
# Reti di Calcolatori - Laboratorio

## Lezione 4

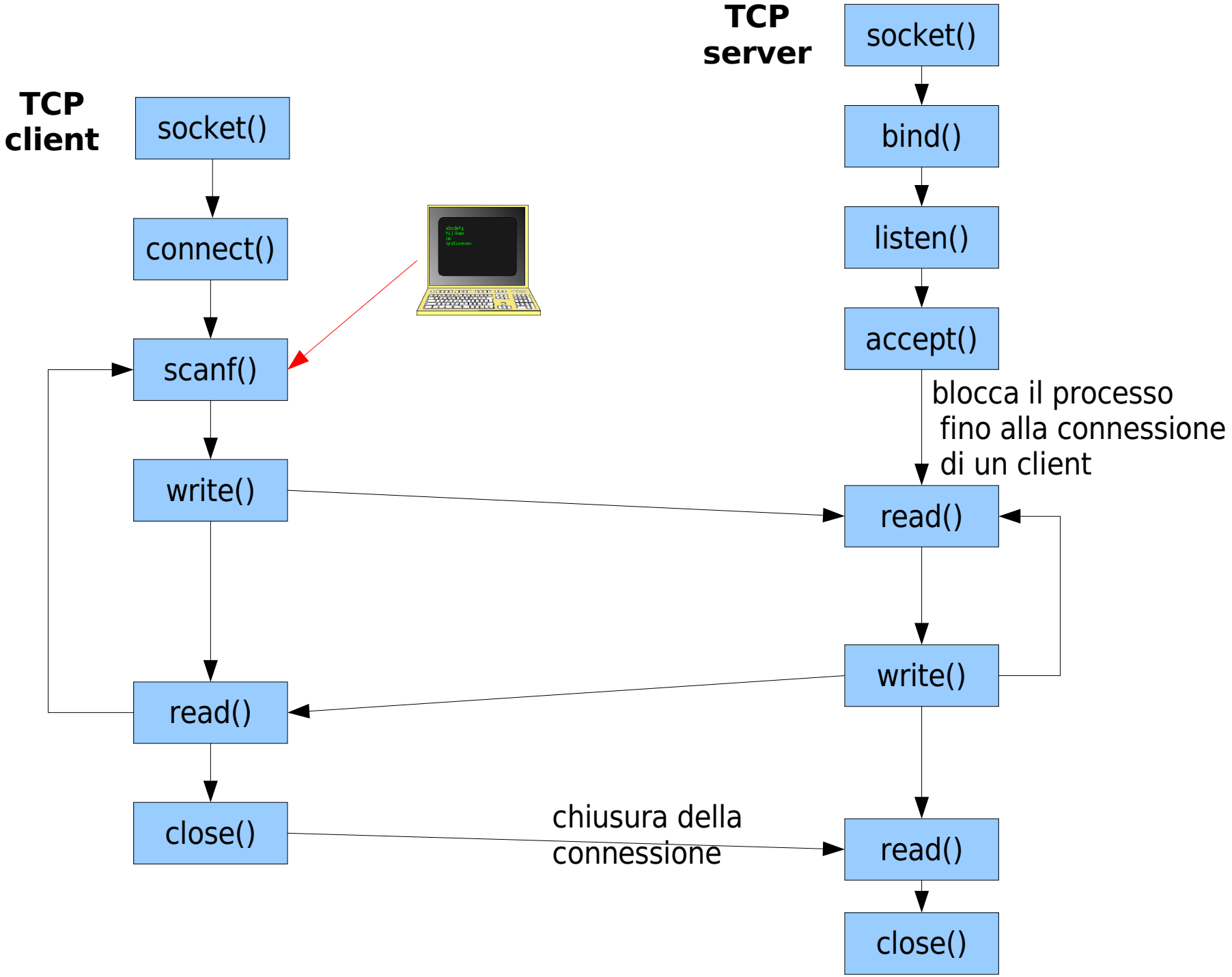
Gennaro Oliva

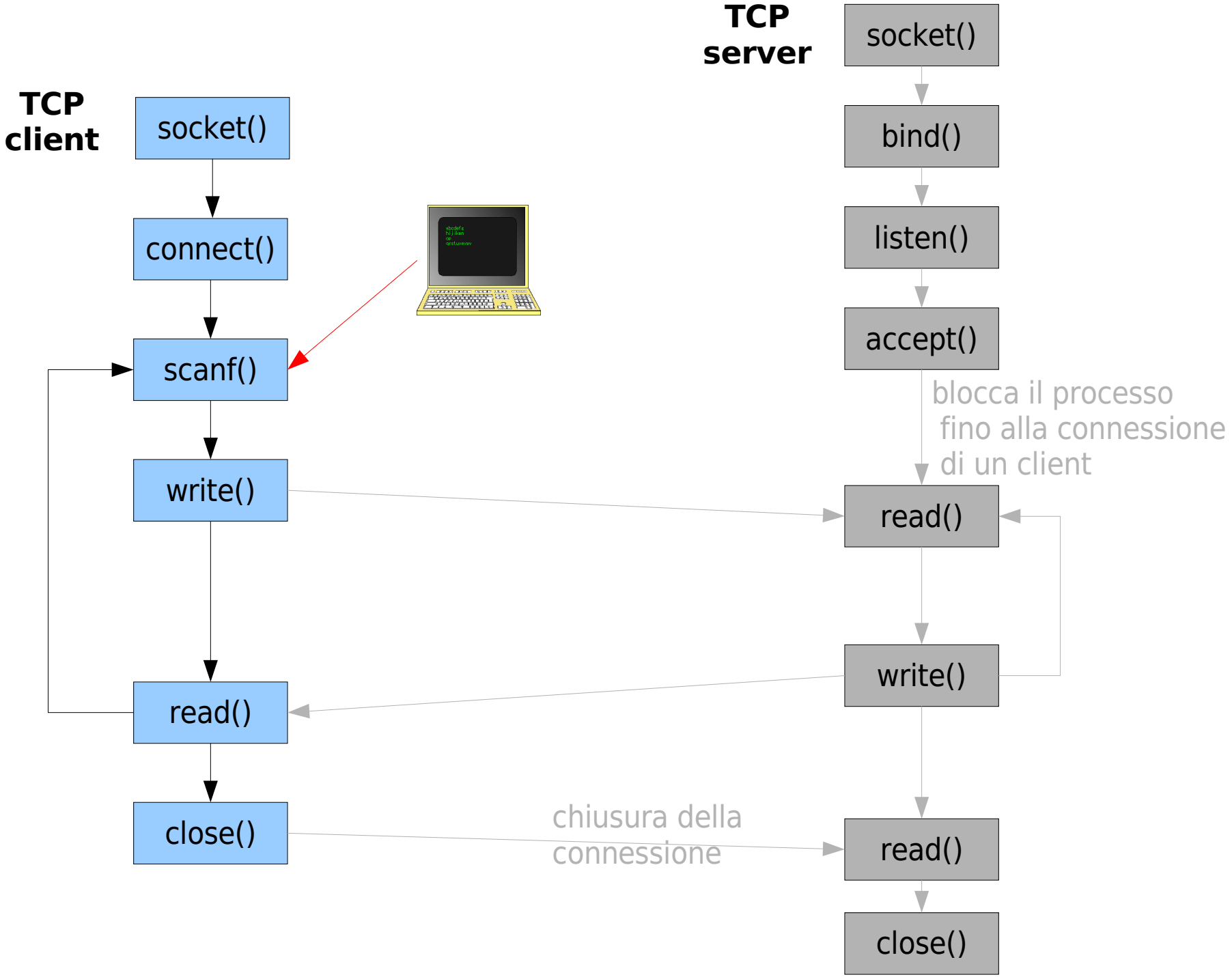
# Applicazioni che gestiscono diversi input

- Esercizio: Scrivere un server concorrente che accetti dai client stringhe di caratteri e restituisca il numero di caratteri in esse contenute
- L'applicazione client deve gestire più input simultaneamente
  - Standard input (leggere da tastiera)
  - Un socket (leggere dal socket)

# Applicazioni che gestiscono diversi input

- Mentre l’applicazione è bloccata in operazione di lettura non si accorge cosa accade nell'altro canale di comunicazione
- Nel nostro caso: cosa accade se il client in e' attesa di input da parte dell'utente e il server cessa l'esecuzione
- E' necessario un meccanismo che consenta di esaminare più canali di input contemporaneamente e accedere primo canale che produce dati


TCP server
socket()
bind()
listen()
accept()
blocca il processo fino alla connessione di un client
read()
write()
read()
TCP client
socket()
connect()
scanf()
write()
read()
close()
chiusura della connessione


TCP server
socket()
bind()
listen()
accept()
blocca il processo fino alla connessione di un client
read()
write()
read()
TCP client
socket()
connect()
scanf()
write()
read()
close()
chiusura della connessione

# Modelli I/O in ambiente UNIX

- Esistono vari modelli di I/O disponibili in ambiente Unix:
  - Bloccante
  - Non Bloccante
  - I/O Multiplex
  - I/O attivato da segnali
  - I/O asincrono

# Le fasi di un operazione di input

- Distinguiamo due fasi per le operazioni di input:
  - l'attesa per la disponibilità dei dati
  - la copia dei dati dalla memoria del kernel al processo
- Nel caso dei socket la prima fase e' l'attesa che i dati arrivino dalla rete
- La seconda fase puo' essere gestita dalla funzione recvfrom

# recvfrom

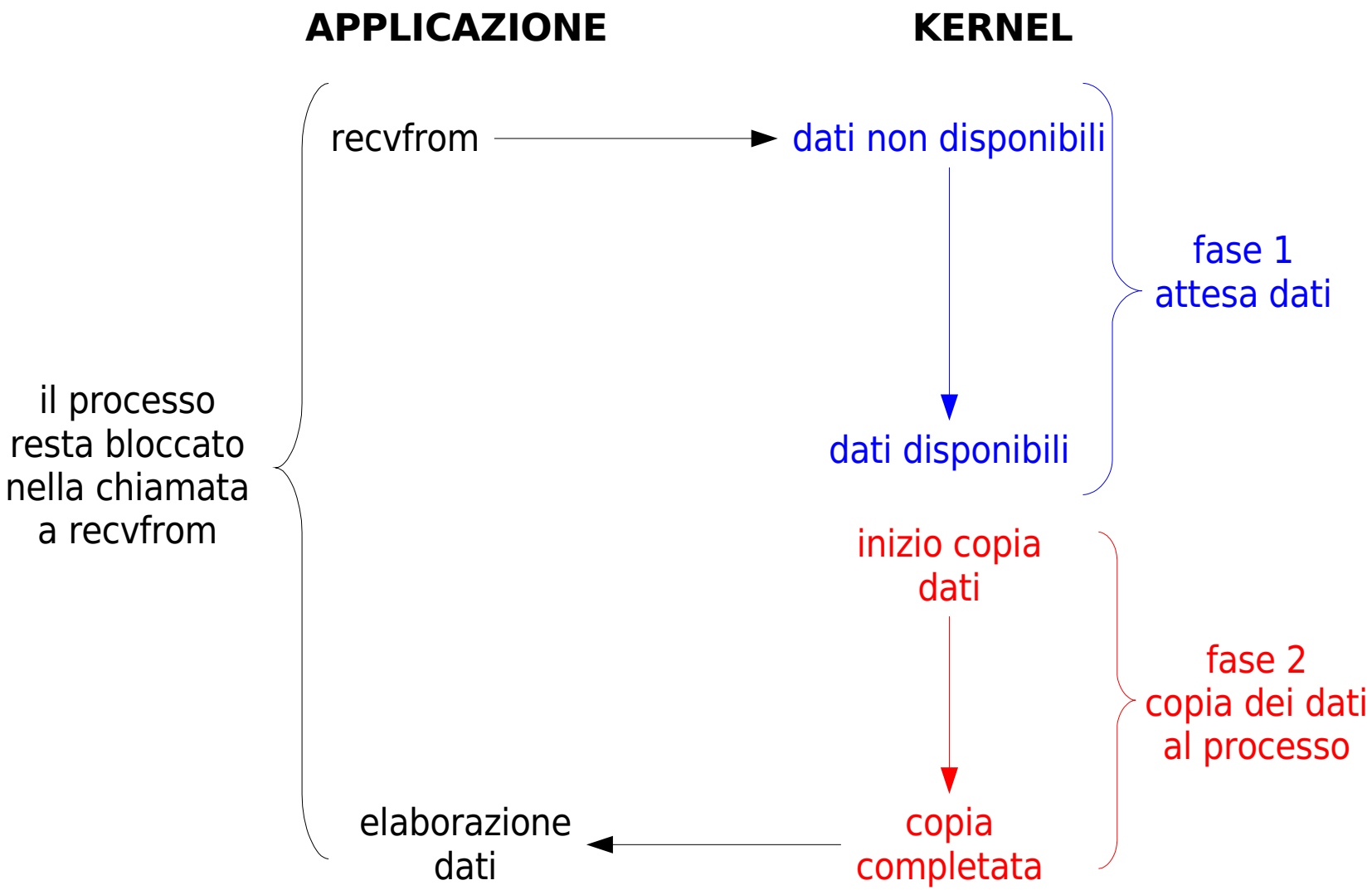
ssize_t recvfrom(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *from, socklen_t *fromlen)

- Riceve un messaggio da un socket
  - sockfd - socket descriptor
  - buf - buffer di memorizzazione del messaggio
  - len - lunghezza del buffer
  - flags - imposta la modalità di funzionamento della comunicazione
  - from - memorizza l'indirizzo del mittente
  - fromlen – memorizza la dimensione di from

# I/O bloccante

- Modello predominante
- Il processo  attende durante entrambi le fasi
- L'esecuzione del processo utente si blocca nella chiamata a recvfrom e riprende l'esecuzione solo quando questa viene soddisfatta o si verifica un errore
- Lo stato del processo in attesa e' sleeping
- I socket sono di default bloccanti

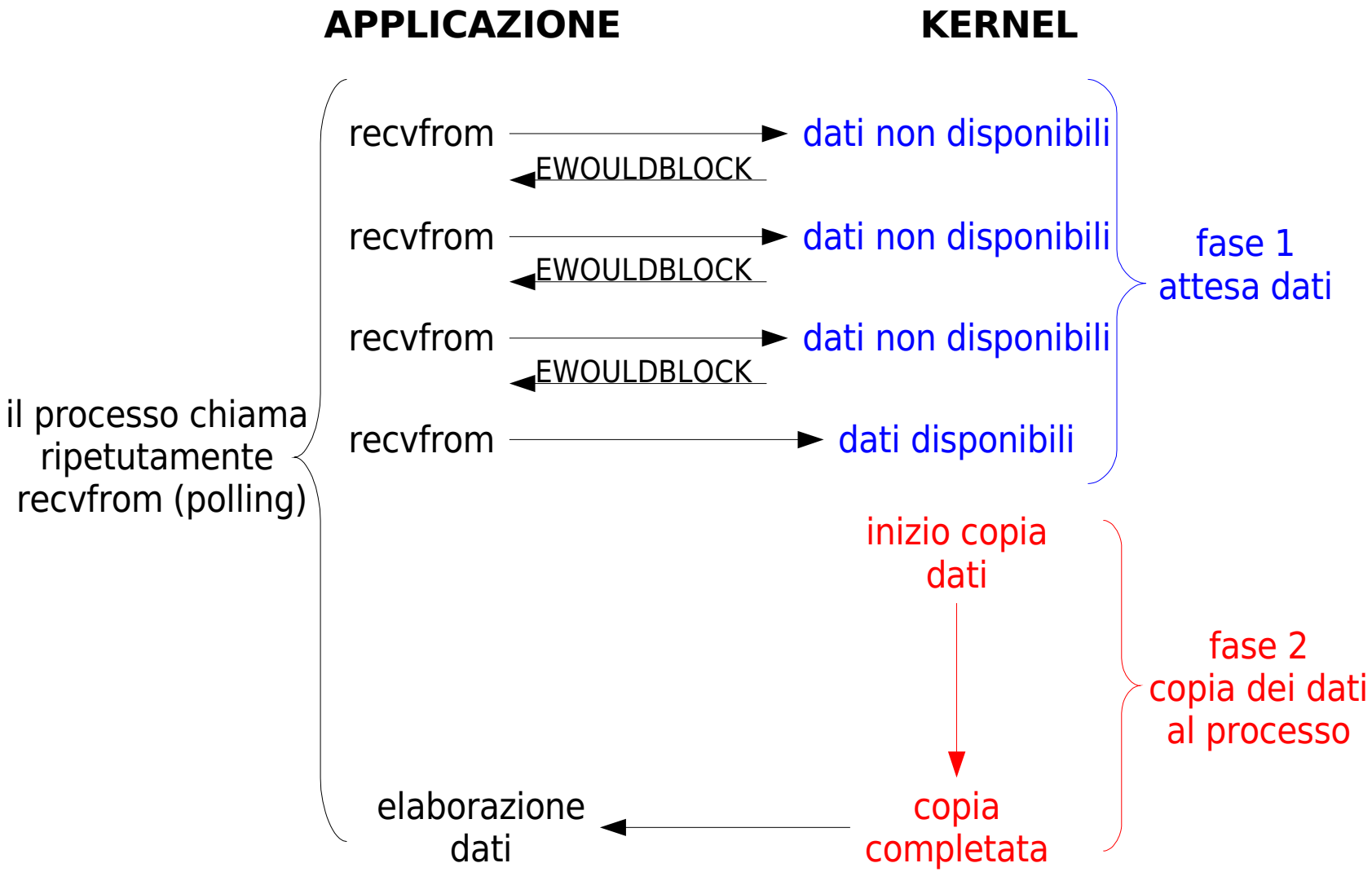
# I/O bloccante



# I/O non bloccante

- Un socket non bloccante restituisce un errore ogni volta che si richiede un'operazione di I/O non ancora effettuabile
- Solitamente la richiesta viene reiterata fino a quando non si ottiene una risposta positiva
- Questa pratica viene detta polling
- Il polling e' uno spreco di tempo di CPU
- Viene utilizzato principalmente nei sistemi dedicati

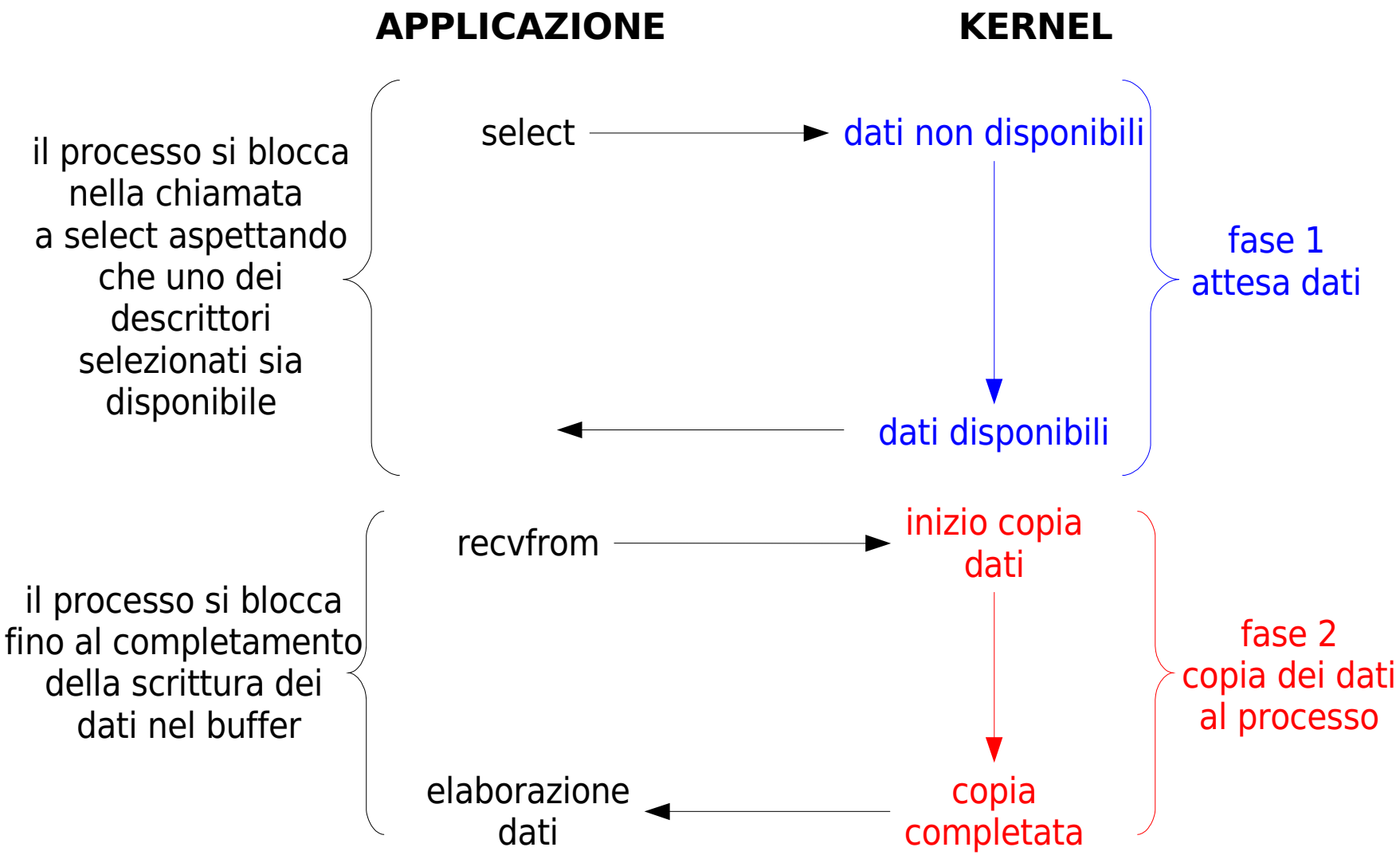
# I/O non bloccante



# I/O Multiplex

- Nel modello di I/O multiplex il processo resta in attesa di eventi su uno o più descrittori
- l'esecuzione si blocca fino a quando uno dei descrittori diventa pronto
- Il vantaggio nell'uso di questo modello e' che si possono monitorare piu' canli di comunicazione

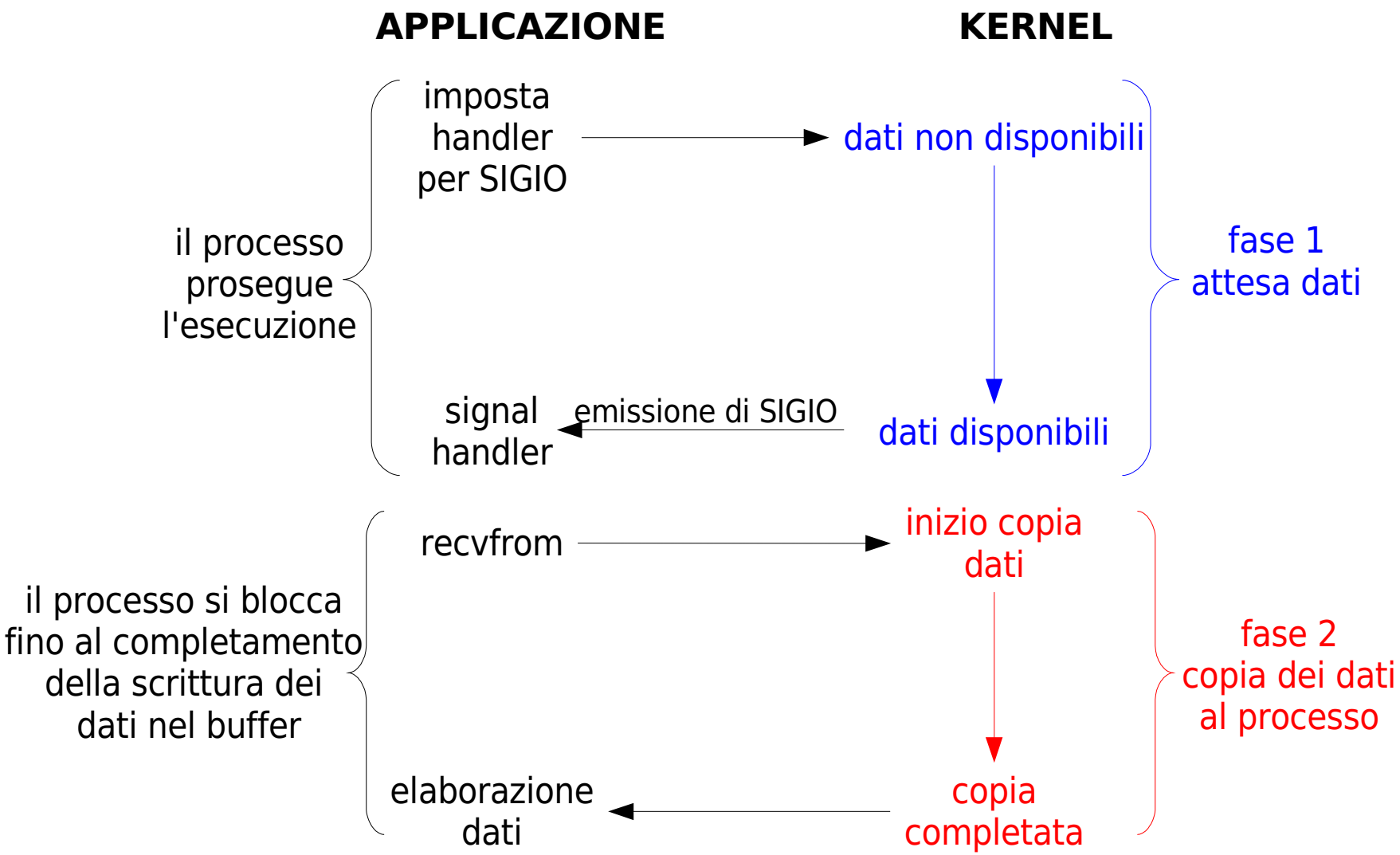
# I/O Multiplex



# I/O controllato da segnali

- Il kernel notifica al processo in escecuzione che il canale di comunicazione è pronto inviando il segnale SIGIO
- Il processo deve impostare un handler per SIGIO

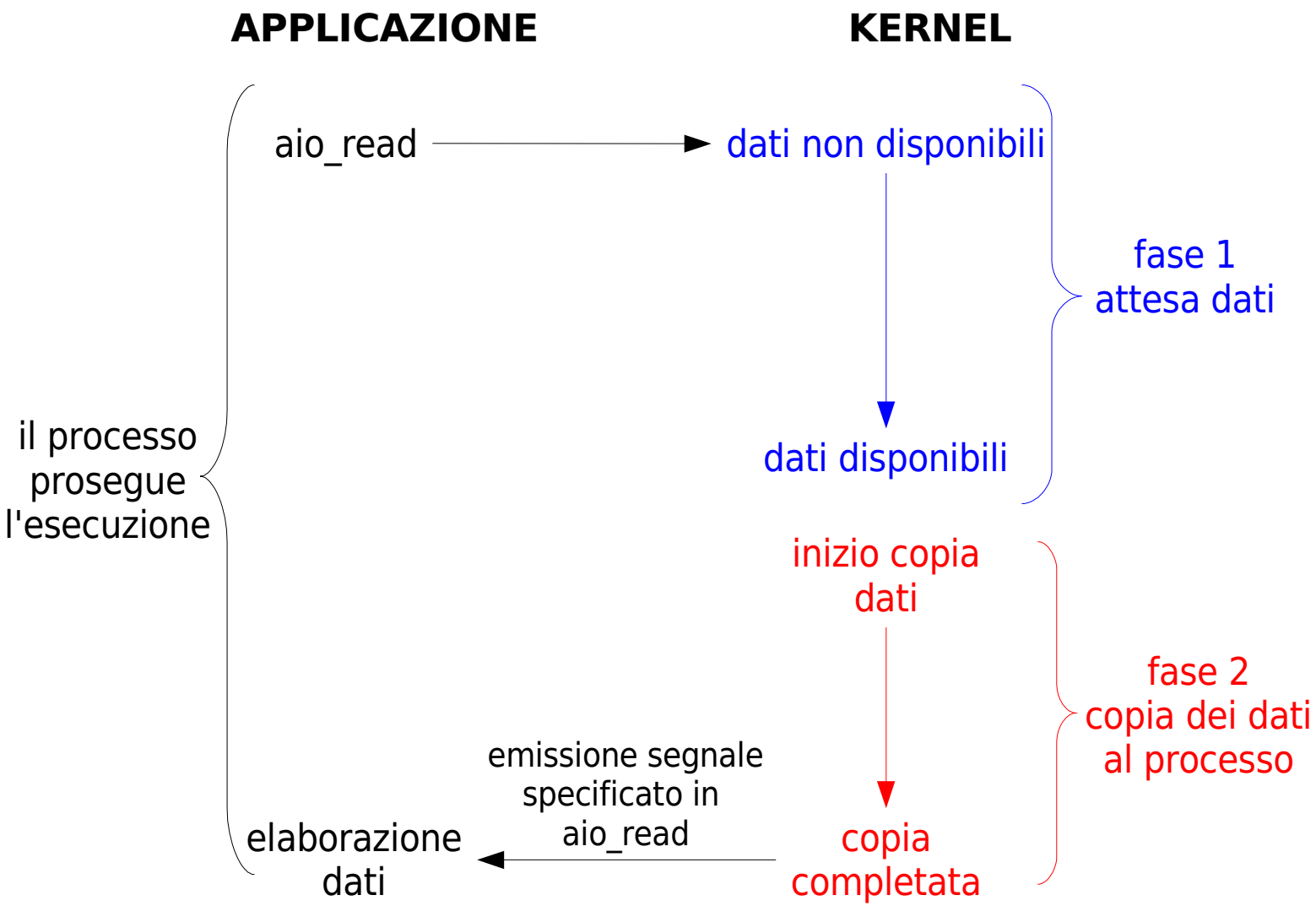
# I/O controllato da segnali



# I/O asincrono

- Definito dallo standard POSIX
- Il kernel si prende carico di entrambi le fasi dell'operazione di I/O
- Una volta che l'operazione e' stata completata viene invato un segnale al processo
- il processo avvia l'operazione comunicando al kernel il descritore, il buffer e il segnale con cui notificare la completazione dell'operazione

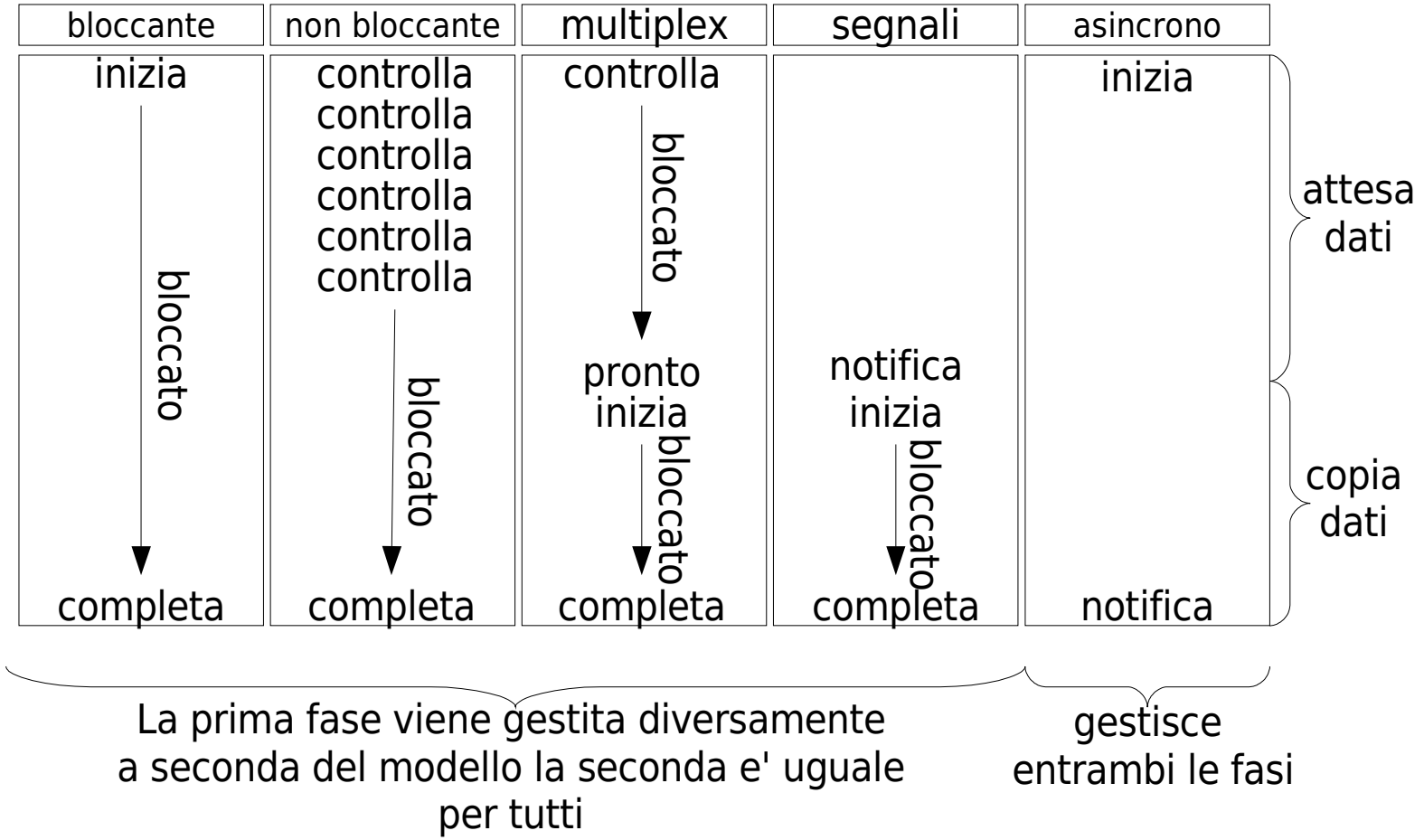
# I/O asincrono



# Thread ed I/O

- Un'altro modello popolare per gestire I/O multipli è quello che utilizza i thread
- Ogni thread
  - gestisce un file descriptor
  - rimane bloccato sul proprio file descriptor

# Riepilogo


bloccante
non bloccante
multiplex
segnali
asincrono
inizia
bloccato
completa
controlla
controlla
controlla
controlla
controlla
controlla
bloccato
completa
controlla
bloccato
pronto
inizia
bloccato
completa
notifica
inizia
bloccato
completa
inizia
notifica
attesa
dati
copia
dati
La prima fase viene gestita diversamente
a seconda del modello la seconda e' uguale
per tutti
gestisce
entrambi le fasi

# select

- La funzione select comunica al kernel di monitorare un insieme di descrittori, ponendo il processo in sleeping e risvegliandolo quando si verifica un evento
- Esempi di eventi sono:
  - Uno dei descrittori {1, 4, 5} e' pronto per la lettura
  - Uno dei descrittori {2, 7} è pronto per la scrittura
  - Su uno dei descrittori {1, 4} si e' verificata un'eccezione
  - sono passati 10.2 secondi

# Descrittori

- I descrittori che possono essere specificati non devono essere necessariamente socket ma e' possibile specificare anche file descriptor
- Il kernel identifica ogni file aperto con un file descriptor ovvero un interno non negativo
- Quando si apre un file esistente o si crea un un nuovo file il kernel restituisce un file descriptor al processo

# file descriptor

- Il file descriptor identifica il file nelle operazioni di lettura e scrittura
- Per convenzione le shell Unix associano:
  - 0 standard input (STDIN_FILENO)
  - 1 standard output (STDOUT_FILENO)
  - 2 standard error (STDERR_FILENO)
- Per conoscere il file descriptor associato ad un file sream (FILE *) si utilizza la funzione fileno

# select

int select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, const struct timeval *timeout);

- Restituisce
  - -1 in caso di errore
  - 0 in caso di timeout
  - il numero di descrittori pronti
- Permette di controllare contemporaneamente uno o più descrittori per lettura, scrittura o la presenza di errori

# timeout

- timeout è il tempo massimo che la system call attende per individuare un descrittore pronto
- Il timeout si specifica in una struttura di tipo timval

```
struct timeval {
long tv_sec; /* numero di secondi */
long tv_usec; /* numero di microsecondi */
};
```

# timeout

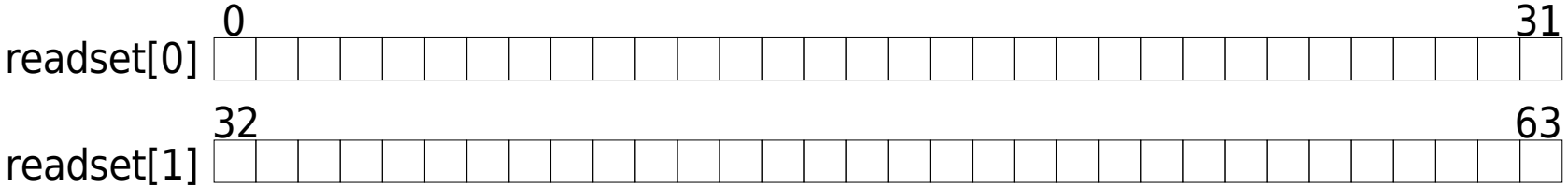
- Esistono tre alternative di timeout per la select:
  - Passando un puntatore NULL  equivale a non impostare un timeout pertanto la select blocca l'esecuzione fino a quando non c'e' un descrittore pronto
  - Passando una specifica quantita' di tempo espressa in secondi e microsecondi si imposta un tempo massimo di attesa
  - Passando un tempo pari a 0 la select verifica se ci sono descrittori pronti e poi ritorna (polling)

# Gli insiemi di descrittori

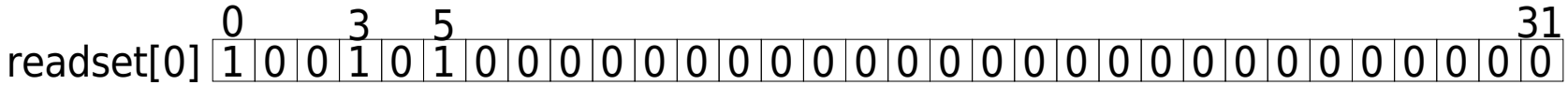
- I parametri 2,3 e 4 della select specificano gli insiemi di descrittori da controllare:
  - readset: pronti per la lettura
  - writeset: pronti per la scrittura
  - exceptionset: condizioni di eccezione
- readset, writeset e exceptionset sono variabili di tipo *fd_set
- in genere array di interi in cui ogni bit rappresenta un descrittore

# Gli insiemi di descrittori

- primo elemento dell’array rappresenta descrittori da 0 a 31
- secondo elemento dell’array rappresenta descrittori da 32 a 63




- Per specificare l'insieme {0,3,5}



# Macro per variabili fd_set

- Nell'header file select.h vengono definte alcune macro per la lettura e la manipolazione di variabili fd_set
- FD_ZERO( fd_set *fdset );
  - Inizializza a zero l’insieme di descrittori fdset facendolo corrispondere all’insieme vuoto
- FD_SET( int fd, fd_set *fdset );
  - Aggiunge fd all’insieme di descrittori fdset ponendo a 1 il bit corrispondente

# Macro per variabili fd_set

- FD_CLR( int fd, fd_set*fdset );
  - Rimuove fd dall'insieme di descrittori fdset ponendo a 0 il bit corrispondente
- FD_ISSET( int fd, fd_set*fdset );
  - Controlla se fd appartine all'insieme di descrittori fdset verificando il valore del bit corrispondente
  - Restituisce 0 in caso negativo ed un valore diverso da 0 in caso affermativo

# Macro per variabili fd_set

- Esempi d'uso delle macro per manipolare le variabile di tipo fd_set

  fd_set readset;

  FD_ZERO ( &readset ); inizializza a 0 tutti i bit

  FD_SET ( 1, &readset ); /* 1 appartiene all’insieme */

  FD_SET ( 4, &readset ); /* 4 appartiene all'insieme */

  FD_ISSET ( 4, &readset ); /* verifica che 4 appartiene all'insieme e restituisce un valore non nullo */

  FD_ISSET ( 3, &readset ); /* verifica che 3 appartiene all'insieme e restituisce zero */

# select

- I 3 insiemi di descrittori vengono passati per riferimento in quanto utilizzati come input ed output
- In input rappresentano i descrittori da testare
- In output rappresentano i descrittori pronti
- Per conoscere i descrittori pronti dopo la select e' necessario controllare ciascuno dei 3 fd_set mediante la macro FD_ISSET

# select

- Il primo argomento della funzione select limita numero di descrittori da testare
- Il suo valore e' in massimo descrittore da testare più uno (maxfdp1)
- La select testa i descrittori 0,1,2,...,maxfdp1-1
- Esempio: se si vogliono testare i descrittori {0,5,8} e' necessario specificare maxfdp1=9

# Descrittori pronti in lettura

- Il descrittore di un socket e' “pronto” in lettura quando si verifica una delle 4 condizioni:
  1) Il numero di byte nel buffer di ricezione del socket e' uguale o maggiore di un valore massimo chiamato low-water mark “LWM” per il buffer di ricezione
     - il valore di LWM per un determinato socket puo' essere impostato dal programmatore
     - Il valore di default per UDP e TCP e' 1
     - In questo caso l'accesso in lettura al descrittore non bloccherà l'esecuzione del processo

# Descrittori pronti in lettura

- Il descrittore di un socket e' “pronto” in lettura quando si verifica una delle 4 condizioni:
  2) La connessione e' stata chiusa
     - In questo caso l'accesso in lettura al socket non bloccherà l'esecuzione del processo e restituira' 0 (EOF)
  3) Il socket e' in ascolto e ci sono nuove connessioni da gestire
     - in questo caso una chiamata ad accept non blocchera' l'esecuzione
  4) Si e' verificato un errore
     - in questo caso l'accesso in lettura restituira' -1

# Descrittori pronti in scrittura

- Il descrittore di un socket e' “pronto” in scrittura quando si verifica una delle 4 condizioni:
  1) Il numero di byte liberi nel buffer di spedizione del socket è maggiore del LWM per il buffer di spedizione
     - il valore di LWM per un determinato socket puo' essere impostato dal programmatore
     - Il valore di default per UDP e TCP e' 2048
     - L’operazione di scrittura restituisce il numero di byte effettivamente passati al sottosistema di rete

# Descrittori pronti in scrittura

- Il descrittore di un socket e' “pronto” in scrittura quando si verifica una delle 4 condizioni:

  2) La connessione e' stata chiusa

     - In questo caso l'accesso in scrittura al socket generera' un SIGPIPE

  3) Un socket che ha utilizzato una connect non bloccante ha completato la connessione o ha riscontrato un'errore

  4) Si e' verificato un errore

     - in questo caso l'accesso in scrittura restituira' -1

# Schema di un applicazione che utilizza l'I/O Multiplex

```
while (1) {
  FD_ZERO (&set);
  FD_SET (STDIN_FILENO,&set);
  FD_SET (sockfd,&set);
  if ( sockfd > STDIN_FILENO )
    maxd = sockfd + 1;
  else
    maxd = STDIN_FILENO + 1;
  if( select(maxd, &set, NULL, NULL, NULL) < 0 )
    exit(1);
  if( FD_ISSET(sockfd, &set) ) {
    ... /*leggi da sockfd */
  }
  if( FD_ISSET(STDIN_FILENO, &set) ) {
  ... /*leggi da standard input */
  }
}
```

# Esercizi

- Dato il server concorrente che accetta in input stringhe dai client e restituisce il numero di caratteri per stringa si modifichi il client relativo in modo che gestista lo standard input e il socket di connessione con il server mediante mutliplex