

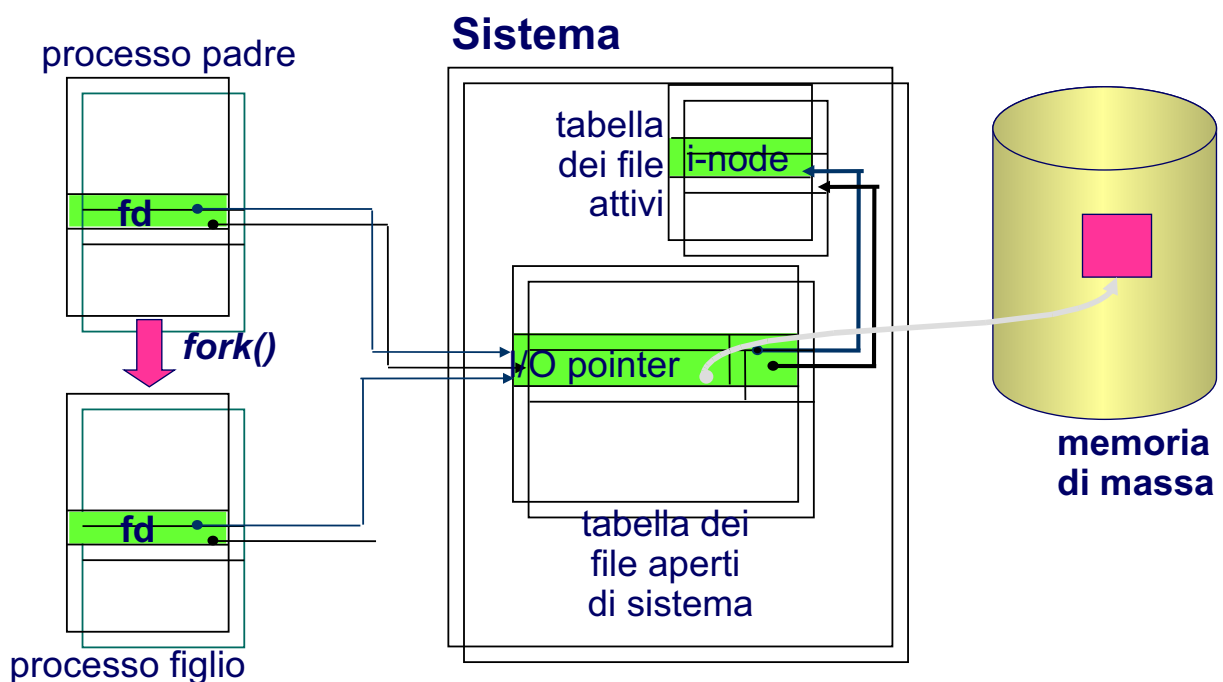
Sesta Esercitazione

Gestione File-System
Segnali

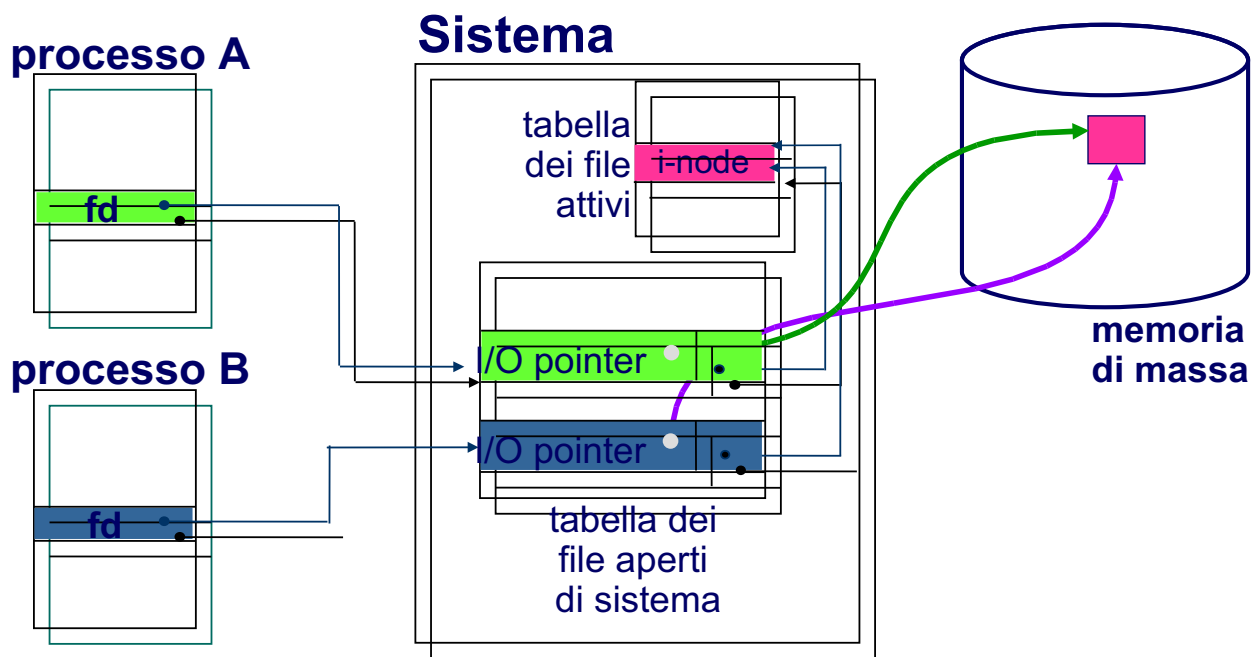
Stefano Monti

smonti@deis.unibo.it

Esempio: processi padre e figlio **condividono**
I/O pointer di file aperti **PRIMA** della creazione



Esempio: processi padre e figlio **NON** condividono I/O pointer di file aperti **DOPPO** la creazione



3

Principali primitive di accesso ai File

- `int creat(char nomefile[], int mode)`
- `int open(char nomefile[], int flag, [int mode])`
- `int close(int fd)`
- `int read(int fd, char* buf, int n)`
- `int write(int fd, char* buf, int n)`
- `int lseek(int fd, int offset, int origin)`

4

Segnali UNIX

■ Gestione di un segnale:

- Associare una funzione handler

```
void(*signal(int sig, void(*func)(int)))(int);
```

Ritorna il
precedente gestore

Numero associato
al segnale **da gestire**

Numero associato
al segnale **ricevuto**

- Ignorare il segnale **SIG_IGN**
- Eseguire una azione di default **SIG_DFL**

■ Invio segnali

```
int kill(int pid, int sig);
```

5

Segnali: altre system call

sleep()

```
unsigned int sleep(unsigned int N)
```

- provoca la sospensione del processo per N secondi (al massimo)
- se il **processo riceve un segnale durante il periodo di sospensione, viene risvegliato prematuramente**
- restituisce 0, se la sospensione non è stata interrotta da segnali; se il risveglio è stato causato da un segnale al tempo **x**, **sleep()** restituisce il numero di secondi non utilizzati dell intervallo di sospensione (**N-x**)

6

Segnali: altre system call

alarm()

`unsigned int alarm(unsigned int N)`

- imposta un timer che **dopo N secondi invierà allo stesso processo il segnale SIGALRM**
- ritorna:
 - 0, se non vi erano time-out impostati in precedenza
 - il numero di secondi mancante allo scadere del time-out precedente

NB: *default* associato a ricezione di SIGALRM è la terminazione

7

Segnali: altre system call

pause()

`int pause(void)`

- sospende il processo **fino alla ricezione di un qualunque segnale**
- ritorna -1 (`errno = EINTR`)

Segnali & `fork()`

Le **associazioni segnali-azioni** vengono registrate nella **User Area del processo**

Siccome:

- `fork()` copia **User Area** del padre in **User Area** del figlio
- padre e figlio condividono lo stesso codice

quindi

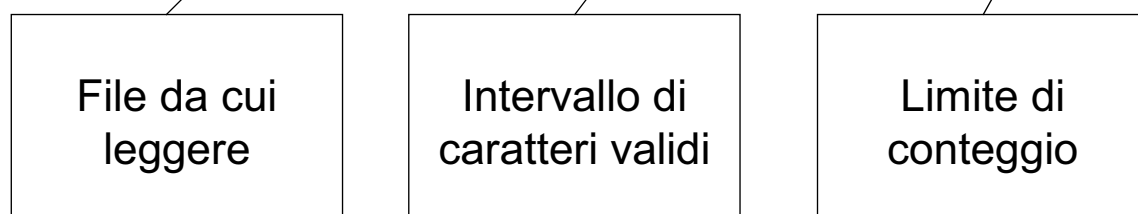
- il figlio eredita dal padre le informazioni relative alla gestione dei segnali:
 - **ignora gli stessi segnali ignorati** dal padre
 - **gestisce con le stesse funzioni gli stessi segnali gestiti** dal padre
 - **segnali a default del figlio sono gli stessi del padre**
- ovviamente `signal()` del figlio successive alla `fork()` non hanno effetto sulla gestione dei segnali del padre

9

Esercizio

Scrivere un programma che abbia la sintassi

`count.c <file> <mincar> <maxcar> <limit>`

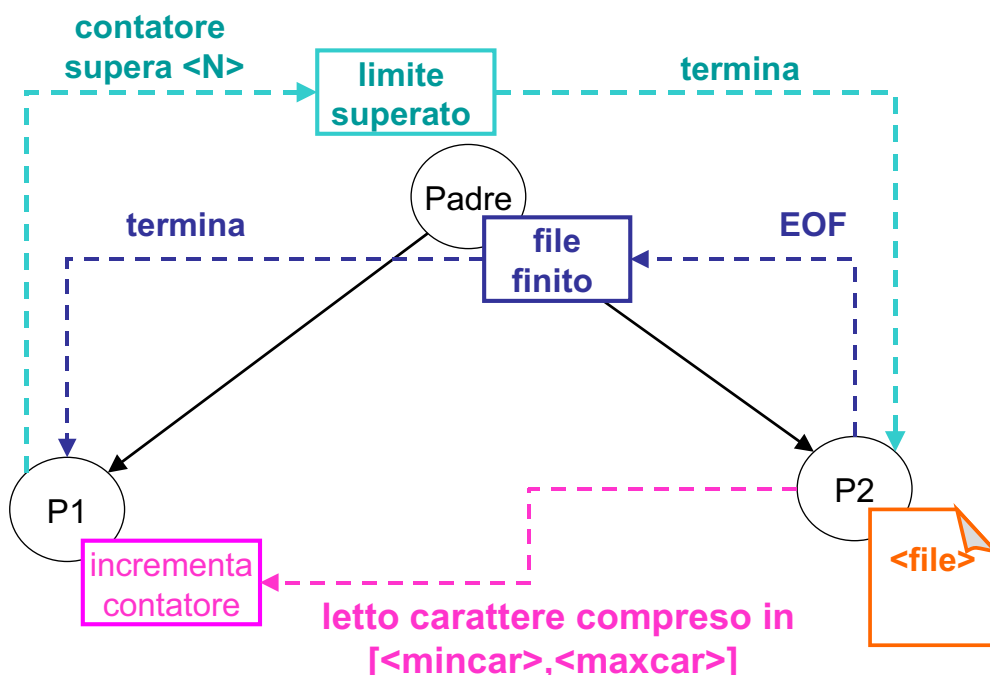


Esercizio

- Il processo padre (P0) deve creare due figli (P1 e P2)
- P2 ha il compito di leggere <file> carattere per carattere (stando in pausa per un secondo dopo ogni lettura)
- Quando **P2 legge un carattere che è compreso nell'intervallo [<mincar>,<maxcar>]** deve **avvertire P1**
 - P1 risponde incrementando un contatore
- Se il valore del contatore controllato da P1 raggiunge <limit>
 - P1 avverte P0** e termina volontariamente
 - P0 risponde all'avvertimento
 - Stampando a video la RESTANTE parte del file
 - Avvertendo P2 di terminare**
- Se invece il file viene completamente consumato da P2
 - P2 avverte P0** e termina volontariamente
 - P0 risponde all'avvertimento **indicando a P1 di terminare**

11

Schema di esecuzione



12

Note e punti cruciali

- Come deve essere aperto il file?
- È possibile invertire i ruoli, ovvero fare in modo che P1 legga il file e P2 gestisca il contatore?
 - Può P1 inviare dei segnali a P2?
- Il padre può terminare direttamente i figli?

13

Estensione

- Estendere il comando nel seguente modo
`count2.c <file> <mincar1> <maxcar1> <lim1>
 <mincar2> <maxcar2> <lim2>`
- Il comportamento è simile al caso precedente
 - Stavolta P2 controlla sia la lettura di caratteri nell'intervallo [`<mincar1>`,`<maxcar1>`] che in [`<mincar2>`,`<maxcar2>`]
 - Il conteggio dei caratteri compresi in [`<mincar2>`,`<maxcar2>`] con lo stesso tipo di reazione del caso precedente nel caso di superamento di `<lim2>`, è **gestito dal padre (P0)**

14

Note sull'estensione

- Ora il padre avrà SIGUSR1 occupato per il conteggio
 - Quindi deve gestire l'avvertimento di fine file da P2 e quello di limite superato da P1 sempre con SIGUSR2
 - Come fa il padre a capire chi gli ha mandato il segnale?
 - Chi invia questo segnale, poi termina ...

15

Affidabilità dei segnali

Aspetti:

- il gestore rimane **installato**?

In caso negativo, è possibile comunque reinstallare all'interno dell'handler

```
void handler(int s)
{ signal(SIGUSR1, handler);
  printf("Processo %d: segnale %d\n", getpid(), s);
  ... }
```

Che cosa succede se qui arriva un nuovo segnale?

- che cosa succede se arriva il segnale durante l'esecuzione dell'handler?
 - *innestamento delle routine di gestione*
 - *perdita del segnale*
 - *accodamento dei segnali* (segnali *reliable*, *BSD 4.2*)

16