# Corso di Sistemi Operativi - Progetto

Studenti: Degiacomi Simone, Faieta Alessio e Simone Mattia
Id gruppo: LABSO1-2017_2018--187538_187899_187906

In questo documento è presente una breve descrizione dell'implementazione del progetto (Progetto 2 - logger) del corso di Sistemi Operativi.

---

Il tool sviluppato si compone principalmente di quattro fasi di esecuzione:

1. Controllo dell'esistenza di un istanza del demone e eventuale avvio;
2. Parsing del comando da eseguire;
3. Esecuzione del comando
4. Stampa dei risultati;

Il tool, attraverso delle opzioni da riga di comando permette di scegliere il percorso del file di log, il formato, operazioni di controllo del demone e una pagina di aiuto.

## Demone

Tutte le scritture su file dei log, indipendentemente dal formato, sono eseguite da un demone, ovvero un processo eseguito in background.
Il demone si comporta quasi come un server e quindi dovrà gestire le richieste di diverse esecuzioni del tool, che potranno avvenire anche contemporaneamente.

Ad ogni avvio del tool viene creato un demone che, come prima cosa, controlla se un'altra istanza del demone è già in esecuzione. In tal caso il nuovo demone termina la sua esecuzione.
Il controllo viene fatto verificando la presenza di un file (posizionato nella cartella /tmp) e di un relativo lock (gestito attraverso la funzione fcntl()) in scrittura su di esso. I lock in scrittura sono associati al processo che li crea, e quindi vengono persi quando il processo muore. Questo garantisce che, anche nel caso in cui il demone si interrompa a causa di un errore e non riesca ad eliminare il file di lock, il nuovo demone riuscirà a identificare che nessun demone è in esecuzione e quindi acquisirà il lock in scrittura.
Nel caso in cui due istanze del demone vengano avviate in parallelo, solo una di esse riuscirà a ottenere il lock sul file, mentre l'altra terminerà.

Quando il tool deve scrivere delle statistiche su un file, esso si prenota inviando un messaggio al demone (utilizzando le message queue) che include la directory attuale del tool e la path in cui mettere il file di log. Dato che più istanze del tool posso esistere

concorrentemente, il demone risponderà ad un tool alla volta, e lo farà tramite un segnale. Il segnale verrà scelto tra i seguenti tre:

- SIGCONT: se è possibile creare il file di log;
- SIGUSR1: se il file di log esiste già e quindi i nuovi dati verranno appesi alla fine del file esistente;
- SIGUSR2: se non è possibile creare il file. In questo caso l'ultimo le operazioni elencate successivamente non verranno eseguite;

Una volta che il tool avrà ricevuto il segnale, esso potrà aprire in scrittura una named pipe (sempre posizionata nella cartella /tmp) creata dal demone. Il demone copierà i dati dalla named pipe al file di log. Una volta completata la scrittura dei dati il demone tornerà in attesa di un nuova prenotazione.

Per la terminazione del demone è possibile inviare un segnale (SIGINT o SIGTERM) attraverso il comando kill o utilizzando un apposito parametro da linea di comando da passare al tool (--stop-daemon).

## Parsing e strutture dati

Il tool permette di eseguire comandi utilizzando una sintassi ripresa dalla shell bash. Questi comandi permettono di far collaborare tra di loro diversi eseguibili, concatenando stream di input e di output e facendo operazioni logiche sui loro exit code.
Per poter implementare un buon numero di operatori e per mantenere un design flessibile (per dare la possibilità di aggiungere nuovi operatori in modo semplice) abbiamo implementato un parser che crea un albero dei comandi.
Ogni eseguibile e quasi ogni operatore corrisponde ad un nodo nell'albero. Questo nodo è di tipo Node, una struct che contiene il tipo di nodo, informazioni dipendenti dal tipo di nodo, stream di quel nodo e statistiche sulla sua esecuzione.
Il parser utilizza un array in cui inserite delle struct che associano delle regular expression al relativo costruttore del nodo. Ricorsivamente per ogni stringa viene fatto scorrere questo vettore e viene chiamata la funzione associata (costruttore) alla prima regular expression che riesce a spezzare la stringa. Nel caso in cui il costruttore sia quello di un eseguibile la ricorsione termina, altrimenti continua.

Alla fine del parsing vengono processati i nodi che corrispondono ad alcuni comandi builtin che non hanno una corrispondenza a un nodo. Esempio di questi comandi sono 'cd' e '>'. In particolare il comando per cambiare directory verrà eliminato nella sua forma di nodo eseguibile e verranno modificati i nodi successivi.

## Esecuzione dell'albero dei comandi

Anche l'esecuzione dei comandi è implementata utilizzando una funzione ricorsiva che va a eseguire tutti i nodi. I nodi sono eseguiti in modo asincrono, per qualsiasi tipo di nodo c'è

una funzione che si occupa di lanciare l'eseguibile e poi di ritornare immediatamente. Una funzione si occuperà poi di attendere la terminazione dei figli per raccogliere i risultati.
Nel caso dell'esecuzione di semplici eseguibili o eseguibili all'interno dell'operatore pipe questi vengono lanciati uno dopo l'altro (con relative fork), mentre nel caso di operatori sincroni (and, or o semicolon)  una funzione dedicata si occupa di analizzare se e quando deve essere eseguito il nodo successivo.
I nodi i cui output dovranno essere concatenati sono associati ad un processo chiamato "Appender", il quale legge gli stream da un nodo all'altro e scrive i dati letti in un unico nodo di uscita. Queste operazioni sono implementate utilizzando delle pipe.

## Stampa dei risultati

Il tool permette di ottenere i log dell'esecuzione in tre diversi formati: file di testo semplice, file csv e una rappresentazione grafica in html.
Anche per questa parte di tool si è puntato ad un design flessibile e quindi è presente una struct chiamata Printer che mima le interfacce dei linguaggi di programmazione a oggetti. Sono poi definite tre istanze di questa struct che specificano diversi puntatori a funzioni: queste definizioni corrispondono alle tre implementazioni.