Esercizio 1

Scrivere un programma in C che prende in input (su riga di comando) il nome di un file e visualizza:

- “Il file <nomefile> uid=<uid dell'owner del file> gid=<gid del file>”
- Se la uid dell'owner del file coincide con la uid del processo controlla il bit x per le protezioni per l'owner e visualizza il messaggio:
  - “Il bit x <e'/non e'> settato per l'owner”
- Se il gid dell'owner del file coincide con il gid del processo controlla il bit x per le protezioni per il gruppo e visualizza:
  - “Il bit x e'/non e' settato per il gruppo”
- Altrimenti controlla il bit x per le protezioni per gli altri e stampa:
  - “Il bit x e'/non e' settato per gli altri”

Esercizio 2

Scrivere un programma C che:

- Prende in input da linea di comando il nome di una directory ed il nome di un file.
- Se non e' possibile aprire la directory, visualizza un messaggio d'errore e termina.
- Se il file non esiste all'interno della directory, visualizza il messaggio: “File non esistente”
- Se il file esiste nella directory visualizz su STANDARD OUTPUT:
  - “File:” seguito dal nome del file, uid e gid del file

Esercizio 3

Scrivere un programma C che prende in input da linea di comando il nome di una directory.

- Se non e' possibile aprire la directory, visualizza un messaggio d'errore e termina.
- Per ogni file contenuto nella directory visualizza su STANDARD OUTPUT:
  - Per i file regolari: La stringa “File:” seguita dal nome del file e dalla sua dimensione;
  - Per le sottodirectory: La stringa “Directory:” seguita dal nome della sottodirectory.

Esercizio 4

Scrivere un programma in C che, dato in input il nome di una directory ed un intero x ed una stringa y, visualizza:

- L'elenco di tutti I file, contenuti nella directory indicata od in una sua sottodirectory, la cui dimensione sia maggiore di x.
- L'elenco di tutte le directory che contengono un file di nome y.

Esercizio 5

Scrivere un programma in C che visualizza:

- Per ognuno dei 7 tipi di file definiti da Unix:
  - il numero totale di file del tipo dato contenuti all'interno dell'albero radicato nella directory corrente.
  - il numero di file del tipo dato, contenuti all'interno dell'albero radicato nella directory corrente, ai quali il processo ha accesso in lettura.
  - il numero di file del tipo dato, contenuti all'interno dell'albero radicato nella directory corrente, ai quali il processo ha accesso in scrittura.
  - il numero di file del tipo dato, contenuti all'interno dell'albero radicato nella directory corrente, ai quali il processo ha accesso in esecuzione.

Esercizio 6

Si assume l'esistenza di un file di testo in cui ogni riga sia composta al più caratteri 15 caratteri nell'insieme {a-z, A-Z, .}

Scrivere un programma in C che legge il file una riga per volta ed esegue le seguenti operazioni:

- Se la stringa x letta dal file contiene il carattere ".", crea nella directory corrente un file vuoto con nome x;
- Se la stringa x letta dal file (a) non contiene il carattere "." e (b) termina per "-", crea nella directory corrente una sottodirectory di nome x.
- Se la stringa x letta dal file (a) non contiene il carattere "." e (b) NON termina per "-", crea nella directory corrente una sottodirectory di nome x che diventa la nuova directory corrente (si veda la funzione chdir).

Esercizio 7

Si considerino le seguenti strutture:

```
struct albero{
    char nome[5];
    struct albero *sx;
    struct albero *dx;
}
struct lista{
    char nome[100];
    struct lista*next;
}
```

Scrivere una funzione in linguaggio C con la seguente firma:

void crea_directory(struct nodo *);

La funzione prende in input il puntatore ad un albero binario e ricrea su disco un filesystem con la stessa struttura. La funzione costruisce, inoltre, una lista contenente l'elenco dei percorsi creati.

Esercizio 8

- Scrivere un programma in linguaggio C che visualizza:
  - PID, GID, PPID
  - Effective UID e GID
  - REAL UID e GID
- Modificando opportunamente gli attributi dell'eseguibile, identificare I casi in cui
  - Effective UID e real uid differiscono
  - Effective GID e real gid differiscono

Esercizio 9

- Scrivere un programma in linguaggio C in cui:
  - Un processo crea un figlio e si mette in attesa della sua terminazione
  - Il figlio visualizza un messaggio e termina con exit(1)
  - Il padre crea un secondo figlio e si mette in attesa della sua terminazione
  - Il figlio visualizza un messaggio e termina con abort()
  - Il padre visualizza un messaggio e termina.
- Quando un figlio termina, il padre visualizza le informazioni “di terminazione del figlio”.

Esercizio 10

Scrivere un programma in linguaggio C che riceve su riga di comando un intero n.

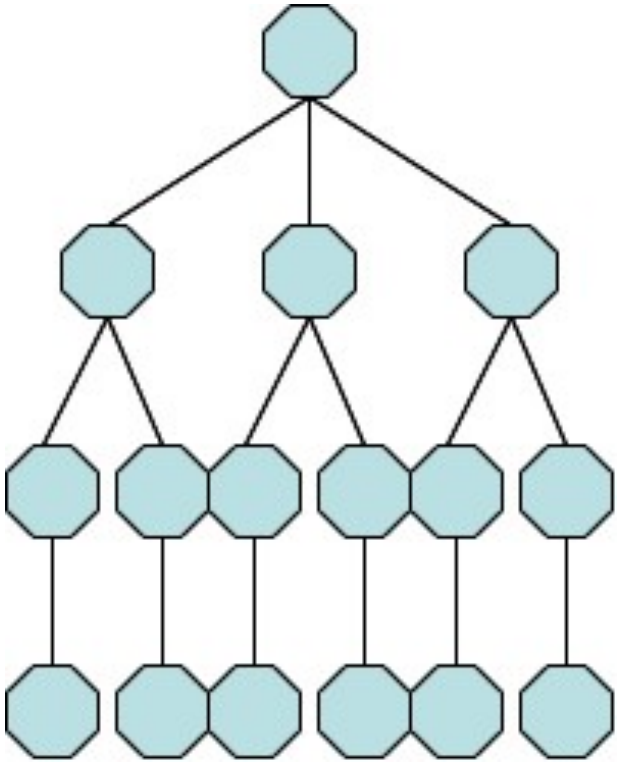
- Il processo crea n figli.
- Il figlio i-esimo visualizza il messaggio “Figlio” seguito dal valore di i e termina.
- Il padre aspetta la terminazione di tutti I figli.

Esercizio 11

Scrivere un programma in C che ricrea l'albero dei processi rappresentato in figura:

- Il processo al livello zero ha tre processi figli
- Ogni processo al livello uno ha due figli
- Ogni processo al livello due ha un unico figlio

Ogni processo nell'albero visualizza l'elenco dei pid dei processi che collegano il proprio nodo alla radice.



Esercizio 12

- Generalizzare il programma dell'esercizio precedente come segue:

- Il programma riceve in input un intero n.
- Il processo al livello zero crea n processi figli
- Ogni processo al livello 1 crea n-1 figli
- Ogni processo al livello 2 crea n-2 figli
- …
- Ogni processo al livello l crea n-l figli.

Ogni processo nell'albero visualizza l'elenco dei pid dei processi che collegano il proprio nodo alla radice.

Esercizio 13

- Scrivere un programma “mini-shell” in linguaggio C che:
  - Riceve un input una stringa da standard input
  - Crea un processo figlio che “esegue” la stringa
- Il processo termina quando l’utente digita il comanda “exit”.

Esercizio 14

- Modificare l'esercizio 4 in modo da consentire l'utilizzo della redirezione.
  - E.g., la "mini-shell+" interpreta correttamente comandi del tipo

    Comando par1 par2… < input > output 2> error