Funzioni bloccanti e soluzioni

- La funzione accept() e le funzioni per la gestione dell'I/O (ad es., read() e write()) sono bloccanti
  - Ad es., le funzioni read() e recv() rimangono in attesa finché non vi sono dati da leggere disponibili sul descrittore del socket
- Server ricorsivo tradizionale:
  - Il server si blocca su accept() aspettando una connessione
  - Quando arriva la connessione, il server effettua fork(), il processo figlio gestisce la connessione ed il processo padre si mette in attesa di una nuova richiesta
- Soluzioni possibili:
  - Usare le opzioni dei socket per impostare un timeout
  - Usare un socket non bloccante tramite la funzione fnctl() nel modo seguente

    fcntl(sockfd, F_SETFL, O_NONBLOCK);
    - Polling del socket per sapere se ci sono informazioni da leggere

Funzioni bloccanti e soluzioni (2)

- Soluzione alternativa:
  - Usare la funzione select() che permette di esaminare più canali di I/O contemporaneamente e realizzare quindi il **multiplexing dell'I/O**
- Nel caso del server
  - Invece di avere un processo figlio per ogni richiesta, c'è un solo processo che effettua il multiplexing tra le richieste, servendo ciascuna richiesta il più possibile
  - Vantaggio: il server può gestire tutte le richieste tramite un singolo processo
    - No memoria condivisa e primitive di sincronizzazione
  - Svantaggio: il server non può agire come se ci fosse un unico client, come avviene con la soluzione del server ricorsivo che utilizza fork()
- Nel caso del client
  - Può gestire più input simultaneamente
    - Ad es., il client echo gestisce due flussi di input: lo standard input ed il socket
    - Usando select(), il primo dei due canali che produce dati viene letto

Funzione select()

```
int select (int numfds, fd_set *readfds, fd_set *writefds,
            fd_set *exceptfds, struct timeval *timeout);
```

- Header file

  sys/time.h, sys/types.h, unistd.h
- Permette di controllare contemporaneamente lo stato di uno o più descrittori degli insiemi specificati
- Si blocca finché:
  - non avviene un'attività (lettura o scrittura) su un descrittore appartenente ad un dato insieme di descrittori
  - non viene generata un'eccezione
  - non scade un timeout
- Restituisce
  - -1 in caso di errore
  - 0 se il timeout è scaduto
  - Altrimenti, il numero totale di descrittori pronti

Parametri della funzione select()

- Insiemi di descrittori da controllare
  - readfds: pronti per operazioni di lettura
    - Es: un socket è pronto per la lettura se c'è una connessione in attesa che può essere accettata con accept()
  - writefds: pronti per operazioni di scrittura
  - exceptfds: per verificare l'esistenza di eccezioni
    - un'eccezione non è un errore (ad es., l'arrivo di dati urgenti fuori banda, caratteristica specifica dei socket TCP)
- readfds, writefds e exceptfds sono puntatori a variabili di tipo fd_set
  - fd_set è il tipo di dati che rappresenta l'insieme dei descrittori (è una bit mask implementata con un array di interi)
- numfds è il numero massimo di descrittori controllati da select()
  - Se maxd è il massimo descrittore usato, numfds = maxd + 1
  - Può essere posto uguale alla costante FD_SETSIZE

Timeout della funzione select()

- timeout specifica il valore massimo che la funzione select() attende per individuare un descrittore pronto

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

- Se impostato a NULL (timeout == NULL)
  - si blocca indefinitamente fino a quando è pronto un descrittore
- Se impostato a zero (timeout->tv_sec == 0 && timeout->tv_usec == 0 })
  - non si attende affatto; modo per effettuare il polling dei descrittori senza bloccare
- Se diverso da zero (timeout->tv_sec != 0 || timeout->tv_usec != 0 }),
  - si attende il tempo specificato
  - select() ritorna se è pronto uno (o più) dei descrittori specificati (restituisce un numero positivo) oppure se è scaduto il timeout (restituisce 0)

Operazioni sugli insiemi di descrittori

- Macro utilizzate per manipolare gli insiemi di descrittori

  `void FD_ZERO(fd_set *set);`
  - Inizializza l'insieme di descrittori di *set* con l'insieme vuoto

  `void FD_SET(int fd, fd_set *set);`
  - Aggiunge *fd* all'insieme di descrittori *set*, mettendo ad 1 il bit relativo a *fd*

  `void FD_CLR(int fd, fd_set *set);`
  - Rimuove *fd* dall'insieme di descrittori *set*, mettendo ad 0 il bit relativo a *fd*

  `int FD_ISSET(int fd, fd_set *set);`
  - Al ritorno di select(), controlla se *fd* appartiene all'insieme di descrittori *set*, verificando se il bit relativo a *fd* è pari a 1 (restituisce 0 in caso negativo, un valore diverso da 0 in caso affermativo)

Descrittori pronti in lettura

- La funzione select() rileva i descrittori pronti
  - Significato diverso per i tre gruppi (lettura, scrittura, eccezione)
- Un descrittore è pronto in lettura nei seguenti casi:
  - Nel buffer di ricezione del socket sono arrivati dati in quantità sufficiente (soglia minima per default pari a 1, modificabile con opzione del socket SO_RCVLOWAT)
  - Per il lato in lettura è stata chiusa la connessione
    - select() ritorna con quel descrittore di socket pari a "pronto per la lettura" (a causa di EOF)
    - Quando si effettua read() su quel socket, read() restituisce 0
  - Si è verificato un errore sul socket
  - Se un socket è nella fase di listening e ci sono delle connessione completate
    - E' possibile controllare se c'è una nuova connessione completata ponendo il descrittore del socket d'ascolto nell'insieme readfds

Descrittori pronti in scrittura

- Un descrittore è pronto in scrittura nei seguenti casi:
  - Nel buffer di invio del socket è disponibile uno spazio in quantità sufficiente (soglia minima per default pari a 2048, modificabile con opzione del socket SO_SNDLOWAT) ed il socket è già connesso (TCP) oppure non necessita di connessione (UDP)
  - Per il lato in scrittura è stata chiusa la connessione (segnale SIGPIPE generato dall'operazione di scrittura)
  - Si è verificato un errore sul socket

Multiplexing dell'I/O nel server

- Il multiplexing dell'I/O può essere usato sul server per ascoltare su più socket contemporaneamente
  - Un unico processo server (iterativo) ascolta sul socket di ascolto e sui socket di connessione
- Struttura generale di un server che usa select()
  - riempire una struttura fd_set con i descrittori dai quali si intende leggere
  - riempire una struttura fd_set con i descrittori sui quali si intende scrivere
  - chiamare select() ed attendere finché non avviene qualcosa
  - quando select() ritorna, controllare se uno dei descrittori ha causato il ritorno. In questo caso, servire il descrittore in base al tipo di servizio offerto dal server (ad es., lettura della richiesta per una risorsa Web)
  - ripetere il ciclo forever

Client TCP echo con select

- Il client deve controllare due diversi descrittori in lettura
  - Lo standard input, da cui legge il testo da inviare al server
  - Il socket connesso con il server, su cui scriverà il testo e dal quale riceverà la risposta
- L'implementazione con I/O multiplexing consente al client di accorgersi di errori sulla connessione mentre è in attesa di dati immessi dall'utente sullo standard input
- La fase iniziale in cui viene stabilita la connessione è analoga al caso precedente (vedere codice client_echo.c)

Client TCP echo con select (2)

```
void str_cli_echo_sel(FILE *fd, int sockfd)
{
  int               maxd, n;
  fd_set            rset;
  char              sendline[MAXLINE], recvline[MAXLINE];

  FD_ZERO(&rset);           /* inizializza a 0 il set dei descrittori in lettura */
  for ( ; ; ) {
    FD_SET(fileno(fd), &rset);  /* inserisce il descrittore del file (stdin) */
    FD_SET(sockfd, &rset);     /* inserisce il descrittore del socket */
    maxd = (fileno(fd) < sockfd) ? (sockfd + 1): (fileno(fd) + 1);
    if (select(maxd, &rset, NULL, NULL, NULL) < 0 ) { /* attende descrittore pronto
      in lettura */
      perror("errore in select");
      exit(1);
    }
```


Client TCP echo con select (3)

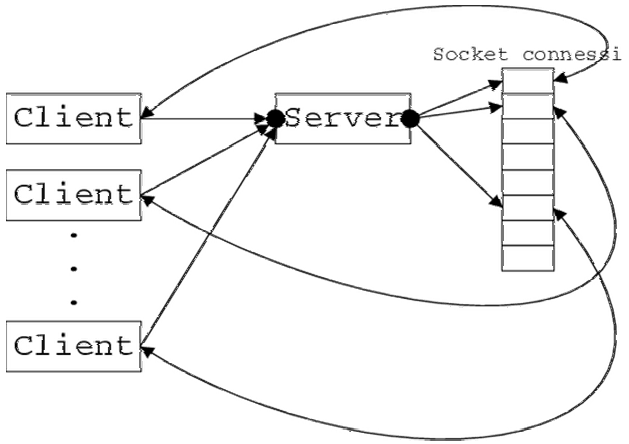
```
  /* Controlla se il file (stdin) è leggibile */
  if (FD_ISSET(fileno(fd), &rset)) {
    if (fgets(sendline, MAXLINE, fd) == NULL)
      return; /* non vi sono dati perché si è concluso l'utilizzo del client */
    if ((writen(sockfd, sendline, strlen(sendline))) < 0) {
      fprintf(stderr, "errore in write");
      exit(1);
    }
  }
```

Client TCP echo con select (4)

```
    /* Controlla se il socket è leggibile */
    if (FD_ISSET(sockfd, &rset)) {
      if ((n = readline(sockfd, recvline, MAXLINE)) < 0) {
        fprintf(stderr, "errore in lettura");
        exit(1);
      }
      if (n == 0) {
        fprintf(stdout, "str_cli_echo_sel: il server ha chiuso la connessione");
        return;
      }
      /* Stampa su stdout */
     recvline[n] = 0;
     if (fputs(recvline, stdout) == EOF) {
        perror("errore in scrittura su stdout");
        exit(1);
      }
    }
  }
}
```


Server TCP echo con select

- Schema del server TCP echo basato sull'I/O multiplexing



Server TCP echo con select (2)

```
#include "basic.h"
#include "echo_io.h"

int main(int argc, char **argv)
{
  int                   listensd, connsd, socksd;
  int                   i, maxi, maxd;
  int                   ready, client[FD_SETSIZE];
  char                  buff[MAXLINE];
  fd_set                rset, allset;
  ssize_t               n;
  struct sockaddr_in    servaddr, cliaddr;
  int                   len;

  if ((listensd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("errore in socket");
    exit(1);
  }
```


Server TCP echo con select (3)

```
memset((void *)&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(SERV_PORT);

if ((bind(listensd, (struct sockaddr *)&servaddr, sizeof(servaddr))) < 0) {
  perror("errore in bind");
  exit(1);
}

if (listen(listensd, QLEN) < 0 ) {
  perror("errore in listen");
  exit(1);
}

/* Inizializza il numero di descrittori */
maxd = listensd;   /* maxd è il valore massimo dei descrittori in uso */
maxi = -1;
```

Server TCP echo con select (4)

```
/* L'array di interi client contiene i descrittori dei socket connessi */
for (i = 0; i < FD_SETSIZE; i++)
  client[i] = -1;

FD_ZERO(&allset); /* Inizializza a zero l'insieme dei descrittori */
FD_SET(listensd, &allset); /* Inserisce il descrittore di ascolto */

for ( ; ; ) {
  rset = allset;  /* Imposta il set di descrittori per la lettura */
  /* ready è il numero di descrittori pronti */
  if ((ready = select(maxd+1, &rset, NULL, NULL, NULL)) < 0) {
    perror("errore in select");
    exit(1);
  }
  /* Se è arrivata una richiesta di connessione, il socket di ascolto
     è leggibile: viene invocata accept() e creato un socket di connessione */
  if (FD_ISSET(listensd, &rset)) {
    len = sizeof(cliaddr);
```


Server TCP echo con select (5)

```
  if ((connsd = accept(listensd, (struct sockaddr *)&cliaddr, &len)) < 0) {
     perror("errore in accept");
     exit(1);
   }

   /* Inserisce il descrittore del nuovo socket nel primo posto libero di client */
   for (i=0; i<FD_SETSIZE; i++)
     if (client[i] < 0) {
       client[i] = connsd;
       break;
     }
   /* Se non ci sono posti liberi in client, errore */
   if (i == FD_SETSIZE) {
     fprintf(stderr, "errore in accept");
     exit(1);
   }
```


Server TCP echo con select (6)

```
    /* Altrimenti inserisce connsd tra i descrittori da controllare
       ed aggiorna maxd */
    FD_SET(connsd, &allset);
    if (connsd > maxd) maxd = connsd;
    if (i > maxi) maxi = i;
    if (--ready <= 0) /* Cicla finché ci sono ancora descrittori da controllare */
     continue;
  }
  /* Controlla i socket attivi per controllare se sono leggibili */
  for (i = 0; i <= maxi; i++) {
    if ( (socksd = client[i]) < 0 )
      /* Se il descrittore non è stato selezionato, viene saltato */
      continue;
    if (FD_ISSET(socksd, &rset)) {
   /* Se socksd è leggibile, invoca la readline */
      if ((n = readline(socksd, buff, MAXLINE)) == 0) {
        /* Se legge EOF, chiude il descrittore di connessione */
        if (close(socksd) == -1) {
```


Server TCP echo con select (7)

```
        perror("errore in close");
        exit(1);
        }
        /* Rimuove socksd dalla lista dei socket da controllare */
      FD_CLR(socksd, &allset);
      /* Cancella socksd da client */
      client[i] = -1;
    }
    else /* echo */
        if (writen(socksd, buff, n) < 0 ) {
          fprintf(stderr, "errore in write");
          exit(1);
        }
        if (--ready <= 0) break;
    }
   }
  }
}
```

Il demone inetd

- Il demone inetd (Internet super-server) è un programma sempre in esecuzione che attende (mediante select) messaggi su un insieme specifico di porte
- Quando arriva un messaggio, inetd accetta la connessione (se necessario) ed effettua il fork di un processo figlio che esegue il programma server corrispondente al servizio richiesto
- Le porte ed i programmi corrispondenti sono specificati nel file /etc/inetd.conf
- Una riga del file di configurazione inetd.conf ha il formato

  service style protocol wait username program arguments

  Ad es: telnet tcp stream nowait root /usr/sbin/tpcd in.telnetd