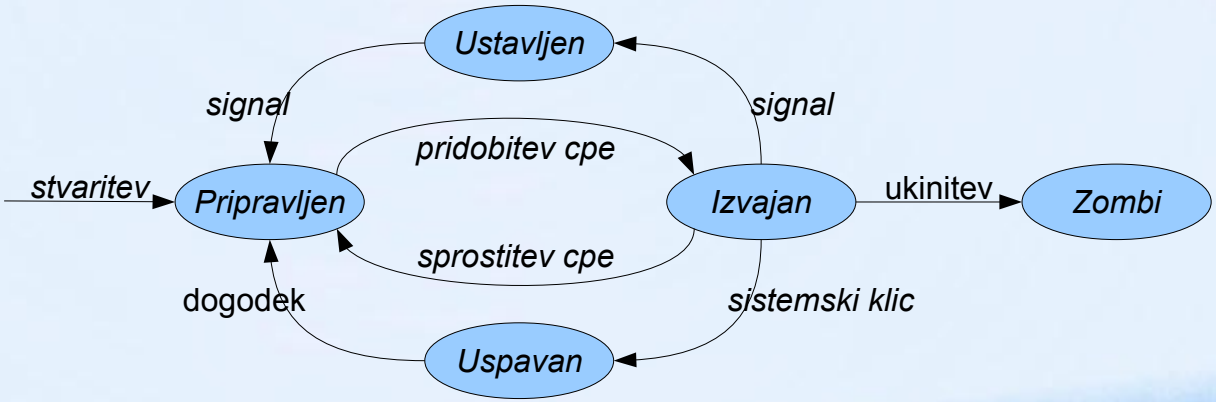
Vzporednost procesov

Vaje Operacijski sistemi
Jure Mihelič

Stanje procesa

- Linux stanja.
  - Pripravljen ali izvajan – `TASK_RUNNING`.
    - Z uporabniškega vidika se proces v obeh primerih izvaja.
  - Ustavljen – `TASK_STOPPED`.
  - Uspavan – `TASK_INTERRUPTIBLE`, `TASK_UNINTERRUPTIBLE`.
  - Zombi.



Sistemski klici

- Info o procesu.
  - PID procesa: `int getpid()`.
  - PPID procesa: `int getppid()`.
- Razno.
  - Spanje: `int sleep(unsigned int seconds)`.
    - Spanje za `seconds` sekund.
    - Lahko se zbudi prej, če prejme signal.

Stvaritev procesa

- Sistemski klic `int fork()`.
  - Ustvari se nov proces (otrok), katerega starš je tekoči proces.
  - Otrok je kopija oz. klon starša.
    - Kopira se koda, podatki, sklad, rokovalniki signalov, itd.
    - Kopirajo se deskriptorji odprtih datotek.
    - Ključavnice se ne kopirajo.
    - *Copy-on-write* – podatki se ne kopirajo dokler jih starš ali otrok ne spreminja.

Stvaritev procesa

- Sistemski klic `int fork()`.
  - V primeru neuspeha vrne -1.
  - Sicer pa se iz funkcije vrneta dva procesa.
    - Starš, kateremu vrne PID otroka.
    - Otrok, kateremu vrne 0.

```
int pid = fork();
if (pid < 0)
    // NAPAKA
else if (pid == 0)
    // OTROK
else
    // STARŠ
```

Zagon programa

- Družina funkcij `int exec(...)`.
  - Argumenti funkcije.
    - Pot do izvršljive datoteke, argumenti programa.
    - Lahko podamo tudi okoljske spremenljivke.
  - Nadomestitev trenutnega procesa.
    - PID in PPID se ne spremenita.
    - Podeduje odprte datoteke, trenutni in korenski imenik.
    - Zagon izvršljive datoteke → nova koda in podatki.
    - Nova sta tudi kopica in sklad.

Zagon programa

- Družina funkcij `int exec(...)`.
  - `execl(...)`, `execlp(...)`, `execle(...)`.
  - `execv(...)`, `execvp(...)`, `execve(...)`.
  - Argumenti ukaza so argumenti funkcije (`l`) ali v tabeli (`v`).
  - Iskanje ukaza preko $PATH (`p`).
  - Podajanje okoljskih spremenljivk (`e`).

```
execl("/bin/ls", "ls", "-alp", "/home/jure", NULL);

char* args[] = { "ls", "-alp", "/home/jure", NULL };
execvp("ls", args);

execvp(argv[1], &argv[1]);
```

Končanje procesa

- Sistemski klic `exit(int status)`.
  - Funkcija, iz katere se nikoli ne vrnemo.
  - Končanje procesa.
    - Sprostitev virov, zaprtje datotek, itd.
    - Staršu se pošlje signal `SIGCHLD`.
  - Kaj se zgodi z otroki procesa?
    - Če gre za „*lupino*“, se otrokom pošlje `SIGHUP` in `SIGCONT`.
      - Privzeti odziv na ta signal je ukinitev procesa.
    - Morebitne otroke posvoji proces **`init`**.
      - Posvojenim procesom pravimo **sirote**.

Končanje procesa

- Sistemski klic `exit(int status)`.
  - Izhodni status.
    - Se shrani v jedru v deskriptorju procesa.
    - Prevzame ga starš procesa z `wait()`.
    - Dokler starš ne prevzame statusa je proces **zombi**.
  - Proces `init` kot krušni starš / sirotišnica.
    - Skrbi za sirote.
    - Privzeti odziv na `SIGCHLD` je izvedba wait().

Čakanje otroka

- Družina funkcij `int wait(...)`.
  - Čakanje na otroka, da se konča; in
  - prevzem njegovega izhodnega statusa.
  - Čakanje na določenega otroka.
    - `int waitpid(pid, &status, opcije)`
  - Čakanje na poljubnega otroka.
    - `int wait(&status)`
      - enako kot `waitpid(-1, &status, 0)`

Čakanje otroka

- Družina funkcij `int wait(...)`.
  - Izhodni status se *skriva* v spremenljivki *status*.
  - Branje izhodnega statusa.
    - `man waitpid`
    - makro `WIFEXITED(status)`.
      - Se je program končal z `exit()`?
    - makro `WEXITSTATUS(status)`.
      - Iz spremenljivke `status` se izlušči izhodni status.

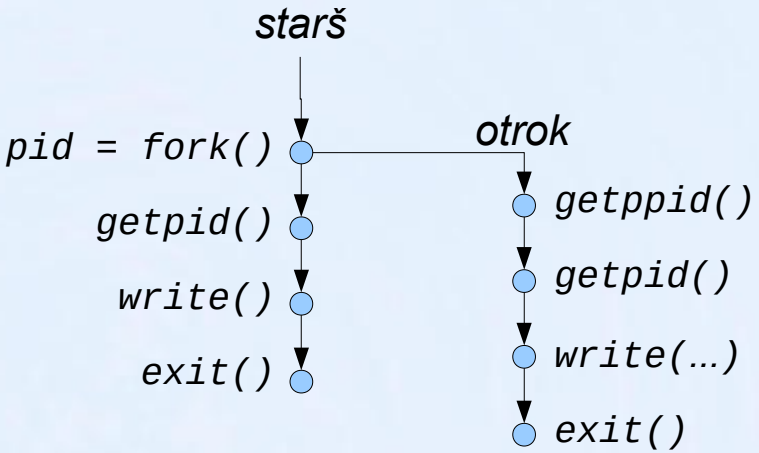
```
if (WIFEXITED(status))
    status = WEXITSTATUS(status));
```

Primeri

Vaje Operacijski sistemi
Jure Mihelič

Procesi v C

- Vejitev.

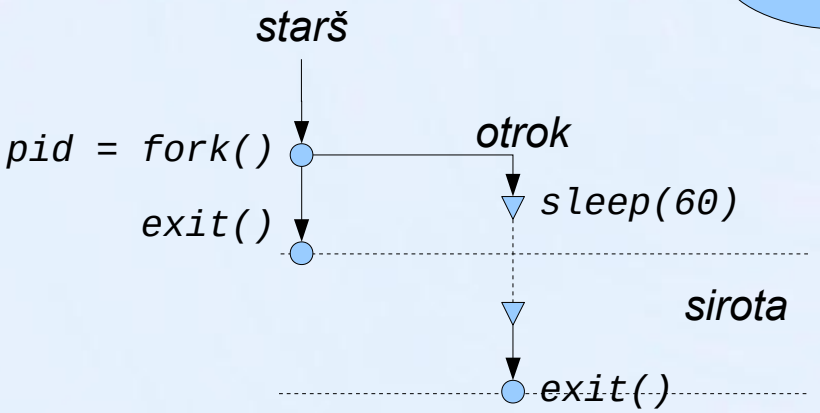



```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int pid = fork();
    if (pid < 0)
        perror(argv[0]);
    else if (pid == 0)
        printf("Sem otrok %i s staršem %i.\n", getpid(), getppid());
    else
        printf("Sem starš %i z otrokom %i,\n", getpid(), pid);
}
```

Procesi v C

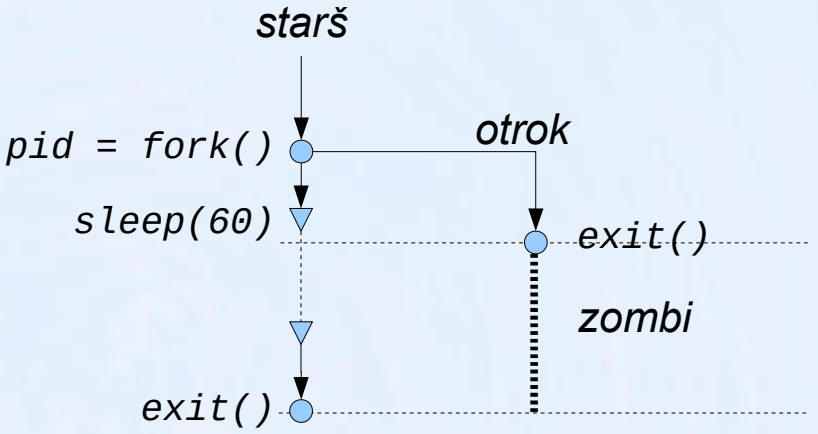
- Sirota.

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int pid = fork();
    if (pid < 0)
        perror(argv[0]);
    else if (pid == 0)
        // otrok zaspi za 60 sekund
        sleep(60);
}
```



Procesi v C

- Zombi.

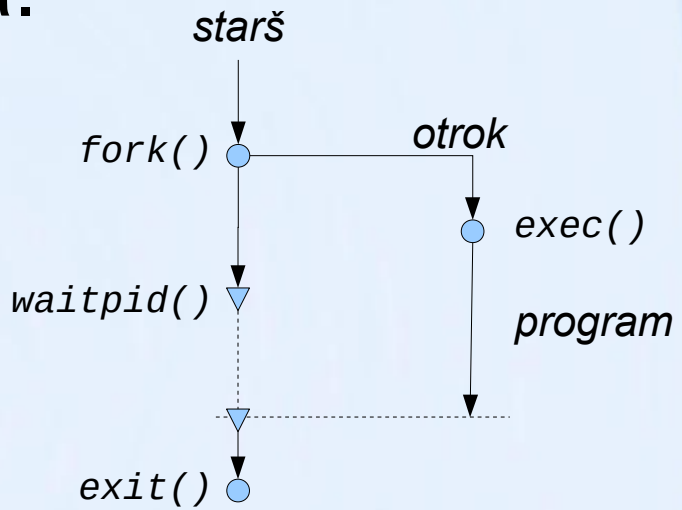



```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int pid = fork();
    if (pid < 0)
        perror(argv[0]);
    else if (pid > 0)
        // starš zaspi za 60 sekund
        sleep(60);
}
```

Procesi v C

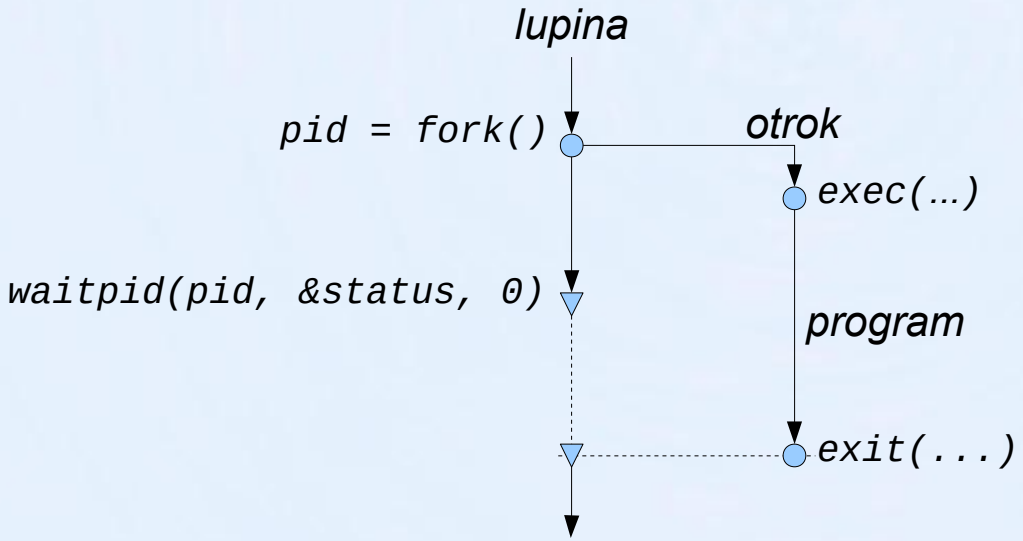
- Zagon procesa oz. programa.

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>

int main(int argc, char* argv[]) {
    int pid = fork();
    if (pid < 0) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        execvp(argv[1], &argv[1]);
        perror("exec");
        exit(EXIT_FAILURE);
    } else {
        int status;
        if (waitpid(pid, &status, 0) < 0) {
           perror("waitpid");
           exit(EXIT_FAILURE);
        }
        if (WIFEXITED(status))
           printf("Izhodni status otroka: %i\n",
               WEXITSTATUS(status));
    }
    exit(EXIT_SUCCESS);
}
```



Procesi v lupini

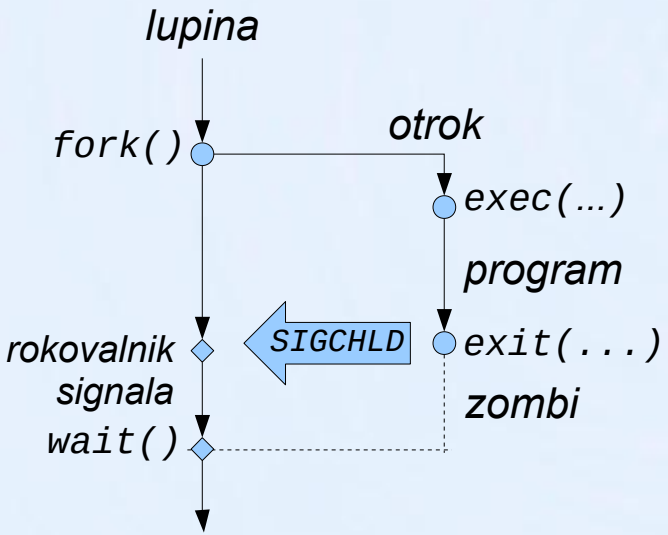
- Zagon programa v ospredju.



Procesi v lupini

- Zagon programa v ozadju.

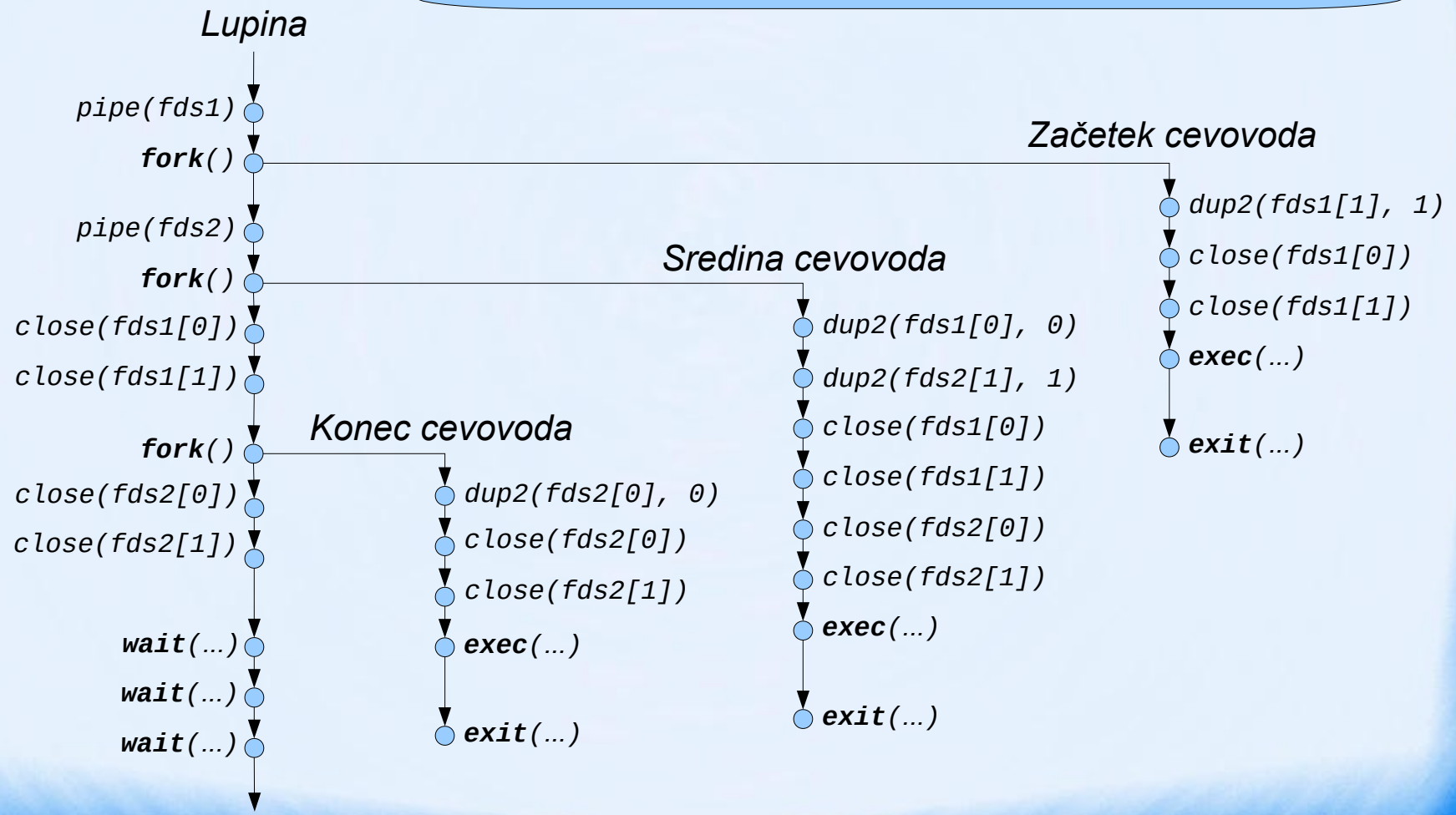
```
xeyes &
```



Procesi v lupini

- Cevovod.

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



Procesi v lupini

- Lupina in sistemski klici.
  - `$$`
    - `getpid()` lupine
  - `$PPID`
    - `getppid()` lupine
  - `$BASHPID`
    - `getpid()`

Procesi v lupini

- Lupina in sistemski klici.
  - Zagon zunanjega ukaza.
    - `fork(), exec(), waitpid()`
  - Zagon zunanjega ukaza v ozadju.
    - `fork(), exec()`
  - Izvajanje v podlupini: (*vgrajeni_ukaz*).
    - `fork(), waitpid()`
  - Izvajanje v podlupini v ozadju: (*vgrajeni_ukaz*) &.
    - `fork()`

```
ls
```

```
xeyes &
```

```
( read line; echo $line )
```

```
( echo {24..42} >fri.txt ) &
```

Procesi v lupini

- Lupina in sistemski klici.
  - Spanje za določen čas: `sleep 42`.
    - `sleep(42)`
  - Čakanje vseh otrok: `wait`.
    - Ponavljanje `wait()`.
  - Čakanje otroka: `wait 12345`.
    - `waitpid()`
  - Zaključek programa: `exit 42`.
    - `exit(42)`