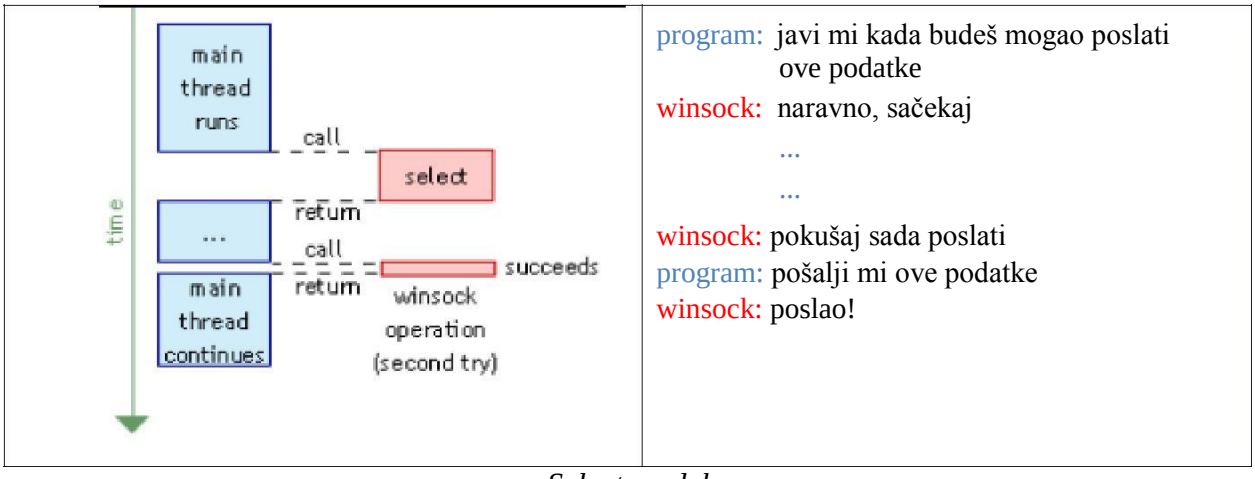
Vežba 9 – Multipleksiranje operacija zasnovano na događajima

U prethodnoj vežbi upoznali smo neblokirajući režim soketa, koji omogućava da se izbegne blokiranje programa ako operacija nad soketom nije spremna da se izvrši. Međutim, teško je bilo odrediti kada je pravi trenutak za novi pokušaj. Zbog toga smo periodično iznova pokušavali izvršiti operaciju trošeći procesorsko vreme.

Funkcija *select* omogućava praćenje stanja čitanja, pisanja i grešaka na jednom ili više soketa čime se omogućava pravovremena reakcija na više događaja od interesa. Na ovaj način, izbegava se potreba za periodičnom proverom stanja objekata od interesa. Na primer, u slučaju TCP servera može se istovremeno čekati prispeće poruka od konektovanih klijenata i novih zahteva za uspostavom konekcije.



Select model

Na samom početku sledi kratak opis `fd_set` strukture koju ćemo koristiti da se na elegantan način navede skup soketa koji će se grupisati za neku posebnu namenu. Ova struktura sadrži dva polja: `fd_array` - niz soketa koji se nalaze u setu i `fd_count` - broj soketa u setu.

```
typedef struct fd_set {
  unsigned int      fd_count;
  SOCKET            fd_array[FD_SETSIZE];
} fd_set;
```

U nastavku je data deklaracija funkcije select, opis parametara i povratne vrednosti funkcije:

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

Funkcija:	Opis:
`select`	Omogućava praćenje stanja čitanja, pisanja i grešaka na jednom ili više soketa.
Parametri:	**Opis:**
`int nfds[in]`	Ovaj parametar se ignoriše.
`fd_set *readfds [in][out]`	Skup soketa na kojima se proverava mogućnost čitanja (za operacije recv, recvfrom , accept i close).
`fd_set * writefds [in][out]`	Skup soketa na kojima se proverava mogućnost pisanja (za operacije send, sendto i connect)
`fd_set * exceptfds [in][out]`	Skup soketa na kojima se proveravaju greške.
`const struct timeval * timeout [in]`	Maksimalno vreme čekanja izraženo preko `timeval` strukture.
Povratna vrednost:	**Opis:**
`int`	Ako se funkcija uspešno izvršila pre tajmauta, ona će vratiti broj soketa na kojima se desio događaj. U slučaju da je vreme čekanja isteklo, funkcija će vratiti 0. U suprotnom, vraća se SOCKET_ERROR, a kod konkretne greške se dobija nakon poziva funkcije WSAGetLastError.

Može se primetiti da postoje tri različita *fd_set* parametra: jedan za praćenje čitanja, jedan za praćenje pisanja i jedan za praćenje grešaka. Ovi ulazno-izlazni parametri reprezentuju skup soketa za koji je potrebno pratiti određeni status. Nakon registrovanja događaja od interesa, *select* funkcija će u setovima ostaviti samo deskriptore na kojima su događaji zabeleženi. Na taj način moguće je doznati koji se od događaja desio na osnovu toga da li se deskriptor soketa i dalje nalazi u setu statusa.

Na primer, ako želimo doznati mogućnost čitanja sa soketa, potrebno je deklarisati i inicijalizovati `readfds` set za operacije čitanja pomoću `FD_ZERO` funkcije. U `readfds` set bi trebalo dodati soket

na kome želimo slušati informacije prispeće paketa pomoću `FD_SET` funkcije i sačekati da se završi poziv funkcije *select*. Nakon provere rezultata funkcije `select`, trebalo bi pomoću funkcije `FD_ISSET` proveriti da li se soket i dalje nalazi u `readfds` setu. Ukoliko se nalazi, moguće je započeti neblokirajući prijem paketa na soketu.

U nastavku su date operacije koje se koriste za rad sa fd_set strukturom:

- FD_ZERO (fd_set *set) - inicijalizuje set
- FD_SET (SOCKET sock, fd_set *set) - dodaje deskriptor soketa u set
- FD_CLR (SOCKET sock, fd_set *set) - uklanja deskriptor soketa is seta
- FD_ISSET (SOCKET sock, fd_set *set) - proverava da li se deskriptor soketa nalazi u setu (povratna vrednost: 1 - da, 0 - ne)

Parametar *timeout* kontroliše maksimalno dozvoljeno vreme koje će funkcija *select* čekati da se neki od događaja desi. Ovaj parametar pokazuje na strukturu tipa *timeval*:

```
typedef struct timeval {
	long tv_sec;
	long tv_usec;
} timeval;
```

Postoje tri mogućnosti:

1. Čekati koliko god je potrebno - poziv funkcije select će blokirati nit dokle god neki od deskriptora soketa ne bude spreman. Za ovaj slučaj potrebno je pokazivač *timeout* postaviti na *null*.

2. Čekati definisani period vremena - povratak *select* funkcije će uslediti nakon što neki od deskriptora bude spreman, ali ne duže od maksimalno definisanog vremena čekanja na koje ukazuje parametar timeout.

3. Ne čekati uopšte - povratak odmah nakon provere deskriptora svih soketa. U ovom slučaju period čekanja je potrebno podesiti na 0 sekundi i 0 milisekundi.

Alogritam za multipleksiranje operacija koje je zasnovano na događajima

1. Postavljanje soketa u neblokirajući režim

```
unsigned long mode = 1;
iResult = ioctlsocket(clientSocket, FIONBIO, &mode);
```

2. Inicijalizacija fd_set strukture korišćenjem *FD_ZERO* makroa:

```
fd_set readfds;
FD_ZERO(&readfds);
```

3. Dodavanje deskriptora soketa u set pomoću *FD_SET* makroa

```
FD_SET(clientSocket, &readfds);
```

4. Poziv *select* funkcije, koja će sačekati da se desi neki od događaja koji želimo sačekati

```
// maksimalni period cekanja select funkcije
timeval timeVal;
timeVal.tv_sec  =  1;
timeVal.tv_usec = 0;

// poziv select funkcije koja omogucava da se saceka definisani
// dogadjaj u period od 1 sekunde
int result = select(0, &readfds , NULL , NULL , &timeVal);
```

5. Provera rezultata poziva select funkcije

```
if (result == 0){
// vreme za cekanje je isteklo
}
else if (result == SOCKET_ERROR){
	//desila se greska prilikom poziva funkcije
}
else {
	// rezultat je jednak broju soketa koji su zadovoljili uslov
}
```

6. Određivanje koji soketi imaju čekajućih događaja korišćenjem *FD_ISSET* makroa i pozivi neblokirajućih funkcija.

```
if (FD_ISSET(clientSocket, &readfds)){
	// izvrsenje operacije
}
```

7. Čišćenje seta za novu iteraciju

```
FD_CLR(clientSocket, &readfds);
```

Zadaci

Koristeći primer implementacije UDP klijenta i servera koji je dat u prilogu materijala za vežbu, potrebno je omogućiti na serveru multipleksiranje operacija čitanja poruka sa mreže i praćenje grešaka zasnovano na događajima.

1. Na serveru napraviti dva UDP soketa i povezati ih sa adresama i portovima koji su dati u nastavku: **(0.4 poena)**
   - serverSocket1
     - IP adresa: sve dostupne adrese
     - Port: 15011
   - serverSocket2
     - IP adresa: 127.0.0.1
     - Port: 15012
2. Obezbediti neblokirajući režim izvršavanja operacija nad serverskim soketima. **(0.3 poena)**
3. Kreirati set za operacije čitanja, inicijalizovati ga i dodati odgovarajuće sokete u set. **(0.3 poena)**
4. Omogućiti istovremeno čekanje prispeća paketa na oba serverska soketa korišćenjem select funkcije. Nakon izvršenja *select* funckije proveriti koliko događaja se desilo i da li se desila greška prilikom poziva funkcije. **(0.3 poena)**
5. Ukoliko je pristigao paket na neki od soketa, potrebno je proveriti na koji od soketa je pristigao paket i pozvati funkciju koja će omogućiti njegov prijem. Ispisati sadržaj primljene poruke. **(0.3 poena)**
6. Omogućiti klijentu da može poslati više poruka serveru. Ukoliko klijent upiše "end" potrebno je ugasiti klijenta. **(0.2 poena)**
7. Na klijentskoj strani potrebno je učitati iz komandne linije port servera, kako bismo omogućili da se sa različitih klijenata izvršavanjem istog koda mogu slati poruke ka različitim portovima na serveru (portovi 15011 i 15012). **(0.2 poena)**

   Napomena: konverziju stringa u broj moguće je realizovati korišćenjem atoi funkcije čija deklaracija je data u nastavku:

   int atoi(char buffer);*

8. Uz multipleksiranje operacija čitanja poruka, omogućiti na serverskim utičnicama i praćenje statusa grešaka zasnovano na događajima.
   - Kreirati set za praćenje grešaka, inicijalizovati ga i dodati odgovarajuće sokete u njega. **(0.2 poena)**
   - Uz pomoć *select* funkcije pratiti operacije čitanja poruka i status grešaka na utičnicama. **(0.1 poena)**
   - Nakon izvršenja *select* funkcije proveriti da li se na nekoj utičnici desila greška. Ako jeste, zatvoriti datu utičnicu. **(0.2 poena)**