Relazione progetto Sistemi Operativi
Anno Accademico 2012-2013

Giuliano Tortoreto (152183)
Andrea Gottardi (151955)

Il progetto scelto riguarda l'implementazione di alcune utility di GNU/Linux. Nell'archivio oggetto di consegna sono presenti le cartelle delle utility e il Makefile, la cui esecuzione porta alla creazione di una cartella **/files**, all'interno della quale sono presenti i file eseguibili compilati.
È anche possibile compilare solo un programma, attraverso il comando **make** nome_programma, inoltre con **make** clean è possibile pulire la cartella da file compilati e da file .o.

Le cartelle presenti nel programma sono le seguenti:

- **mkbkp**: gestione della utility omonima di creazione archivio
- **equal**: gestione della utility omonima di confronto file
- **plive**: gestione della utility omonima di visualizzazione processi
- **managelogs**: gestione della scrittura dei log che le utility producono.

Di seguito una descrizione delle varie utility implementate, comprensiva di motivazioni riguardanti azioni non specificate espressamente nella consegna, ma inserite per una maggiore usabilità.

- **MKBKP**

Questa utility, dati un nome di archivio e una serie di path, effettua la creazione di un archivio. L'archivio viene salvato come un file binario, strutturato nella seguente maniera:

%LIST% file1 file2 file3 %LIST%
%DIRS% dir1 dir2 dir3 %DIRS%
%CONTENT% contenuto file 1 %CONTENT%
%CONTENT% contenuto file 2 %CONTENT%
%CONTENT% contenuto file 3 %CONTENT%

In **%LIST%** è salvata la lista dei file contenuti nei path dati, in **%DIRS%** sono contenute le cartelle che sono contenute nei path dati, e in **%CONTENT%** sono elencati, uno per uno, i contenuti dei vari file.

Questa funzione archivia, lista ed estrae correttamente sia file che cartelle contenenti file di ogni tipo, che vengono sempre interpretati come file binari. E' stata gestita anche l'eventualità che i file o le cartelle passate in argomento abbiano un nome composto da più parole. La funzione ovvia a questo problema sostituendo gli spazi con dei trattini (-): ad esempio, il file "Senza Nome.doc" viene convertito in "Senza-Nome.doc".

Alcuni limiti nell'esecuzione sono stati identificati nella chiamata della funzione utilizzando più opzioni unite: ad esempio, chiamando la funzione con le opzioni **"-fc"** anzichè **"-f -c"** il programma si blocca perchè viene alterato l'ordine di valutazione dei parametri inseriti.

Le system call utilizzate in questo programma sono:

- **fopen**(), per aprire i file;
- **fclose**(), per chiudere i file precedentemente aperti
- **fgetc**(), per leggere i file byte per byte
- **fscanf**(), per leggere i file basandosi su placeholder.
- **access**(), per testare la possibilità di accedere a un file
- **opendir**(), per aprire le directory o testarne l'esistenza dato il path
- **fprintf**(), per scrivere su un file
- **ftw**(), per scorrere l'albero della cartella, sottocartelle e file compresi.
- **getopt**(), permette una gestione più rapida dei parametri che l'utente passa al programma che vuole eseguire;

Utilizzo:

creazione: (path)/mkbkp -f -c nome_arch path1 path2 ...

estrazione: (path)/mkbkp -f -x nome_arch
lista file: (path)/mkbkp -f -t nome_arch

PLIVE

Questa utility, quando viene chiamata, visualizza la lista dei processi che sono maggiormente attivi a livello di utilizzo della CPU. Per far ciò ci siamo avvalsi dell'utilizzo della cartella **/proc/,** all'interno della quale sono presenti tutte i processi, ognuno dei quali ha una cartella dedicata. Leggendo il file **/proc/stat** è possibile ricavare il tempo che la CPU ha passato "lavorando" fino al momento della chiamata. Per i vari processi, invece, è stato sufficiente leggere per ognuno di essi il file **/proc/<pid>/stat** per trovare il tempo di CPU che il processo ha usato dal momento in cui è stato lanciato. Tuttavia, avendo questi dati, è impossibile sapere la percentuale attuale di utilizzo della CPU. E' quindi necessario procedere facendo passare qualche secondo (l'entità dell'attesa varia a seconda di quanto decide l'utente) ed eseguire nuovamente questi calcoli, memorizzando la differenza di utilizzo sia della CPU, che dei processi. In questo modo, facendo il rapporto tra la differenza della CPU e la differenza del processo, è possibile trovare la percentuale di utilizzo della CPU di un dato processo nel tempo definito.

Per poter ricevere l'input da parte dell'utente, senza mettere in attesa il programma, è stato necessario sviluppare una funzione kbhit() che modifica le impostazioni del terminale, in modo che i caratteri inseriti vengano letti direttamente, senza passare per un buffer. In questo modo il programma procede normalmente, modificando il suo comportamento solo nell'eventualità in cui l'utente digiti un carattere.

Le system call utilizzate in questo programma sono:

- **fopen**(), per aprire i file;
- **fclose**(), per chiudere i file precedentemente aperti
- **fgetc**(), per leggere i file byte per byte
- **fscanf**(), per leggere i file basandosi su placeholder.
- **access**(), per testare la possibilità di accedere a un file
- **opendir**(), per aprire le directory o testarne l'esistenza dato il path
- **readdir**(), per leggere il contenuto della directory;

- **fprintf**(), per stampare su un file;
- **sprintf**(), per scrivere su una stringa sulla base di placeholders.
- **getopt**(), permette una gestione più rapida dei parametri che l'utente passa al programma che vuole eseguire;

Utilizzo:

esecuzione default: (path)/plive
esecuzione con n: (path)/plive -n numeroprocessi

- ### Equal

Questa utility verifica se 2 file /cartelle sono uguali altrimenti stampa le differenze.
Il caso in cui viene passata alla funzione una cartella e un file, risponde false in quanto sono di tipo diverso.
In caso siano 2 file, la funzione verifica **byte** per **byte** se sono uguali, se si, stampa 0, altrimenti stampa 1 e la posizione del primo byte a cui vengono trovate le differenze e fra parentesi la posizione dei **bit** in cui differisce, nel file di **log** invece saranno elencate **tutte** le posizioni in cui byte differiscono e fra parentesi quali bit sono diversi.

Nel caso siano cartelle, abbiamo deciso di scorrere separatamente i due path e salvare per ognuno tutti i suoi sottopercorsi(relativi) in un vettore di stringhe. Dopo di che i 2 vettori vengono ordinati in modo che sia più facile verificarne l'uguaglianza.
Una volta fatto questo, viene chiamata una funzione che verifica se i due vettori di stringhe contengono le stesse stringhe.
Quando vengono trovate stringhe uguali, si verifica se sono cartelle o file.
Se sono cartelle, passo agli elementi successivi, altrimenti chiamo l'apposita funzione per i file usata sopra e stampo a video e nel file di log le differenze, se ce ne sono.
Nel caso siano stringhe diverse allora stampo a video l'elemento minore, che sicuramente non può essere contenuto nelle posizioni successive dell'altro vettore. Quindi vado avanti con l'indice dove ho trovato l'elemento minore.
(Se l'elemento apparterrà solo al primo path sarà preceduto da "<<" altrimenti da " >>>>")
Ripeto questa operazione per tutti gli elementi dell'array, alla fine stampo gli elementi in più, che non sono ancora stati visitati dall'array perchè un path potrebbe contenere meno elementi dell'altro.
Se scorrendo il vettore di array non trovo mai neanche un elemento diverso alla fine stampo "0", altrimenti al primo elemento diverso stampo a schermo "1".

Le system call utilizzate in questo programma sono:

- **fopen**(), per aprire i file;

- **fclose**(), per chiudere i file precedentemente aperti
- **getc**(), per leggere i file byte per byte
- **opendir**(), per aprire le directory o testarne l'esistenza dato il path
- **readdir**(), per leggere il contenuto della directory
- **closedir**(), per chiudere le directory
- **access**(), per verificare se un path esiste

Utilizzo:

esecuzione default: (path)/equal path1 path2

- ### ManageLogs

Dato che per ogni utility è necessario un file di log abbiamo creato una cartella managelogs che contiene managelogs.h e managelogs.c qui ci sono le funzioni necessarie per aprire un file di log o (crearlo se inesistente), di scrivere e chiudere il file di log. Nel file .h ci sono gli header e nel .c l'implementazione.

NOTE:

- i file di Log vengono creati solo se si è sudoers, altrimenti il programma non li crea e ogni volta che viene chiamato dice di creare il file di Log, ma in realtà non lo fa;