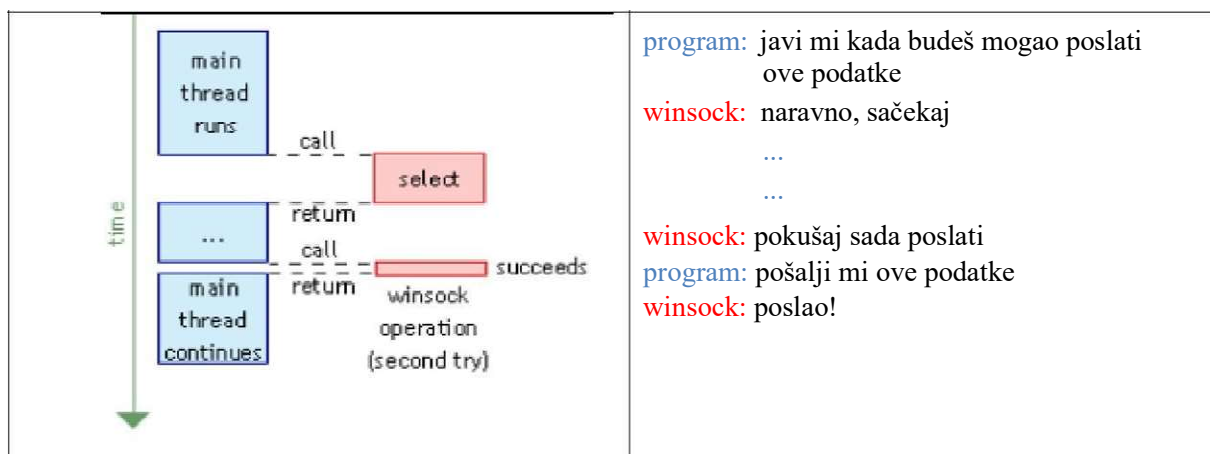


## Vežba 5 – Multipleksiranje operacija zasnovano na događajima

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 nfd, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
    const struct timeval *timeout);
```

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

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 deklarirati 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 omogućava da se saceka definisani
// događaj 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)){
    // izvršenje operacije
}
```

7. Čišćenje seta za novu iteraciju

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

## Zadatak 1

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:
  - `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.
3. Kreirati set za operacije čitanja, inicijalizovati ga i dodati odgovarajuće sokete u set.
4. Omogućiti istovremeno čekanje prispeća paketa na oba serverska soketa korišćenjem `select` funkcije. Nakon izvršenja `select` funkcije proveriti koliko događaja se desilo i da li se desila greška prilikom poziva funkcije.
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.
6. Omogućiti klijentu da može poslati više poruka serveru. Ukoliko klijent upiše "end" potrebno je ugasiti klijenta.
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).

*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 ipraćenje statusa grešaka zasnovano na događajima.
  - Kreirati set za praćenje grešaka, inicijalizovati ga i dodati odgovarajuće sokete unjega.
  - Uz pomoć `select` funkcije pratiti operacije čitanja poruka i status grešaka nautičnicama.
  - Nakon izvršenja `select` funkcije proveriti da li se na nekoj utičnici desila greška. Ako jeste, zatvoriti datu utičnicu.

## Zadatak 2

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

1. Obezbediti neblokirajući režim izvršavanja operacija nad serverskim soketom kojisluzi za prijem zahteva za uspostavom veze sa serverom.
2. Kreirati `fd_set` za operacije čitanja, inicijalizovati ga i dodati soket zaprijem konekcija u set.
3. Omogućiti čekanje prispeća novih konekcija na soket korišćenjem `select` funkcije. Maksimalni interval čekanja podesiti na 1 sekundu.
4. Nakon izvršenja `select` funkcije proveriti koliko događaja se desilo i da li se desila greška prilikom poziva funkcije.
  - Ukoliko se desila greška ugasiti server.
  - Ukoliko se nije desio događaj nastaviti sa čekanjem događaja
  - U suprotnom, ispisati broj događaja koji se dogodio.
5. Proveriti da li događaj koji se desio predstavlja zahtev za uspostavom konekcija. Posle toga prihvatiti konekciju i ispisati port i adresu klijenta koji je zahtev poslao.
6. Obezbediti neblokirajući režim izvršavanja operacija nad soketom koji jedodeljen klijentu. Nakon toga omogućiti čekanje novih događaja.
7. Umesto jednog soketa, deklarirati niz od 3 soketa i brojač koji će pamtit koji je poslednji soket dodeljen novom klijentu. Prilikom odobravanja zahteva za uspostavom veze, soket koji se dodeljuje klijentu se smešta u prvo slobodno mesto u nizu soketa.
8. U `fd_set` u koji se pre svakog poziva dodavao soket za prijem novih konekcija potrebno je dodati i sokete svih klijenata koji su se konektovali na server, kako bismo pomoću `select` funkcije mogli osluškivati poruke koje stignu na njih.
9. Dodati nakon izvršenja `select` funkcije funkcionalnost koja će omogućiti proveru da li je stigla poruka na neki od soketa koji su dodeljeni klijentima. Ukoliko jeste, primiti poruku i ispisati je na ekran.
10. Omogućiti na klijentskoj strani da se šalju poruke dokle god klijent ne izabere opciju da ne želi više da šalje poruke.
11. Omogućiti da klijent u okviru jedne poruke pošalje :
  - ime i prezime studenta, i
  - broj poena osvojenih na testu.

Za ovu namenu kreirati odgovarajuću strukturu (koja će se popuniti potrebnim podacima) i poslati je u okviru poruke.

12. Kada poruka stigne na server, ispisati je na ekranu u formatu datom u primeru:

*Ime i prezime :*        *Dragan Jovanović*

*Poeni:*                *40*

13. Omogućiti prijem zahteva za prekid veze od klijenata i u tom slučaju potrebno je poravnati sokete u nizu tako da se stvori jedno mesto na kraju i umanjiti brojač zauzetih soketa. Poravnanje soketa u nizu je potrebno uraditi i u slučaju da se desila greška prilikom prijema poruke.
14. Omogućiti da se na ekran server ispiše ime predmeta i studijska godina ukoliko se u toku čekanja događaja pritisne bilo koja tipka tastature. Iskoristiti: povratnu vrednost `select` funkcije, koja ukazuje na to da je vreme čekanja isteklo i funkciju `_kbhit()` koja vraća `true` ako je neki od taster bio pritisnut.
15. Prilikom prekida rada servera potrebno je poslati poruku prekida uspostave veze čime bi se svi konektovani klijenti obavestili o završetku rada servera i zatim i klijentizavršili svoj rad.
16. Obezbediti da se ne primi novi zahtev za uspostavom konekcija, ukoliko su već uspostavljene 3 konekcije sa klijentima. U tom slučaju u setu u kome se nalaze soketi na kojima želimo slušati prijemne događaje ne bi trebalo da se nalazi soket za prijem novih konekcija.
17. ***Za one koji hoće da znaju više:*** Operativni sistem po default-u prihvata konekciju ukoliko je zahtev za uspostavom veze upućen soketu koji se nalazi u stanju slušanja. Time se zahtev odobrava pre nego što se pozove i izvrši *Accept* funkcija na našem serveru. Samim tim, klijent će dobiti informaciju da je *Connect* funkcija uspešno izvršena, jer će mu operativni sistem odobriti zahtev iako ga serverska aplikacija još nije obradila. Dati zahtev naša implementacija servera će obraditi kada se oslobodi jedno mesto za konekciju (tj. neki od klijenata prekine vezu).

*Takođe, ukoliko je upravo konektovani klijent već poslao neku poruku, ona će biti primljena nakon uspostave veze.*

*Ovo podrazumevano ponašanje moguće je izmeniti promenom parametra `SO_CONDITIONAL_ACCEPT`, koji će obezbediti da se zahtevi za uspostavom veze prihvaju od strane serverske aplikacije, a ne operativnog sistema. Isprobajte datu mogućnost i proverite u čemu je razlika!*

```
int bOptLen = sizeof(bool);
iResult = setsockopt(listenSocket, SOL_SOCKET,
                     SO_CONDITIONAL_ACCEPT, (char *) &bOptVal, bOptLen);
if (iResult == SOCKET_ERROR)
{
    printf("setsockopt for SO_CONDITIONAL_ACCEPT failed with error:
           %u\n", WSAGetLastError());
}
```

***Od svakog studenta očekuje se da razume gradivo koje je korišćeno prilikom izrade ovog zadatka. On predstavlja izazov jer sadrži znatno više pod-zadataka nego što će biti dato na kolokvijumu, sa ciljem da se objedini svo znanje kojeje potrebno za kolokvijum.***