Domande a Risposta Multipla

Scegliere la risposta più appropriata per ciascuna domanda.

1. Fondamenti di FIFO

1. Qual è la differenza fondamentale tra una Named Pipe (FIFO) e una Unnamed Pipe (Pipe anonima)?
    a. La Pipe anonima è più veloce, mentre la FIFO gestisce solo dati di grandi dimensioni.
    b. La FIFO è un file speciale nel filesystem ed è usata da processi non correlati1, mentre la Pipe anonima è usata solo da processi correlati (genitore/figlio).
    c. La FIFO supporta la comunicazione bidirezionale, mentre la Pipe anonima è solo unidirezionale.
    d. La FIFO non si blocca mai durante l'apertura, mentre la Pipe anonima si blocca sempre.
2. Quale funzione deve essere usata per creare la FIFO sul filesystem, specificando i permessi di accesso?
    a. open(PATH_FIFO, O_CREAT)
    b. pipe()
    c. mkfifo(PATH_FIFO, 0666)
    d. connect(PATH_FIFO)
3. Qual è la conseguenza dell'aprire una FIFO in sola scrittura (O_WRONLY) in modalità bloccante se nessun processo l'ha ancora aperta in lettura?
    a. L'operazione open() restituisce immediatamente -1 e l'esecuzione prosegue.
    b. Il processo scrivente si blocca (attende) finché un processo non apre la FIFO in lettura.
    c. Il sistema operativo genera un errore e rimuove la FIFO.
    d. La FIFO viene automaticamente aperta in lettura dal sistema.
4. Se la funzione read() restituisce 0, cosa significa in relazione ad un processo in scrittura?
    a. La FIFO è temporaneamente vuota (risposta tipica in non bloccante con EAGAIN).
    b. Si è verificato un errore critico (perror).
    c. Il processo che stava scrivendo ha chiuso la sua estremità di scrittura, terminando la comunicazione.
    d. Il processo ha letto tutti i byte disponibili nel buffer.
5. Perché il Produttore in produttore.c utilizza strlen(buffer) + 1 quando chiama la funzione write()?
    a. Per includere un byte di controllo che indica la fine del messaggio.
    b. Per scrivere solo la stringa utile (senza il terminatore nullo).
    c. Per aggiungere un'intestazione al messaggio.
    d. Per includere il terminatore nullo (\0) nella trasmissione.

Completamento di Codice (Sintassi e Applicazione)

Es1

Il seguente codice mostra la creazione e l'apertura della FIFO:

```
#define PATH_FIFO "[... 1 ]"

if (mkfifo (PATH_FIFO, [ ... 2 ] ) == -1) {
    perror("Errore nella creazione della fifo");
    return 1;
}

printf("Produttore: In attesa dell'apertura del Consumatore...\n");
fd = open (PATH_FIFO, [ ... 3 ] );
if (fd == -1) {
    perror("Errore nell'apertura della FIFO in scrittura");
    return 1;
}
```

Completamento:

1. [... 1]:
2. [... 2]:
3. [... 3]:

Es2:

```
fd = open(PATH_FIFO, [ ... 4 ] | O_NONBLOCK);

} else { // bytes_input == -
    //Errore dovuto a FIFO vuota
    if (errno == [ ... 5 ] || errno == EWOULDBLOCK) {
        printf("Consumatore: FIFO vuota. Nessun dato disponibile (attesa di 1s)...\n");
        sleep(1);
    } else {
        perror("Errore durante la lettura dalla FIFO");
        close(fd);
        return EXIT_FAILURE;
    }
}
```

Completamento:

4. [... 4]:
5. [... 5]:

ESERCIZI RIPASSO

1. Creare due programmi, **Produttore** e **Consumatore**, che utilizzano una FIFO per scambiare un piccolo **array** di numeri interi.

   Produttore:
   - Crea la FIFO (es. `/tmp/my_fifo`).
   - Definisce un array di 5 numeri interi.
   - Apre la FIFO in **scrittura**.
   - Scrive l'intero array nella FIFO (tutti e 5 gli interi in un'unica operazione di `write`).
   - Chiude e rimuove la FIFO.

   Consumatore:
   - Apre la FIFO in **lettura**.
   - Legge un blocco di dati pari alla dimensione dell'array (5 interi).
   - Stampa i numeri interi ricevuti.

- Come si calcola la dimensione esatta in **byte** dell'array per la `read`/`write`?

2. Creare un **Client** che invia continuamente una **struttura** (array logico) di coordinate al **Server**, che le elabora in tempo reale.

```
typedef struct {
   int x;
   int y;
} Coordinate;
// O un array di due interi int coord[2];
```

   Server (Consumatore):
   - Crea la FIFO.
   - Apre la FIFO in lettura e si mette in attesa.
   - Entra in un ciclo infinito.
   - In ogni iterazione, legge una singola struttura `Coordinate` dalla FIFO.
   - Stampa la coordinata ricevuta e il suo **quadrante** (es. "I Quadrante", "Asse X", ecc.).

   Client (Produttore):
   - Apre la FIFO in scrittura.
   - Entra in un ciclo che invia, diciamo, 10 coordinate casuali o predefinite (una alla volta, con una piccola pausa `sleep(1)` tra un invio e l'altro).
   - Chiude la FIFO.

- Il Server deve gestire la condizione in cui la `read` ritorna **0** (il Client ha chiuso la FIFO), uscendo dal ciclo.
- Il Client deve gestire il caso in cui la FIFO non è ancora stata aperta dal Server.

3. Simulare un protocollo in cui il Produttore invia messaggi di diversa lunghezza, preceduti da un intero che ne specifica la dimensione.

   Produttore:
   - Crea la FIFO.
   - Prepara 3 stringhe (array di caratteri) di lunghezza diversa (es. "Ciao", "Messaggio più lungo", "Fine").

    - Per ogni stringa:
      a. Calcola la lunghezza della stringa (es. strlen(str)).
      b. Scrive l'intero che rappresenta la lunghezza nella FIFO.
      c. Scrive la stringa stessa nella FIFO.
    - Chiude e rimuove la FIFO.
2. **Consumatore:**
    - Apre la FIFO.
    - Entra in un ciclo che si interrompe solo quando la read per la lunghezza fallisce (o ritorna 0).
    - Legge un intero (la **lunghezza** del messaggio successivo).
    - Alloca dinamicamente un buffer di quella dimensione (malloc).
    - Legge il numero esatto di byte specificato (il **messaggio** vero e proprio).
    - Stampa il messaggio.

- Questa è la variante più realistica per lo streaming di dati complessi.
- È necessario effettuare **due** operazioni di read consecutive (una per la lunghezza, una per il dato).