Uvod u distribuirane informacijske sustave

Komunikacija

Da ponovimo

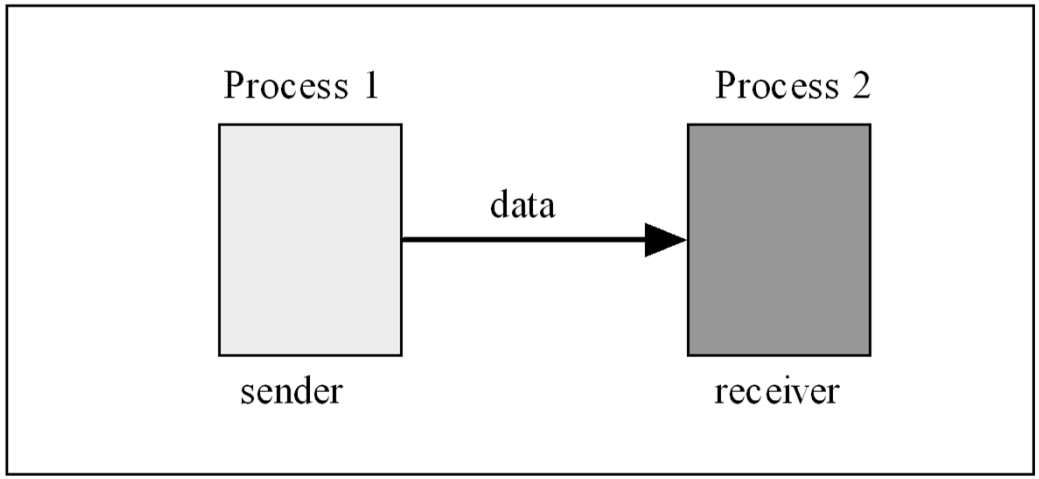
- Vrste distribuiranih sustava
- Arhitekture
- Uloga procesa i niti
- Procesi klijenta
- Procesi servera

Komunikacija

- Komunikacija među procesima je srž distribuiranih sustava
- Razmjena podataka između dva ili više neovisnih procesa / niti.
- Komunikacija na razini operacijskog sustava:
  - Signali, Cijevi, redovi poruka, dijeljena memorija , semafori
- Distribuirani računalni sustavi
  - RPC, RMI, MOM, data streaming

Komunikacija među procesima

- Arhetipski API : connect , send, recieve, disconect
- Sinkronizacija : sinkronizirati operacije (send prije recieve)
- U praksi, traži podršku sustava



Sinkrona i asinkrona komunikacija

- Sinkronizacija blokirajućim pozivima: da bi se operacije uskladile, jedan proces mora čekati da drugi završi sa radom
- Asinkrona komunikacija: nema čekanja, proces nastavlja sa svojim radom nakon poziva. Evantualno može biti obavješten kada se operacija izvrši

- Sinkrono=blokirajuće
- Asinkrono=ne blokirajuće

Sinkrona i asinkrona komunikacija

- Blokirajući pozivi
  - Ako su pogrešno sinkronizirnani mogu izazvati **deadlock**
  - **Deadlock** je situacija u kojoj dvije niti ili procesa obe čekaju da druga završi prije nego nastavi sa radom
  - „*kokoš ili jaje*” problem
  - Korištenje *timeouta* može pomoći

Signali

- Najstariji oblik komunikacije u Unix OS
- Mogu se generirati ili tipkovnicom ili generiranom greškom u izvođenju programa

1)	SIGHUP	2)	SIGINT	3)	SIGQUIT	4)	SIGILL
5)	SIGTRAP	6)	SIGIOT	7)	SIGBUS	8)	SIGFPE
9)	SIGKILL	10)	SIGUSR1	11)	SIGSEGV	12)	SIGUSR2
13)	SIGPIPE	14)	SIGALRM	15)	SIGTERM	17)	SIGCHLD
18)	SIGCONT	19)	SIGSTOP	20)	SIGTSTP	21)	SIGTTIN
22)	SIGTTOU	23)	SIGURG	24)	SIGXCPU	25)	SIGXFSZ
26)	SIGVTALRM	27)	SIGPROF	28)	SIGWINCH	29)	SIGIO
30)	SIGPWR						

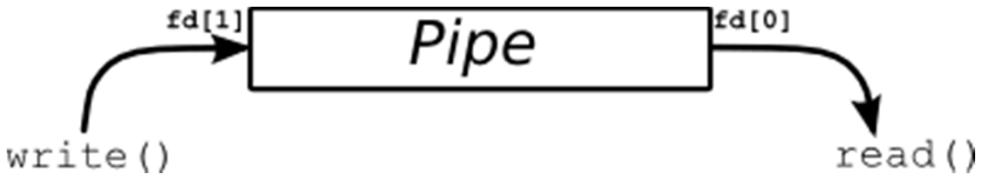
```
#include  <signal.h>
#include  <stdio.h>

static void		sig_usr(int);		/* one handler for both signals */

int main(void)
{
		if (signal(SIGUSR1, sig_usr) == SIG_ERR)
			{ printf("can't catch SIGUSR1\n"); exit(1); }
		if (signal(SIGUSR2, sig_usr) == SIG_ERR)
			{ printf("can't catch SIGUSR2\n"); exit(1); }
		for ( ; ; )
			pause();
}
static void sig_usr(int signo)   { /* argument is signal number */
		if (signo == SIGUSR1)
			printf("received SIGUSR1\n");
		else if (signo == SIGUSR2)
			printf("received SIGUSR2\n");
		else{
			 printf("received signal %d\n", signo); exit(1); }
		return;
}
```

System V IPC

- System V je jedna od prvih komercijalnih verzija Unix OS
- Podržava 3 metode za dijeljenje podataka među procesima
  - message queues (dvo smjerne cijevi)
  - semaphore sets (dijeljenji brojači resursa)
  - shared memory segment

Cijevi (pipes)

- U linuxu
  - ls | pr | lpr
- Preusmjeravanje rezultata jednog procesa drugom procesu
- Tok informacija u jednom smijeru
- Imenovane cijevi - FIFO



```
/***** KEYBOARD HIT PROGRAM *****/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>
#include <ctype.h>

int filedes[2];

void *read_char(){
    char c;
    printf("Entering routine to read character.....\n");
    while(1) {
        /* Get a character in 'c' except '\n'. */
        c = getchar();
        if(c == '\n')
            c = getchar();
        write(filedes[1], &c, 1);
        if(isalnum(c)) {
            sleep(2);
            exit(1);
        }
    }
}
```

```
void *check_hit(){
    char c;
    printf("Entering routine to check hit.....\n");
    while(1) {
        read(filedes[0], &c, 1);
        if(isalnum(c)) {
        printf("The key hit is %c\n", c);
        exit(1);
        } else {
        printf("key hit is %c\n", c);
        }
    }
}
int main(){
    int i;
    pthread_t tid1, tid2;
    pipe(filedes);
    /* Create thread for reading characters. */
    i = pthread_create(&tid1, NULL, read_char, NULL);
    /* Create thread for checking hitting of any keyboard key. */
    i = pthread_create(&tid2, NULL, check_hit, NULL);
    if(i == 0) while(1);
    return 0;
}
```

Fifo – imenovane cijevi

```
/*** speak.c -- writes into a FIFO*/
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

#define FIFO_NAME "myPipe„

int main(void){
        char s[300];
        int num, fd;
```

```
mknod(FIFO_NAME, S_IFIFO | 0666, 0);
printf("waiting for readers...\n");
fd = open(FIFO_NAME, O_WRONLY);

printf("got a reader--type some stuff\n");
while (gets(s), !feof(stdin)) {

if ((num = write(fd, s, strlen(s))) == -1)
perror("write");
else
printf("speak: wrote %d bytes\n", num);
}

return 0;
}
```

```
/*** tick.c -- reads data from a FIFO*/
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

#define FIFO_NAME "myPipe„

int main(void){
        char s[300];
        int num, fd;
mknod (FIFO_NAME, S_IFIFO | 0666, 0);
printf("waiting for writers...\n");
fd = open(FIFO_NAME, O_RDONLY);
printf("got a writer\n");
do {
if ((num = read(fd, s, 300)) == -1)
        perror("read");
else {
        s[num] = '\0';
printf("tick: read %d bytes: \"%s\"\n", num, s);
                }
        } while (num > 0);

        return 0;
}
```

Redovi poruka (*message queues*)

- Jedna od načina komunikacije među procesima **System V**
- Asinkrono slanje poruka između dva procesa
- Red se implementira i održava od strane kernela

Operacije sa redom poruka

- Msgget - kreira se red poruka,pošiljatelj koristi zastavicu IPC_CREAT, dok primatelj samo specificira prava pristupa. Za identifikaciju se koristi *key*
- Msgctl - upravljanje operacijamana redu poruka
- Msgsnd - pisanje poruke u red
- Msgrcv - primanje poruke

```
/************ send.c *********/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <errno.h>
#include <string.h>

#define NMSGS          5
extern int errno;

struct msgbuf {
        long mtype;
        char mtext[100];
};
int main(){
        int msgid;
        int i, nloop;
        struct msgbuf msgp;
        char tmp_msg[100];
        tmp_msg[0] = '\0';
msgid = msgget(9999, IPC_CREAT | 0666);
        if(msgid < 0) {
printf("%d : Error number is %d\n", __LINE__, errno);
exit(1);
        }
for(nloop=0;nloop<NMSGS;nloop++) {
msgp.mtype = 1;
fgets(tmp_msg, 100, stdin);
strncpy(msgp.mtext, tmp_msg, strlen(tmp_msg));
i = msgsnd(msgid, &msgp, strlen(tmp_msg), IPC_NOWAIT);
    if(i < 0) {
printf("%d : Error number is %d\n", __LINE__, errno);
exit(1);
        }
   tmp_msg[0] = '\0';
        }

        return 0;
}
```

```
/************** recv.c *************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <errno.h>
#include <string.h>

extern int errno;
struct msgbuf {
        long mtype;
        char mtext[100];
};
int main(){
        int msgid;
        int i, nloop;
        struct msgbuf msgp;
msgid = msgget(9999, 0444);//read
        if(msgid < 0) {
printf("%d : Error number is %d\n", __LINE__,
errno);
        exit(1);      }
for(nloop=0;nloop<5;nloop++) {
bzero(msgp.mtext, 100);
i = msgrcv(msgid, &msgp, 100, 1, IPC_NOWAIT);
        if(i < 0) {
printf("%d : Error number is %d\n", __LINE__,
errno);
exit(1);
                }

msgp.mtext[strlen(msgp.mtext)] = '\0';
fwrite(msgp.mtext, sizeof(char), strlen(msgp.mtext),
stdout);
printf("message is: %s\n", msgp.mtext);
        }
        if(msgctl(msgid, IPC_RMID, NULL) < 0) {
printf("%d : Error number is %d\n", __LINE__,
errno);
                exit(1);
        }

        return 0;
}
```

Dijeljena memorija

- Jedna od načina komunikacije među procesima **System V**
- Komunikacija djeljenom memorijom izvodi se kroz sljedeće korake:
  - Dohvaćanje identifikatora zajedničke memorije - shmget (shared memory get)
  - Korištenje identifikatora adrese zajedničke memorije - shmat (shared memory attach),
  - oslobađanje zajedničke memorije nakon upotrebe - shmdt (shared memory detach)
  - Korištenje adrese za upravljanej pristupom, pravima primanje podataka i uništenje zajedničke memorije - shmctl (shared memory control).

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <errno.h>
#include <string.h>
#include <ctype.h>

extern int errno;

#define SIZE 1

char *read_key;
int shmid;

int shared_init()
{
if((shmid = shmget(9999, SIZE, IPC_CREAT | 0666)) < 0) {
        printf("Error in shmget. errno is: %d\n", errno);
        return -1;
        }
        if((read_key = shmat(shmid, NULL, 0)) < 0) {
                printf("Error in shm attach. errno is: %d\n",
        errno);
                return -1;
        }
        return 0;
}

void read_char()
{
        char c;
        while(1) {
                c = getchar();
                if(c == '\n') {
                        c = getchar();
                }
                strncpy(read_key, &c, SIZE);
                printf("read_key now is %s\n",
        read_key);
                if(isalnum(*read_key)) {
                shmdt(read_key);
                shmctl(shmid, IPC_RMID, NULL);
                        exit(1);
                }
        }
}

int main()
{
        if(shared_init() < 0) {
                printf("Problems with shared
        memory\n");
                exit(1);
        }
        read_char();
        return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <errno.h>
#include <string.h>
#include <ctype.h>

extern int errno;

#define SIZE 1

char *detect_key;
int shmid;

int shared_init()
{
	if((shmid = shmget(9999, SIZE, 0444)) < 0) {
	printf("Error in shmget. errno is: %d\n", errno);
		return -1;
	}
	if((detect_key = shmat(shmid, NULL,
	SHM_RDONLY)) < 0) {
	printf("Error in shm attach. errno is: %d\n", errno);
		return -1;
	}
//	detect_key = NULL;
	return 0;
}
```

```
void detect_hit()
{
	char c;
	c = *detect_key;
	while(1) {
		if(c != *detect_key) {
		if(isalnum(detect_key[0])) {
		printf("detect_key is %s\n", detect_key);
		shmdt(detect_key);
		shmctl(shmid, IPC_RMID, NULL);
		exit(1);
		} else {
		printf("detect_key is %s\n", detect_key);
				}
		c = *detect_key;
		}
	}
}

int main()
{
	if(shared_init() < 0) {
	printf("Problems with shared memory\n");
		exit(1);
	}
	detect_hit();
	return 0;
}
```

Semafori

- Semafor je sinkronizacijska varijabla među procesima koji komuniciraju
- Mogu komunicirati podacima ili koristiti dijeljenji podatkovni objekt – tada se treba osigurati od istovremenog pisanja u objekt
- Zaključavajući mehanizam
- Semget
- Semop
- semctl

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <string.h>
#include <errno.h>
#include <signal.h>
#include <ctype.h>

#define NUM       1

#define SEMAPHORE_KEY 2004

extern int errno;

int semid;
struct sembuf lock_sem_var = {0, -1,
IPC_NOWAIT};
struct sembuf ulock_sem_var = {0, 1,
IPC_NOWAIT};
```

```
union semun {
            int val;
            struct semid_ds *buf;
            unsigned short int *array;
            struct seminfo *__buf;
};
void init_semaphore()
{
if((semid = semget(SEMAPHORE_KEY,
NUM,      IPC_CREAT | 0666)) < 0)
             {
printf("semget error: errno is %d\n",
errno);
exit(1);
            }
}

void set_sem_val()
{
union semun semopts;
semopts.val = 1;
semctl(semid, 0, SETVAL, semopts);
}
```

```
int semaphore_lock(int flag)
{
lock_sem_var.sem_num = flag;
if(semop(semid, &lock_sem_var, 1) == -1) {
        return -1;
}
return 0;
}

int semaphore_unlock(int flag)
{
ulock_sem_var.sem_num = flag;
if(semop(semid, &ulock_sem_var, 1) == -1) {
printf("unlock error. errno is %d flag is
%d\n", errno, flag);
        return -1;
}
        return 0;
}

void remove_semaphore()
{
semctl(semid, 0, IPC_RMID);
}
```

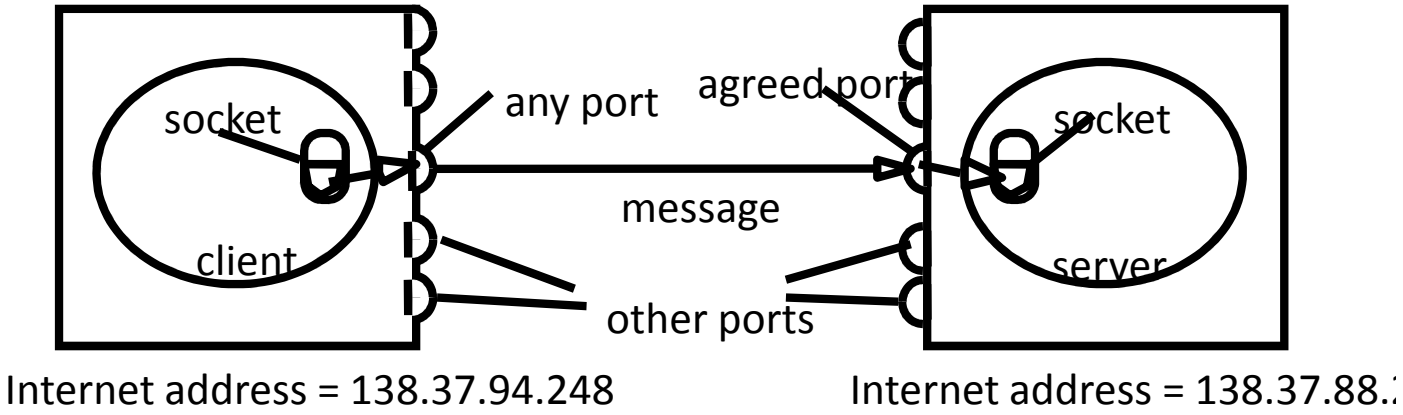
```
void critical_resource(){
char c;
printf("request from %d\n", getpid());
while(1) {
c = getchar();
if(c != '\n')     break;
}
if(isdigit(c) != 0) {
printf("found a number. Press CTL + C to exit\n");
while(1) {}
}}
void process_for_write(){
int i;
while(1) {
        i = semaphore_lock(0);
        if(i == 0) {
        printf("resource locked.....\n");
        critical_resource();
        semaphore_unlock(0);
        printf("resource unlocked.....\n");
}
sleep(1);}
}
```

```
void process_exit()
{
        remove_semaphore();
        exit(1);
}

void init_process_one()
{
        if(fork() == 0) {
        printf("pid is %d\n", getpid());
        signal(SIGINT, process_exit);
        process_for_write();
        while(1);
        }
}
```

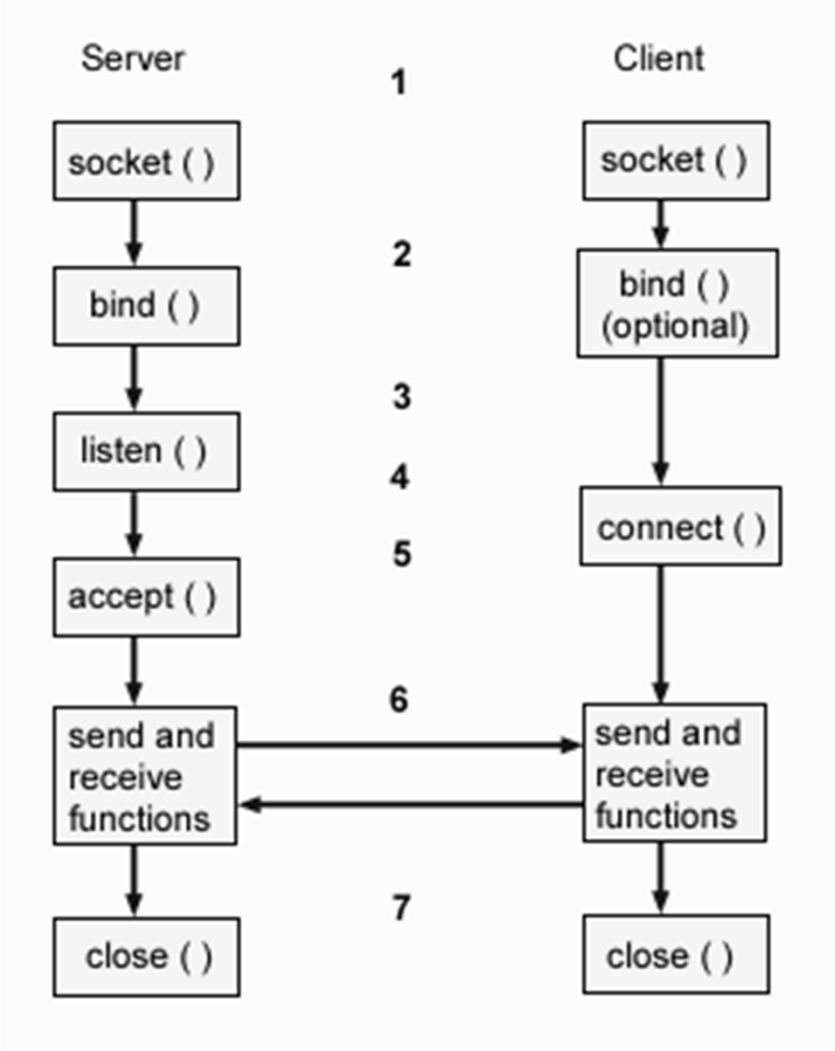
```
void init_process_two()
{
        if(fork() == 0) {
        printf("pid is %d\n", getpid());
        signal(SIGINT, process_exit);
        process_for_write();
        while(1);
        }
}

int main()
{
        signal(SIGINT, SIG_DFL);
        signal(SIGINT, process_exit);
        init_semaphore();
        set_sem_val();
        init_process_one();
        init_process_two();
        while(1);
        return 0;
}
```

Socketi

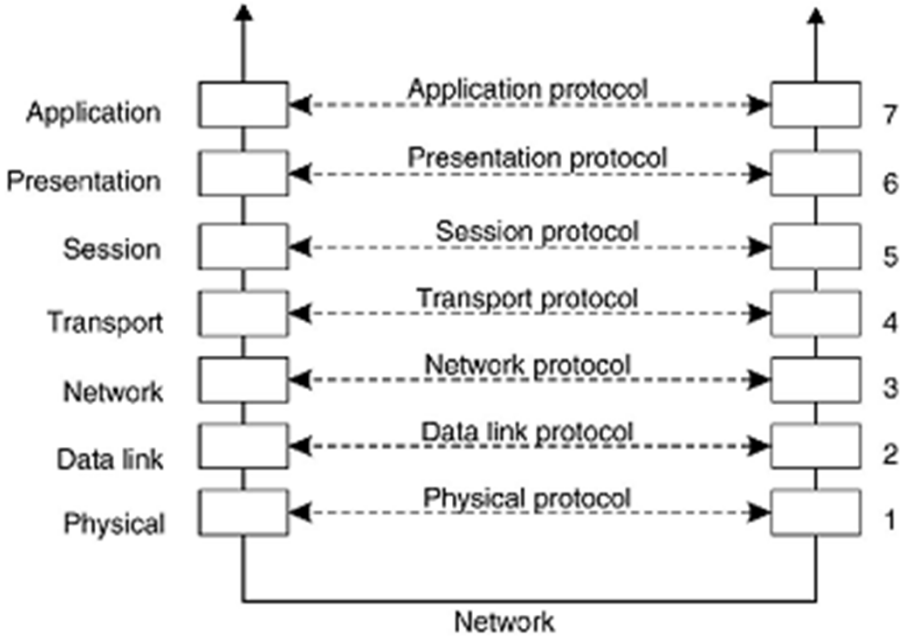
- IPC metoda koja omogućava komunikaciju među procesima koji se odvijaju na različitim mašinama
- Na razini TCP
- Definiran IP adresom i portom
- Koriste se sljedeći pozivi
  - Socket
  - Bind
  - Listen
  - accept
  - connect

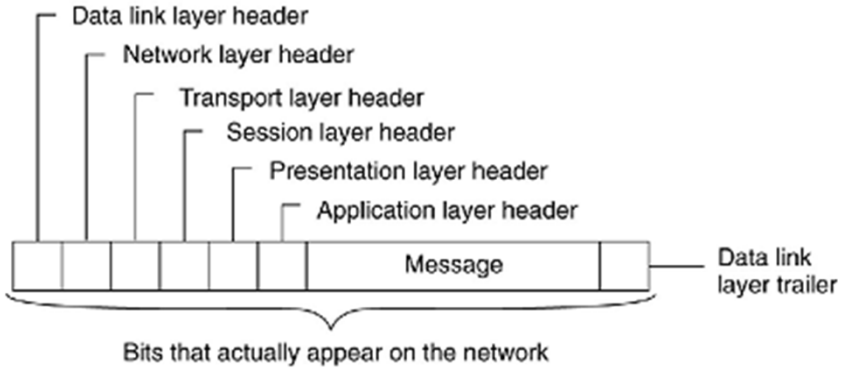


Socketi


Server
Client
1
socket ( )
socket ( )
2
bind ( )
bind ( )
(optional)
3
listen ( )
4
connect ( )
5
accept ( )
6
send and
receive
functions
send and
receive
functions
7
close ( )
close ( )

Slojevita arhitektura – ISO OSI model





Slojeviti modeli komunukacije

- Svaka razina modela nudi svoje protokole
- Skup protokola koje koristi konkretan sustav naziva se paket protokola (*protocol suite* ili *protocol stack*)
- Najpoznatiji protocol suite je *Internet protocol suite* (TCP/IP)

Protokoli niže razine

- **Fizička razina**: protokol definira električna, mehanička i signalna sučelja.
  - Koliko su naponi za 0 i 1, oblik i veličina konektora, broj pinova. Prenos bitova
- **Data link razina**: provjera ispravnosti podataka.
  - Grupira bitove u okvire (frame) i provjerava (najčešće pomoću checksume) ispravnost framea
- **Network razina**: brine se o usmjerevanju podataka do odredišta odabirom najboljeg puta (ne nužno najkraći).
  - IP protokol – conectionless protokol (svaka poruka se usmjerava individualno)

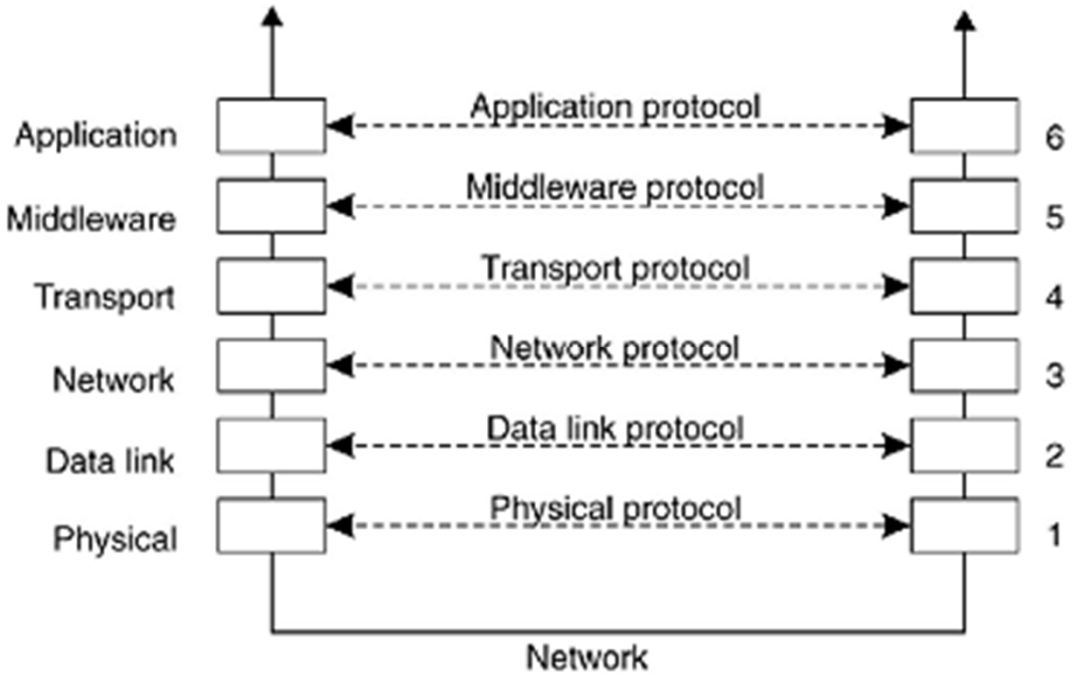
Transportni protokoli

- Zadnja neophodna razina osnovnog mrežnog skupa protokola
- Brine se o prenosu poruke od pošiljatelja do primatelja bez gubitaka
- Rastavlja poruku na pakete i sastavlja pakete u poruku
  - TCP protokol
  - UDP – nije connection oriented
  - RTP (Real Time transport protocol) – omogućava prenos podataka u realnom vremenu ali bez garancije isporuke
    - Dodatno mogu se definirati protokoli za nadzor paketa RTP

Protokoli više razine

- **Session razina** – osigurava kontrolu dialoga sinkronizacija, postavljanej checkpointa za vrijeme prenosa velikih količina podataka
- **Prezentacijska razina** – brine se o identificiranju dijelova informacija iz bitova podataka – format podataka
- **Aplikacijska razina** - najčešće se koriste protokoli samo ove razine.
  - Sa aspekta OSI modela, svi distribuirani sustavi su samo aplikacije
  - Protokoli specifični za pojedine aplikacije ili protokoli generalne namjene
  - FTP protokol
  - HTTP protokol

Adaptirani model



Middleware protokoli

- Logički se uglavnom nalazi na aplikacijskoj razini
- Različiti protokoli za različite svrhe
  - Protokol za autentikaciju –nije dio aplikacije već aplikacije koriste usluge middlewarea
  - Protokol za izvođenje operacija (commit) osigurava atomarnost operacija (transakcije)
  - Protokol za zaključavanje resursa
- Osiguravaju komunikaciju na višoj razini
  - Poziv procedure ili objekta,sinkronizacija streama, multicast

Middleware

- osnovne funkcionalnosti middlewarea :
  - Pozivi udaljenih procedura
  - Usluga stvaranja reda poruka
  - Podrška komunikaciji kroz kontinuirane medije putem streama
  - Multikastiranje – slanje prouka na više adresa
- Vrste komunikacije
  - ***Persistent***: poruka se čuva trajno dok je primatelj ne primi (kao email)
  - ***Transistent***: poruka se čuva dok se pošiljatelj ii primatelj aktivni
  - Sinkrona i asinkrona

RPC

- Usluge su ponuđene u obliku procedure
  - Tijelo procedure se izvršava na serveru
- Client stub - pakira parametre u poruku koja se predaje udaljenom serveru i ekstraktira povratnu vrijednost iz primljene poruke odgovora
- Sinkrona komunikacija – klijent je blokiran dok se procedura ne izvrši
- RMI – *Remote method invocation*

Message oriented modeli

- perzistentna asinkorna komunikacija
  - Socket

Streaming

- Kada dvije sljedne poruke imaju vremensku povezanost
- Specificira se maksimalno kašnjenje
- Npr audio ili video stream

Multicasting

- Komunikacija jedan pošiljatelj na više primatelja
- Definiranje stabla slanja od pošiljatelja do više primatelja – (samo organizacija - p2p)
- Protokoli epidemije (gossip)

Zaključak

- Lokalno dostribuirani sustavi
  - Signali,cijevi, imenovane cijevi, redovi poruka, dijeljena memorija, semafori
- Mrežno distribuirani sustavi
  - Socketi
  - Pozivi udaljenih procedura ili metoda, message orijentirana, streaming i multicasting