Esercitazione Processi

Laboratorio Sistemi Operativi

Antonino Staiano
Email: antonino.staiano@uniparthenope.it

File I/O

SOLUZIONI DI ALCUNI ESERCIZI

Esercizio 1

- Scrivere un programma C che:
  - Prende in input coppie di interi utilizzando la system call read
  - Calcola la somma degli interi
  - Stampa a video il risultato utilizzando la write
  - Termina quando il primo input e' -1
- Assumere che gli interi consistano di una sola cifra

Esercizio 2

- Modificare l'esercizio 1 in modo che prenda l'input dal file “testfile” e scriva l'output nel file “outputfile”
- Utilizzare le funzioni per la duplicazione dei file descriptor
  - TUTTE LE READ SU STANDARD INPUT
  - TUTTE LE WRITE SU STANDARD OUTPUT

Processi

ESERCIZI

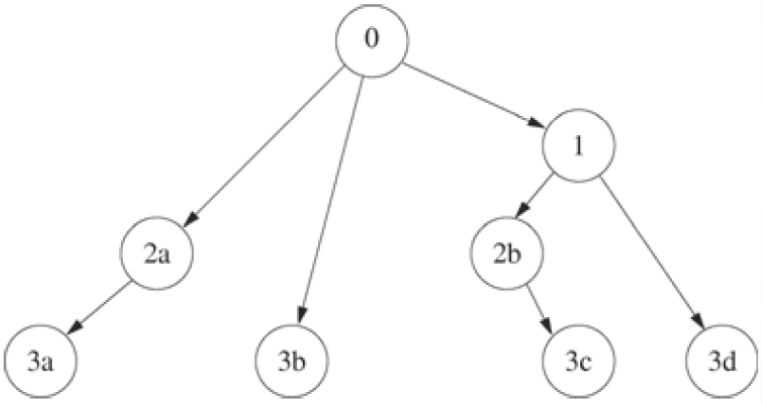
Esempio

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main (int argc, char *argv[]) {
pid_t childpid = 0;
int i, n;
if (argc != 2){ /* Controllo argomenti*/
fprintf(stderr, "Uso: %s processi\n", argv[0]);
return 1;
}
n = atoi(argv[1]);
for (i = 1; i < n; i++)
if ((childpid = fork()) <= 0)
break;
fprintf(stderr, "i:%d processo ID:%d padre ID:%d figlio
ID:%d\n",
i, getpid(), getppid(), childpid);
return 0;
}
```


Osservazioni

- Cosa succede se sostituiamo il test
  (childpid = fork()) <= 0 con (childpid = fork()) == -1
- **Risposta:**
- Tutti I processi restano nel ciclo a meno che la fork fallisca. Ogni iterazione del ciclo raddoppia il numero di processi, formando un albero del tipo riportato in figura per n=4
  - In figura, ogni processo è rappresentato con un cerchio la cui etichetta è il valore di i al momento della sua creazione
  - Il processo originario ha etichetta 0
  - Le lettere distinguono processi creati con lo stesso valore di i
- In questo programma non si distingue tra padre e figlio dopo la fork
  - Entrambi procedono a creare figli alla successiva iterazione del ciclo

Esempio: albero dei processi creati (n=4)



Esercizio 1

```
int glob=5;
int pid=0;
pid=fork();
glob--;
pid=fork();
glob--;
if (pid!=0) {
     pid=fork();
     glob--;
}
printf("Valore di glob=%d\n",glob);
```


Esercizio 2

```
int glob=5;
int pid=0;
int main() {
      int i=0;
      for (i=1;i<3;i++) {
            pid=fork();
                  if (pid==0) {
                        glob=glob*2;
                        sleep(i+1);
                  }
            glob=glob+1;
            printf("Valore di glob=%d\n",glob);
}
}
```


Esercizio 3

- Scrivere un programma C che:
  - Crea un processo figlio, stampa il messaggio “In attesa” ed attende la terminazione del figlio.
  - Il figlio esegue il comando “ls -l”
  - Quando il figlio termina, il padre visualizza il messaggio “nuovo figlio” e crea un secondo processo figlio.
  - Il secondo figlio aspetta per 5 secondi, stampa a video un messaggio e termina.
  - Quando il processo figlio termina, il padre stampa a video il pid del processo terminato.

Esercizio 4

- Scrivere un programma C in cui un processo crea un processo figlio
  - Il processo figlio calcola la sequenza di Fibonacci di ordine n (n<=12). Quando termina restituisce il valore calcolato come codice di terminazione
  - Il padre attende la terminazione del figlio ed esamina lo stato di terminazione
    - Se lo stato di terminazione è relativo ad una terminazione con successo e il codice di terminazione è un valore minore di 50
      - Crea un secondo figlio che esegue il comando ls –al a.out
      - Attende il secondo figlio, stampa un messaggio e termina
    - Altrimenti, stampa un messaggio e termina

Esercizio 5

- Realizzare un programma in C e Posix sotto Linux che realizzi una struttura di processi ad albero ternario, tale che ogni processo si metta in attesa che i suoi figli terminino. Ogni figlio termina dopo aver atteso per un numero di secondi pari al livello dell'albero al quale si trova, allo scadere del quale stampa a schermo la stringa "Concluso! e comunica la genitore la sua terminazione.