Progetto II: Il linguaggio C e le chiamate di sistema sui processi

Scrivere un programma C `psstat.c` che svolga quanto segue:

- se invocato senza parametri deve stampare su standard output un istogramma orizzontale che rappresenti il numero di processi in esecuzione per ogni utente (ovviamente vanno considerati solo gli utenti che hanno almeno un processo in esecuzione). Esempio:

```
> ./psstat
root  : **********
rossi : **
verdi : ****
...
```

- se invocato, passando come parametro il nome di un utente, deve stampare su standard output il numero di processi di quell'utente. Esempio:

```
> ./psstat verdi
utente verdi : 4 processi
```

Progetto II: Il linguaggio C e le chiamate di sistema sui processi

Suggerimenti:

- Per costruire l'istogramma conviene lanciare, tramite una system call della famiglia `exec`, il comando `ps` con gli opportuni argomenti e salvarne l'output in un file di testo per poi aprire quest'ultimo ed elaborarne il contenuto.

- Per salvare l'output di `ps` in un file bisogna effettuare una redirezione dell'output: tale operazione non si può fare direttamente tramite le system call della famiglia `exec`. Bisogna prima creare uno script della shell `psscript` con il comando `ps [opzioni] > <file>` e richiamare quest'ultimo dal programma C come segue:

  ```
  execl("/bin/sh", "sh", "-c", "./psscript", NULL);
  ```

  L'opzione `-c` dell'eseguibile della shell permette di eseguire i comandi contenuti nel file che viene specificato (in questo caso il file `psscript`).

L'ambiente di un processo (I)

- L'ambiente di un processo è un insieme di stringhe (terminate da `\0`).
- Un ambiente è rappresentato da un vettore di puntatori a caratteri terminato da un puntatore nullo.
- Ogni puntatore (che non sia quello nullo) punta ad una stringa della forma:

$$\textit{identificatore} = \textit{valore}$$

- Per accedere all'ambiente da un programma C, è sufficiente aggiungere il parametro `envp` a quelli del main:

```
/* showmyenv */
#include <stdio.h>

main(int argc, char **argv, char **envp)
{
    while(*envp)
        printf("%s\n",*envp++);
}
```

oppure usare la variabile globale seguente:

```
extern char **environ;
```

L'ambiente di un processo (II)

- L'ambiente di default di un processo coincide con quello del processo chiamante.

- Per specificare un nuovo ambiente è necessario usare una delle due varianti seguenti della famiglia `exec`:

  - `execle(path, arg0, arg1, ..., argn, (char *)0, envp);`

  - `execve(path, argv, envp);`

  memorizzando in `envp` l'ambiente desiderato.

L'ambiente di un processo (III)

```
/* setmyenv */
#include <unistd.h>
#include <stdio.h>

main()
{ char *argv[2], *envp[3];

  argv[0] = "setmyenv";
  argv[1] = (char *)0;

  envp[0] = "var1=valore1";
  envp[1] = "var2=valore2";
  envp[2] = (char *)0;

  execve("./showmyenv", argv, envp);
  perror("execve fallita");
}
```

Eseguendo il programma precedente si ottiene quanto segue:

```
>./setmyenv
var1=valore1
var2=valore2
>
```

L'ambiente di un processo (IV)

- Esiste una funzione della libreria standard che consente di cercare all'interno di `environ` il valore corrispondente ad una specifica variabile d'ambiente:

```
#include <stdlib.h>
char *getenv(const char *name);
```

`getenv` prende come argomento il nome della variabile da cercare e restituisce il puntatore al valore (ciò che sta a destra del simbolo =) o `NULL` se non lo trova:

```
#include <stdio.h>
#include <stdlib.h>
main()
{ printf("PATH=%s\n", getenv("PATH"));
}
```

- Dualmente, `putenv` consente di modificare o estendere l'ambiente:
  `putenv("variabile=valore");`

Current working directory e root directory

- Ad ogni processo è associata una **current working directory** che è ereditata dal processo padre. La chiamata di sistema seguente consente di cambiarla:

```
#include <unistd.h>

int chdir(const char *path);
```

- Ad ogni processo inoltre è associata una **root directory** che specifica il punto di inizio (/) del file system visibile dal processo stesso. Per cambiare la root directory si può usare la chiamata di sistema seguente:

```
#include <unistd.h>

int chroot(const char *path);
```

User-id e group-id (I)

- Ad ogni processo sono associati un **real user-id** ed un **real group-id** che coincidono con quelli dell'utente che ha lanciato il processo.

- Esistono anche l'**effective user-id** e l'**effective group-id** che determinano i privilegi del processo. Nella maggior parte dei casi essi coincidono, rispettivamente, con il real user-id ed il real group-id.

- Tuttavia se il file di un programma ha attivo il bit dei permessi noto come set-user-id (set-group-id), allora, quando sarà invocato con una chiamata `exec`, l'effective user-id (effective group-id) diventerà quello del proprietario del file e non quello dell'utente che lo ha lanciato.

User-id e group-id (II)

```
#include <unistd.h>

main()
{
  uid_t uid, euid;
  gid_t gid, egid;

  uid = getuid();
  euid = geteuid();

  gid = getgid();
  egid = getegid();

  uid_t newuid;
  gid_t newgid;
  int status;
  ...
  status = setuid(newuid); /* imposta l'effective user-id */
  status = setgid(newgid); /* imposta l'effective group-id */
}
```

Esercizi

Modificare il programma `smallsh` della quindicesima lezione (23/11/2004), aggiungendo le seguenti funzionalità (dopo aver implementato le funzioni `gettok` e `runcommand` lasciate per esercizio):

- aggiungere il comando `exit` per terminare l'esecuzione della shell;
- implementare il comando `cd` per cambiare la directory corrente;
- aggiungere la possibilità di riconoscere ed ignorare i commenti (i.e., tutto ciò che segue il carattere speciale `#`);
- fare in modo che, usando il metacarattere di escape (\), sia possibile includere i caratteri speciali (e.g. spazi, tabulazioni, `&`, `;`, `#`) negli argomenti dei comandi.