Comunicazione Inter-Processo (IPC) con FIFO (Named Pipes)

Analisi del Modello Produttore-Consumatore

17 novembre 2025

1 Introduzione all'IPC e FIFO

La **Comunicazione Inter-Processo (IPC)** è un meccanismo che consente ai processi di un sistema operativo di scambiarsi informazioni. Le **FIFO**, o *Named Pipes*, sono uno dei metodi di IPC più semplici e potenti.

- A differenza delle pipe anonime, una FIFO ha un nome nel filesystem (es. `/tmp/dati_fifo`), il che permette a processi non imparentati (ovvero non creati tramite `fork()` da un antenato comune) di utilizzarla.
- Opera in modalità **half-duplex**: i dati scorrono in una sola direzione (o si legge o si scrive).
- Il meccanismo è di tipo **bloccante** di default: un processo che tenta di aprire una FIFO in lettura attenderà che un altro processo l'apra in scrittura, e viceversa.

2 Il Programma Produttore (`produttore.c`)

Il ruolo del Produttore è quello di creare la FIFO, scriverci dati e, una volta terminato il trasferimento, rimuoverla dal filesystem.

2.1 Creazione e Apertura

Il codice C utilizza la funzione `mkfifo()` per creare la named pipe e `open()` in modalità `O_WRONLY` per la scrittura.

```
// 1. Crea la FIFO con permessi 0666
if (mkfifo(FIFO_PATH, 0666) == -1) {
    perror("Errore nella creazione della FIFO (mkfifo)");
}

// 2. Apre la FIFO in modalità scrittura (bloccante)
fd = open(FIFO_PATH, O_WRONLY);
if (fd == -1) {
    perror("Errore nell'apertura della FIFO per scrittura");
    return 1;
}
```

Listing 1: Estratto: Creazione e Apertura FIFO nel Produttore

2.2 Invio dei Dati (Stringhe)

Il Produttore invia `NUM_MESSAGES` stringhe. È cruciale che la chiamata `write()` includa il byte nullo (`\0`) per garantire che il Consumatore possa interpretare correttamente il buffer come una stringa C.

```
// Prepara la stringa
snprintf(buffer, MAX_MSG_SIZE, "Messaggio %d dal Produttore (PID %d)", i, getpid());

// Invia la stringa, incluso il terminatore '\0'
if (write(fd, buffer, strlen(buffer) + 1) == -1) {
    perror("Errore nella scrittura sulla FIFO");
    break;
}
```

Listing 2: Estratto: Scrittura di Messaggi Stringa

2.3 Chiusura e Pulizia

Dopo aver inviato tutti i messaggi, il Produttore chiude il descrittore di file (`close(fd)`) e rimuove la FIFO dal filesystem tramite `unlink()`.

3 Il Programma Consumatore (`consumatore.c`)

Il Consumatore attende che la FIFO sia disponibile, legge i dati e li processa.

3.1 Apertura e Lettura

Il Consumatore apre la FIFO in modalità `O_RDONLY`. Questa chiamata si bloccherà fino a quando il Produttore non apre la FIFO in scrittura, rispettando la semantica bloccante.

```
// 1. Apre la FIFO in modalità lettura (bloccante)
fd = open(FIFO_PATH, O_RDONLY);
if (fd == -1) {
    perror("Errore nell'apertura della FIFO per lettura");
    return 1;
}

// 2. Legge le stringhe in loop
bytes_read = read(fd, buffer, MAX_MSG_SIZE);

if (bytes_read > 0) {
    // Il Produttore ha inviato '\0', quindi lo sostituiamo per la stampa.
    buffer[bytes_read - 1] = '\0';
    printf("Consumatore: Ricevuto: %s\n", buffer);
}
```

Listing 3: Estratto: Apertura e Lettura FIFO nel Consumatore

3.2 Segnalazione di Fine Flusso

La condizione `bytes_read == 0` è fondamentale in quanto indica che tutti i processi che avevano aperto la FIFO in scrittura hanno chiuso il loro descrittore di file. Questo è il segnale per il Consumatore che non ci saranno più dati.

4 Variazione: Scambio di Strutture Dati

Sebbene l'esempio iniziale scambi stringhe, le FIFO sono ottimali per trasferire blocchi di dati a dimensione fissa, come le strutture C. In questo caso, il Consumatore deve leggere esattamente la dimensione del tipo di dato atteso.

4.1 Definizione e Invio della Struttura

```c
struct record {
    int id;
    float valore;
};
#define NUM_RECORDS 5
```

Listing 4: Definizione della Struttura (Identica per Entrambi i Processi)

Nel Produttore, l'invio avviene con: `write(fd, &rec, sizeof(struct record))`.

4.2 Lettura della Struttura

Nel Consumatore, la lettura avviene con `read(fd, &rec, sizeof(struct record))`. La logica di lettura cambia: non si cerca più il terminatore `\0`, ma si verifica che i byte letti siano esattamente `sizeof(struct record)`.

```c
// Legge esattamente la dimensione della struttura
bytes_read = read(fd, &rec, sizeof(struct record));

if (bytes_read == sizeof(struct record)) {
    // Successo: si processa la struttura
    printf("Consumatore: Ricevuto Record -> ID: %d, Valore: %.2f\n", rec.id, rec.valore);
} else if (bytes_read == 0) {
    // Fine del flusso
}
```

Listing 5: Estratto: Lettura di Strutture Dati Fisse

5 Esecuzione

Per eseguire il modello Produttore-Consumatore si utilizzano due terminali separati.

1. **Compilazione:**

   ```
   gcc produttore.c -o produttore
   gcc consumatore.c -o consumatore
   ```

2. **Esecuzione (Terminali Separati):**

   - **Terminale 1:** Eseguire il Consumatore. Si bloccherà in attesa che la FIFO venga aperta in scrittura.

     ```
     ./consumatore
     ```

   - **Terminale 2:** Eseguire il Produttore. Dopo l'apertura in scrittura, inizierà a inviare i dati.

     ```
     ./produttore
     ```

6 Soluzioni Complete del Codice C

Questa sezione fornisce le implementazioni complete dei programmi Produttore e Consumatore per i modelli base discussi.

6.1 Versione 1: Scambio di Stringhe

6.1.1 Produttore (Stringhe)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>

#define FIFO_PATH "/tmp/dati_fifo"
#define MAX_MSG_SIZE 50
#define NUM_MESSAGES 10

int main() {
    int fd;
    char buffer[MAX_MSG_SIZE];

    if (mkfifo(FIFO_PATH, 0666) == -1 && errno != EEXIST) {
        perror("mkfifo");
        return 1;
    }

    fd = open(FIFO_PATH, O_WRONLY);
    if (fd == -1) {
        perror("open O_WRONLY");
        return 1;
    }

    for (int i = 1; i <= NUM_MESSAGES; i++) {
        snprintf(buffer, MAX_MSG_SIZE, "Messaggio %d dal Produttore (PID %d)", i, getpid
    ());

        if (write(fd, buffer, strlen(buffer) + 1) == -1) {
            perror("write");
            break;
        }
        sleep(1);
    }

    close(fd);
    unlink(FIFO_PATH);
    return 0;
}
```

Listing 6: produttore_stringhe.c

6.1.2 Consumatore (Stringhe)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

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

#define FIFO_PATH "/tmp/dati_fifo"
#define MAX_MSG_SIZE 50
#define NUM_MESSAGES 10

int main() {
    int fd;
    char buffer[MAX_MSG_SIZE];
    int bytes_read;
    int received_count = 0;

    fd = open(FIFO_PATH, O_RDONLY);
    if (fd == -1) {
        perror("open O_RDONLY");
        return 1;
    }

    while (received_count < NUM_MESSAGES) {
        bytes_read = read(fd, buffer, MAX_MSG_SIZE);

        if (bytes_read > 0) {
            buffer[bytes_read - 1] = '\0';
            received_count++;
            printf("Consumatore: Ricevuto [%d/%d]: %s\n", received_count, NUM_MESSAGES,
    buffer);
        } else if (bytes_read == 0) {
            break;
        } else {
            perror("read");
            break;
        }
    }

    close(fd);
    return 0;
}
```

Listing 7: consumatore_stringhe.c

6.2 Versione 2: Scambio di Strutture Dati

6.2.1 Produttore (Strutture)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>

#define FIFO_PATH "/tmp/dati_fifo"
#define NUM_RECORDS 5

struct record {
    int id;
    float valore;
```

```c
};

int main() {
    int fd;
    struct record rec;

    if (mkfifo(FIFO_PATH, 0666) == -1 && errno != EEXIST) {
        perror("mkfifo");
        return 1;
    }

    fd = open(FIFO_PATH, O_WRONLY);
    if (fd == -1) {
        perror("open O_WRONLY");
        return 1;
    }

    for (int i = 1; i <= NUM_RECORDS; i++) {
        rec.id = i * 100;
        rec.valore = (float)i * 3.14;

        if (write(fd, &rec, sizeof(struct record)) == -1) {
            perror("write");
            break;
        }
        sleep(1);
    }

    close(fd);
    unlink(FIFO_PATH);
    return 0;
}
```

Listing 8: produttore_strutture.c

6.2.2 Consumatore (Strutture)

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

#define FIFO_PATH "/tmp/dati_fifo"
#define NUM_RECORDS 5

struct record {
    int id;
    float valore;
};

int main() {
    int fd;
    struct record rec;
    int bytes_read;
    int received_count = 0;

    fd = open(FIFO_PATH, O_RDONLY);
    if (fd == -1) {
        perror("open O_RDONLY");
```

```
26          return 1;
27      }
28
29      while (received_count < NUM_RECORDS) {
30          bytes_read = read(fd, &rec, sizeof(struct record));
31
32          if (bytes_read == sizeof(struct record)) {
33              received_count++;
34              printf("Consumatore: Ricevuto Record [%d/%d] -> ID: %d, Valore: %.2f\n",
35                     received_count, NUM_RECORDS, rec.id, rec.valore);
36          } else if (bytes_read == 0) {
37              break;
38          } else if (bytes_read == -1) {
39              perror("read");
40              break;
41          } else {
42              fprintf(stderr, "Errore: Lettura incompleta (letti %d bytes, attesi %zu)\n",
    bytes_read, sizeof(struct record));
43              break;
44          }
45      }
46
47      close(fd);
48      return 0;
49  }
```

Listing 9: consumatore_strutture.c