Operacijski sustavi - vježba 2

1. Ostvarivanje višezadaćnog rada pomoću više jednodretvnenih procesa

Program je skup instrukcija i podataka koji se nalaze na disku. Kako bi se pokrenuo novi program treba se pozvati novi proces koji je *okolina u kojem se program izvršava.*

Proces se sastoji od tri segmenta; *segment instrukcija, segment korisničkih podataka i segment sustavskih podataka.* Program inicijalizira instrukcije i korisničke podatke, te nakon inicijalizacije više ne postoji čvrsta veza između procesa i programa koji on izvodi.

1.1. Sustavski poziv `fork()`

Sustavskim pozivom `fork()` se zahtijeva stvaranje novog procesa iz postojećeg. Ako dođe do greške pri stvaranju 'dijete' procesa, vraćena je vrijednost -1, a dijete nije stvoreno. `fork()` nema nikakvih argumenata, pa programer ne može biti odgovoran za bilo kakvu grešku pri izvedbi, već je kriva jezgra.

1.2. Sustavski pozivi `exit()`, `wait()` i `getpid()`

Poziv `exit()` završava izvođenje procesa koji poziva tu funkciju. Sustavski poziv `wait()` čeka de jedan od procesa dijece završi sa radom. Ne može se specificirati koji proces se čeka, te se najčešće piše `wait(NULL)`. Poziv `getpid()` vraća identifikacijski broj procesa.

1.3. Sustavski poziv za stvaranje i rad sa zajedničkim prostorom

`int shmget(key_t key, int size, int flags)` - ovaj sustavski poziv pretvara ključ nekog segmenta zajedničkog prostora u njegov identifikacijski broj ili stvara novi segment. Novi segment barem `size` bitova bit će stvoren ako se kao ključ uporabi `IPC_PRIVATE`. U treći argument stavljaju se dozvole pristupa - `0600` znači da korisnik može čitati i pisati u segment, a grupa ne.

`shmget` vraća identifikacijski broj segmenta zajedničkog memorijskog prostora, ili vraća -1 u slučaju greške.

Proces veže segment na svoj adresni prostor pomoću `shmat(int segid, const void *addr, int flags)`. `segid` je broj dobiven pomoću sustavskog poziva `shmget`, `addr` se najčešće određuje kao NULL da jezgra sama odredi prostor u koji će spremiti segment, i `flags` se također najčešće zadaju kao 0.

Segment se može otpustiti od procesa pozivom `shmdt(void *addr)`. Segment ostaje ne dirnut i može mu se opet pristupiti tako da se ponovno veže na proces.

Uništavanje segmenta se postiže pozivom `shmctl(int segid, int cmd, struct shmid_ds *sbuf)`. Kroz `segid` se prosljeđuje adresa prostora koji želimo uništiti, u `cmd` se prosljeđuje `IPC_RMID`, a `sbuf` može ostati prazan, tj. NULL.

2. Višedretvenost

Ideja višedretvenog programiranja je to da se program sastoji od više jedinica koje se mogu samostalno izvoditi. Programer ne mora razmišljati o redoslijedu njihova izvođenja, već to obavlja operacijski sustav.

U praksi bi stvaranje novog procesa trebalo kopirati sve postojeće dretve u 'dijete' proces, no u stvarnosti se kopira samo glavna dretva.

2.1. Stvaranje dretava

Sve dretve, osim prve i glavne, nastaju pozivanjem sustavskog poziva `pthread_create`.

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
    void *(start_routine)(void *), void *arg);
```

`thread` je kazaljka na mjesto gdje se u memoriji sprema ID novonastale dretve, `addr` je adresa strukture koja sadrži podatke o atributima s kojom želi stvoriti dretvu (ako se za `addr` postavi NULL, onda se uzimaju pretpostavljene vrijednosti). `start_routine` predstavlja pokazivač na funkciju koju će novonastala dretva imati kao početnu, a `arg` je adresa parametra koji se dretvi prenosi - može biti NULL.

2.2. Završetak rada dretve

Normalan završetak rada dretve je izlazak iz njene inicijalne funkcije ili sustavskim pozivom `pthread_exit`.

```
int pthread_exit(void *status);
```

`status` je kazaljka na stanje s kojim dretva završava. U pravilu se ne koristi, nego se koristi `pthread_cancel`. Završavanjem dretve iz glavne funkcije se vraća NULL.

Dretva čeka na završetak druge dretve pozivom `pthread_join`.

```
int pthread_join(pthread_t cekana_dr, void **stanje);
```

`cekana_dr` je identifikacijski broj dretve na koju se čeka, a `stanje` je pokazivač na pokazivač izlaznog statusa dočekane dretve.