LINUX 101: COME FUNZIONA LA SHELL?

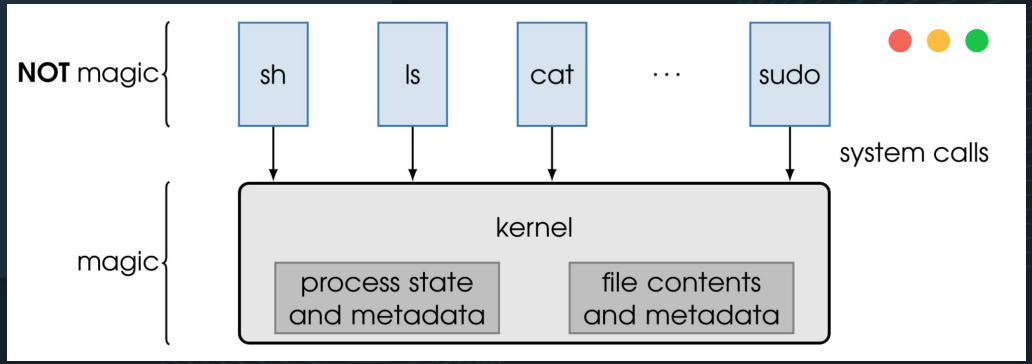
```
> cat talk_title.txt
```

01 LA STRUTTURA DI UNIX

I fondamenti per capire cosa fa (e cosa non fa) il Kernel

MAGIC VS
NOT MAGIC

I programmi non sono magici, ma sfruttano la magia del kernel



COSA GESTISCE IL KERNEL

Processi

- ID, UserID
- Primitive
- *current working directory*
- file aperti

File

- Tipo di file
- Owner e gruppi
- Permessi
- Primitive

PRIMITIVE PER I PROCESSI

fork()

L'unico modo per creare nuovi processi

execve()

Per eseguire nuovi programmi

wait()
exit()

Gestione della terminazione

LA SHELL È UN PROCESSO?

02 ELEMENTARY SHELL

Come (e perché) la shell funziona

01-basic

La shell più semplice

- Lettura da stdin
- Creazione di un nuovo processo
- Esecuzione con un wrapper (più semplice) di execve()

```c
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

#define MAX_LINE 1024

int main()
{
    char buf[MAX_LINE];
    int n;

    while ( (n = read(0, buf, MAX_LINE)) > 0 ) {
        buf[n - 1] = '\0';

        if (fork()) {
            wait(0);
        } else {
            execl(buf, buf, NULL);
            perror(buf);
            exit(1);
        }
    }
    return 0;
}
```

01-basic

La shell più semplice

- Lettura da stdin
- Creazione di un nuovo processo
- Esecuzione con un wrapper (più semplice) di execve()

$PATH non viene usata

```c
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

#define MAX_LINE 1024

int main()
{
    char buf[MAX_LINE];
    int n;

    while ( (n = read(0, buf, MAX_LINE)) > 0 ) {
        buf[n - 1] = '\0';

        if (fork()) {
            wait(0);
        } else {
            execl(buf, buf, NULL);
            perror(buf);
            exit(1);
        }
    }
    return 0;
}
```

02-path

Eseguire programmi da qualunque posizione

- Passaggio del PATH tramite execlp()

```c
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

#define MAX_LINE 1024

int main()
{
    char buf[MAX_LINE];
    int n;

    while ( (n = read(0, buf, MAX_LINE)) > 0 ) {
        buf[n - 1] = '\0';

        if (fork()) {
            wait(0);
        } else {
            execlp(buf, buf, NULL);
            perror(buf);
            exit(1);
        }
    }
    return 0;
}
```

02-path

Eseguire programmi da qualunque posizione

- Passaggio del PATH tramite execlp()

Il passaggio degli argomenti non è ancora implementato

```c
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

#define MAX_LINE 1024

int main()
{
    char buf[MAX_LINE];
    int n;

    while ( (n = read(0, buf, MAX_LINE)) > 0 ) {
        buf[n - 1] = '\0';

        if (fork()) {
            wait(0);
        } else {
            execlp(buf, buf, NULL);
            perror(buf);
            exit(1);
        }
    }
    return 0;
}
```

03-words

Argomenti per programmi più complessi

- buildargv() prepara argc e argv per l'esecuzione
- execvp() permette il passaggio dei parametri al nuovo programma

```c
int buildargv(char *argv[], word_t words[], int nwords)
{
    int i, argc;

    argc = 0;
    for (i = 0; i < nwords && i < MAX_ARGS; i++) {
        word_t *nw = &words[i];
        argv[argc] = nw->w;
        argc++;
    }
    if (argc >= MAX_ARGS) {
        fprintf(stderr, "too many arguments\n");
        return 0;
    }
    argv[argc] = NULL;
    return argc;
}
```

03-words

Argomenti per programmi più complessi

- buildargv() prepara argc e argv per l'esecuzione
- execvp() permette il passaggio dei parametri al nuovo programma

Alcuni "programmi" non sono programmi!

```c
int buildargv(char *argv[], word_t words[], int nwords)
{
    int i, argc;

    argc = 0;
    for (i = 0; i < nwords && i < MAX_ARGS; i++) {
        word_t *nw = &words[i];
        argv[argc] = nw->w;
        argc++;
    }
    if (argc >= MAX_ARGS) {
        fprintf(stderr, "too many arguments\n");
        return 0;
    }
    argv[argc] = NULL;
    return argc;
}
```

04-builtin

Quando è necessario modificare l'environment

- Confronto del nome programma chiamato con le built-in
- Esecuzione nello stesso Thread
- Si evita l'overhead per alcuni programmi

```
while ( (n = read(0, buf, MAX_LINE)) > 0 ) {
    buf[n - 1] = '\0';

    nwords = getwords(buf, words);
    if (!nwords)
        continue;
    buildargv(c_argv, words, nwords);

    if (!strcmp(c_argv[0], "cd")) {
        if (c_argv[1] == NULL) {
            fprintf(stderr, "argument missing");
            continue;
        }
        if (chdir(c_argv[1]) < 0) {
            perror(c_argv[1]);
        }
        continue;
    }
```

LA SHELL È COMPLETA?

Funzionalità come I/O redirect, gestione dell'environment e scripting devono essere implementate nella shell.

Anche le funzioni complesse vengono gestite direttamente dalla shell.

GRAZIE PER L'ATTENZIONE!

Materiale fornito dal prof. Giuseppe Lettieri

docenti.ing.unipi.it/g.lettieri/

genrico.loni[at]gmail.com
@genricoloni