Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative
Istituto sistemi informativi e networking

SUPSI

I/O Asincrono

Amos Brocco, Ricercatore, DTI / ISIN

13 marzo 2012

Input / Output sincrono

- Operazione I/O sincrona nel thread principale
  - Il programma è bloccato durante I/O


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

#define NBYTES 10000

void main(void)
{
        char* buffer[NBYTES];
        unsigned int bytes;

        /* Apro il file da leggere */
        FILE* file = fopen("/var/log/syslog", "r");

        bytes = read(fileno(file), buffer, NBYTES);

        printf("Lettura sincrona, bytes letti %d.\n", bytes);

        close(fileno(file));
}
```

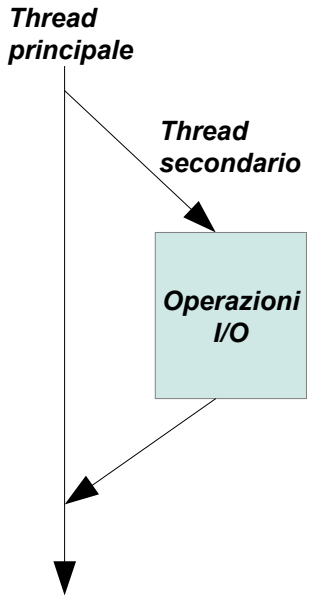
Input / Output in un thread separato

- Operazione di I/O con un thread separato

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#define NBYTES 10000
FILE* file;
char* buffer[NBYTES];
unsigned int bytes;

void* lettore() {
          bytes = read(fileno(file), buffer, NBYTES);
}

void main(void) {
          pthread_t lett;
          /* Apro il file da leggere */
          file = fopen("/var/log/syslog", "r");
          pthread_create(&lett, NULL, &lettore, NULL);
          printf("Thread lettore creato... posso fare altro\n");
          sleep(5);
          pthread_join(lett, NULL);
          printf("Lettura sincrona, bytes letti %d.\n", bytes);
          close(fileno(file));
}
```



IO Asincrono POSIX

- POSIX definisce una API specifica per le operazioni di input/output asincrone
  - la richiesta di input/output termina senza attendere la conclusione dell'operazione di trasferimento dati
  - La disponibilità dei dati (input) o la fine del trasferimento (output) viene comunicata al programma mediante un segnale
  - Il segnale può essere associato a una funzione asincrona che può servire a compiere operazioni sui dati disponibili o trasferiti
  - Il meccanismo è applicabile a molti canali I/O seriali (periferiche seriali, schede di rete, pipe,...)

- Per compilare è necessario specificare l'utilizzo della libreria LibRt

```
gcc -o programma programma.c -lrt
```

Input / Output asincrono di POSIX

- È un insieme di funzioni e strutture dati definite nel modulo `aio.h`
- Funzioni disponibili:
  - Richiesta di lettura
  - Scrittura
  - Annullamento della richiesta
  - Trasferimento dei dati disponibili (input)
  - Attesa della conclusione dell'operazione
  - Analisi degli errori

Lettura e scrittura

- Lettura asincrona

```
#include <aio.h>

int aio_read   struct aiocb   *aiocb
```

- Scrittura asincrona

```
int aio_write   struct aiocb   *aiocb
```

- Annullamento di un'operazione in corso

```
int aio_cancel   int            fildes
                 struct aiocb   *aiocb
```

Siccome sullo stesso file ci possono essere più operazioni in corso è necessario specificare quale operazione annullare; se il secondo argomento è NULL, vengono annullate tutte)

Strutture dati

Tipo	Membro	Significato
int	`aio_fildes`	descrittore
volatile void	`aio_buf`	indirizzo *buffer* di input/output
size_t	`aio_nbytes`	medesima definizione di `nbytes` di `read()/write()`
off_t	`aio_offset`	*offset* all'interno del file
int	`aio_opcode`	vedi `lio_listio()`
struct sigevent	`aio_sigevent`	

Tabella 3.4: Alcuni campi della struttura `aiocb`

Tipo	Membro	Valore/Significato
int	sigevent_notify	`SIGEV_SIGNAL`→ notifica con segnale
		`SIGEV_NONE`→ nessuna notifica
union sigval	sigevent_value	valore accompagnatorio
int	sigev_signo	numero del segnale da inviare

Tabella 3.5: Campi della struttura `sigevent`

Risultato e errori

- Valore di ritorno

```
#include <aio.h>

ssize_t aio_return   struct aiocb   *aiocb
```

- Analisi degli errori

```
#include <aio.h>

int aio_error   const struct aiocb   *aiocb
```

I/O asincrono con attesa attiva

```
#include <aio.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define NBYTES 10000
struct aiocb callback;

void main(void)
{
  char* buffer[NBYTES];

  /* Apro il file da leggere */
  FILE* file = fopen("/var/log/syslog", "r");
```

I/O asincrono con attesa attiva

```
  /* Gestione lettura asincrona */
  callback.aio_buf = buffer;
  callback.aio_fildes = fileno(file);
  callback.aio_nbytes = NBYTES; /* Quanti bytes leggere */
  callback.aio_offset = 0; /* Da che posizione  */

  /* Inizio la lettura asincrona */
  aio_read(&callback);

  printf("Attesa attiva...\n");
  /* Attesa attiva */
  while(aio_error(&callback) == EINPROGRESS) {
    sleep(1);
  }
  printf("Lettura asincrona, bytes letti %d.\n",
                (int) aio_return(&callback));

  close(fileno(file));
}
```

I/O asincrono con segnale di callback

```
#include <aio.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#define NBYTES 10000
struct sigaction action;
struct aiocb callback;

void lettura_terminata(int signal, siginfo_t *info, void *uap)
{
  int operazione = info->si_value.sival_int;
  /* Nota: non è sicuro usare printf qui, ma per questo esempio va bene */
  printf("Lettura asincrona %d, bytes letti %d. Premi CTRL+C per uscire. \n",
                operazione,
                (int) aio_return(&callback));

  close(callback.aio_fildes);
}
```

I/O asincrono con segnale di callback

```
void main(void)
{
  char* buffer[NBYTES];

  /* Gestore segnale di callback */
  action.sa_sigaction = lettura_terminata;
  action.sa_flags = SA_SIGINFO;
  sigemptyset(&action.sa_mask);
  sigaction(SIGRTMIN+7, &action, NULL);

  /* Apro il file da leggere */
  FILE* file = fopen("/var/log/syslog", "r");

  /* Gestione lettura asincrona */
  callback.aio_buf = buffer;
  callback.aio_fildes = fileno(file);
  callback.aio_nbytes = NBYTES; /* Quanti bytes leggere */
  callback.aio_offset = 0; /* Da che posizione  */
```

I/O asincrono con segnale di callback

```
  /* Definizione segnale da inviare */
  callback.aio_sigevent.sigev_notify = SIGEV_SIGNAL;
  callback.aio_sigevent.sigev_signo = SIGRTMIN+7;
  callback.aio_sigevent.sigev_value.sival_int = 13; /* Id operazione */

  /* Inizio la lettura asincrona */
  aio_read(&callback);

  /* In attesa (termina con CTRL+C) */
  printf("Mi metto in attesa\n");
  while(1) {
    sleep(1);
  }
}
```