La comunicazione fra processi: PIPE e FIFO

dr. Andrea E. Naimoli

Università degli Studi di Trento

Dipartimento di Ingegneria e Scienza dell'Informazione

via Sommarive 14

I - 38050 Trento - Povo, Italy

FIFO

- FIFO = First In, First Out (come una coda)
- Sono dette anche named pipes
- Le FIFO non sono strutture all’interno del kernel
- Si comportano per certi aspetti come i file, per altri come le pipe

FIFO - caratteristiche

- Come i file:
  - ✓ Hanno un nome
  - ✓ Tutti i processi con i permessi adeguati possono accedervi (non soltanto i discendenti del processo che le ha aperte)
- Come le pipe:
  - ✓ I primi dati scritti sono i primi ad essere letti
  - ✓ Non si può usare lseek
  - ✓ I dati possono essere letti una sola volta

Utilizzo

- Si devono innanzitutto creare, poi vanno aperte in lettura o in scrittura.

- Quando una FIFO viene aperta in scrittura, il programma si blocca fino a quando qualcuno non la apre in lettura (e viceversa)
  - *Blocking I/O: questo permette il loro uso per una sincronizzazione tra diversi processi*.

- Le FIFO come tubi
  - *FIFO = un tubo in cui l'acqua che scorre è rappresentata dai dati.*
  - *Aprire la FIFO è come aprire un rubinetto: non ha senso immettere dati (acqua) finché entrambe le estremità sono state collegate.*

Una nuova system call

int mkfifo(char *pathname, mode_t mode);

- pathname è il nome della nuova FIFO
- mode sono i permessi della FIFO

Esempio:

```
#define FILE_MODE (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
…
mkfifo("myFIFO",FILE_MODE)
```

Come usare una FIFO

- Una volta creata, la FIFO va aperta in lettura ed in scrittura
  - ✓ N.B. **`mkfifo`** si limita a creare la FIFO, ma il processo non può ancora accedervi.

- Si può usare la system call **`open`**, oppure le funzioni della libreria **`<stdio.h>`**

```
lettura   = open(“myFIFO",O_RDONLY | O_NONBLOCK);
scrittura = open(“myFIFO",O_WRONLY | O_NONBLOCK);
```

solo se non si vuole bloccare fino all’apertura dell’altro capo della FIFO

Come cancellare una FIFO

- Nota Bene
  - una pipe viene cancellata dal sistema (kernel) una volta che vengono chiusi tutti i suoi canali di I/O oppure con la chiamata **exit()**
  - una FIFO rimane nel file-system anche dopo una **exit()**
  - per cancellarla occorre esplicitare da programma la sua cancellazione con la system call:

    unlink (“myFIFO”);

read FIFO

- Utilizzo analogo alle pipe:
  - ✓ **int read (fifo_fd, char *buf, int nbytes)**
  - ✓ Se ci sono dati, vengono letti nell'ordine in cui sono stati scritti e una volta letti vengono cancellati dalla FIFO (non possono essere riletti); quando finiscono i dati (o si raggiunge **nbytes**) la **read** ritorna il numero di byte letti
  - ✓ Se si prova a leggere da una FIFO vuota, il programma si blocca fino a quando qualcuno ci scrive qualcosa

write FIFO

```
int write (fifo_fd, char *buf, int nbytes)
```

- Se si prova a scrivere su una FIFO piena, il programma si blocca fino a quando qualche altro processo non libera del posto.
- La **write** è un'operazione *atomica*
- Se si deve scrivere un numero di byte che non supera la dimensione della FIFO, questi vengono scritti tutti in una volta (Non si hanno quindi processi diversi che mescolano parte dei loro dati nella FIFO).

Esercizio (30 min.)

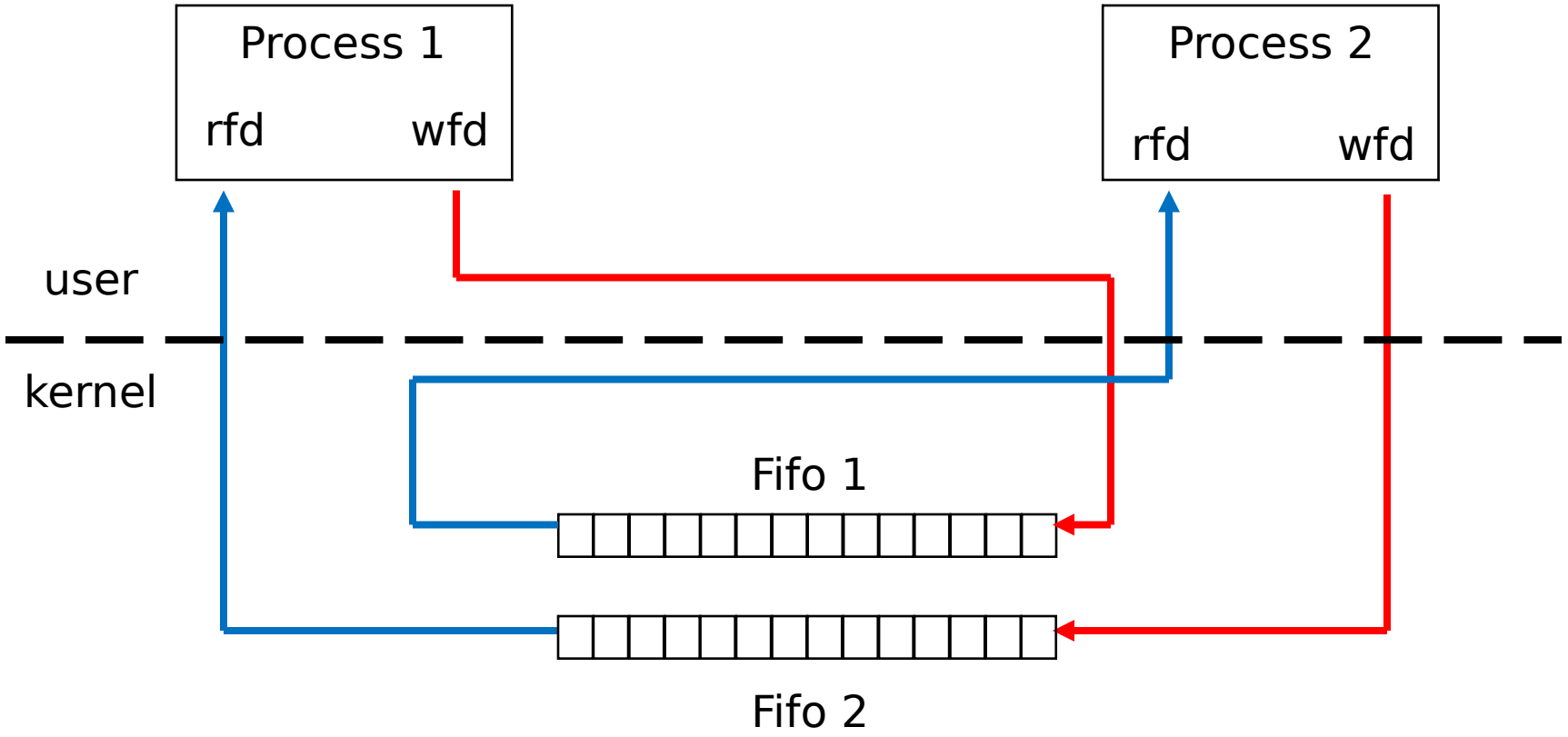
- Scrivere un programma che crea la FIFO con nome “**myFIFO**”
- Da questa FIFO il figlio legge mentre il padre scrive un semplice messaggio

(Esempio “05-fifo.c”)

Comunicazione nei 2 sensi (full-duplex)

Poiché le FIFO, una volta aperte, si comportano come le pipe, per realizzare una comunicazione nei 2 sensi tra 2 processi sono necessarie 2 FIFO

Implementazione



FIFO come pipe

Vantaggi

- Permettono la comunicazione tra processi che non siano legati da vincoli di parentela

Svantaggi

- Servono 5 system call diverse per stabilire il canale di comunicazione
  - Create, write/read per PIPE
  - Create, open, write/read, close, unlink per FIFO
- Posso avere dei problemi coi nomi dei file (ad es. collisioni)