Operacijski sistemi

Medprocesna komunikacija API

Jurij Mihelič, FRI, Uni LJ

Vsebina

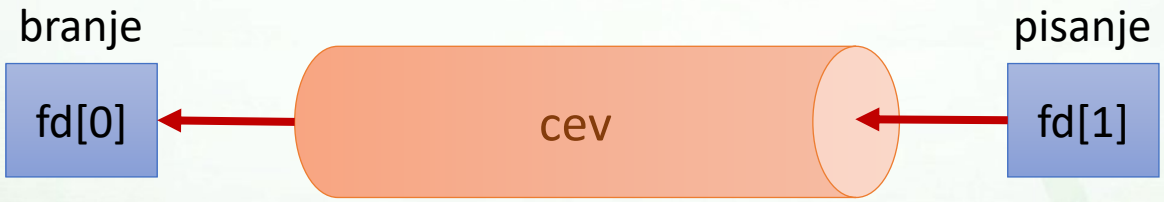
- Anonimne cevi
- Imenovane cevi
- Sporočilne vrste
- Sporočila
- Deljeni pomnilnik – System V
- Deljeni pomnilnik – POSIX
- Sporočila – QNX
- Vtičnice

Anonimne cevi

- Kaj je **cev** (pipe)?
  - posredni sinhroni način
  - enosmerna komunikacija
    - podatke pošiljamo v začetek cevi, na drugem koncu prihajajo ven
    - princip FIFO – first in first out
  - podatki se medpomnijo
    - OS poskrbi za medpomnilnik

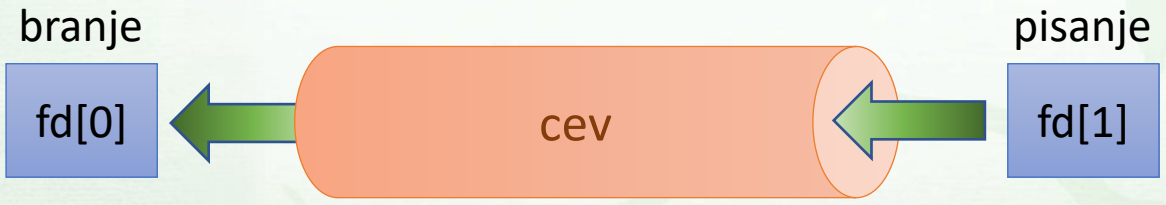
Anonimne cevi

- Ustvarjanje cevi
  - `int pipe(int fd[2])`
    - ustvari in odpre cev (bralni in pisalni konec cevi)
    - `fd[0]` … datoteka, ki predstavlja bralni konec cevi
    - `fd[1]` … datoteka, ki predstavlja pisalni konec cevi



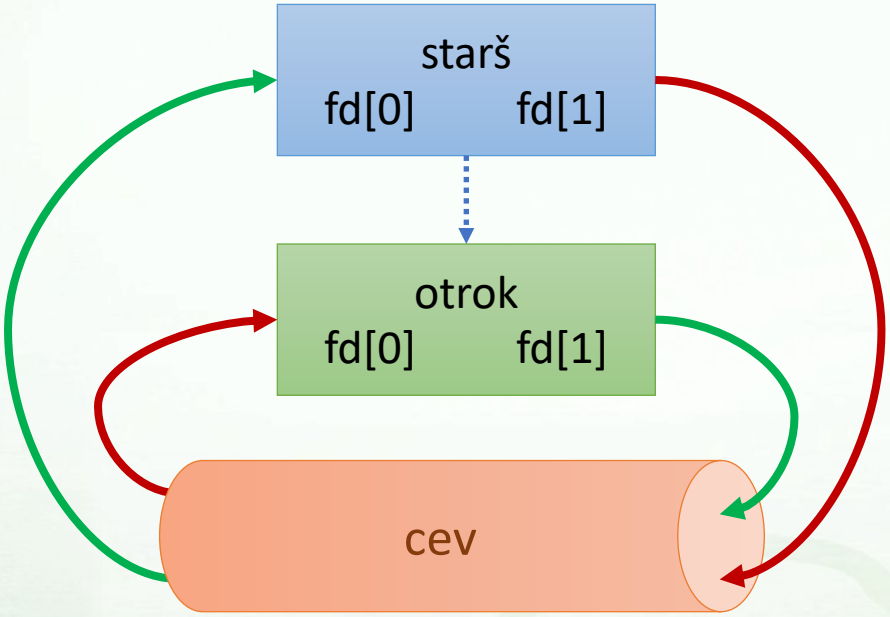
Anonimne cevi

- Uporaba cevi
  - branje
    - iz cevi beremo z `read(fd[0], …)`
  - pisanje
    - v cev pišemo z `write(fd[1], …)`
  - oba klica lahko blokirata
    - branje iz prazne cevi (prazen medpomnilnik)
    - pisanje v polno cev (poln medpomnilnik)



Anonimne cevi

- Dedovanje cevi
  - otroci dedujejo odprte datoteke, torej tudi cev
  - kombinacija klicev `pipe()` & `fork()`

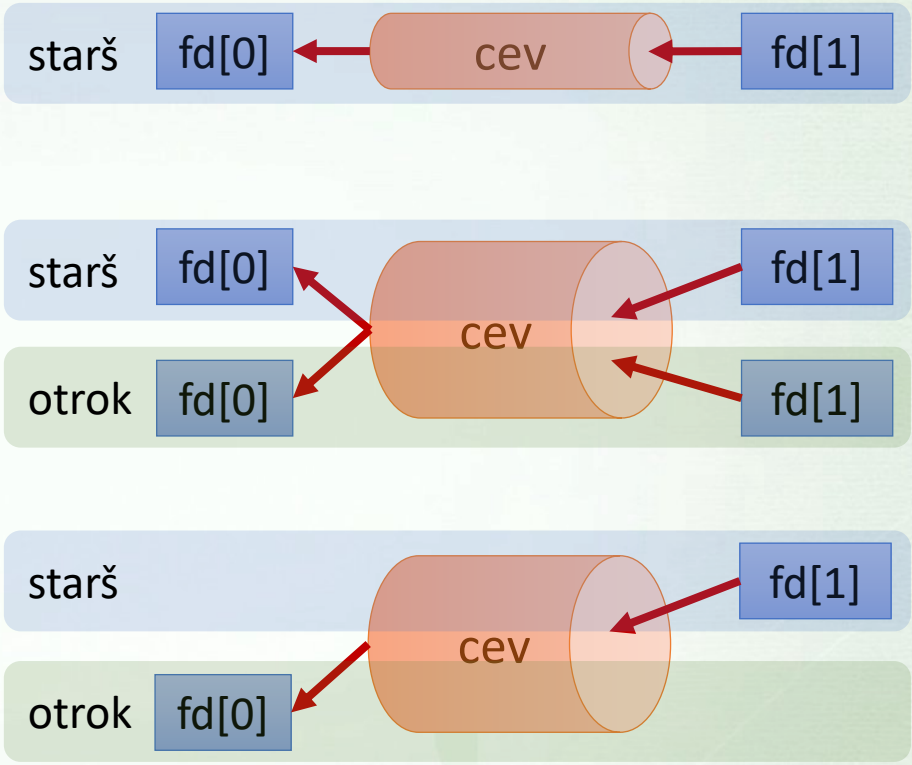


Anonimne cevi

- Dedovanje cevi
  - komunikacija je omejena na procese s skupnim prednikom
  - splošno
    - starš in potomci (otroci, vnuki, itd.)
  - tipično
    - starš in otroci
    - otroci med seboj

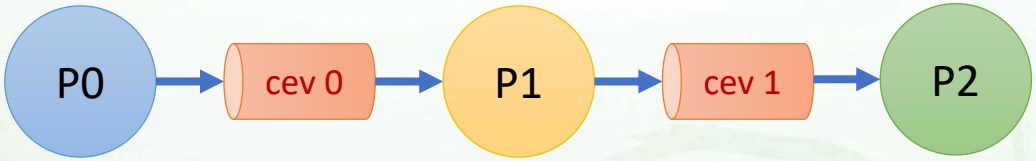
Anonimne cevi

- Primer: starš piše, otrok bere
  - ustvarjanje cevi
    - `int fd[2]`
    - `pipe(fd)`
  - razvejitev starša
    - `fork()`
  - priprava:
    - starš: `close(fd[0])`
    - otrok: `close(fd[1])`
  - branje in pisanje
    - starš: `write(fd[1])`
    - otrok: `read(fd[0])`
  - sprostitev cevi
    - starš: close(fd[1])
    - otrok: close(fd[0])



Anonimne cevi

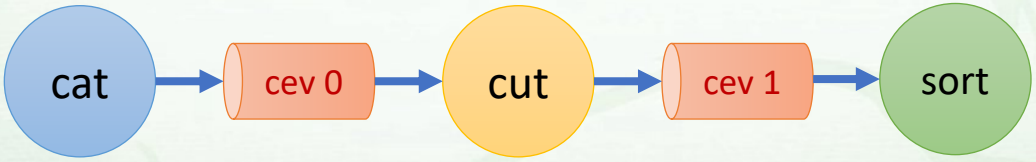
- Cevovod (pipeline)
  - zaporedje otrok istega starša
    - vhode in izhode otrok med seboj povežemo s cevmi
    - dodatna uporaba preusmerjanja
    - izhod procesa preusmerimo v vhod ustrezne cevi
    - izhod cevi preusmerimo v vhod naslednjega procesa



Anonimne cevi

- Cevovod v lupini
  - zaporedje ukazov ločeno z znakom |
  - lupina celoten cevovode pripravi sama
    - ustvari cevi: `pipe()`
    - ustvari otroke: `fork()`
    - naredi preusmeritve: `dup2()`
    - zažene ukaze (v otrocih): `exec()`
    - počaka, da se vsi otroci končajo: `waitpid()`

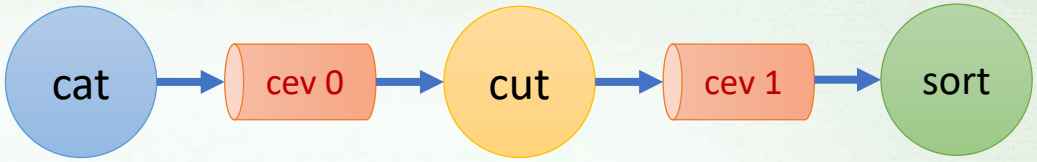
```
cat /etc/passwd | cut -d: -f7 | sort -u
```



Anonimne cevi

```
cat /etc/passwd | cut -d: -f7 | sort -u
```

- Cevovod v lupini



Lupina

pipe(fds1)
fork()
pipe(fds2)
fork()
close(fds1[0])
close(fds1[1])
fork()
close(fds2[0])
close(fds2[1])
wait(...)
wait(...)
wait(...)

Začetek cevovoda

dup2(fds1[1], 1)
close(fds1[0])
close(fds1[1])
exec(...)
exit(...)

Sredina cevovoda

dup2(fds1[0], 0)
dup2(fds2[1], 1)
close(fds1[0])
close(fds1[1])
close(fds2[0])
close(fds2[1])
exec(...)
exit(...)

Konec cevovoda

dup2(fds2[0], 0)
close(fds2[0])
close(fds2[1])
exec(...)
exit(...)

Imenovane cevi

- Imenovane cevi (named pipe, fifo)
  - podobne anonimnim cevem
  - poseben tip datoteke (pipe)
    - datoteke imajo ime (imeniški vnos)
    - naslavljanje cevi preko imena datoteke
    - zaščitni mehanizem kot za datoteke
  - neomejena komunikacija
    - proces, ki pozna ime cevi, jo lahko uporabi

Imenovane cevi

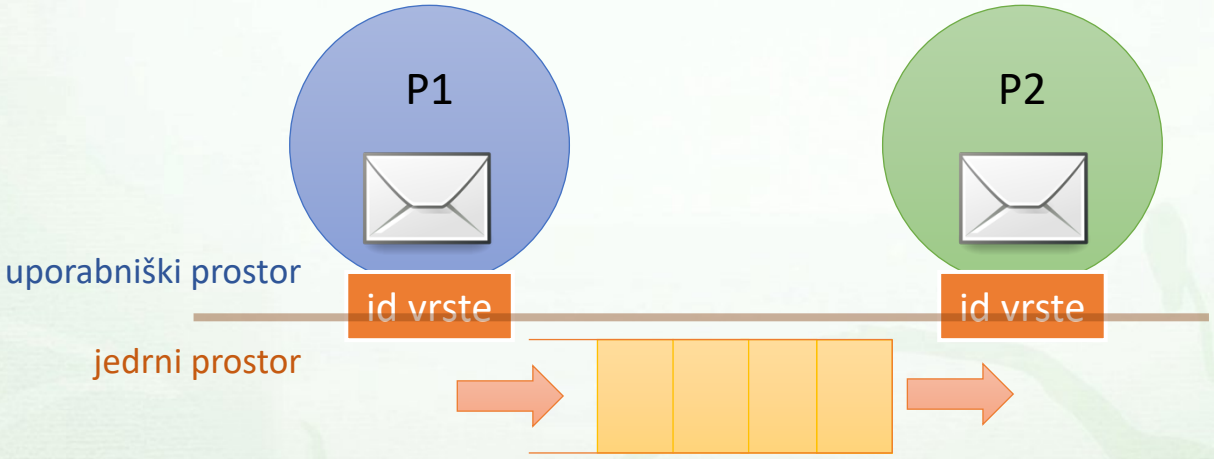
- Ustvarjanje imenovane cevi
  - sistemska klica: `mkfifo()` in `mknod()`
  - ukaza: `mkfifo` ***cev*** in `mknod` ***cev*** `p`
- Uporaba imenovane cevi
  - odpiranje in zapiranje: `open()` in `close()`
  - branje in pisanje: `read()` in `write()`

POSIX in System V

- POSIX medprocesna komunikacija
  - uporablja datoteke za identifikacijo kanala
    - datotečni deskriptor: kanal pripravljen za komunikacijo
    - veljajo datotečni zaščitni mehanizmi
- System V medprocesna komunikacija
  - uporablja ključe za identifikacijo kanala
    - ključ predhodno ustvarimo z `ftok(`*`pathname`*`, …)`
    - preko ključa nato ustvarimo kanal
    - veljajo datotečni zaščitni mehanizmi

Sporočilne vrste

- Sporočilna vrsta (message queue)
  - posredna asinhrona oblika komunikacije
  - hramba sporočil v vrsti
    - vrsta v jedru: povezan seznam sporočil
    - sporočila so lahko različno dolga
    - vsaka vrsta ima identifikator



Sporočilne vrste – System V

- Uporaba sporočilnih vrst
  - `int msgget(key, …)`
    - odpre (ali ustvari) sporočilno vrsto z oznako *key*
    - vrne deskriptor vrste *qid*
  - `int msgctl(qid, IPC_RMID, …)`
    - zapiranje vrste (in tudi nadzor obnašanja)
  - `int msgsnd(qid, msg, size, …)`
    - pošlje sporočilo *msg* v vrsto z deskriptorjem *qid*
  - `int msgrcv(qid, msg, size, …)`
    - prejeme sporočilo, ki čaka v vrsti z deskriptorjem *qid*

Deljeni pomnilnik

- Deljeni pomnilnik
  - segment: deljeni kos pomnilnika
    - niso nujno sklenjeni okviri strani
    - z vidika procesa gre za sklenjeni kos
  - deljeni pomnilnik je sistemsko pogojen
    - OS omejuje število segmentov in skupno velikost deljenega pomnilnika
    - deljeni pomnilnik lahko *preživi* proces
  - potek komunikacije
    - ustvarjanje in sprostitev segmenta
    - priklop in odklop segmenta

Deljeni pomnilnik – System V

- Uporaba
  - `int shmget(`*key*`, ...)`
    - odpre (ali ustvari) deljeni pomnilnik (segment)
    - povezava do segmenta z oznako *key*
    - vrne deskriptor segmenta *shmid*
  - `int shmctl(`*shmid*`, IPC_RMID, …)`
    - sprostitev deljenega pomnilnika
    - pomaga tudi ponovni zagon OS ☺

Deljeni pomnilnik – System V

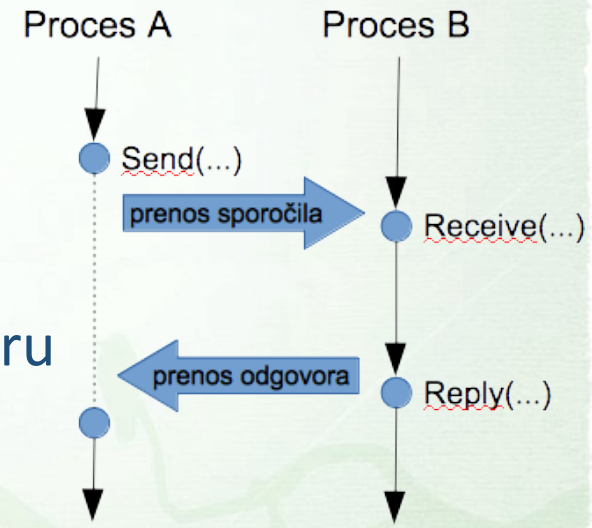
- Uporaba
  - `int shmat(`*`shmid, …`*`)`
    - priklop (attach) segmenta z deskriptorjem *shmid*
    - segment *shmid* preslika v naslovni prostor procesa
    - vrne kazalec *addr* na deljeni pomnilnik
  - `int shmdt(`*`addr, …`*`)`
    - odklop (detach) segmenta
    - navidezni pomnilnik: odstrani preslikavo na deljeni pomnilnik na naslovu *addr*
  - možen večkratni priklop in odklop

Deljeni pomnilnik – POSIX

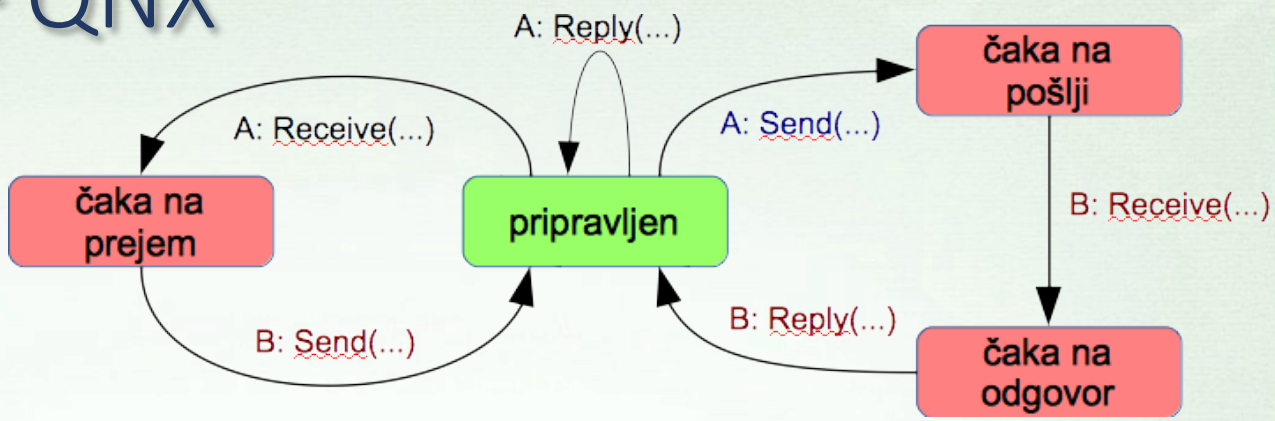
- Uporaba
  - povezava do segmenta
    - `int shm_open(`*`fname`*`, ...)`
    - vrne datotečni deskriptor
  - sprostitev povezave do segmenta
    - `close()` ali `shm_close()`
  - sprostitev deljenega pomnilnika
    - `int shm_unlink(`*`fname`*`)`
  - priklop segmenta
    - `void* mmap(...)`
  - odklop segmenta
    - `int munmap(`*`addr`*`, ...)`

Sporočila – QNX

- Sporočila (v QNX OS)
  - neposredna sinhrona oblika komunikacije
    - neposredna: naslavljanje preko PID
    - sinhrona: operacije blokirajo (povzročijo čakanje)
  - vsako sporočilo zahteva tudi odgovor
    - pošlji-prejmi-odgovori
    - dokler ni odgovora izvorni proces blokira
  - hranjenje sporočila
    - v procesovem pomnilniškem prostoru
    - zaradi sinhronosti vrsta ni potrebna



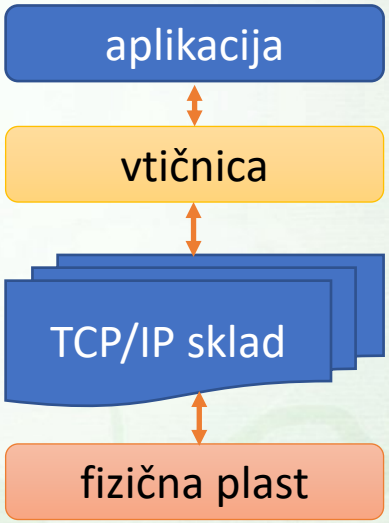
Sporočila – QNX



- Sporočila (v QNX OS)
  - Send(pid, msg, reply, …)
    - pošlje sporočilo `msg` procesu `pid`
    - nato blokira dokler ne dobi odgovora v `reply`
  - Receive(0, msg, …)
    - prejme sporočilo v `msg` kateregakoli procesa
    - čaka, če sporočilo ni na voljo
    - kot rezultat vrne `pid` izvornega procesa
  - Reply(pid, reply, …)
    - odgovori procesu `pid` z odgovorom `reply`

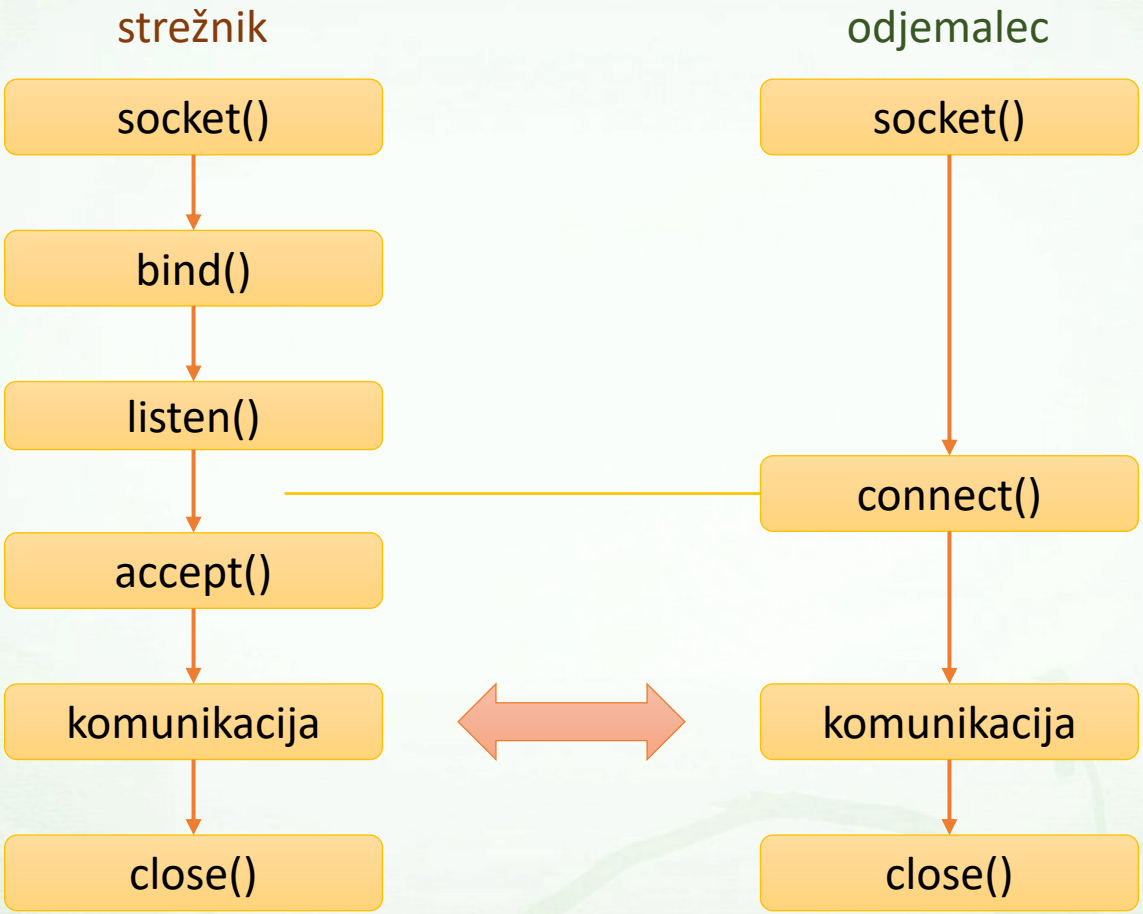
Vtičnice

- Vtičnice
  - medprocesna in tudi mrežna komunikacija
  - posredna dvosmerna komunikacija
  - tipična uporaba: odjemalec / strežnik
    - strežnik vzpostavi vtičnico na znanem naslov
    - nato čaka na zahteve odjemalcev
    - odjemalec se poveže na vtičnico
    - nato sledi izmenjava sporočil



Vtičnice

- Potek dogodkov – povezavna komunikacija



Vtičnice

- Vrste vtičnic
  - AF_LOCAL: lokalna vtičnica
    - ne omogoča omrežnih povezav
    - naslavljanje preko datotek (imenski vnos)
    - ni posebnih protokolov
  - AF_INET: internetni protokol v4
    - SOCK_STREAM: TCP
    - SOCK_DGRAM: UDP
    - SOCK_RAW: IP, ICMP
  - AF_INET6: internetni protokol v6
    - podobno kot AF_INET
  - itd.