CUSTOM SHELL

Anno accademico 2017/2018 – Università degli studi di Trento
Progetto Laboratorio Sistemi Operativi 2018 – Custom Shell
Realizzato da: Tommaso Bosetti 185286 – Sebastiano Chiari 185858 – Marta Toniolli 187839

Task

Lo scopo del progetto era quello di realizzare una mini shell interattiva che supportasse l'utilizzo di comandi basilari (come per esempio **`ls`**, **`wc`** e **`date`**) ed accettasse il carattere di piping. Una livello più elevato di implementazione comprendeva il supporto di comandi con parametri avanzati. L'output e/o l'error dei comandi devono essere salvati all'interno di file di "log", gestiti separatamente oppure come un unico file in base alla scelta dell'utente.

Struttura

Il nostro progetto è organizzato in diversi files:

- "**`Makefile.makefile`**", file make per avviare la compilazione dei file sorgente, con una regola di default che descrive brevemente le altre regole del makefile, una regola '**`build`**' per la compilazione e una regola '**`clean`**' per l'eliminazione dell'eseguibile e dei file di "log" di default (nel caso fossero stati precedentemente creati)
- "**`main.c`**", file sorgente di avvio del programma nel quale sono incluse le librerie e i `file.h` contenenti le funzioni di inizializzazione e la logica di esecuzione dei comandi. Qui si richiama la funzione di creazione dei file (di default o personalizzati) e si avvia il prompt, dentro il quale viene utilizzata una funzione di lettura interattiva dei comandi ed eventuali parametri
- "**`init.c`**" (e il relativo file header "**`init.h`**"), definisce le variabili globali e le macro. In questo file troviamo le funzioni di creazione, inizializzazione e gestione delle strutture utilizzate dalla mini shell (per esempio, file di "log")
- "**`functions.c`**" (e il relativo file header "**`functions.h`**"), contiene tutte le funzioni che sono richiamate durante l'esecuzione del prompt per eseguire i comandi passati dall'utente e salvare gli output dei comandi sui file di log di output, error o file unico
- "**`utilities.c`**" (e il relativo file header "**`utilities.h`**"), contiene le funzioni di servizio per l'inizializzazione dei flag e per la gestione delle funzionalità aggiuntive durante l'esecuzione del prompt, come ad esempio il cambio di livello o l'help

I comandi digitati dall'utente vengono eseguiti in modo interattivo, richiamando nella maggior parte dei casi una system call, contenente come argomento il valore passato dalla shell. I comandi particolari ed avanzati vengono trattati in modo leggermente diverso (help, quit, cd, pico, changeLevel, vim).

I file di log sono salvati nella stessa cartella del makefile. L'eseguibile generato dalla regola build compila i **`file.c`** e **`.h`** (inclusi in "**`main.c`**"), posti all'interno della cartella **`src`**, creando una cartella **`bin`** contenente l'eseguibile denominato "**`shell`**".

Problemi riscontrati

Avendo studiato come linguaggio principale **C++**, abbiamo subito notato sostanziali differenze con il **C**, come ad esempio l'utilizzo di flag di interi per la simulazione di variabili booleane, l'utilizzo delle stringhe e le modalità di allocazione dinamica di memoria.

Un'altra delle criticità riscontrata è stata il mal funzionamento del comando "**cd**" tramite system call; nonostante sembri funzionare stampando correttamente sul file di output, scrive anche una stringa di errore all'interno del file di error.

Annotazioni

Se, durante l'esecuzione della custom shell, l'utente elimina uno dei file di log, l'errore verrà gestito.

Abbiamo inoltre realizzato la possibilità per l'utente di cambiare il livello di specifiche dell'output direttamente a livello di esecuzione o a run time.

Eseguendo la regola clean vengono eliminati solamente l'eseguibile generato dalla regola build nella cartella **bin** e i file di "log" di default. Eventuali file di "log" personalizzati NON verranno eliminati. L'utente dovrà eventualmente intervenire manualmente.