

# $\mu$ bash: laboratorio di Sistemi di Elaborazione e Trasmissione dell'Informazione (SETI)

a.a. 2020/2021

## Introduzione

Lo scopo di questo laboratorio è implementare una piccola shell, che chiameremo  $\mu$ bash, per prendere familiarità con le *system call* POSIX di base per la gestione dei processi.

La sintassi dei comandi è stata pensata per facilitarne il “*parsing*” e non la comodità d’uso da parte degli utenti. Per esempio, richiediamo che non ci sia nessun *blank* fra il carattere `>` e il nome del file dove redirigere lo standard output di un comando, quindi `ls >foo` è un comando valido, mentre `ls > foo` non lo è (nella vera *bash* entrambe le forme sono equivalenti). Se volete implementare il *parsing* diversamente da come viene suggerito, dovrete motivarne le ragioni; nota bene: cose del tipo “l’ho trovato così (in rete/da un amico/da mio cugggino/...)” e non sono in grado di modificarlo” non sono considerate buone ragioni.

Inoltre, a differenza delle shell comunemente usate, non gestiremo l’espansione dei nomi di file, le sequenze di escape, le stringhe, i gruppi di processi, processi in *foreground/background* e tante altre cose.

## Librerie e strumenti

Dovrete sviluppare un programma in C che sfrutti le *system call* POSIX senza appoggiarsi a nessuna libreria particolare, fatta ovviamente eccezione per la libreria standard e, se volete, GNU Readline (nel vostro *makefile* potete assumere che sia già stata installata, per cui basterà usare `-lreadline`).

Dovrete consegnare solo i sorgenti e *makefile* della vostra implementazione di  $\mu$ bash, creando un archivio zip/tar e facendone l’upload su AulaWeb (in un’apposita consegna che verrà creata). Come dialetto del C potete assumere ISO 2011, con estensioni GNU (ovvero, `-std=gnu11`).

**Consigli** Per facilitare la risoluzione di problemi, ricordatevi di compilare con i simboli per il debug (`-ggdb`) e abilitare ottimizzazioni compatibili con il debugging (`-Og`).

Sono, inoltre, *fortemente consigliati* i flag `-Wall -pedantic -Werror`.

Ricordatevi di controllare sempre il valore di ritorno di ogni funzione/syscall e rilasciare immediatamente le risorse (per esempio, i *file descriptor*) quando non più necessarie. Fate particolare attenzione alla gestione della memoria dinamica, l’uso di strumenti come *address sanitizer* (cioè usare l’opzione `-sanitize=address`) o *valgrind* (se, per qualche ragione, *address sanitizer* non fosse disponibile) è consigliato.

## Descrizione di $\mu$ bash

$\mu$ bash processa i comandi, leggendoli da *standard input*, linea per linea, finché non raggiunge la fine del file (`ctrl-D` da terminale). Prima di leggere una linea, stampa un *prompt* che visualizza la directory corrente, vedete `getcwd(2)`, seguita dalla stringa " \$ ". Di `getcwd` potete usare la versione di glibc (la GNU libc è la libreria C standard sotto Linux), che estende POSIX.1-2001<sup>1</sup>. Per leggere le linee potete usare `fgets(3)` o, volendo offrire funzionalità di editing, GNU `readline(3)`.

Come le shell “vere”,  $\mu$ bash offre sia comandi *built-in* (ma uno solo!), sia la possibilità di eseguire comandi/programmi esterni, passando argomenti e redirigendo I/O in file o *pipe*.

L’unico comando built-in è `cd`, che prende un solo argomento: il *pathname* della directory di destinazione (quindi, a differenza di quello in *bash*, non ci sono argomenti opzionali e non dovete modificare le variabili d’ambiente). Per semplicità, il comando `cd` può essere usato solo come primo e unico comando di una linea, senza nessuna redirezione dell’I/O (nel caso l’utente cerchi di usare redirezioni o usi `cd` in *pipe* con altri comandi, dovete segnalare un errore). Per esempio, sono comandi legali:

- `cd foo`
- `cd /non/importa/se/non/esiste`

---

<sup>1</sup>Scoprite da soli perché dovrete volerlo fare ☺

mentre non lo sono:

- `cd foo >bar` — errore: redirectione con comando `cd`
- `cd /etc | grep pippo` — errore: `cd` usato con altri comandi

Per la sua implementazione, vedete `chdir(2)`.

## Comandi esterni

Tutte le linee, non vuote, vengono suddivise in una sequenza di comandi separati dal carattere pipe (|):  $l = c_1 | c_2 | \dots | c_n$ . Ovviamente, nel caso  $n = 1$  non ci sarà nessun separatore. Per processare in questo modo le stringhe, vedete `strtok_r(3)`<sup>2</sup>.

Dopo aver separato  $l$  in una sequenza di comandi, dovreste separare ogni comando  $c$  in una sequenza di argomenti separati da blank (spazi o tab):  $c = a_1 a_2 \dots a_k$ . A questo punto, se un certo  $a_j$  inizia con il carattere...

- dollaro (\$), allora va sostituito con il valore della variabile d'ambiente corrispondente. Per esempio, se un argomento fosse `$foo`, andrebbe sostituito con il valore della variabile d'ambiente `foo`, si veda `getenv(3)`. Segnalate un errore se la variabile non esiste.
- minore (<), allora va tolto dalla lista degli argomenti e considerato una redirectione dello *standard input*. Per esempio, se un argomento fosse `<foo` (notare l'assenza di spazi fra `<` e `foo`), per l'esecuzione del comando corrispondente lo standard input dovrebbe corrispondere al file `foo`. È un errore specificare più di una redirectione dell'input per ogni comando. Per la redirectione vedete `open(2)`, `dup/dup2(2)` e `close(2)`.
- maggiore (>), allora va tolto dalla lista degli argomenti e considerato una redirectione dello *standard output*. Per esempio, se un argomento fosse `>foo`, per l'esecuzione del comando corrispondente lo standard output dovrebbe corrispondere al file `foo`. È un errore specificare più di una redirectione dell'output per ogni comando.

In una sequenza di comandi, solo il primo comando può redirigere lo standard input e solo l'ultimo comando può redirigere lo standard output (e nessuno è costretto a farlo). Ovviamente, se  $n = 1$  il singolo comando può redirigere entrambi.

Per tutti i comandi da  $c_2$  a  $c_n$ , lo standard input di  $c_i$  deve corrispondere allo standard output di  $c_{i-1}$ , si veda `pipe(2)`.

Dopo aver rimosso le redirectioni, si considerano gli argomenti rimanenti:  $a'_1 a'_2 \dots a'_x$ . Deve essere  $0 < x \leq k$ , altrimenti, se  $x = 0$ , vuol dire che in  $c$  non è stato specificato nessun vero comando, ma solo redirectioni.

A questo punto,  $a'_1$  è il nome del file da eseguire e  $a'_2 \dots a'_x$  i suoi argomenti. Ricordate che, per convenzione, `argv[0]=a'_1`, `argv[1]=a'_2`, etc. Per eseguire i comandi esterni vedete `exec(3)` (ovvero, i *wrapper* di `execve(2)`), ricordando che i comandi vanno eseguiti in processi figli, si veda `fork(2)`.

Dopo aver eseguito i comandi specificati in una linea, aspettate la terminazione di tutti i processi figli, si veda `wait(2)`, segnalando se un processo termina con uno *status* diverso da 0 (usare `WIFEXITED` e `WEXITSTATUS`), oppure è stato ucciso da un segnale (usare `WIFSIGNALED` e `WTERMSIG`).

## Esempi

Alcuni esempi di linee che la `μbash` deve poter eseguire:

- `cd foo` — cambia la directory di lavoro
- `ls -l | grep foo >bar` — filtra l'elenco dei file tenendo solo le linee che contengono la stringa "foo" e scrive il risultato nel file "bar"
- `cat /proc/cpuinfo | grep processor | wc -l` — conta il numero di processori (core) presenti nel sistema
- `cat </proc/cpuinfo | grep processor | wc -l` — come il precedente, ma stavolta `cat` legge da standard input (che è stato rediretto)

E alcuni esempi di linee sbagliate (si deve segnalare un errore di "parsing"):

- `cd foo bar` — errore: il comando `cd` ha un solo argomento
- `cd foo <bar` — errore: il comando `cd` non supporta la redirectione
- `ls | cd foo` — errore: il comando `cd` deve essere usato da solo

---

<sup>2</sup>Con `strtok_r` una sequenza di separatori viene trattata come un singolo separatore, ignorate pure questo problema.

- `ls -l | grep foo > bar` — errore: non è specificato il file per la redirectione dello standard output (c'è uno spazio fra `>` e `bar`)
- `cat /proc/cpuinfo | | grep processor | wc -l` — errore: comando "vuoto" (doppia `|` senza niente in mezzo)
- `ls | grep foo <bar | wc -l` — errore: solo il primo comando può avere la redirectione dell'input