Relazione laboratorio bush--

Relazione a cura di:
Brandi Eugenio S4064172
Nolberto Felix Cruces Hidalgo S4056658
Pellaco Francesco S4228743

Funzionamento generale della bush --

Lo scopo della `bush--` è quello di simulare le funzioni base di una shell linux, ossia l'esecuzione di alcuni comandi.
I comandi accettati dalla `bush--` sono di due tipologie:

- i comandi esterni alla `bush--` stessa, i cui quali files eseguibili risiedono nei percorsi presenti nella variabile di sistema path;
- i comandi interni alla `bush--` stessa come : `@cd` per cambiare cartella, `@show-variables` per la stampa a video delle variabili di sistema, `@quit` per terminare l'esecuzione della bush--.

Una volta avviata, la `bush`-- resterà in attesa di un comando da input; una volta ricevuto procederà con la sua esecuzione o con una segnalazione di errore.
Per riconoscere i comandi e verificarne la correttezza, la bush-- effettua un'analisi delle stringhe inserite attraverso un parsing. Estraendo dalla stringa i vari token che la compongono, la `bush--` ne verifica la compatibilità con le grammatiche memorizzate al suo interno e crea una albero utile per l'esecuzione di comandi. In caso la stringa sia riconosciuta, la `bush--` provvede all'esecuzione del comando e alla sua gestione. Alla fine dell'esecuzione del comando la `bush--` si rimette in attesa. Nel caso un token non venga riconosciuto come parte del linguaggio, la `bush--` segnala a video un messaggio di errore e si rimette in attesa.
Per terminare l'esecuzione si deve digitare il comando `@quit`.

Descrizione del nostro operato :

- ***var_table.c***

vt_to_envp(...)

Questa funzione, deve copiare il contenuto di una struttura (composta da due array di caratteri) in una variabile char**. Per allocare la variabile necessaria, è stata usata 2 volte la funzione `malloc()` nel seguente modo:

- la prima volta, per creare un vettore di puntatori a char passando alla funzione la dimensione moltiplicata per la costante `sizeof(char*);`
- la seconda volta, per allocare lo spazio necessario alla memorizzazione della stringa, passando alla funzione la dimensione delle due stringhe da copiare più la costante +2 necessaria per aggiungere il carattere '=' e '\0'.

Attraverso le funzioni `strcpy()` e `strcat()` abbiamo settato la variabile prima inizializzata nel modo voluto. Come richiesto dalle specifiche, abbiamo settato l'ultimo valore dell'array a NULL.

- **ast.c**

cd_execute(...)

Questa funzione esegue il comando `@cd`, che ci sposta nella cartella indicata. In base al path specificato avremo i seguenti comportamenti:

- path == NULL -> spostamento in una cartella di default;
- path != NULL -> spostamento nella cartella specificata.

Per poter eseguire queste due azioni, per prima cosa si è analizzato il contenuto del path passato:
nel caso in cui fosse stato pari a NULL, tramite la funzione `vt_lookup()`, si sarebbe settato con la destinazione di default HOME; se diverso sarebbe rimasto invariato.
A questo punto, attraverso la syscall `chdir()` viene eseguito il comando specificato. In caso di fallimento della syscall, viene segnalato a video un messaggio di errore senza interruzione del codice. Dopo di che, tramite la funzione `vt_set_value()`, abbiamo aggiornato il contenuto della variabile PWD. Per aggiornare il suo contenuto, abbiamo usato la syscall `getcwd()` passandogli NULL e 0. Così facendo la syscall gestisce da sola l'allocazione del vettore di caratteri necessario.

find_in_path(...)

Questa funzione analizza la variabile `name` passata e agisce nei seguenti due modi :

- se la variabile rappresenta un percorso, la funzione ritorna una copia della variabile stessa;
- se la variabile rappresenta il nome di un file, la funzione cerca la cartella in cui risiede il file e ne ritorna il path.

Per poter eseguire queste due azioni, per prima cosa si è analizzato il contenuto della variabile `name` attraverso la funzione `strchr()`.
Per capire se la variabile contiene un percorso o meno, si è specificato alla funzione il delimitatore '/' .
Nel caso in cui la variabile rappresenti un percorso, ritorna la copia della variabile `name` attraverso la funzione `strdup()`; in caso contrario, specificandogli come delimitatore ":", abbiamo estratto i vari path dalla variabile `likePath`.
Tramite le funzioni `strncpy()` e `strcat()`, si è costruita, per ogni path estratto, la stringa da analizzare tramite la syscall `access()`, che specificando la modalità `F_OK`, verifica l'esistenza del file. In caso affermativo viene restituita la stringa creata, altrimenti si procede all'analisi degli altri path estratti.

ext_cmd_execute(...)

Per eseguire il comando specificato, si sono usate la syscall `execve()` e la syscall `fork()`. Poiché in caso di successo della syscall `execve()` si interrompe l'esecuzione del codice, l'altra syscall è necessaria per creare un processo figlio al quale affidare il compito dell'esecuzione della `execve()`.
In questo modo, alla fine dell'esecuzione del `execve()` il processo figlio termina e il processo padre continua la sua normale esecuzione.

pipe_execute(...)

Questa funzione, esegue il comando della pipe e quindi gestisce la redirezione dell'input e dell'output.
Per gestire la redirezione dell'input e dell'output, è stata utilizzata la syscall `pipe()`. Attraverso la syscall `fcntl()` abbiamo cambiato alcuni parametri dei due file descriptor appena creati, specificando il comando `F_SETFD` e l'argomento `FD_CLOEXEC` settando così la chiusura automatica dei file descriptor dopo l'esecuzione di una `exec()`.
Attraverso la chiamata ricorsiva, abbiamo esplorato l'albero creato ed eseguito i vari comandi reindirizzando i vari input ed output.

redirect_fd(...)

Questa funzione, controlla se la variabile `from_fd` e' diversa dalla costante `NO_REDIR`; in caso affermativo, tramite la syscall `dup2()` duplica il file descriptor `from_fd` creandone uno nuovo associandolo alla variabile `to_fd`; altrimenti si salta questo passaggio.
Se la syscall non riesce a duplicare il file descriptor, genera un errore che provoca l'interruzione del programma.

- **shell.c**

free_envp(...)

Con questa funzione, dealloca un vettore di puntatori a `char`.
Per fare ciò, si è usata la funzione `free()` in un ciclo per deallocare le singole celle del vettore, dopodiché è stata richiamata per deallocare il vettore stesso.

wait_for_termination_of_children(...)

Questa funzione blocca il processo padre sino a che i processi figli creati non hanno finito l'esecuzione del comando.
Per fare ciò, si è usata la syscall `wait()`. Questa syscall e' stata inserita in un ciclo poiché era necessario attendere che tutti i processi figli terminassero prima di riprendere la normale esecuzione del processo padre. Come condizione di uscita del ciclo, si e' usato il codice di errore `ECHILD`, poiché questo errore, indica che non ci sono più figli da attendere. Non appena la variabile `errno` e' uguale a `ECHILD` siamo sicuri che tutti i figli in esecuzione sono terminati.

- **bmm.c**

make_sure_PWD_is_set(...)

Nel caso in cui la variabile d'ambiente `PWD` non sia inizializzata, si deve inizializzarla.
Per fare ciò, abbiamo usato la syscall `getcwd()`, la quale ci restituisce il path della cartella in cui siamo.
Dato che non conoscevamo la dimensione del path, abbiamo specificato alla syscall NULL e 0.
Così facendo la syscall si gestisce da sola l'allocazione dell'array di caratteri necessario.
Ottenuta l'informazione, tramite la funzione `vt_set_value()`, abbiamo aggiornato la variabile `PWD`.