Progetto di Sistemi Operativi Modulo II (Canale 1) A.A 2011/2012

Scelte progettuali:

Nelle specifiche del progetto, si chiede di definire tre programmi per la gestione di un sistema di esecuzione di comandi in ambiente controllato. Non essendo a conoscenza dei possibili comandi che il nostro sistema dovrà processare, l'idea seguita è quella di creare una gabbia contenente i file binari e le librerie presenti in */bin, /lib /usr/bin, /usr/lib, /lib64*.
Questo compito è svolto dall'executor, che acquisisce come primo parametro una directory (`jail_dir`), controlla se è la root directory e in caso negativo monta (in sola lettura) nella gabbia */bin, /lib /usr/bin, /usr/lib, /lib64*.
Oltre a svolgere i comandi a disposizione in *bin, usr/bin, ./*, l'executor è in grado di eseguire dei comandi interni, definiti in `comandi.c`. I comandi **`shutdown`** e **`abort`** sono stati implementati direttamente in `executor.c`, e non in `comandi.c`, in quanto sono strettamente legati all'executor. Shutdown si occupa di terminare l'executor correttamente, smontando le cartelle qualora fossero state montate, rimuovendo la FIFO e chiudendo i file descriptor aperti. Abort esegue le stesse operazioni, ma termina con errore. Questa funzione viene utilizzata dall'executor stesso anche in caso di errori durante la sua esecuzione.
L'executor si occupa di creare una FIFO (`fifo_path`) su cui il requester scriverà i comandi richiesti, questo perché nella realtà potremmo avere più requester che scrivono sulla stessa FIFO per essere serviti da un executor.

La notifica al monitor di un errore o della corretta esecuzione di un comando da parte dell'executor è stata eseguita tramite l'invio di due segnali:

-**`SIGUSR1`** per segnalare un errore,
-**`SIGUSR2`** per segnalare la corretta esecuzione.

Con i segnali **`SIGINT`** o **`SIGILL`** l'executor termina rispettivamente correttamente o con errore.

Il requester acquisisce come unico parametro il path della FIFO su cui verrano scritti i comandi da eseguire che saranno letti da stdin. Questo termina alla ricezione del segnale **`SIGINT`** (es. da parte dell'utente), o di **`SIGPIPE`** quando l'executor rimuove la FIFO.

Il monitor avvia l'executor che deve trovarsi nella sua stessa directory e rimane in attesa dei segnali da parte dell'executor. Quando executor sarà terminato, il monitor si occuperà di scrivere il file di log.

Lo script `monitor-wrapper.sh`, individuerà per una profondità massima di tre directory, tutte le directory con nome *LabSOMod2Canale1*, e se presente, avvierà il monitor arrestando la sua ricerca.

Contributi ed organizzazioni:

- **Lista dei membri del gruppo:**
  - Domenico Citera – Matricola: 1186200

- **Istruzioni per la comilazione ed esecuzione dei programmi:**

  Per eseguire il programma digitare in ordine i seguenti comandi:

  - **`make`** per compilare il tutto
  - **`make install`** per l'installazione nella cartella **`./`**
  - **`make clean`** per cancellare i file oggetto nella cartella locale (`objfile`)
  - **`make cleanall`** per cancellare i file oggetto e gli eseguibili nella cartella locale (`objfile, binfile`)
  - **`make uninstall`** per disinstallare il tutto.

  Gli eseguibili possono essere avviati con

  ```
  ./monitor&
  ./requester fifo
  ```

  dove `fifo` è il path della fifo che creerà l'executor, definito in `executor.h` (appunto `fifo`).

- **Esempi di test:**
  - **`test1.sh`** – avvia il monitor in bg, attende due secondi, e avvia il requester leggendo i comandi dal file `com1.txt`. I comandi da eseguire sono: **`ls -l, ls, ps, date, dir, whoami, echo`** Ciao mondo (comando interno)*,* **`shutdown`***.* Quindi il monitor termina correttamente.

  - **`test2.sh`** – avvia il monitor in bg, attende due secondi, e avvia il requester leggendo i comandi dal file `com2.txt.` I comandi da eseguire sono: **`help`** (comando interno), **`comando`** *non valido,* **`pwd`** (comando interno), **`mkdir`** *prova,* **`rmdir`** *prova,* **`abort`***.* Quindi il monitor termina con errore.

  - **`test3.sh –`** avvia il monitor in bg, attende due secondi e invia all'executor il segnale **`SIGINT`**. Quindi il monitor termina correttamente.

  - **`test4.sh –`** avvia il monitor in bg, attende due secondi e invia all'executor il segnale **`SIGILL`**. Quindi il monitor termina con errore.