ESERCIZIO 1

Un file regolare di nome "pippo" è eseguibile da tutti, leggibile ed eseguibile dal gruppo, leggibile, modificabile ed eseguibile dal proprietario.

a) riportare la notazione Unix che rappresenta correttamente i permessi e le protezioni associate a pippo

-rwxr-x--x pippo

b) indicate il comando necessario per togliere l'eseguibilità di pippo per il gruppo e tutti gli altri utenti del sistema (l'eseguibilità deve rimanere per il proprietario del file)

chmod go-x pippo

1

ESERCIZIO 1 (cont.)

c) indicate il comando necessario per permettere al gruppo di modificare il file

chmod g+w pippo

d) riportare la notazione Unix che rappresenta correttamente i permessi e le protezioni associate a pippo dopo le operazioni dei punti b) e c)

-rwxrw---- pippo

e) qual è il comando Unix normalmente usato per visualizzare i permessi e le protezioni associate ad un file?

ls -l

2

ESERCIZIO 2

Nel seguito, assumiamo che tutti i comandi indicati vengano correttamente eseguiti senza errore

a) All'interno di una directory vuota (di nome dir1), viene dato il comando "mkdir new-folder". Disegnare la struttura interna nella cartella dir1 dopo questa operazione. (scegliete voi dei valori adeguati per gli i-node)

dir1	
45	.
40	. .
70	new-folder

3

ESERCIZIO 2 (cont.)

b) Sempre all'interno di dir1, viene data la seguente sequenza di comandi:

```
cp ../pippo .
ln pippo pluto
rm pippo
```

come è fatto il file "dir1" a questo punto?

dir1

45	.
40	. .
70	new-folder
0	pippo
80	pluto

4

ESERCIZIO 3

a) Qual è l'effetto del seguente comando Unix?

```
ps agx | grep gcc | wc -l > xyz
```

conta il numero di istanze attive nel sistema del compilatore gcc e mette il risultato nel file xyz

b) In questo contesto, qual è la funzione dei "caratteri" "|" e ">"?

sono i metacaratteri "pipe" e "redirezione dell'output".

"|" Mette in comunicazione l'output di un comando con l'input del comando successivo. ">" redirige l'output di un comando sul file specificato, creandolo se non esiste

5

ESERCIZIO 4

a) qual è la differenza tra i seguenti due comandi?

```
1) gcc myfile.c
2) gcc myfile.c &
```

2) è un comando eseguito in background. Mentre è in esecuzione l'utente può continuare ad usare il terminale. 1) è stato lanciato in foreground, per cui viene restituito il controllo del terminale solo alla terminazione del comando stesso.

b) come si può "trasformare" il comando 1), mentre questo è in esecuzione, nel comando 2)?

ctrl-Z (oppure inviando al processo il segnale SIGSTOP)

bg

6

ESERCIZIO 5

Scrivete uno script shell che ha tre argomenti di input. I primi due sono file che esistono. Il terzo argomento è il nome di un file che deve essere creato e deve essere formato dalle prime righe del primo file (ad esempio le prime 10) e dalle ultime righe del secondo file (ad esempio le ultime 10). Non devono essere usati file temporanei.

shell-script:

```
head $1 > $3
tail $2 >> $3
```

7

ESERCIZIO 6

si consideri il seguente programma C:

```
1 main()
2  {
3  int n, i;
4  i = 5;
5  n = fork();
6  if ( n != 0) { i = i+1;
7                 printf("%d",i);}
8  else printf("%d",i);
9  }
```

a) qual è il valore della variabile i stampato alle righe 7 e 8? Perché?

alla riga 7 i vale 6, e alla riga 8 i vale 5. Il processo figlio eredita una copia delle variabili del padre, e quindi anche la variabile i con il suo valore. Modifiche fatte dal processo padre non hanno però effetto sulle variabili del figlio ereditate dal padre.

8

ESERCIZIO 6 (cont.)

b) che cosa fa la system call fork? In che senso possiamo dire che "restituisce due valori"? Che valori?

la fork genera un processo figlio che incomincia ad eseguire il codice a partire dall'istruzione successiva alla fork. Il valore restituito dalla fork è uguale a 0 per il processo figlio, ed è uguale al PID del figlio per il processo padre.

c) nel codice dato, possiamo dire in che ordine verranno eseguite le stampe delle righe 7 e 8?

no, poiché è codice che appartiene a processi concorrenti, l'ordine di esecuzione dipende dall'ordine in cui verranno schedulati dallo scheduler

9

ESERCIZIO 7

si considerino i seguenti programmi C:

P1:

```
void pippo()
{printf("ciao!");
  exit()}

1 main()
2  {
3  int i,n;
4  for (i=0; i<5; i++) sleep(1);
5  n = signal(SIGUSR1,pippo);
6  printf("bye bye");
7 for (;;) sleep(1);}
```

P2:

```
1 main(int argc, char *argv[])
2  {
3  if (argc > 1) kill(PID,SIGUSR1)
4  else kill(PID,SIGKILL)
5  }
```

dove PID è il process-id di P1 (si assuma che sia noto a P2 quando questo viene eseguito)

a) a cosa serve la syscall signal?

predispone il processo chiamante a compiere una certa azione se riceve il segnale specificato.

10

ESERCIZIO 7 (cont.)

b) a cosa serve la syscall kill?

invia il segnale specificato al processo il cui PID è indicato nel primo argomento

c) A quali condizioni P1 stampa la scritta "ciao!"?

Se P2 viene lanciato con almeno un argomento e nel momento in cui viene eseguita la kill della riga 3 è già stata eseguita la signal della riga 5 di P1.

d) A quali condizioni P1 stampa la scritta "bye bye"?

Se l'istruzione della riga 3 di P2 viene eseguita dopo la stampa.

e) A quali condizioni P1 stampa sia "ciao!" che "bye bye"?

mai.

11

ESERCIZIO 8

una cartella "mydir" contiene come unico file l'eseguibile "myproc", di cui questo è il corrispondente sorgente:

myproc.c:

```
1 main()
2  {
3  printf("ciao\n");
4  execl("/bin/ls","/bin/ls",0);
5  printf("bye bye\n");
6  }
```

a) qual è l'output prodotto dall'esecuzione di myproc nella cartella mydir se la execl viene correttamente eseguita?

ciao
myproc

12

ESERCIZIO 8 (cont.)

myproc.c:

```
1 main()
2  {
3  printf("ciao\n");
4  execl("/bin/ls","/bin/ls",0);
5  printf("bye bye\n");
6  }
```

b) e se la execl per qualche ragione fallisce (ad esempio ls non è presente nella directory /bin)?

ciao
bye bye

c) cosa fa la syscall execl?
lancia l'eseguibile specificato nell'argomento, e il processo prosegue usando il codice dell'eseguibile, senza ritornare alla porzione di codice che sta sotto la execl stessa.

13

ESERCIZIO 9

a) Spiegate in che contesto vengono usati e quali sono i vantaggi e gli svantaggi nell'uso dei blocchi marcati "delayed write"

Nel contesto del buffer cache. Sono blocchi di file in ram che sono stati modificati e che devono essere ancora salvati in memoria secondaria. Il SO cerca di ritardarne al massimo il salvataggio in modo da evitare operazioni su disco inutili. Se però manca la corrente prima che il salvataggio sia stato effettuato possono venire persi dei dati.

b) Poichè il buffer cache ruba spazio in ram, potrebbe essere conveniente implementarlo usando la memoria virtuale?

No. vorrebbe dire permettere che alcuni dei buffer in ram siano in realtà tenuti in memoria secondaria. Ma l'obiettivo del buffer cache è proprio quello di limitare in numero di accessi in memoria secondaria.

14

ESERCIZIO 10

a) Che cosa indica il link counter di un i-node? Se leggiamo il link counter di un qualsiasi i-node di un file system, qual è il valore più piccolo che possiamo trovare? E il valore più grande?

Conta il numero di nomi diversi (i link appunto) con cui è conosciuto all'interno del file system il file associato a quell'i-node. 1. Non c'è un limite massimo.

b) Viene dato il comando "rm A". In quale caso i blocchi dei dati del file A vengono efffettivamente rimossi insieme all-i-node di A? (supponendo ovviamente che si abbiano i permessi per rimuovere A)

Quando il link-counter dell'i-node di A vale 1, e viene quindi portato a zero in seguito al comando rm. Non esistono più altri (hard) link al file nel sistema, e il file può essere definitivamente rimosso.

15

ESERCIZIO 11

a) In un sistema sappiamo che alcuni utenti stanno attualmente lanciando il compilatore gcc. Cos'è che ci dice quante istanze del compilatore sono attive in un dato istante? Quante copie dell'eseguibile gcc saranno presenti in RAM? Perché?

il reference-count dell'in-core i-node del file gcc. Sempre solo una copia di un file eseguibile viene portata in RAM, poiché questo è un file di sola lettura che può essere usato da più processi contemporaneamente.

b) In quale caso il codice di gcc non viene rimosso dalla RAM anche se nessuno sta compilando?

se il file che contiene gcc ha lo sticky bit settato.

16

ESERCIZIO 12

a) Nell-i-node di un file A il link-counter vale 3. Viene dato il comando: ln -s A B (B diviene un link simbolico ad A). Che valore assume il link-counter del file?

Rimane a 3, dato che i link simbolici non modificano il link counter.

b) Viene dato il comando "rm A". Cosa succede di B?

B fa riferimento ad un pathname (A) che non esiste più, per cui qualsiasi riferimento a B darà un errore del tipo "borken link".

Notate che il file (di cui A era un hard link) non è stato rimosso, perché il suo link-counter ha ancora valore 2 dopo l'esecuzione del comando rm.

17

ESERCIZIO 13

In un sistema Unix i blocchi dell'hard disk sono grandi 512 byte, e vengono usati due byte per scrivere il numero di un blocco. Un file A è lungo 5140 byte.

- La lettura del byte 1000 richiede più, meno, o lo stesso tempo della lettura del byte 5135 di A (motivate la risposta, assumendo che non sia presente il buffer cache).

  meno tempo. Infatti basta leggere nell-i-node il numero del secondo blocco di dati del file, che contiene il byte 1000, e poi portare in ram il blocco stesso.

  Il byte 5135 è memorizzato nel primo blocco ad indirizzamento indiretto. Occorre quindi leggere il numero dell'undicesimo blocco puntato dall'i-node, (il puntatore *single-indirect*), in modo da poter leggere in ram tale blocco. Al suo interno, il primo puntatore è il numero del blocco che contiene il byte 5135, che deve essere portato in ram per essere letto.

18

ESERCIZIO 13(cont.)

b) Qual è la frammentazione interna prodotta da A?
L'ultimo blocco del file contiene solo 20 byte, per cui 492 byte rimangono inutilizzati. Inoltre, il blocco a singola indirezione contiene solo un puntatore (che punta all'ultimo blocco del file) per cui altri 512 - 2 byte = 510 byte rimangono inutilizzati.

c) Quanti blocchi diversi possono essere indirizzati usando 2 byte?
2^{16} blocchi diversi.

d) Qual è la massima dimensione di un file in questo sistema?
$512 \times 10 + 512 \times 256 + 512 \times 256^2 + 512 \times 256^3$ byte

19

IN DEFINITIVA:

a) se avete fatto e capito gli esercizi sui comandi unix, e se vi siete abituati ad usarli per lavorare sul sistema quando sviluppate i programmi in C,

b) Se avete capito come funzionano le principali system call (fork, wait, signal, kill, execl) e avete fatto gli esercizi che abbiamo dato durante il corso,

non avrete problemi a risolvere gli esercizi dell'esame.

ATTENZIONE: non ci saranno domande/esercizi sull'IPC, ma ci potranno essere domande sulle cose viste in aula sul kernel Unix, ad esempio: come funziona un file di tipo pipe, che cosa è una user file table, cosa è il set-uid bit, quali regioni di un processo possono essere condivise, come viene calcolata la priorità di un processo unix, e così via.

20