettuare la seguente chiamata: `perror("Si e' verificato il seguente errore nella chiamata a %s", syscall_name);`

ESERCIZIO 16
Quale delle seguenti affermazioni sulla comunicazione tra processi in Linux è vera?

- Usando le *named pipes*, è possibile far comunicare solo processi parenti (ad es., padre con figlio)
- Per far comunicare qualunque coppia di processi è necessario metterli in pipeling da shell
- **Usando le *socket*, è possibile far comunicare qualsiasi coppia di processi**
- Nessuna delle altre opzioni è vera
- Usando la *syscall* `pipe`, è possibile far comunicare qualunque coppia di processi

ESERCIZIO 17
Quale delle seguenti affermazioni sui segnali Linux è vera?

- Tutti i segnali, se non opportunamente catturati, provocano la terminazione del processo, con l'eccezione del segnale STOP
- Per un processo è sempre possibile ridefinire il comportamento di un qualsiasi segnale
- **È possibile per un qualunque processo inviare un segnale ad un qualsiasi altro processo dello stesso utente**
- Nessuna delle altre opzioni è vera

ESERCIZIO 18
Quale delle seguenti affermazioni sulle *pipe* di Linux è vera?

- Usando la *syscall* `mkfifo`, viene aperto un solo file descriptor
- Per aprire una *named pipe*, è sempre necessario crearla nello stesso programma usando la *syscall* `mkfifo`
- **Usando la *syscall* `pipe`, vengono automaticamente aperti 2 file descriptor**
- Nessuna delle altre opzioni è vera

ESERCIZIO 19
Quale delle seguenti affermazioni sui comandi della `bash` è vera?

- **Eseguendo il comando `echo `date`` viene stampata la data e l'ora corrente (secondo l'orologio di sistema)**
- Il comando `man cmd` restituisce in sequenza tutte le pagine di manuale per il comando `cmd` contenute nelle varie sezioni del manuale
- Il comando `clear` può essere usato per pulire completamente lo schermo: dopo l'esecuzione, il terminale non conterrà alcuna scritta
- Il comando `cat stringa` può essere usato per scrivere `stringa` su stdout

ESERCIZIO 20
Quale delle seguenti affermazioni sui comandi della `bash` è falsa?

- Il comando `type file` mostra il tipo del file file (regolare, directory, etc)
- Il comando `whoami` mostra lo username dell'utente attualmente loggato nel terminale in cui viene digitato il comando (potrebbe non coincidere con l'utente che ha effettuato il login grafico)
- Il comando `id` può essere usato per visualizzare i gruppi cui un utente appartiene
- Il comando `which cmd` mostra qual è il file eseguibile che viene eseguito quando si lancia il comando cmd, ma solo per i comandi che non sono builtin

ESERCIZIO 21
Quale delle seguenti affermazioni sui comandi `cat` e `od` è falsa?

- Il comando `cat` interpreta ogni sequenza di byte letta come un carattere (tipicamente UTF-8), e lo stampa
- Il comando `od` scrive ogni singolo byte letto usando il suo valore numerico
- I comandi `od file` e `cat file` non possono mai dare lo stesso risultato
- I comandi `od file` e `cat file` danno lo stesso risultato se `file` è un file di testo ASCII

ESERCIZIO 22
Quale delle seguenti affermazioni sui comandi `cat` e `od` è falsa?

- L'opzione `-A` di `od` permette di scegliere la base (decimale, ottale od esadecimale) di tutti i bytes da stampare
- L'opzione `-n` di `cat` fa precedere ogni riga stampata con il numero della riga stessa (a partire da 1)
- L'opzione `-E` di `cat` stampa anche il carattere `$` alla fine di ogni riga
- L'opzione `-j B` di `od` permette di cominciare la visualizzazione a partire dal (`B` + 1)-esimo byte

ESERCIZIO 23
Quale delle seguenti affermazioni sui comandi `less` e `more` è falsa?

- Sono specialmente utili quando si vuole visualizzare un output molto lungo (che non è possibile visualizzare in un'intera schermata di terminale)
- Sono entrambi interattivi
- Per terminarli occorre premere CTRL+C
- Entrambi permettono di ricercare espressioni regolari

ESERCIZIO 24
Quale delle seguenti affermazioni sul comando `kill` è falsa?

- Per mandare il segnale SIGTERM al processo con PID 19330, è sufficiente scrivere il comando `kill -SIGTERM 19330`
- Per mandare il segnale 9 al processo con PID 10, è sufficiente scrivere il comando `kill -KILL %10`
- Per mandare il segnale SIGINT al processo in background con job id 3, è sufficiente scrivere il comando `kill -` `` `kill -l SIGINT` `` `%3`
- Per mandare il segnale 9 al processo in background con job id 3, è sufficiente scrivere il comando `kill -KILL %3`

ESERCIZIO 25
Quale delle seguenti affermazioni sul comando `kill` è vera?

- È obbligatorio specificare il segnale da inviare, come numero intero
- Lanciato senza nessun argomento, manda SIGKILL all'ultimo processo lanciato
- Può essere usato per ottenere lo stesso risultato tanto del CTRL+C quanto del CTRL+Z
- Nessuna delle altre affermazioni è vera

ESERCIZIO 26
Quale delle seguenti affermazioni sul comando `time` è falsa?

- Il comando `/usr/bin/time cmd` può solo mostrare il tempo (di CPU, di sistema, e reale)
- Esistono 2 comandi `time`: uno è una keyword della bash e l'altro corrisponde ad un file eseguibile (solitamente `/usr/bin/time`)
- Il comando `/usr/bin/time cmd` ha anche l'effetto di eseguire il comando `cmd`
- Il comando `time cmd`, eseguito dalla bash, può solo mostrare il tempo (di CPU, di sistema, e reale)

ESERCIZIO 27
Quale delle seguenti affermazioni sul comando `find` è falsa?

- È possibile cercare anche directory, e non solo file
- È obbligatorio che gli starting point siano delle directory
- È possibile restringere la ricerca ai soli file che sono link simbolici
- È possibile cercare nomi di file che rispettino un dato pattern o una data regular expression

ESERCIZIO 28
Quale delle seguenti affermazioni sul comando `top` è vera?

- Se lanciato con il comando `top -b`, per terminarlo è sufficiente premere il tasto `Q`
- Se lanciato con il comando `top`, per terminarlo è necessario premere `CTRL+C`
- Nessuna delle altre affermazioni è vera
- Il suo output è uguale a quello di `ps`, ma le opzioni sono diverse

ESERCIZIO 29
Quale delle seguenti affermazioni sui comandi `cmp`, `diff` e `patch` è vera?

- È possibile usare il comando `patch` solo se si ha l'output del comando `diff`
- L'opzione `-b` ha lo stesso significato sia per `diff` che per `cmp`
- È possibile usare il comando `patch` solo se si ha, indifferentemente, l'output del comando `diff` o del comando `cmp`
- L'opzione `-i` di `cmp` permette di considerare come uguali le differenze sul solo minuscolo/maiuscolo

ESERCIZIO 30
Quale delle seguenti affermazioni sul comando `ps` è vera?

- [x] Senza nessun argomento, mostra tutti i processi lanciati dall'utente attuale nel terminale attuale
- [ ] Per ogni processo, mostra sempre il suo PID, indipendentemente dagli argomenti con cui viene lanciato
- [ ] Non è possibile usarlo per vedere i processi lanciati dall'utente root
- [ ] È possibile usarlo per vedere solo i processi che superano un certo uso della RAM

ESERCIZIO 31
Quale delle seguenti affermazioni sulle funzioni `malloc`, `calloc`, `realloc` e `free` è falsa?

- [x] Le due chiamate `calloc(N, sizeof(int))` e `realloc(NULL, N*sizeof(int))` hanno sempre lo stesso effetto
- [ ] Le due chiamate `malloc(N*sizeof(int))` e `realloc(NULL, N*sizeof(int))` hanno sempre lo stesso effetto
- [ ] I risultati di `malloc, calloc` e `realloc` possono essere passati alla funzione `free` per poter essere riallocati da future `malloc, calloc` e/o `realloc`
- [ ] Il primo argomento di `realloc`, quando non `NULL`, deve contenere il risultato di una precedente chiamata a `malloc, calloc` o `realloc`

ESERCIZIO 32
Quale delle seguenti affermazioni sugli script `sed` è vera?

- [ ] Sono file di testo composti da sequenze di linee del tipo *condizione azione*, dove la *condizione* può essere, ad esempio, un numero di linea o una espressione regolare
- [ ] Sono file di testo composti da sequenze di linee che possono essere del tipo *N azione*, dove *N* è il numero di linea del file in ingresso che `sed` sta processando ed *azione* è un comando Unix da eseguire sulla linea *N*
- [ ] Sono script (es: bash) al cui interno viene invocato il comando `sed`
- [x] Nessuna delle altre opzioni è vera

ESERCIZIO 33
Quale delle seguenti affermazioni sulla variabile `IFS` è vera?

- [ ] Deve contenere, come valore, un solo carattere, da usare per la separazione in parole nella `bash` (word splitting)
- [ ] Nessuna delle altre opzioni è vera
- [x] Può essere usato per cambiare l'esecuzione di un ciclo `for` della `bash`
- [ ] Contiene il carattere utilizzato per la separazione in token da `awk`

ESERCIZIO 34
Relativamente alla programmazione bash, quali delle seguenti affermazioni è vera?

- [ ] Nessuna delle opzioni è vera
- [ ] il comando `$count[3]` stampa l'elemento con indice 3 dell'array `count`. Gli array in bash sono sparsi quindi non c'è garanzia che si tratti del terzo o quarto elemento dell'array
- [ ] il comando `$count[3]` stampa il terzo elemento dell'array `count`
- [ ] il comando `$count[3]` stampa il quarto elemento dell'array `count`

ESERCIZIO 35
Relativamente alla programmazione bash, quale delle seguenti affermazioni è vera?

- [x] Nessuna delle altre opzioni è vera
- [x] Una volta dichiarato il tipo di una variabile, lo si può cambiare solo dopo aver invocato il comando unset
- [ ] Si può dichiarare esplicitamente il tipo di una variabile, anteponendolo al nome della variabile stessa (es: `int count`)
- [ ] Ad una variabile di tipo intero non è possibile assegnare un valore di tipo diverso, altrimenti il programma termina con un errore
- [ ] Non è possibile dichiarare una variabile in sola lettura: una variabile definita in uno script bash è sempre modificabile

ESERCIZIO 36
Relativamente alla programmazione bash, quale delle seguenti affermazioni è vera?

- [ ] È possibile inizializzare/definire array associativi con la seguente sintassi: `declare -A myArray= {'key1'='value1' ... 'keyn'='valuen'}`
- [ ] Non è possibile definire array associativi
- [x] Nessuna delle altre opzioni è vera
- [ ] L'acronimo BASH sta per *Bourne Advanced Shell*

ESERCIZIO 37
Relativamente alla programmazione bash, quale delle seguenti affermazioni sul carattere `#` è vera?

- [ ] Quando è seguito dal carattere `!`, non rappresenta *mai* un commento
- [x] Nessuna delle altre opzioni è vera
- [ ] Se presente in uno script, tutto quello che lo segue è sempre considerato commento
- [ ] Rappresenta sempre l'inizio di un commento, con un'unica eccezione: quando è preceduto dal carattere `$`

ESERCIZIO 38
Relativamente alla programmazione bash, la variabile `IFS` :

- [ ] Nessuna delle opzioni è vera
- [ ] È l'acronimo per Internal Field Splitting
- [x] Rappresenta una variabile contenente la sequenza di tutti i caratteri utilizzati per la separazione in parole (word splitting)
- [ ] Rappresenta il carattere utilizzato per la separazione in parole (wordsplitting)

ESERCIZIO 39
Quale delle seguenti affermazioni sulle applicazioni client-server realizzate tramite *socket* è vera?

- [ ] Il client deve sempre chiamare la *syscall* `bind`
- [ ] Il client deve sempre chiamare la *syscall* `listen`
- [ ] Il server deve chiamare la *syscall* `connect`
- [x] Sia il server che il client devono sempre chiamare la *syscall* `socket`

ESERCIZIO 40
Si supponga di voler vedere, per tutti i processi dell'utente utente, il suo PID, il suo PPID, il comando usato per lanciare il processo (con tutti gli argomenti), la usa occupazione totale di memoria in kB e la sua attuale occupazione di memoria in RAM (senza considerare quindi la parte eventualmente swappata su disco), sempre in kB. Quale dei seguenti comandi è quello corretto?

- `ps -uutente -o pid,ppid,cmd,rss,sz`
- `ps -e -o pid,ppid,cmd,vsz,rss`
- `ps -uutente -o pid,ppid,cmd,rss,vsz`
- ● `ps -uutente -o pid,ppid,cmd,vsz,rss`

ESERCIZIO 41
Per eliminare tutte le linee duplicate in un file di testo (senza preoccuparsi dell'ordinamento delle righe) occorre:
~~Quale delle seguenti affermazioni è vera?~~

- ● utilizzare congiuntamente i comandi `sort` e ~~`cat`~~ ***uniq***
- utilizzare congiuntamente i comandi `cat` e `grep`
- utilizzare il comando `uniq` con opzione `-u`
- utilizzareilcomando `uniq`

ESERCIZIO 42
Quale delle seguenti affermazioni sulle espressioni regolari è vera?

- Il metacarattere + consente di concatenare due regex
- Non è mai possibile definire un match con il carattere
- ● Un carattere *literal* ha un match con se stesso
- Nessuna delle altre opzioni è vera

ESERCIZIO 43
Una espressione regolare:

- È composta da caratteri literal e caratteri di punteggiatura
- Non può essere formata da soli caratteri non literal
- ● Descrive implicitamente un insieme di stringhe che hanno almeno un match con se stessa
- Non può essere formata da soli caratteri literal

ESERCIZIO 44
Esiste in unix un comando che consente di stampare il numero di occorrenze di una riga in un file?

- Nessuna delle altre opzioni è vera
- No, occorre utilizzare congiuntamente i comandi `uniq` e `wc`
- No, occorre utilizzare congiuntamente i comandi `sort` e `wc`
- ● Sì€

ESERCIZIO 45
Quale delle seguenti affermazioni sul Linguaggio C è vera?

- Richiede che ogni modulo sia scritto in un file separato, come nel Java
- ● È stato definito presso i laboratori di ricerca di una compagnia telefonica americana
- Nasce per risolvere le ambiguità e i problemi di portabilità su architetture diverse di cui soffrono gli altri linguaggi di programmazione finora noti
- È stato definito come linguaggio Open Source da Dennis Ritchie
- ● Richiede che i programmi siano sempre scritti in file con estensione .c
- È incompatibile con i Sistemi Operativi della famiglia Windows
- Nessuna delle altre opzioni è vera
- ● È un linguaggio strutturato e compilato
- È un linguaggio non strutturato e compilato
- Viene definito per la creazione del primo Sistema Operativo Unix
- Nasce negli anni 70 per sviluppare programmi portabili su diverse architetture hardware
- Nasce come linguaggio di programmazione proprietario per l'implementazione dei programmi sui sistemi DEC PDP-11
- ● Nasce negli anni 70 come evoluzione del linguaggio B

ESERCIZIO 46
Quale delle seguenti affermazioni sui programmi scritti in Linguaggio C è vera?

- Rappresenta le stringhe ESCLUSIVAMENTE come array di caratteri terminate dal carattere '\n'
- ● Rappresenta tipicamente le stringhe come array di caratteri terminate dal carattere '\0'
- Rappresenta le stringhe ESCLUSIVAMENTE come array di caratteri terminate dal carattere '0'
- Rappresenta le stringhe ESCLUSIVAMENTE come array di caratteri terminate dal carattere '^M'

ESERCIZIO 47
Quale dei seguenti sistemi operativi ***non*** è un antenato di Linux?

- MULTICS
- ● MacOSX
- Unix
- Le altre risposte contengono tutte degli antenati di Linux

ESERCIZIO 48
Quale dei seguenti linguaggi ***non*** è mai stato usato per implementare Unix?

- ● Le altre risposte contengono tutte dei linguaggi usati per implementare Unix
- Il B
- L'assembler del PDP-7
- Il C

ESERCIZIO 49
Quale dei seguenti *non* è un possibile stato di un processo Linux?

- Sleeping
- Orphaned
- Zombie
- Ready o Runnable
- Uninterruptible sleep
- Stopped
- Running
- Continued

ESERCIZIO 50
Quale dei seguenti campi *non* è presente nel process control block?

- Current working directory
- Change time
- GID reale ed effettivo
- Nice

ESERCIZIO 51
Per modificare tutte le occorrenze della lettera o ed i rispettivamente in O ed I di un file di testo, quale comando è più appropriato utilizzare?

- awk
- sed
- grep
- tr

ESERCIZIO 52
La stringa `Informatica9000` ha un match con la seguente REGEX:

- `Informatica9000$`
- `^Informatica900$`
- `Informatica9[0^3]`
- `[^IKU]nformatica`

ESERCIZIO 53
Si supponga che sia appena stata eseguita la seguente riga di codice di un processo: `int pid = fork();`. Quale delle seguenti affermazioni è vera?

- Nel processo padre, la variabile pid assume 2 diversi valori
- Nel processo figlio, la variabile `pid` vale assume 1 solo valore, corrispondente al PID del padre
- C'è un nuovo processo pronto per andare in esecuzione, a meno che la variabile `pid` non valga -1
- Nel processo padre, la variabile `pid` vale assume 1 solo valore, corrispondente al suo stesso PID

ESERCIZIO 54
Si supponga di voler scrivere un programma immune al CTRL+C. Quale dei seguenti frammenti di codice realizza quanto detto sopra?

- `signal(SIGINT, SIG_IGN);`
- `signal(SIGINT, SIG_DFL);`
- Non è possibile essere immuni al CTRL+C
- `signal(SIGTERM, SIG_DFL);`

ESERCIZIO 55a
Si consideri il comando:

```
find / \( -name 'Doc*' -a -type d \) -o -newer Documenti -exec touch '{}' \;
```

Quale delle seguenti affermazioni è vera?

- Il comando modifica tutti i tempi (atime, mtime e ctime) delle directory il cui nome comincia con `Doc`, e fa lo stesso anche con tutti i file e/o directory che siano stati modificati più recentemente della directory Documenti
- Il comando modifica tutti i tempi (atime, mtime e ctime) di tutte le directory il cui nome comincia con `Doc` e che siano state modificate più recentemente della directory Documenti
- Il comando modifica tutti i tempi (atime, mtime e ctime) delle directory il cui nome comincia con `Doc`, e fa lo stesso anche con tutti i file regolari che siano stati modificati più recentemente della directory Documenti
- Nessuna delle altre opzioni è vera

ESERCIZIO 55b
Si consideri il comando:

```
find Doc* \( -name 'Doc*' -a -type d \) -o -newer Documenti -exec touch '{}' \;
```

Quale delle seguenti affermazioni è vera?

- L'azione non è specificata correttamente, quindi la bash restituirà un messaggio d'errore
- Il comando stampa su schermo tutte le directory il cui nome comincia con Doc e che siano state modificate più recentemente della directory Documenti
- Ilcomandomodificatuttiitempi(atime,mtimeectime)dituttedirectoryilcui nome comincia con Doc e che siano state modificate più recentemente della directory Documenti
- Nessuna delle altre opzioni è vera

ESERCIZIO 56a

Si vuole scrivere un programma equivalente al seguente script:

```
echo -n "Esecuzione in corso..."
/bin/ls -la /
echo " fatto"
```

Quale dei seguenti frammenti di codice realizza quanto detto sopra?

```
char **argv = {"/bin/ls", "-la", "/"};
printf("Esecuzione in corso...");
execv("/bin/ls", argv);
printf("fatto\n");
```

```
char **argv = {"/bin/ls", "-la", "/", 0};
printf("Esecuzione in corso...");
execv("/bin/ls", argv);
printf("fatto\n");
```

```
char **argv = {"-la", "/", 0};
printf("Esecuzione in corso...");
execv("/bin/ls", argv);
printf("fatto\n");
```

- [x] Nessuna delle altre opzioni è corretta

ESERCIZIO 56b

Si vuole scrivere un programma equivalente al seguente script:

```
echo -n "Esecuzione in corso..."
/bin/ls -la /
echo " fatto"
```

Quale dei seguenti frammenti di codice realizza quanto detto sopra?

```
printf("Esecuzione in corso...");
execl("/bin/ls", "/bin/ls", "-la", "/", NULL);
printf("fatto\n");
```

```
printf("Esecuzione in corso...");
execl("/bin/ls", "/bin/ls", "-la", "/");
printf("fatto\n");
```

```
char **argv = {"-la", "/", 0};
printf("Esecuzione in corso...");
execv("/bin/ls", "-la", "/", NULL);
printf("fatto\n");
```

- [x] Nessuna delle altre opzioni è corretta

ESERCIZIO 57a

Si supponga di voler avere in esecuzione in background i comandi `cmd1` e `cmd2` (si supponga anche che tali comandi non terminino, a meno che non gli si mandi un opportuno segnale). Quale dei seguenti modi è corretto?

```
cmd1
#premere Ctrl+Z
bg
cmd2
#premere Ctrl+Z
bg
```

```
cmd1
#premere Ctrl+Z
cmd2
#premere Ctrl+Z
```

```
cmd1
bg
#premere Ctrl+Z
cmd2
#premere Ctrl+Z
bg
```

```
cmd1
bg
#premere Ctrl+Z
cmd2
bg
#premere Ctrl+Z
```

ESERCIZIO 57b

Si supponga di voler avere in esecuzione in background i comandi cmd1 e cmd2. Quale dei seguenti modi è corretto?

```
cmd1
#premere Ctrl+Z
bg
cmd2
#premere Ctrl+Z
bg
```

```
cmd1
#premere Ctrl+Z
fg
cmd2
#premere Ctrl+Z
fg
```

```
cmd1
#premere Ctrl+Z
bg
cmd2
#premere Ctrl+Z
```

```
cmd1
#premere Ctrl+Z
cmd2
#premere Ctrl+Z
fg
```

ESERCIZIO 58

Si supponga di voler lanciare in background i comandi `cmd1` e `cmd2` (si supponga anche che tali comandi non terminino, a meno che non gli si mandi un opportuno segnale). Quale dei seguenti modi è corretto?

- `cmd1 & cmd2;`
- `cmd1 & cmd2`
- `cmd1 && cmd2`
- `( cmd1; cmd2 ) &`
- `cmd1 & cmd2 &`
- `cmd1; cmd2 &`
- Nessuna delle altre opzioni è corretta

se non c'e quello, questo

ESERCIZIO 59

Ignorando eventuali memory leaks, quale dei seguenti frammenti di codice può portare ad un segmentation fault?

```
char *p = calloc(10, sizeof(char));
p[9] = 'a';
free(p);
p = realloc(p, 10*sizeof(char));
free(p);
```

```
char *p = calloc(10, sizeof(char));
p[9] = 'a';
p = realloc(p, 10*sizeof(char));
free(p);
```

```
char *p = malloc(10*sizeof(char));
p[1] = 'a';
free(p);
```

```
char *p = malloc(10*sizeof(char));
p[9] = 'a';
p = NULL;
p = realloc(p, 10*sizeof(char));
free(p);
```

ESERCIZIO 60

Quale dei seguenti frammenti di codice è corretto?

```
int pid = fork();
if (pid == 0) { /* fai qualcosa, sei il figlio */}
else if (pid > 0) { /* fai qualcosa, sei il padre */}
else {
  perror("fork failed");
}
```

```
int pid = fork();
if (pid == 0) { /* fai qualcosa, sei il padre */}
else if (pid > 0) { /* fai qualcosa, sei il figlio */}
else {
  perror("fork failed");
}
```

```
int pid = fork();
if (pid < 0) { /* fai qualcosa, sei il figlio */}
else if (pid > 0) { /* fai qualcosa, sei il padre */}
else {
  perror("fork failed");
}
```

Tutte le altre opzioni sono sbagliate

ESERCIZIO 61

Si supponga di avere il seguente frammento di codice:

```
int var = somefunction()%100;
printf("%d\n%.2lf\n", var, (double)var);
```

dove `somefunction` ritorna un intero. Quale dei seguenti frammenti di codice scrive gli stessi caratteri sullo stdout, senza errori?

```
int var = somefunction()%100;
char buf[7];
sprintf(buf, "%d", var);
write(1, buf, strlen(buf));
write(1, "\n", 1);
sprintf(buf, "%.2lf", (double)var);
write(1, buf, strlen(buf));
write(1, "\n", 1);
```

```
int var = somefunction()%100;
char *buf;
sprintf(buf, "%d", var);
write(1, buf, strlen(buf));
write(1, "\n", 1);
sprintf(buf, "%.2lf", (double)var);
write(1, buf, strlen(buf));
write(1, "\n", 1);
```

```
int var = somefunction()%100;
double var2 = (double)var;
write(1, (char *)&var, sizeof(var));
write(1, (char *)&var2, sizeof(var2));
```

```
int var = somefunction()%100;
char *buf = (char *)calloc(sizeof(int), sizeof(char));
sprintf(buf, "%d", var);
write(1, buf, strlen(buf));
write(1, "\n", 1);
sprintf(buf, "%.2lf", (double)var);
write(1, buf, strlen(buf));
write(1, "\n", 1);
free(buf);
```

ESERCIZIO 62

Si supponga di avere il seguente frammento di codice:

```
int var = somefunction1()%1000;
fprintf(stream, "%d\n%.2lf\n", var, (double)var);
```

Quale dei seguenti frammenti di codice scrive gli stessi caratteri sullo stdout, senza errori?

```
int var = somefunction1()%1000;
char buf[7];
sprintf(buf, "%d", var);
write(1, buf, strlen(buf));
write(1, "\n", 1);
printf("%d\n", sprintf(buf, "%.2lf", (double)var));
write(1, buf, strlen(buf));
write(1, "\n", 1);
```

```
int var = somefunction1()%1000;
char buf[6];
sprintf(buf, "%d", var);
write(1, buf, strlen(buf));
write(1, "\n", 1);
printf("%d\n", sprintf(buf, "%.2lf", (double)var));
write(1, buf, strlen(buf));
write(1, "\n", 1);
```

```
int var = somefunction1()%1000;
char *buf = (char *)calloc(sizeof(int), sizeof(char));
sprintf(buf, "%d", var);
write(1, buf, strlen(buf));
write(1, "\n", 1);
printf("%d\n", sprintf(buf, "%.2lf", (double)var));
write(1, buf, strlen(buf));
write(1, "\n", 1);
free(buf);
```

```
int var = somefunction1()%1000;
double var2 = (double)var;
write(1, (char *)&var, sizeof(var));
write(1, (char *)&var2, sizeof(var2));
```

ESERCIZIO 63

Si supponga di avere il seguente frammento di codice:

```
int var = somefunction1();
double var2 = somefunction2();
fprintf(stdout, "%d\n%lf\n", var, var2);
```

Quale dei seguenti frammenti di codice ha lo stesso effetto?

```
int var = somefunction1();
double var2 = somefunction2();
write(1, (char *)&var, sizeof(var));
write(1, (char *)&var2, sizeof(var2));
```

```
int var = somefunction1();
double var2 = somefunction2();
char *buf = calloc(sizeof(var2) > sizeof(var)? sizeof(var2) :
sizeof(var), sizeof(char));
sprintf(buf, "%d\n", var);
write(1, buf, sizeof(var) + 1);
sprintf(buf, "%lf\n", var2);
write(1, buf, sizeof(var2) + 1);
```

```
int var = somefunction1();
double var2 = somefunction2();
char *buf = calloc(sizeof(var2) > sizeof(var)? sizeof(var2) :
sizeof(var), sizeof(char));
sprintf(buf, "%d", var);
write(1, buf, sizeof(var));
write(1, "\n", 1);
sprintf(buf, "%lf", var2);
write(1, buf, sizeof(var2));
write(1, "\n", 1);
```

```
int var = somefunction1();
double var2 = somefunction2();
char buf[4];
sprintf(buf, "%d", var);
write(1, buf, sizeof(var));
write(1, "\n", 1);
sprintf(buf, "%lf", var2);
write(1, buf, sizeof(var2));
write(1, "\n", 1);
```

ESERCIZIO 64

Si supponga di avere il seguente frammento di codice:

```
FILE *stream = fopen("file_esistente.txt", "r");
fseek(stream, -100, SEEK_END);
long pos = ftell(stream);
```

Quale dei seguenti frammenti di codice ha lo stesso effetto?

```
int fd = open("file_esistente.txt", O_RDONLY);
lseek(fd, -100, SEEK_END);
long pos = lseek(fd, 0, SEEK_CUR);
```

```
int fd = open("file_esistente.txt", O_RDONLY);
lseek(fd, -100, SEEK_END);
long pos = ltell(fd);
```

```
int fd = open("file_esistente.txt", O_RDONLY);
lseek(fd, -100, SEEK_END);
long pos = lseek(fd, -100, SEEK_SET);
```

```
int fd = open("file_esistente.txt", O_RDONLY);
lseek(fd, -100, SEEK_END);
long pos = lseek(fd, 0, SEEK_END);
```

ESERCIZIO 65

Si supponga di avere il seguente frammento di codice:

```
FILE *stream = fopen("file_esistente.txt", "r"); int var;
double var2;
fscanf(stream, "%d\n%lf\n", &var, &var2);
```

```
e che il file file_esistente.txt abbia il seguente contenuto:
4567
34.56
```

Quale dei seguenti frammenti di codice ha lo stesso effetto?

```
int fd = open("file_esistente.txt", O_RDONLY); int var;
double var2;
char *buf = calloc(sizeof(var2) > sizeof(var)? sizeof(var2) : sizeof(var), sizeof(char));
read(fd, buf, sizeof(var));
var = atoi(buf);
read(fd, buf, sizeof(var2));
var2 = atof(buf);
```

```
int fd = open("file_esistente.txt", O_RDONLY); int var;
double var2;
read(fd, &var, sizeof(var));
read(fd, &var2, sizeof(var2));
```

```
int fd = open("file_esistente.txt", O_RDONLY); int var;
double var2;
char buf[4];
read(fd, buf, sizeof(var)); var = atoi(buf);
read(fd, buf, sizeof(var2));
```

```
intfd=open("file_esistente.txt",O_RDONLY);
int var;
double var2;
fscanf(fd, "%d\n%lf\n", &var, &var2);
```

ESERCIZIO 66

Si supponga di avere il seguente frammento di codice:

```
FILE *stream = fopen(NOMEFILE, "w");
```

Quale dei seguenti frammenti di codice ha lo stesso effetto?

```
int fd = open(NOMEFILE, O_WRONLY | O_CREAT, 0666);
```

```
int fd = open(NOMEFILE, O_WRONLY | O_TRUNC);
```

```
int fd = open(NOMEFILE, O_WRONLY);
```

```
int fd = open(NOMEFILE, O_WRONLY | O_CREAT | O_TRUNC, 0666);
```