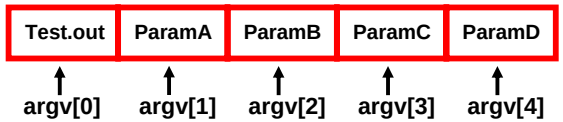
Esercitazione 3
segnali

Richiami

Utilizzo di argc e argv:

- argc (int argc) rappresenta il numero di stringhe distinte che caratterizzano l'invocazione del programma, quindi il numero di parametri passati più 1 (il nome del programma)
- argv (char**argv o char * argv[]) è un puntatore ad un vettore di stringhe che contengono effettivamente le stringhe che caratterizzano la corrente l'invocazione del programma, quindi il nome del programma e tutti i parametri che gli sono stati passati

ESEMPIO
argc = #parametri + 1 (nome programma) = 5



argv = vettore di lunghezza 5
ATTENZIONE : argv[i] è un puntatore!

Richiami

La funzione atoi permette di convertire una stringa nell'intero che rappresenta:

```
int atoi(char * nptr)
```

Un paio di esempi:

- ```
  char car;
  scanf("%c",&car);
  int val=atoi(&car);
  ```
- ```
  char * str[10];
  scanf("%s",str);
  int val=atoi(str);
  ```

NOTA:
per utilizzare atoi deve essere inclusa la libreria `stdlib.h`

Richiami

La funzione sprintf permette di utilizzare una stringa come stdout di una printf, quindi memorizziamo il risultato della stampa all'interno dell'array di caratteri:

```
int sprintf(char* str, char* format,…)
```

dove `str` è la stringa usata come buffer e l'intero ritornato è il numero di caratteri effettivamente stampati.

Esempio:

```
char* str[50];
int a=2;
int b=3;
sprintf(str,"il risultato di %d + %d è %d",a,b,(a+b));
printf("%s",str); → il risultato di 2 + 3 è 5
```

Richiami

La funzione sscanf permette di utilizzare una stringa come stdin di una scanf; utilizza la stessa invocazione ed ha lo stesso significato della scanf, con la differenza che il primo argomento `str` è l'array di caratteri (ovvero la stringa) che vogliamo leggere:

```
int sscanf(char* str, char* format,…)
```

Esempio:

```
int x;
sscanf( argv[1], "%d", &x );  → equivalente a
                                x = atoi(argv[1]);
```

Problema

Scrivere un programma C che faccia la somma di un numero non noto a priori di interi passati come parametri di invocazione e memorizzi il risultato in una variabile di tipo stringa di caratteri.

Suggerimenti:

- Attenzione all'uso di argc e argv
- Leggere il man di atoi
- Leggere il man di sprintf

Esempio di soluzione

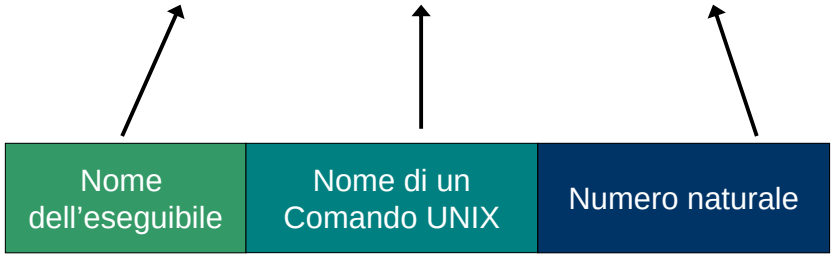
```
# include<stdio.h>
# include<stdlib.h>

int main(int argc,char**argv){
  printf("Nome comando%s\n",argv[0]);
  printf("Numero argomenti %d\n",argc-1);
  int variabile_appoggio=0;
  int accumulatore=0;
  char* risultato[10];
  for(int i=1;i<argc;i++)
    {
      variabile_appoggio=atoi(argv[i]);
      //printf("argomento %d-esimo:%s\n",i,argv[i]);
      //printf("variabile appoggio:%d\n",variabile_appoggio);
      accumulatore=accumulatore+variabile_appoggio;
    }
  sprintf(risultato,"%d",accumulatore);
  printf("Risultato:%s\n",risultato);
}
```

Obiettivo 1/2

- Scrivere un comando che abbia la sintassi

`execute <nome_comando> <N>`



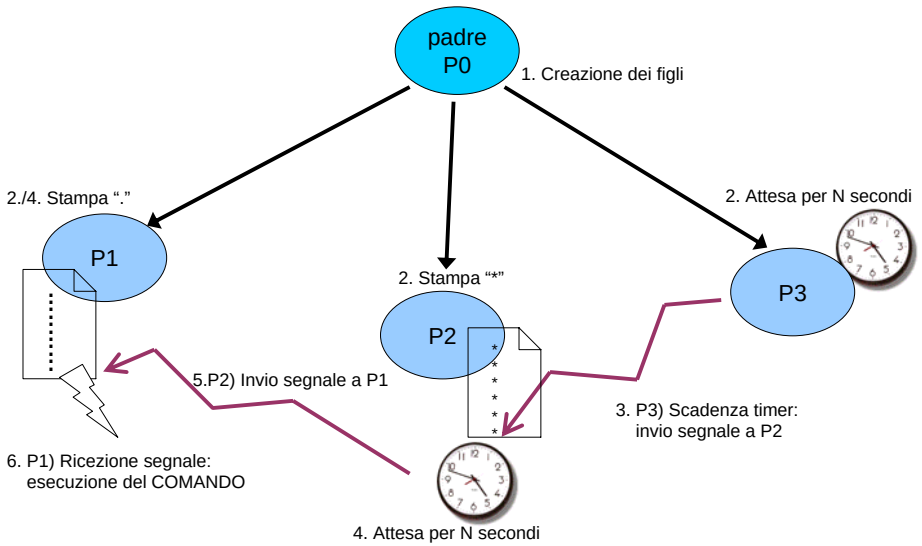
Obiettivo 2/2

- In particolare:
  - Il comando deve creare tre processi figli, P1, P2 e P3; il processo padre si mette poi in attesa della terminazione dei tre figli
  - P1 deve mettersi in un ciclo di attesa infinito, stampando, ogni secondo, la stringa ".\n"
  - P2 deve mettersi in un ciclo di attesa infinito, stampando, ogni secondo, la stringa "*\n"
  - P3 deve attendere **<N>** secondi, poi deve inviare un segnale a P2
  - Alla ricezione del segnale, P2 deve attendere passivamente (senza stampare sullo stdout nulla) altri **<N>** secondi per poi inviare un segnale a P1 e terminare
  - Alla ricezione del segnale, P1 deve eseguire il comando specificato in **<nome_comando>**

Note

- Nell'esercizio facciamo in modo che il processo 3 comunichi con il processo 2 e che il processo 2 comunichi con il processo 1, ma sarebbe stato equivalente se avessimo voluto un ordine invertito? NO!
- Lavorando in questo modo è possibile memorizzare in un vettore i pid dei processi al momento della generazione, in modo che:
  - P3 conosca il pid di P2 e P1
  - P2 conosca il pid di P1
- Ciascun processo può conoscere i pid dei processi figli creati in precedenza in quanto alla creazione ottiene una copia dei dati del padre in quell'istante (compreso quindi l'eventuale vettore di pid)

Schema



Variante

- Vogliamo ora apportare alcune modifiche all'esercizio:
  - Come prima il comando deve creare tre processi figli, P1, P2 e P3; il processo padre si mette poi in attesa della terminazione dei tre figli
  - P1 deve mettersi in un ciclo di attesa infinito, stampando, ogni secondo, la stringa ".\n"
  - P2 deve mettersi in un ciclo di attesa infinito, stampando, ogni secondo, la stringa "*\n"
  - P3 deve attendere **<N>** secondi, poi deve inviare un segnale a P2
  - Alla ricezione del segnale, P2 deve attendere altri **<N>** secondi *continuando a stampare a video "*\n"* per poi inviare un segnale a P1 e terminare
    - La `sleep` in questo caso non è utile ai nostri scopi
    - Potremmo utilizzare la system call `alarm`
  - Alla ricezione del segnale, P1 deve eseguire il comando specificato in **<nome_comando>**

Schema della variante