

CORSO DI SISTEMI OPERATIVI - ESERCITAZIONE 7

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

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

La primitiva `select` attende il cambiamento di stato di un numero specificato di descrittori di file. Tre diversi insiemi di descrittori vengono controllati:

1. I descrittori contenuti in `readfds` vengono controllati per verificare la possibilità di eseguire una operazione di `read` non bloccante.
2. I descrittori contenuti in `writefds` vengono controllati per verificare la possibilità di eseguire una operazione di `write` non bloccante.
3. I descrittori contenuti in `exceptfds` vengono controllati per verificare eventuali eccezioni.

Il parametro `timeout` contiene il tempo di attesa della `select`. Se tale parametro viene fissato a zero la `select` ritorna immediatamente.

In uscita la `select` modifica gli insiemi per indicare quali descrittori hanno cambiato il loro stato. La primitiva ritorna il numero di descrittori il cui stato è cambiato.

Esistono alcune macro che consentono di manipolare gli insiemi dei descrittori dei file.

- `FD_ZERO(fd_set *set)`
cancella un insieme;
- `FD_SET(int fd, fd_set *set)`
aggiunge un descrittore ad un insieme;
- `FD_CLR(int fd, fd_set *set)`
rimuove un descrittore da un insieme;
- `FD_ISSET(int fd, fd_set *set)`
controlla se un descrittore appartiene ad un insieme;

1 Esercizi Proposti

Esercizio 1:

Si progetti in ambiente Unix/C la seguente interazione tra il processo server `Ps` e i suoi clienti `Pi`:

1. il processo Ps viene eseguito sul proprio host;
2. Ps offre due servizi differenti su due porte il cui valore viene specificato come argomento di invocazione del programma;
3. i processi clienti Pi (anch'essi vengono eseguiti sul proprio host) inviano richieste di servizio al server Ps attraverso socket di tipo STREAM;
4. il processo Ps si avvale della primitiva select ponendosi in attesa delle richieste di servizio (con un timeout fissato di 5 minuti oltre il quale il server termina la propria attivit'a);
5. il primo servizio offerto da Ps prevede la restituzione della data e dall'ora correnti ai processi clienti Pi;
6. il secondo servizio offerto da Ps prevede la restituzione di informazioni di rete (DNS) su un particolare host ai clienti. Il nome dell'host viene inviato dai processi clienti stessi a Ps.
7. Per soddisfare il secondo tipo di servizi Ps esegue le seguenti operazioni:
 - attiva un processo figlio per ogni nuova richiesta di servizio accettata;
 - il processo figlio esegue il comando `nslookup <nome host>` (si consiglia l'utilizzo della primitiva `system`. Esempio: `system(stringa);` dove `stringa="nslookup localhost"`);
 - il processo figlio salva l'output del comando precedente su un file;
 - il processo figlio in risposta al cliente scrive sulla socket il contenuto del file;
8. I processi Pi utilizzano il comando `telnet host <porta>` per connettersi al server Ps (nel caso di richiesta del secondo tipo di servizi i processi Pi inviano a Ps la stringa `<nome host>` dallo shell in seguito al comando `telnet`)

Di seguito viene riportata una traccia per l'esecuzione dell'attesa bloccante del server su una socket (FD_SETSIZE è il massimo numero di descrittori che possono essere contenuti all'interno di insiemi del tipo `fd_set`).

```
.....
struct timeval timeRec;
int sock1,sock2,rc;
fd_set select_set;

timeRec.tv_sec=60*5;
timeRec.tv_usec=0;
FD_ZERO(&select_set);
FD_SET(sock1,&select_set);
FD_SET(sock2,&select_set);

/* Test di lettura sulla socket */
rc=select(FD_SETSIZE,&select_set,0,0,&timeRec);
if (rc>0 && FD_ISSET(listenSocket,&select_set))
.....
```

Esercizio 2:

Si realizzi in ambiente Unix/C il server della seguente interazione tra processi:

1. il sistema consiste di due tipi di processi: un processo server P_s e i processi clienti P_{ci} ;
2. per la comunicazione tra P_s e i processi clienti P_{ci} vengono utilizzate socket Stream;
3. il server P_s offre un servizio **concorrente** (un figlio per ogni connessione) alla porta 987 oppure 9876 (N.B. : motivare la scelta della porta utilizzata);
4. il client invia una stringa del tipo `cmd1 | cmd2` (digitata dall'utente) affinché un opportuna coppia di processi sul server realizzi il piping dei comandi `cmd1` e `cmd2` per semplicità ipotizzare che i comandi `cmd1` e `cmd2` **non abbiano argomenti**.
5. lo standard output del processo che realizza `cmd2` deve essere ridiretto sulla socket connessa affinché sia ricevuto e visualizzato dal cliente;
6. a partire dal momento in cui riceve un segnale SIGUSR1, il server P_s deve memorizzare in un file di testo (denominato `server-log.txt`) gli indirizzi IP dei clienti che richiedono il servizio.

Deve essere utilizzata la gestione affidabile dei segnali. Come generico client P_{ci} , si suggerisce l'utilizzo del programma `telnet`, invocato come

```
telnet localhost numeroportaserver
```

Una volta connessi al server, scrivere ad esempio `ls|wc` [Invio] nel terminale del `telnet` per verificare il funzionamento del server. Si tenga conto che `telnet` invia oltre alla stringa digitata i caratteri di fine linea `\r \n`. Si rammenta la presenza nella libreria C delle funzioni:

- `char *strchr(const char *s, int c)`: resituisce la prima occorrenza del carattere `c` nella stringa `s`;
- `char *strstr(const char *haystack, const char *needle)`: resituisce la prima occorrenza della sottostringa `needle` nella stringa `haystack`.

Esercizio 3:

Si realizzi in ambiente Unix/C il server della seguente interazione tra processi:

1. il sistema consiste di un processo padre P_f e N processi figli P_{ci} ;
2. per la comunicazione tra P_f e P_{ci} vengono utilizzate delle pipe;
3. il processo padre tiene un array `pids[]` con i PID dei processi figli e un indice p compreso tra 0 e $N - 1$; tutti i processi (padre e figli) tengono un contatore `count` inizializzato a 1;
4. ad ogni ricezione di un segnale SIGUSR1, ciascun processo incrementa il valore della propria variabile `count`;

5. alla prima invocazione il padre inizializza l'indice p compreso tra 0 e $N - 1$ e comunica al processo figlio p -esimo il PID del processo $(p+1)$ -esimo ($p+1$ è da intendersi modulo N);
6. il processo p -esimo invia un segnale SIGUSR1 al processo con il PID comunicato dal padre e risponde al padre inviando il proprio valore di *count*;
7. il processo padre, ricevuto dal figlio il valore del contatore *count*, incrementa l'indice $p = (p + count) \% N$ e ripete la stessa operazione.

Deve essere utilizzata la gestione affidabile dei segnali.

Esercizio 4:

Si realizzi un server concorrente su socket STREAM che abbia il seguente comportamento:

1. il server riceve dai clienti una stringa che rappresenta un intero N ($2 \leq N \leq 10$);
2. il server deve quindi creare N processi figli che attendono un intervallo di durata casuale D_i ($1 \leq D_i \leq 5$) secondi, visualizzano un messaggio con il loro PID e terminano;
3. il server comunica al client il PID del processo figlio che è terminato per primo;
4. se il server riceve un segnale SIGUSR2 durante l'attesa dei processi figli, deve provvedere a terminarli tutti e ad inviare il messaggio "Tutti i processi figli sono stati terminati" al cliente.

Si richiede l'utilizzo della gestione affidabile dei segnali. Si suggerisce l'utilizzo del comando telnet come client per la verifica del funzionamento del server.