

Ambiente di un processo

Programmi e Processi

- ▶▶ Un **programma** è un file ordinario contenente delle istruzioni e dei dati
- ▶▶ Un **processo** è un ambiente nel quale un programma viene eseguito. Fanno parte dell'ambiente, dei segmenti (istruzioni, dati) inizializzati dal programma

Funzione **main**

```
int main(int argc, char *argv [ ]);
```

Quando si esegue un programma:

- ▶▶ si esegue prima una routine di start-up speciale che prende
 - ▶ valori passati dal kernel in *argv*[] dalla linea di comando
 - ▶ variabili d'**ambiente**
- ▶▶ successivamente viene chiamata la funzione **main**

Terminazione di un processo

- ▶▶ Terminazione normale
 - ▶ ritorno dal **main**
 - ▶ chiamata a **exit**
 - ▶ chiamata a **_exit**
- ▶▶ Terminazione anormale
 - ▶ chiamata **abort**
 - ▶ arrivo di un segnale

Funzioni exit

```
▶▶ #include <stdlib.h>
void exit (int status);
```

Descrizione: restituisce *status* al processo che chiama il programma includente exit ;

effettua prima una *pulizia* e poi ritorna al kernel

- ▶ effettua lo shutdown delle funzioni di libreria standard di I/O (fclose di tutti gli stream lasciati aperti) => tutto l'output è flushed

```
▶▶ #include <unistd.h>
void _exit (int status);
```

Descrizione: ritorna immediatamente al kernel

```
#include <stdio.h>

int main(void)
{
    printf("Ciao a tutti");

    _exit(0);
}
```

Exit handler

```
#include <stdlib.h>
int atexit (void ( *funzione) (void));
```

Restituisce: 0 se O.K.
diverso da 0 su errore

funzione = punta ad una funzione che e' chiamata per cleanup il processo alla sua normale terminazione.

▶▶ Il numero di exit handlers che possono essere specificate con atexit e' limitato dalla quantita' di memoria virtuale.

```
int main(void)
{
    atexit(my_exit2);    atexit(my_exit1);
    printf("ho finito il main\n");
    return(0);
}

static void my_exit1(void)
{
    printf("sono il primo handler\n");
}

static void my_exit2(void)
{
    printf("sono il secondo handler\n");
}
```

```
$ a.out
ho finito il main
sono il primo handler
sono il secondo handler
```

Sostituiamo return(0)
con _exit(0).

Che cosa succede
mandando in
esecuzione

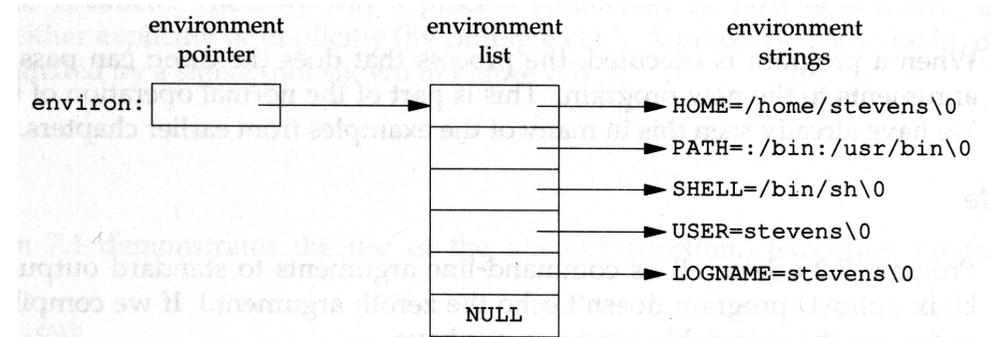
Esercizio

- ▶ Scrivere un programma che chiede all'utente di scegliere se uscire bruscamente oppure lasciando eseguire al programma alcuni `exit_handler`

Environment List

ad ogni programma è passato anche una lista di variabili di ambiente individuata dalla variabile

`extern char **environ`



```
#include <stdio.h>

extern char **environ;
int main()
{
    int i=0;
    while (environ[i])
        printf("%s\n", environ[i++]);
}
```

Manipolare le variabili di ambiente

```
#include <stdlib.h>
```

```
char *getenv (const char *name);
```

Descrizione: restituisce il puntatore al contenuto della variabile *name* tra le variabili di ambiente

```
int putenv (const char *str);
```

Descrizione: prende una stringa di char *str* che è del tipo *name=value* e la inserisce nella environment list, se *name* già esiste sostituisce il suo valore con *value*

Manipolare le variabili di ambiente

```
#include <stdlib.h>
```

```
int setenv (const char *name, const char *value, int rewrite);
```

Descrizione: inserisce *name= value* e

se *name* già esiste e *rewrite*≠0 allora sostituisce

se *name* già esiste e *rewrite*=0 allora non sostituisce

Layout di memoria di programmi C

1. segmento di testo
 - istruzioni macchina eseguite.
2. segmento di dati inizializzati
3. segmento di dati non inizializzati
4. stack
 - vbls automatiche, funzioni e sue informazioni (e.g. dove ritornare)
5. heap
 - allocazione dinamica della memoria