

Laboratorio di Sistemi Operativi

Le FIFO

pipe vs fifo

- ▶ la *pipe* può essere usata solo tra processi "imparentati" (che hanno un antenato comune che ha creato la pipe)
- ▶ la *fifo* consente di scambiare dati tra processi qualsiasi

FIFO (named pipes)

```
#include <sys/types.h>
#include <sys/stat.h>
```

```
int mkfifo(const char *pathname, mode_t mode);
```

Restituisce: 0 se OK

-1 in caso di errore

Apertura di un FIFO

- ▶ Creare una FIFO è come creare un file; infatti FIFO è un tipo di file (codificato in `st_mode` di `stat`)
- ▶ Una volta creata, una FIFO può essere aperta con `open` oppure `fopen` (l'argomento `mode` è lo stesso della `open` e `creat`).
- ▶ Anche se somiglia ad un file (si utilizzano le stesse funzioni di I/O, risiede sul filesystem) ha le caratteristiche di una **pipe**:
 - ▶ I dati scritti vengono letti in ordine **first-in-first-out**
 - ▶ Le chiamate in lettura e scrittura sono atomiche se la quantità di dati è minore di `PIPE_BUF`
 - ▶ Non è possibile rileggere i dati già letti né posizionarsi all'interno con `lseek`

Sincronizzazione in una FIFO

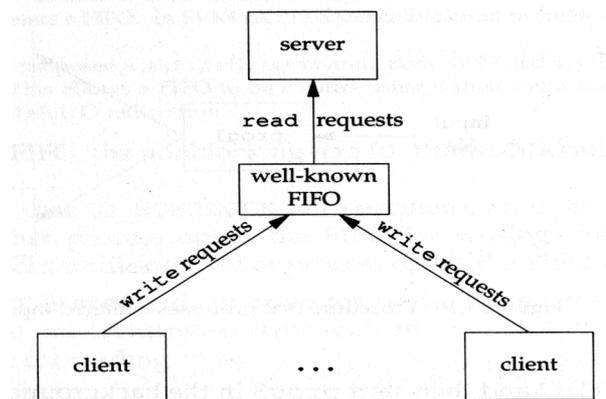
- ▶ Una **open** in lettura si blocca fino a che un processo non effettua una **open** in scrittura (e viceversa).
- ▶ Se viene utilizzato il flag **O_NONBLOCK**,
 - ▶ una **open** in lettura ritorna immediatamente se non c'è un processo che ha effettuato una **open** in scrittura.
 - ▶ una **open** in scrittura restituisce un errore se non c'è un processo che ha effettuato una **open** in lettura.

Sincronizzazione in una FIFO

- ▶ Se si effettua una **write** su una FIFO che non è più aperta in lettura da alcun processo, viene generato il segnale **SIGPIPE**
- ▶ Quando l'ultimo degli scrittori chiude una FIFO, viene generato un **EOF** per il processo lettore

esempio: Comunicazione da *Client* a *Server*

i client mandano richieste e il server le deve leggere dalla FIFO.



esempio 1

- ✍ i client scrivono nella FIFO del server un msg in cui si presentano e chiedono di essere uccisi
- ✍ il server li uccide!!!
- ✓✍ il server legge dalla FIFO

Soluzione: client

```
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<fcntl.h>

int main(){
    int fds, pid;
    char c[5],f[9];

    if((fds=open("FIFO",O_WRONLY))<0){ //Apre la well-known FIFO
        printf("Error: impossibile aprire la fifo in scrittura\n");
        exit(0);
    }

    pid=getpid();

    write(fds,&pid,sizeof(pid)); //Invia il messaggio al server

    close(fds); //Chiude la well-known FIFO

    while(1); //Attende di essere ucciso
}
```

Soluzione: server...

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<fcntl.h>
#include<signal.h>

int done;

void alm(int s);
void server(int fd);

int main(){
    int fds;
    int pidc;

    done=1;

    signal(SIGALRM,alm);
    alarm(30); //Il server rimarrà attivo per 30 secondi

    umask(0);
```

Soluzione: ...server...

```
//Crea la well-known FIFO
if(mkfifo("FIFO",S_IRWXU|S_IRGRP|S_IROTH)<0){
    printf("Impossibile creare la FIFO\n");
    exit(0);
}

while(done){

    if((fds=open("FIFO",O_RDONLY))<0){ //Apri la well-known FIFO
        printf("Error: impossibile aprire la FIFO in sola lettura\n");
        exit(0);
    }
    server(fds); //Esegue le operazioni richieste
    close(fds); //Chiude la well-known FIFO
}
remove("FIFO"); //Rimuove la well-known FIFO
exit(0);
}
```

Soluzione : ...server

```
/*Segnala il passaggio dei 30 secondi e indica al server di arrestarsi*/
void alm(int s){
    done=0;
}

void server(int fd){
    char c[5],f[9];
    int pid;
    int fdc;

    if(read(fd,&pid,sizeof(pid))<0){ //Legge dalla well-known FIFO
        printf("Error: il messaggio non è valido\n");
        return;
    }
    printf("Il server ha letto dalla FIFO %d\n",pid);
    //sleep(1);

    kill(pid,SIGKILL); //Uccide il processo client
}
}
```

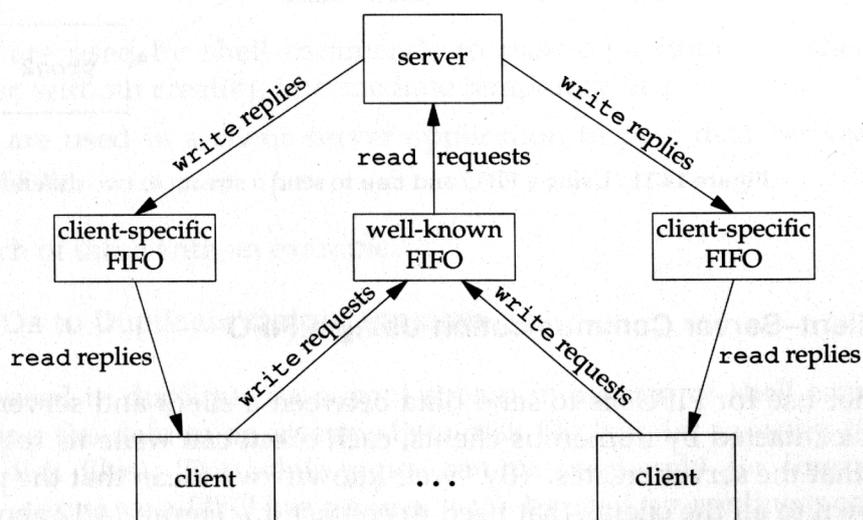
esempio: Comunicazione da *Client a Server*

...ma come puo' rispondere il server?

► Soluzione:

- Ciascun client manda insieme alla sua richiesta anche il suo pid
- Il server crea una FIFO per ciascun client usando un nome con una parte prefissata ed una contenente il pid del processo

Client-Server con FIFO



esempio-2

-  i client scrivono nella FIFO del server un msg in cui si presentano e chiedono di essere uccisi
-  il server legge e scrive nelle loro FIFO un **BANG**
-  il client legge la risposta del server
-  il server li uccide!!!

Soluzione: client...

```
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<fcntl.h>

int main(){

    int fds, pid;
    char c[5],f[9];

    if((fds=open("FIFO",O_WRONLY))<0){ //Apre la well-known FIFO
        printf("Error: impossibile aprire la fifo in scrittura\n");
        exit(0);
    }

    pid=getpid();

    write(fds,&pid,sizeof(pid)); //Invia il messaggio al server

    close(fds); //Chiude la well-known FIFO
```

Soluzione: ...client

```
sprintf(f,"FIFO%d\0",pid); //Setta il nome della client-FIFO

//Attende finchè la client-FIFO non è creata e la apre
while((fds=open(f,O_RDONLY))<0);

if(read(fds,c,5)<0) //Legge dalla client-FIFO
printf("Impossibile leggere BANG\n");

printf("Il client ha letto %s e sta per morire\n",c);

close(fds); //Chiude la client-FIFO

while(1); //Attende di essere ucciso
}
```

Soluzione: server...

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<fcntl.h>
#include<signal.h>

int done;

void alm(int s);
void server(int fd);

int main(){
int fds;
int pidc;

done=1;

signal(SIGALRM,alm);
alarm(30); //Il server rimarrà attivo per 30 secondi

umask(0);
```

Soluzione: ...server...

```
//Crea la well-known FIFO
if(mkfifo("FIFO",S_IRWXU|S_IRGRP|S_IROTH)<0){
    printf("Impossibile creare la FIFO\n");
    exit(0);
}

while(done){

    if((fds=open("FIFO",O_RDONLY))<0){ //Apre la well-known FIFO
        printf("Error: impossibile aprire la FIFO in sola lettura\n");
        exit(0);
    }
    server(fds); //Esegue le operazioni richieste
    close(fds); //Chiude la well-known FIFO
}
remove("FIFO"); //Rimuove la well-known FIFO
exit(0);
}

/*Segnala il passaggio dei 30 secondi e indica al server di arrestarsi*/
void alm(int s){
    done=0;
}
}
```

Soluzione : ...server...

```
void server(int fd){
    char c[5],f[9];
    int pid;
    int fdc;

    if(read(fd,&pid,sizeof(pid))<0){ //Legge dalla well-known FIFO
        printf("Error: il messaggio non è valido\n");
        return;
    }
    printf("Il server ha letto dalla FIFO %d\n",pid);

    sprintf(f,"FIFO%d\0",pid); //Componi il nome della client-FIFO

    if(mkfifo(f,S_IRWXU|S_IRGRP|S_IROTH)<0){ //Crea la client-FIFO
        printf("impossibile creare la FIFO\n");
        return;
    }
    if((fdc=open(f,O_WRONLY))<0){ //Apre la client-FIFO
        printf("Impossibile aprire la FIFO in scrittura\n");
        return;
    }
}
```

Soluzione :...server

```
if(write(fdc,"BANG",5)<0) //Scrive nella client-FIFO
    printf("Impossibile scrivere\n");

close(fdc); //Chiude la client-FIFO

/*Può essere usato per far sì che il client
abbia il tempo di leggere dalla FIFO*/
//sleep(1);

kill(pid,SIGKILL); //Uccide il processo client
remove(f); //Rimuove la client-FIFO
}
```

Esercizio

Sia P1 un processo che crea una pipe ed un figlio F1.

Sia P2 un altro processo che comunica con P1 con una FIFO. P2 genera ogni secondo un numero tra 1 e 100 e lo scrive nella FIFO insieme al proprio PID.

P1 per 20 secondi legge i numeri trovati nella FIFO e scrive sulla pipe il proprio PID più il numero letto. P1 dopo 20 sec dalla creazione del primo figlio, scrive sulla pipe il PID di P2 ed il numero -1; quindi stampa un messaggio sullo schermo e termina la sua esecuzione.

F1 legge il numero dalla pipe: se esso è -1 uccide P2 e termina; altrimenti stampa sul terminale il proprio PID seguito dal numero letto dalla pipe.