## Documentazione

Implementare un programma che riceva in input tramite argv[2] un numero intero N maggiore o uguale ad 1 (espresso come una stringa di cifre decimali), e generi N nuovi processi. Ciascuno di questi leggerà in modo continuativo un valore intero da standard input, e lo comunicherà al processo padre tramite memoria condivisa. Il processo padre scriverà ogni nuovo valore intero ricevuto su di un file, come sequenza di cifre decimali.
I valori scritti su file devono essere separati dal carattere ' ' (blank).
Il pathname del file di output deve essere comunicato all'applicazione tramite argv[1].

L'applicazione dovrà gestire il segnale SIGINT in modo tale che se il processo padre venga colpito il contenuto del file di output venga interamente riversato su standard-output.

---

# *OUTPUT*

***Per facilitare la comprensione del software di seguito viene riportato un esempio di output su sistemi UNIX.***

```
gcc prova2.c -lpthread
./a.out file 5
```

**Come è facile notare l'applicazione prevede che vengano passati come parametri due tipologie di argomenti:**

- **Il nome di un file, in questo caso è il nome "file" stesso;**
- **Un numero intero che identifichi il numero di processi.**

***In totale vi sono 3 argomenti: ad esclusione dei parametri appena discussi viene aggiunto il nome del programma appena compilato che per default è a.out.***

***Passati questi argomenti come bisogna procedere?***

***Vogliamo realizzare un software di questa tipologia:***

```
Valore letto : 3
Dati scritti correttamente [*]
10
Valore letto : 10
Dati scritti correttamente [*]
55
Valore letto : 55
Dati scritti correttamente [*]
102
Valore letto : 102
Dati scritti correttamente [*]
```

***Visualizziamo il file:***

```
3 10 55 102
```

***Ed effettivamente i valori interni al file sono separati.***

**Il codice sfrutta l'ausilio di una regione di memoria mappata tramite la funzione mmap.**

```
address_mmap = mmap(NULL, N_pages*PAGE_SIZE, PROT_WRITE|PROT_READ, MAP_SHARED|MAP_ANONYMOUS, 0, 0);
```

**Ed è proprio questa regione che verrà usata per comunicare al processo padre il valore appena letto dal processo figlio.**

**La creazione del file verrà effettuata tramite le seguenti istruzioni:**

```
file_descriptor = open(nome_file, O_CREAT|O_RDWR, 0666);
file = fdopen(file_descriptor, "r+");
```

**Il "nome_file" passato come primo parametro alla funzione open è impostato nelle righe precedenti nel seguente modo:**

```
char *nome_file = argv[1];
```

**Perciò il nome assegnato al file viene deciso dall'utente a run-time.**

***Dopo questa lunga premessa ed effettuati vari controlli di natura banale si giunge alla porzione viva del codice.***

***La struttura presenta la gestione di due semafori: un semaforo che viene preso in gestione dai processi e un semaforo che viene utilizzato esclusivamente dal processo padre.***
***Non sono presenti array semaforici – data la semplicità del codice.***

```
semaforo_per_processi = semget(IPC_PRIVATE,1,IPC_CREAT|IPC_EXCL|0666);
semaforo_daddy = semget(IPC_PRIVATE, 1,IPC_CREAT|IPC_EXCL|0666);
ret = semctl(semaforo_daddy,0, SETVAL, 0); //si blocca
ret = semctl(semaforo_per_processi, 0, SETVAL, 1);
```

***Sono entrambi singoli distributori di gettoni.***
***Sono impostati nel seguente modo:***


***semaforo_per_processi***


***semaforo_daddy***

***Il semaforo dei processi è settato ad 1 pertanto sarà proprio il processo 1 a partire per primo.***
***Anche il processo padre "vorrebbe" partire per primo ma si ritrova impossibilitato nel prelievo di un gettone perciò si bloccherà in attesa che un processo figlio lo sblocchi.***
***Il processo 1 scriverà dentro la memoria condivisa il valore numerico:***

*Dopodiché chiederà lo sblocco del processo padre che leggerà questo valore e lo scriverà sul file come sequenza di cifre decimali.*


*semaforo_per_processi*


*semaforo_daddy*

*Parte il padre.*


*semaforo_per_processi*


*semaforo_daddy*

*Scrive sul file.*
*Reimposta ad 1 il semaforo dei processi.*


*semaforo_per_processi*


*semaforo_daddy*

*E ricomincia.*

*La gestione del segnale è notevolmente facile.*
**Appena verrà premuto il segnale CTRL + C bisognerà stampare il contenuto del file in output.**
**Questo lo si può semplicemente implementare tramite l'ausilio di una signal:**
**signal(SIGINT, printer);**
*La funzione che implementa il segnale è la seguente:*

```
void printer(int sign)
{
    system(command);
    return;
}
```

*Dove command è implementato nel seguente modo:*

```
sprintf(command,"cat %s", nome_file);
```

**Documentazione didattica a cura di Simone Remoli.**