DOMANDE/RISPOSTE GRUPPO A

1. Descrivere l'effetto dei seguenti comandi di shell:
i) cat, ii) more, iii) less, iv) head, v) tail
Inoltre, descrivere l'effetto del flag "-n" per head e tail.

Risposta domanda 1:

cat - Il comando shell "cat" (dall'inglese "concatenate") viene utilizzato per visualizzare il contenuto di uno o più file di testo direttamente sullo schermo del terminale. Quando si utilizza il comando "cat" senza specificare alcuna opzione o argomento, il suo comportamento predefinito è quello di concatenare il contenuto dei file di input e di mostrarli a schermo.

more: Il comando shell "more" viene utilizzato per visualizzare il contenuto di un file di testo direttamente sullo schermo del terminale in modo interattivo. A differenza del comando "cat", che mostra tutto il contenuto di un file in una sola volta, il comando "more" divide il contenuto in pagine e mostra ogni pagina alla volta. Quando si utilizza il comando "more", il contenuto del file viene visualizzato una pagina alla volta, e il prompt del terminale mostra il numero della pagina corrente e il totale delle pagine. L'utente può scorrere avanti e indietro tra le pagine utilizzando i tasti della tastiera, come ad esempio la barra spaziatrice per passare alla pagina successiva e il tasto "b" per tornare alla pagina precedente.

less: Il comando shell "less" è un'alternativa al comando "more" per visualizzare il contenuto di un file di testo direttamente sullo schermo del terminale in modo interattivo. A differenza del comando "more", che visualizza il contenuto del file in modo progressivo a pagina per volta, il comando "less" permette all'utente di scorrere il contenuto del file in modo più flessibile e preciso. In particolare, il comando "less" consente di scorrere avanti e indietro tra le pagine del file, ma anche di scorrere il contenuto del file riga per riga, di effettuare ricerche all'interno del file e di saltare a una specifica posizione del file utilizzando il numero di riga. Il comando "less" è anche in grado di gestire file di grandi dimensioni in modo efficiente, poiché legge il contenuto del file solo al bisogno e non lo carica tutto in memoria, come invece fa il comando "more".

head: Il comando shell "head" viene utilizzato per visualizzare le prime righe di uno o più file di testo direttamente sullo schermo del terminale. Il comportamento predefinito del comando "head" è quello di mostrare le prime 10 righe di un file di testo. Se si vuole visualizzare un numero diverso di righe, è possibile specificarlo utilizzando l'opzione "-n" seguita dal numero desiderato. Ad esempio, il comando "head -n 5 file.txt" visualizzerà le prime 5 righe del file "file.txt". Inoltre, il comando "head" può essere utilizzato per visualizzare le prime righe di più file contemporaneamente, specificando i nomi dei file come argomenti separati da spazio. In questo caso, il risultato mostrerà le prime righe di ogni file nell'ordine in cui sono stati specificati.

tail: Il comando shell "tail" viene utilizzato per visualizzare le ultime righe di uno o più file di testo direttamente sullo schermo del terminale. Il comportamento predefinito del comando "tail" è quello di mostrare le ultime 10 righe di un file di testo. Se si vuole visualizzare un numero diverso di righe, è possibile specificarlo utilizzando l'opzione "-n" seguita dal numero desiderato. Ad esempio, il comando "tail -n 5 file.txt" visualizzerà le ultime 5 righe del file "file.txt". Inoltre, il comando "tail" può essere utilizzato per visualizzare le ultime righe di più file contemporaneamente, specificando i nomi dei file come argomenti separati da spazio. In questo caso, il risultato mostrerà le ultime righe di ogni file nell'ordine in cui sono stati specificati. Il comando "tail" è molto utile per visualizzare rapidamente la fine di un file di testo e per monitorare eventuali aggiunte di nuove righe. Ad esempio, è possibile utilizzare il comando "tail -f" per "segui" un file di log in tempo reale, visualizzando continuamente le nuove righe che vengono aggiunte al file mentre viene scritto.

2. Descrivere i seguenti comandi di shell:
ls -lh; ls *.c; ls /dev/ cd; cd *nome;* cd. ; cd .. rm *nome*; rm *; rmdir *nome*; rm -r *nome* mkdir *nome*; cp *nome nome1*; mv *nome nome1*

Comando ls-lh: Il comando di shell "ls -lh" viene utilizzato per elencare il contenuto di una directory in cui si trova l'utente corrente."ls" è l'abbreviazione di "list" e viene utilizzato per elencare il contenuto della directory.

L'opzione "-l" indica di visualizzare il contenuto della directory in formato "lungo", ovvero con informazioni dettagliate su ogni file o directory, come il proprietario, i permessi di accesso, la dimensione e la data di creazione o modifica.

L'opzione "-h" (human-readable) rende la visualizzazione delle dimensioni dei file in formato leggibile all'utente, utilizzando le unità di misura più appropriate per le dimensioni dei file, come KB, MB o GB.

In sintesi, il comando "ls -lh" mostrerà il contenuto della directory corrente con informazioni dettagliate sui file e sulle directory, con le dimensioni dei file visualizzate in formato leggibile all'utente. Questo comando è molto utile per ottenere informazioni dettagliate sul contenuto di una directory, in modo da poter prendere decisioni informate su quali file o directory manipolare.

Comando ls *.c: Il comando di shell "ls *.c" viene utilizzato per elencare i file presenti nella directory corrente che hanno estensione ".c". In particolare, "ls" è il comando di elencazione dei file in Unix e Linux, mentre il carattere "" viene utilizzato come carattere jolly per rappresentare qualsiasi stringa di caratteri. Quindi, quando viene utilizzato ".c", il carattere jolly rappresenta qualsiasi stringa di caratteri prima dell'estensione ".c".

Comando ls /dev/cd: Il comando di shell "ls /dev/cd" viene utilizzato per elencare le informazioni sul dispositivo CD/DVD-ROM nella directory "/dev/" del sistema. In particolare, "/dev/cd" si riferisce al dispositivo CD/DVD-ROM del sistema, che è spesso rappresentato come "/dev/cdrom" o "/dev/dvd" a seconda della distribuzione Linux utilizzata. Il comando "ls" è utilizzato per elencare i file e le directory nella posizione specificata, mentre "/dev/cd" indica la posizione specifica del dispositivo CD/DVD-ROM. Quando il comando viene eseguito, elenca le informazioni sul dispositivo CD/DVD-ROM come se fosse un file. Le informazioni possono includere le autorizzazioni, il proprietario, la dimensione del file e così via.

Comando cd nome: Il comando di shell "cd nome" viene utilizzato per cambiare la directory corrente del terminale in una specifica directory con il nome "nome". In questo comando, "cd" è l'abbreviazione di "change directory", ovvero "cambia directory" in italiano. La specifica directory "nome" può essere un percorso assoluto o relativo. Se "nome" è un percorso assoluto, il comando cambia la directory corrente direttamente in quella posizione. Ad esempio, se si esegue il comando "cd /home/user/Desktop", la directory corrente verrà cambiata alla directory "Desktop" all'interno della directory dell'utente. D'altra parte, se "nome" è un percorso relativo, il comando cambia la directory corrente in base alla directory corrente attuale. Ad esempio, se si esegue il comando "cd Documents", la directory corrente verrà cambiata alla sottodirectory "Documents" della directory corrente. È importante notare che il comando "cd" è uno dei comandi di base utilizzati nella shell di Unix e Linux per la gestione delle directory. Inoltre, la directory corrente è la directory di lavoro attuale del terminale e può essere visualizzata utilizzando il comando "pwd".

Comando cd .: Il comando di shell "cd ." viene utilizzato per cambiare la directory corrente del terminale nella stessa directory corrente. In questo comando, "cd" è l'abbreviazione di "change directory", ovvero "cambia directory" in italiano, mentre il punto "." rappresenta la directory corrente. Quando viene eseguito il comando "cd .", il terminale cambia la directory corrente in quella in cui si trova attualmente. Questo comando può essere utilizzato per rinnovare la visualizzazione della directory corrente, in quanto a volte può essere utile ricaricare la visualizzazione della directory per verificare eventuali modifiche. È importante notare che il punto "." è una rappresentazione simbolica della directory corrente in Unix e Linux. Inoltre, questo comando può essere utile per evitare di dover digitare il nome completo della directory corrente quando si vuole tornare alla stessa directory in futuro.

Comando cd .. rm nome: Il comando di shell "cd .. rm nome" è in realtà una combinazione di due comandi separati: "cd .." e "rm nome". Il comando "cd .." viene utilizzato per cambiare la directory corrente del terminale alla directory genitore della directory corrente. In questo comando, ".." rappresenta la directory genitore, ovvero la directory che contiene la directory corrente. Dopo aver eseguito il comando "cd .." e aver cambiato la directory corrente, viene eseguito il comando "rm nome". Questo comando viene utilizzato per rimuovere il file o la directory con il nome specificato "nome" nella directory corrente.

Comando rm *: Il comando di shell "rm " viene utilizzato per eliminare tutti i file nella directory corrente. In questo comando, "rm" è l'abbreviazione di "remove", ovvero "rimuovi" in italiano, mentre l'asterisco "" rappresenta tutti i file nella directory corrente.

Comando rmdir nome: Il comando di shell "rmdir nome" viene utilizzato per rimuovere una directory vuota di nome "nome" nella directory corrente. In questo comando, "rmdir" è l'abbreviazione di "remove directory", ovvero "rimuovi directory" in italiano, mentre "nome" rappresenta il nome della directory che si desidera eliminare. È importante notare che il comando "rmdir" può essere utilizzato solo per rimuovere directory vuote. Se la directory contiene file o altre directory, il comando "rmdir" non funzionerà e verrà visualizzato un messaggio di errore. In questo caso, sarà necessario utilizzare il comando "rm" per eliminare il contenuto della directory prima di utilizzare il comando "rmdir" per rimuovere la directory vuota.

Comando rm -r nome mkdir nome: Il comando di shell "rm -r nome mkdir nome" è un comando di shell che può essere utilizzato per rimuovere una directory di nome "nome" e quindi creare una nuova directory con lo stesso nome. In dettaglio, il comando "rm -r nome" viene utilizzato per rimuovere la directory "nome" e tutti i suoi contenuti in modo ricorsivo, ovvero tutte le sottodirectory e i file contenuti in essa saranno rimossi. Successivamente, il comando "mkdir nome" viene utilizzato per creare una nuova directory vuota di nome "nome" nella directory corrente.

Comando cp nome nome1: Il comando di shell "cp nome nome1" viene utilizzato per copiare il contenuto di un file o di una directory chiamata "nome" nella directory corrente in un nuovo file o directory chiamata "nome1". In particolare, il comando "cp" è l'abbreviazione di "copy", ovvero "copia" in italiano, e la sintassi "cp nome nome1" indica che il contenuto del file o della directory di nome "nome" sarà copiato in un nuovo file o directory chiamata "nome1". Il comportamento del comando "cp" dipende dal fatto che "nome" sia un file o una directory. Se "nome" è un file, il contenuto del file sarà copiato in un nuovo file chiamato "nome1". Se "nome" è invece una directory, la directory e il suo contenuto saranno copiati in una nuova directory chiamata "nome1".

Comando mv nome nome1: Il comando di shell "mv nome nome1" viene utilizzato per rinominare un file o una directory chiamata "nome" nella directory corrente in un nuovo nome "nome1", oppure per spostare il file o la directory "nome" nella directory corrente e rinominarlo con il nuovo nome "nome1". In particolare, il comando "mv" è l'abbreviazione di "move", ovvero "sposta" o "rinomina" in italiano, e la sintassi "mv nome nome1" indica che il file o la directory di nome "nome" verrà spostato o rinominato in "nome1".

3. Descrivere il comando `find`, e fornire un esempio (inclusa la sintassi per l'esecuzione su shell) in cui è combinato con l'azione `exec`.

La sintassi di base per il comando find è la seguente:

find [percorso] [criteri di ricerca]

dove [percorso] specifica la directory di partenza per la ricerca e [criteri di ricerca] sono le opzioni utilizzate per definire le condizioni di ricerca.

Esempio di comando find:

find /home/user/docs -name "*.txt"

In questo esempio, il comando find cerca tutti i file con estensione ".txt" nella directory "/home/user/docs" e in tutte le sue sotto-directory.

L'opzione -name viene utilizzata per specificare il nome del file da cercare, utilizzando il carattere jolly * per indicare qualsiasi carattere o sequenza di caratteri nel nome del file.

Il comando find può essere combinato con l'azione exec per eseguire un comando specifico sui file trovati durante la ricerca. L'azione exec viene utilizzata per eseguire un comando specificato dall'utente sui file trovati durante la ricerca.

Esempio di comando find combinato con l'azione exec:

find /home/user/docs -name "*.txt" -exec grep "example" {} \;

In questo esempio, il comando find cerca tutti i file con estensione ".txt" nella directory "/home/user/docs" e in tutte le sue sotto-directory. L'opzione -exec viene utilizzata per specificare l'azione grep "example" da eseguire sui file trovati.

Il simbolo {} viene utilizzato per indicare il file corrente trovato durante la ricerca e \; viene utilizzato per indicare la fine dell'azione exec.

In questo esempio, il comando grep cerca la parola "example" in ogni file con estensione ".txt" trovato dalla ricerca.

Per associare i permessi al file "prova" è possibile utilizzare il comando chmod della shell. La sintassi generale di chmod è la seguente:

4. Associare, tramite shell, i seguenti permessi al file "prova":
I permessi da applicare saranno definiti in sede di esame. A titolo di esempio, si riportano le possibili modalità della richiesta:
1. Lettura, scrittura, esecuzione per utente e gruppo, solo lettura per tutti gli altri
2. Scrittura per utente, esecuzione per gruppo, lettura e scrittura per tutti gli altri
3. rwr --x -w-
4. --x -wx r--

chmod [opzioni] [permessi] [file]

Dove [opzioni] sono le opzioni che possono essere utilizzate, [permessi] specifica i permessi da applicare e [file] è il file al quale si vuole applicare i permessi.

Esempio:

chmod 760 prova

Questo comando assegna i permessi 760 al file "prova". In particolare, la prima cifra 7 rappresenta i permessi dell'utente proprietario del file (in questo caso, lettura, scrittura e esecuzione), la seconda cifra 6 rappresenta i permessi del gruppo proprietario del file (lettura e scrittura), mentre la terza cifra 0 rappresenta i permessi degli altri utenti (nessun permesso).

Per assegnare i permessi come richiesti nella modalità d'esame, possiamo utilizzare le seguenti combinazioni di permessi e opzioni:

Lettura, scrittura, esecuzione per utente e gruppo, solo lettura per tutti gli altri:

chmod 770 prova

Scrittura per utente, esecuzione per gruppo, lettura e scrittura per tutti gli altri:

chmod 765 prova

rwr --x -w- :

chmod 624 prova

--x -wx r--:

chmod 421 prova

Si noti che la sintassi dei permessi utilizzati è la forma octal, in cui ogni cifra rappresenta i permessi per l'utente, il gruppo e gli altri, rispettivamente. Ad esempio, 7 indica lettura, scrittura e esecuzione, 6 indica lettura e scrittura, 5 indica lettura e esecuzione, 4 indica solo lettura, 3 indica lettura e esecuzione, 2 indica solo scrittura e 1 indica solo esecuzione.

5. Descrivere la differenza tra la concatenazione di comandi tramite “;” , “&&” e “||”

Nella shell Unix/Linux, è possibile concatenare più comandi in una singola linea di comando utilizzando diversi operatori. Le tre opzioni più comuni sono ;, && e ||.

;: questo operatore consente di concatenare due o più comandi indipendenti. Ciò significa che il secondo comando verrà eseguito indipendentemente dal successo o dal fallimento del primo comando.

Esempio:

comando1; comando2

In questo caso, il comando1 verrà eseguito e, indipendentemente dal suo successo o fallimento, il comando2 verrà eseguito subito dopo.

&&: questo operatore consente di concatenare due o più comandi in modo che il secondo comando venga eseguito solo se il primo comando ha avuto successo (cioè ha restituito un valore di uscita 0).

Esempio:

comando1 && comando2

In questo caso, il comando2 verrà eseguito solo se il comando1 ha avuto successo.

||: questo operatore consente di concatenare due o più comandi in modo che il secondo comando venga eseguito solo se il primo comando ha fallito (cioè ha restituito un valore di uscita diverso da 0).

Esempio:

comando1 || comando2

In questo caso, il comando2 verrà eseguito solo se il comando1 ha fallito.

In sintesi, l'operatore ; concatena due o più comandi in modo indipendente, && concatena i comandi solo se il precedente ha avuto successo, mentre || concatena i comandi solo se il precedente ha fallito.

6. Descrivere la struttura del file system Linux, elencando almeno 3 tra le sottodirectory di root (“/”), e specificandone il contenuto atteso.

Il file system di Linux segue una struttura gerarchica a albero, in cui ogni directory è una sotto-directory di un'altra, a partire dalla radice principale del sistema, indicata dal simbolo "/". Qui di seguito sono elencate tre sottodirectory della directory principale "/":

/bin: questa directory contiene i programmi di base necessari per il funzionamento del sistema, come ad esempio i comandi di shell e altri programmi di utilità. Alcuni dei programmi presenti in questa directory includono ls, cp, mv, rm, mkdir, cat, grep, ecc.

/etc: questa directory contiene i file di configurazione di sistema. Qui è possibile trovare i file di configurazione del sistema, come ad esempio passwd, group, hosts, fstab, network, cron, ssh, ecc. Questi file vengono utilizzati dal sistema operativo per configurare vari aspetti del sistema, come le impostazioni dell'utente, le impostazioni di rete, le impostazioni di sicurezza e altro ancora.

/usr: questa directory contiene tutti i programmi e le librerie non essenziali per il funzionamento del sistema. In questa directory si possono trovare programmi, librerie, documentazione e altri file associati alle applicazioni installate. Alcuni esempi di sottodirectory all'interno di /usr includono /usr/bin (programmi non essenziali per il sistema), /usr/lib (librerie condivise), /usr/share (file condivisi) e /usr/local (programmi e librerie locali all'utente).

Queste sono solo alcune delle directory principali all'interno del file system di Linux. Ogni directory ha il proprio scopo e funzione specifici nel sistema operativo, e l'organizzazione gerarchica consente di mantenere il sistema operativo organizzato e facilmente navigabile.

7. Cosa è un puntatore, e come si utilizza? Fornire un esempio di creazione e utilizzo di un puntatore.

In programmazione, un puntatore è una variabile che contiene l'indirizzo di memoria di un'altra variabile. In altre parole, un puntatore "punta" ad una determinata posizione di memoria dove è allocata una variabile. Utilizzando un puntatore, è possibile accedere e manipolare il contenuto della variabile a cui punta.

La dichiarazione di un puntatore avviene indicando il tipo della variabile a cui punta e il nome del puntatore, preceduto dall'operatore "*". Ad esempio, per dichiarare un puntatore a un intero, si può utilizzare la seguente sintassi:

```
int *p;
```

In questo caso, p è un puntatore a un intero.

Per assegnare un indirizzo di memoria ad un puntatore, si utilizza l'operatore "&". Ad esempio:

```
int x = 5;
int *p;

p = &x; // assegna l'indirizzo di memoria di x al puntatore p
```

In questo caso, p punta all'indirizzo di memoria di x.

Per accedere al contenuto della variabile a cui punta un puntatore, si utilizza l'operatore "*". Ad esempio:

```
int x = 5;
int *p;
p = &x;
printf("Il valore di x è %d\n", *p); // stampa "Il valore di x è 5"
```

In questo caso, l'operatore "*" viene utilizzato per accedere al valore di x, a cui punta il puntatore p.

Un altro esempio di utilizzo di un puntatore potrebbe essere quello di utilizzarlo per accedere ad un'array:

```
int arr[5] = {1, 2, 3, 4, 5};
int *p = &arr[0];

for (int i = 0; i < 5; i++) {
  printf("Il valore di arr[%d] è %d\n", i, *(p+i));
}
```

In questo caso, il puntatore p viene inizializzato con l'indirizzo del primo elemento dell'array arr. Il ciclo for utilizza l'operatore "*" per accedere al valore di ciascun elemento dell'array.

8. Descrivere le funzioni fopen, fclose, fseek.

Le funzioni fopen, fclose e fseek sono funzioni standard del linguaggio di programmazione C utilizzate per la gestione dei file.

La funzione fopen viene utilizzata per aprire un file. La sua sintassi è la seguente:

FILE *fopen(const char *filename, const char *mode);

La funzione riceve come primo parametro il nome del file da aprire, come secondo parametro una stringa che indica la modalità di apertura del file ("r" per la sola lettura, "w" per la sola scrittura, "a" per l'aggiunta in coda, ecc.). Restituisce un puntatore a una struttura FILE che rappresenta il file aperto, o NULL in caso di errore.

La funzione fclose viene utilizzata per chiudere un file precedentemente aperto con fopen. La sua sintassi è la seguente:

int fclose(FILE *stream);

La funzione riceve come parametro il puntatore al file da chiudere restituisce 0 in caso di successo, oppure un valore negativo in caso di errore.

La funzione fseek viene utilizzata per spostare il puntatore di lettura/scrittura all'interno di un file. La sua sintassi è la seguente:

int fseek(FILE *stream, long int offset, int whence);

La funzione riceve come primo parametro il puntatore al file su cui effettuare lo spostamento, come secondo parametro un valore intero che indica l'offset di spostamento in byte e come terzo parametro una costante che indica la posizione di partenza da cui effettuare lo spostamento (SEEK_SET per l'inizio del file, SEEK_CUR per la posizione corrente del puntatore, SEEK_END per la fine del file). Restituisce 0 in caso di successo, oppure un valore negativo in caso di errore.

In generale, queste funzioni sono utilizzate insieme per la lettura e la scrittura di file all'interno di un programma C.

9. Cosa è una struttura in C? Fornire un esempio di struttura.

In C, una struttura è un tipo di dato definito dall'utente che permette di raggruppare un insieme di variabili di tipi diversi sotto un unico nome. Una struttura può essere pensata come una sorta di contenitore per informazioni correlate.

La sintassi per la definizione di una struttura è la seguente:

```
struct nome_struttura {
   tipo variabile1;
   tipo variabile2;
   /* ... */
};
```

dove nome_struttura è il nome della struttura, tipo è il tipo di dato di ogni variabile all'interno della struttura e variabile1, variabile2, ecc. sono i nomi delle variabili.

Ad esempio, si può definire una struttura persona come segue:

```
struct persona {
   char nome[50];
   int eta;
   float altezza;
};
```

La struttura persona contiene tre variabili: nome, eta e altezza, che rispettivamente rappresentano il nome, l'età e l'altezza di una persona. Si può poi dichiarare una variabile di tipo persona e assegnare valori alle sue variabili come segue:

```
struct persona p;
strcpy(p.nome, "Mario Rossi");
p.eta = 30;
p.altezza = 1.75;
```

In questo esempio, si dichiara una variabile di tipo persona chiamata p e si assegnano i valori "Mario Rossi" per il nome, 30 per l'età e 1.75 per l'altezza.

Le strutture sono molto utili quando si lavora con dati complessi e correlati, poiché consentono di organizzare i dati in modo più logico e facile da gestire.

10. Descrivere le funzioni per l'allocazione e deallocazione di memoria.

In C, l'allocazione e deallocazione della memoria sono operazioni fondamentali che consentono di gestire la memoria a disposizione del programma.

La funzione principale per l'allocazione di memoria è malloc(), che permette di richiedere al sistema operativo un blocco di memoria di dimensione specificata. La sintassi di malloc() è la seguente:

```
void* malloc(size_t size);
```

dove size è la dimensione del blocco di memoria richiesta in byte. La funzione restituisce un puntatore di tipo void* che punta all'inizio del blocco di memoria allocato. È necessario effettuare un cast esplicito del puntatore restituito a un tipo appropriato.

Ad esempio, si può allocare un blocco di memoria di 10 interi come segue:

```
int* p = (int*) malloc(10 * sizeof(int));
```

In questo esempio, p è un puntatore a intero che punta all'inizio del blocco di memoria allocato. Si moltiplica la dimensione dell'intero per il numero di elementi richiesti per ottenere la dimensione totale del blocco di memoria richiesto.

La deallocazione della memoria avviene tramite la funzione free(), che restituisce la memoria precedentemente allocata al sistema operativo. La sintassi di free() è la seguente:

```
void free(void* ptr);
```

dove ptr è il puntatore alla memoria precedentemente allocata tramite malloc() o altre funzioni di allocazione della memoria.

Ad esempio, per deallocare il blocco di memoria precedentemente allocato, si può utilizzare:

```
free(p);
```

In questo esempio, p è il puntatore al blocco di memoria precedentemente allocato tramite malloc(), che viene liberato tramite la funzione free().

Oltre a malloc() e free(), esistono altre funzioni per l'allocazione e deallocazione della memoria, come calloc() e realloc(). La funzione calloc() è simile a malloc(), ma alloca e inizializza a zero un blocco di memoria. La funzione realloc() consente di modificare la dimensione di un blocco di memoria precedentemente allocato tramite malloc() o altre funzioni di allocazione della memoria.

11. Qual è la differenza tra funzione di libreria e chiamata di sistema? Fornire almeno un esempio di chiamata di sistema e la corrispondente funzione di libreria.

In generale, le funzioni di libreria sono un insieme di funzioni predefinite che sono disponibili come parte di una libreria di codice. Queste funzioni sono scritte in un linguaggio di programmazione ad alto livello e sono progettate per essere utilizzate da altri programmi.

Le chiamate di sistema, d'altra parte, sono funzioni fornite dal sistema operativo stesso, che consentono ai programmi di interagire con il sistema operativo. Queste chiamate di sistema sono generalmente implementate a basso livello e permettono di accedere a funzionalità del sistema operativo come la gestione dei file, la gestione della memoria e l'interazione con l'hardware.

Un esempio di chiamata di sistema è la funzione open(), che consente di aprire un file per la lettura o la scrittura. La sintassi di open() è la seguente:

```
int open(const char *pathname, int flags);
```

dove pathname è il percorso del file da aprire e flags sono le opzioni di apertura del file (come la modalità di accesso, la creazione del file se non esiste, ecc.). La funzione restituisce un file descriptor che viene utilizzato per riferirsi al file in operazioni successive.

Un esempio di funzione di libreria corrispondente a open() è fopen(). La sintassi di fopen() è la seguente:

```
FILE *fopen(const char *pathname, const char *mode);
```

dove pathname è il percorso del file da aprire e mode sono le opzioni di apertura del file (come "r" per la lettura, "w" per la scrittura, ecc.). La funzione restituisce un puntatore a un oggetto FILE, che viene utilizzato per riferirsi al file in operazioni successive.

In questo esempio, open() è una chiamata di sistema, che interagisce direttamente con il sistema operativo per aprire un file. fopen(), d'altra parte, è una funzione di libreria che fornisce un'interfaccia più facile da usare per aprire un file, che utilizza open() sotto il cofano per effettuare la chiamata di sistema.

12. Cosa è e come si utilizza l'utility make?

L'utility make è un programma di build automation per semplificare la compilazione e l'organizzazione dei progetti software. In particolare, make automatizza il processo di compilazione e di creazione dei file eseguibili o delle librerie, tenendo traccia delle dipendenze tra i file sorgente.

Il funzionamento di make si basa su un file di configurazione chiamato "Makefile", che specifica come compilare il progetto e definisce le dipendenze tra i vari file del progetto.

Il formato di un Makefile è costituito da regole che specificano il modo in cui i file di output sono generati a partire dai file di input, utilizzando un particolare compilatore o strumento di build. In una regola, vengono specificati i file di input e i file di output, insieme a eventuali opzioni del compilatore o altri parametri necessari.

Ad esempio, una regola per la compilazione di un file sorgente in C potrebbe avere la seguente sintassi:

```
target: dependencies
    command
```

Dove "target" è il nome del file di output, "dependencies" sono i file di input richiesti per la compilazione del file di output e "command" è il comando da eseguire per generare il file di output.

Una volta definito il Makefile, l'utente può eseguire il comando make dalla linea di comando per avviare il processo di compilazione. make legge il Makefile e determina quali regole devono essere eseguite in base alle dipendenze tra i file. Successivamente, esegue i comandi specificati nelle regole per generare i file di output.

make offre inoltre una serie di opzioni per personalizzare il processo di compilazione, come la specifica di opzioni del compilatore, la selezione di regole specifiche da eseguire e la gestione delle dipendenze tra i file.

In sintesi, l'utilità make semplifica il processo di compilazione e di organizzazione dei progetti software, automatizzando la creazione dei file di output a partire dai file sorgente e tenendo traccia delle dipendenze tra i file.

13. Cosa è il preprocessore e quali sono le sue principali funzionalità?

In linguaggio di programmazione C, il preprocessore è un componente del compilatore che si occupa di elaborare il codice sorgente prima della fase di compilazione vera e propria. Il preprocessore effettua una serie di operazioni di trasformazione del codice, tra cui la sostituzione di macro, la gestione delle direttive di inclusione e la definizione di costanti simboliche.

Le principali funzionalità del preprocessore includono:

Sostituzione di macro: il preprocessore sostituisce ogni occorrenza di una macro con il corrispondente codice definito in fase di pre-compilazione. Ciò consente di definire blocchi di codice riutilizzabili in più parti del programma, semplificando la scrittura del codice.

Gestione delle direttive di inclusione: il preprocessore gestisce le direttive di inclusione (#include) che permettono di importare codice da altri file di header. Ciò consente di separare la definizione delle funzioni, delle variabili e delle costanti dal loro utilizzo, semplificando la gestione del codice.

Definizione di costanti simboliche: il preprocessore permette di definire costanti simboliche, ovvero nomi che rappresentano un valore costante. Questo permette di semplificare la scrittura del codice e di evitare la presenza di numeri "magici" all'interno del codice sorgente.

Compilazione condizionale: il preprocessore permette di condizionare la compilazione di parti del codice a seconda di determinate condizioni. Questo consente di escludere parti di codice in fase di compilazione a seconda del target o delle opzioni di compilazione utilizzate.

In sintesi, il preprocessore è uno strumento importante nella fase di sviluppo del software in C, che permette di semplificare la scrittura del codice, di importare definizioni di funzioni e costanti da altri file e di gestire la compilazione condizionale del codice.

14. Descrivere la sintassi di almeno 3 system call per operare su file.

open(): questa system call viene utilizzata per aprire un file esistente o crearne uno nuovo. La sintassi è la seguente:

```
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>

int open(const char *pathname, int flags, mode_t mode);
```

dove "pathname" rappresenta il percorso del file da aprire, "flags" specifica le opzioni di apertura del file (ad esempio, "O_RDONLY" per aprire il file in sola lettura o "O_CREAT" per crearne uno nuovo), e "mode" specifica i permessi da assegnare al file in caso di creazione. La system call restituisce un file descriptor (un intero che identifica il file aperto) oppure -1 in caso di errore.

read(): questa system call viene utilizzata per leggere il contenuto di un file. La sintassi è la seguente:

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
```

dove "fd" rappresenta il file descriptor del file da leggere, "buf" è un puntatore ad un buffer in cui verrà memorizzato il contenuto letto, e "count" rappresenta il numero di byte da leggere. La system call restituisce il numero di byte letti, oppure 0 se si è raggiunto la fine del file, oppure -1 in caso di errore.

write(): questa system call viene utilizzata per scrivere dati su un file. La sintassi è la seguente:

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

dove "fd" rappresenta il file descriptor del file su cui scrivere, "buf" è un puntatore al buffer contenente i dati da scrivere, e "count" rappresenta il numero di byte da scrivere. La system call restituisce il numero di byte scritti, oppure -1 in caso di errore.

15. Descrivere come si possono creare e distruggere processi, elencando e descrivendo le primitive coinvolte.

In un sistema operativo, un processo è un'istanza in esecuzione di un programma. Esistono diverse primitive per la creazione e la distruzione di processi in un sistema operativo. Di seguito ne descriviamo alcune:

fork(): questa primitiva crea un nuovo processo duplicando il processo chiamante. Il nuovo processo è identico a quello originale, ma ha un PID (Process Identifier) diverso. Il processo figlio viene creato eseguendo un nuovo programma o una funzione differente dalla funzione main() del programma originale. La primitiva fork() restituisce 0 nel processo figlio e il PID del figlio nel processo padre.

exec(): questa primitiva viene utilizzata per sostituire l'immagine del processo corrente con una nuova immagine (cioè, eseguire un nuovo programma). Esistono diverse varianti della primitiva exec(), a seconda della modalità di esecuzione del nuovo programma. Ad esempio, la primitiva execl() esegue un nuovo programma nella stessa immagine del processo corrente, mentre la primitiva execv() esegue un nuovo programma in una nuova immagine del processo.

wait(): questa primitiva viene utilizzata per far attendere un processo padre fino alla terminazione del processo figlio. Quando un processo figlio termina, viene generato un segnale SIGCHLD che viene catturato dal processo padre. Il processo padre può quindi utilizzare la primitiva wait() per ottenere lo stato di uscita del processo figlio e liberare le risorse utilizzate dal processo figlio.

exit(): questa primitiva viene utilizzata per terminare un processo in modo ordinato. Quando viene chiamata, la primitiva exit() termina l'esecuzione del processo corrente e restituisce uno stato di uscita.

kill(): questa primitiva viene utilizzata per inviare un segnale a un processo specifico. Il segnale può essere utilizzato per interrompere l'esecuzione del processo, gestire eventi di sistema, gestire errori, etc.

Per quanto riguarda la distruzione di processi, i processi possono essere distrutti in modo naturale (cioè, quando terminano la loro esecuzione) o tramite la primitiva kill(). In alcuni casi, è possibile che i processi vengano distrutti in modo forzato dal kernel del sistema operativo a causa di un errore di sistema o di una violazione dei permessi.

16. Descrivere cosa è un segnale, fornendo almeno un esempio di segnale inviabile tramite tastiera verso il terminale.

In un sistema operativo, un segnale è un meccanismo asincrono di comunicazione utilizzato per notificare un processo o un thread di un evento. Un segnale può essere inviato da un processo a un altro processo o a se stesso. Il processo che riceve il segnale può decidere come gestirlo, ovvero se ignorarlo, trattarlo o terminare il processo.

Un esempio di segnale inviabile tramite tastiera verso il terminale è il segnale SIGINT (Signal Interrupt), che viene generato quando l'utente preme i tasti Ctrl+C sulla tastiera. Questo segnale viene inviato al processo attualmente in esecuzione nel terminale, solitamente un processo shell. Il segnale SIGINT viene utilizzato per interrompere il processo in modo sicuro e pulito.

17. Descrivere le pipe anonime, incluse le primitive coinvolte per la loro creazione e utilizzo.

Le pipe anonime, chiamate anche pipe senza nome, sono un meccanismo di comunicazione interprocesso (IPC) in cui il flusso di dati viene diretto da un processo all'altro attraverso un canale unidirezionale. Le pipe anonime vengono create utilizzando la funzione di sistema pipe()

La funzione pipe() crea una coppia di file descriptor, uno per la lettura e uno per la scrittura. Il file descriptor per la scrittura viene utilizzato dal processo che scrive i dati nella pipe, mentre il file descriptor per la lettura viene utilizzato dal processo che legge i dati dalla pipe.

La sintassi della funzione pipe() è la seguente:

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

```
int pipe(int fd[2]);
```

dove fd è un array di due interi che conterrà i file descriptor della pipe.

Una volta creata la pipe, i processi possono utilizzare i file descriptor associati per la lettura e la scrittura. Il processo padre, ad esempio, potrebbe scrivere dei dati nella pipe e il processo figlio potrebbe leggere tali dati.

Ecco un esempio di utilizzo di una pipe anonima:

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

int main() {
 int fd[2];
 pid_t pid;
 char message[] = "Hello, world!";

 if (pipe(fd) == -1) {
  perror("pipe");
  return 1;
 }

 pid = fork();

 if (pid == -1) {
  perror("fork");
  return 1;
 }

 if (pid == 0) {
  /* processo figlio */
  char buffer[20];
  close(fd[1]); /* chiude il file descriptor per la scrittura */
```

```
    read(fd[0], buffer, sizeof(buffer));
    printf("Processo figlio: %s\n", buffer);
    close(fd[0]);
  } else {
    /* processo padre */
    close(fd[0]); /* chiude il file descriptor per la lettura */
    write(fd[1], message, sizeof(message));
    printf("Processo padre: %s\n", message);
    close(fd[1]);
  }

  return 0;
}
```

In questo esempio, viene creata una pipe anonima utilizzando la funzione pipe(). Viene quindi generato un nuovo processo utilizzando la funzione fork(). Il processo figlio legge i dati dalla pipe utilizzando la funzione read(), mentre il processo padre scrive i dati nella pipe utilizzando la funzione write(). Infine, i processi chiudono i file descriptor della pipe utilizzati per la lettura e la scrittura, rispettivamente, utilizzando la funzione close().

18. Descrivere le pipe con nome, incluse le primitive coinvolte per la loro creazione e utilizzo.

Le pipe con nome, conosciute anche come FIFO (First In First Out), sono una forma di comunicazione tra processi in cui i dati sono trasmessi attraverso un file system virtuale. Una pipe con nome è creata come un file nel file system e può essere aperta e utilizzata da più processi contemporaneamente.

La creazione di una pipe con nome avviene attraverso la system call mkfifo(), che richiede come argomento il nome della pipe da creare e una maschera di permessi per il file (come per la system call open()). Ad esempio, per creare una pipe con nome chiamata "myfifo" con permessi 0666 (lettura/scrittura per tutti gli utenti) si può utilizzare il seguente codice:

```
#include <sys/stat.h>
#include <fcntl.h>

int main() {
    int fd;
    char *fifo = "myfifo";
    mkfifo(fifo, 0666);
```

```
    return 0;
}
```

Una volta creata, una pipe con nome può essere utilizzata per la comunicazione tra processi. Per aprire la pipe con nome in lettura o scrittura, si utilizza la system call open(). Ad esempio, il seguente codice apre la pipe con nome "myfifo" in scrittura e scrive un messaggio:

```
#include <fcntl.h>
#include <unistd.h>

int main() {
    int fd;
    char *fifo = "myfifo";
    fd = open(fifo, O_WRONLY);
    write(fd, "hello world", 12);
    close(fd);
    return 0;
}
```

Il seguente codice apre la stessa pipe in lettura e legge il messaggio:

```
#include <fcntl.h>
#include <unistd.h>

int main() {
    int fd;
    char *fifo = "myfifo";
    char buf[20];
    fd = open(fifo, O_RDONLY);
    read(fd, buf, 20);
    printf("%s\n", buf);
    close(fd);
    return 0;
}
```

Una volta che tutti i processi che utilizzano la pipe con nome hanno terminato e chiuso la pipe, è possibile rimuoverla dal file system con la system call unlink(). Ad esempio:

```
#include <unistd.h>

int main() {
   char *fifo = "myfifo";
   unlink(fifo);
   return 0;
}
```

Le pipe con nome sono spesso utilizzate per la comunicazione tra processi in modalità client-server o in altre situazioni in cui i processi non condividono lo stesso spazio di indirizzamento.

19. Elencare i passi per la creazione e utilizzo di una socket, sia lato client che lato server.

La creazione e l'utilizzo di una socket è un processo complesso che coinvolge diversi passaggi sia lato client che lato server. Di seguito, sono elencati i passi principali per la creazione e l'utilizzo di una socket:

Lato server:

Creazione della socket del server con la funzione socket(). Si specifica il dominio della socket (es. AF_INET per IPv4), il tipo di socket (es. SOCK_STREAM per TCP), e il protocollo (solitamente 0 per selezionare il protocollo predefinito).

Assegnazione di un indirizzo IP e di una porta alla socket tramite la funzione bind().

Abilitazione della socket a ricevere connessioni tramite la funzione listen().

Accettazione di una connessione in arrivo tramite la funzione accept(). Questa funzione crea una nuova socket che rappresenta la connessione stabilita con il client.

Lato client:

Creazione della socket del client con la funzione socket(). Si specifica il dominio della socket (es. AF_INET per IPv4), il tipo di socket (es. SOCK_STREAM per TCP), e il protocollo (solitamente 0 per selezionare il protocollo predefinito).

Connessione alla socket del server tramite la funzione connect(). Si specifica l'indirizzo IP e la porta del server a cui ci si vuole connettere.

Una volta che la connessione tra client e server è stata stabilita, i dati possono essere scambiati attraverso la socket utilizzando le funzioni send() e recv(). Dopo l'uso, le socket devono essere chiuse tramite la funzione close().

20. Descrivere la differenza tra socket AF-INET ed AF-UNIX.

In Linux, esistono due tipi principali di socket: AF_INET e AF_UNIX (o AF_LOCAL).

AF_INET è utilizzato per la comunicazione su una rete IP. Utilizza l'IPv4 o IPv6 come protocollo di rete e ha un indirizzo IP e una porta associata.

AF_UNIX, al contrario, è utilizzato per la comunicazione all'interno dello stesso sistema operativo. Non utilizza il protocollo di rete IP, ma piuttosto un sistema di file socket all'interno del file system del sistema operativo.

In altre parole, AF_INET viene utilizzato per comunicare su una rete IP, come ad esempio Internet, mentre AF_UNIX viene utilizzato per la comunicazione tra processi all'interno dello stesso sistema operativo.