SVEUČILIŠTE U MOSTARU
FAKULTET STROJARSTVA, RAČUNARSTVA I ELEKTROTEHNIKE
PREDDIPLOMSKI STUDIJ RAČUNARSTVA

OPERACIJSKI SUSTAVI

VJEŽBE

Nastavnik: prof.dr.sc. Sven Gotovac
gotovac@fesb.hr

Asistent: Željko Šeremet
zeljko.seremet@fsre.sum.ba

MOSTAR, SVIBANJ 2024.

Ostvarenje višezadaćnog rada s pomoću više procesa

Sadržaj

- Sustavski poziv ***fork***
- Sustavski pozivi ***exit***, ***wait*** *i* ***getpid***
- Pokretanje paralelnih procesa
- Zajednički adresni prostor
- Sustavski pozivi za stvaranje i rad sa zajedničkim spremnikom
- Struktura programa sa paralelnim procesima i zajedničkim spremnikom

Uvod (1)

- Program je skup instrukcija i podataka koji se nalaze u datoteci na disku.
- U opisu datoteke ona je opisana kao izvršna i njen sadržaj je organiziran prema pravilima jezgre.
- Sve dok sadržaj datoteke odgovara pravilima i dok je označena kao izvršna, program može biti pokrenut.
- Kako bi se pokrenuo novi program prvo treba (pozivom jezgre) stvoriti novi proces koji je okolina u kojem se izvršava program.

Uvod (2)

- Proces se sastoji od tri segmenta: segment instrukcija, segment korisničkih podataka i segment sustavskih podataka.
- Program inicijalizira segment instrukcija i korisničke podatke. Nakon inicijalizacije više nema čvrste veze između procesa i programa koji on izvodi.
- Proces dobiva sredstva (više spremnika, datoteke, itd.) koji nisu prisutni u samom programu, mijenja podatke, itd.
- Iz jednog programa može se inicijalizirati više procesa koji se paralelno izvode.

SUSTAVSKI POZIV ***fork*** (1)

- Sustavskim pozivom ***fork*** zahtijeva se stvaranje novog procesa iz postojećeg.
- Kada proces koji se trenutno izvodi pokrene novi proces, pokrenuti proces postaje "dijete" procesa "roditelja" koji ga je pokrenuo.
- Dijete dobija kopije segmenta instrukcija i segmenta podataka roditelja.
- U stvari, budući se segment instrukcija najčešće ne mijenja, jezgra može uštediti vrijeme i memoriju tako da postavi taj segment kao zajednički za oba procesa (sve dok ga jedan od njih ne odluči inicijalizirati novim programom).
- Također, dijete nasljeđuje većinu sustavskih podataka roditelja.

SUSTAVSKI POZIV *fork* (2)

- **int fork(void) ;**
- U ovaj sustavski poziv ulazi jedan proces, a iz njega izlaze dva odvojena procesa ("dijete" i "roditelj") koji dobivaju svaki svoju povratnu vrijednost.
- Proces dijete dobiva rezultat 0, a roditelj dobiva identifikacijski broj procesa djeteta.
- Ako dođe do greške, vraćena vrijednost je -1, a dijete nije ni stvoreno. ***fork*** nema nikakvih argumenata, pa programer ne može biti odgovoran za grešku već je ona rezultat nemogućnosti jezgre da kreira novi proces zbog nedostatka nekog od potrebnih sredstava.

SUSTAVSKI POZIV ***fork*** (3)

- Dijete nasljeđuje većinu atributa iz segmenta sustavskih podataka kao što su aktualni direktorij, prioritet ili identifikacijski broj korisnika. Manje je atributa koji se ne nasljeđuju:
  - Identifikacijski brojevi procesa djeteta i roditelja su različiti, jer su to različiti procesi.
  - Proces dijete dobiva kopije otvorenih opisnika datoteka (*file descriptor*) od roditelja. Dakle to nisu isti opisnici datoteka, tj. procesi ih ne dijele. Međutim, procesi dijele kazaljke položaja u datotekama (*file pointer*). Ako jedan proces namjesti kazaljku položaja na određeno mjesto u datoteci, drugi proces će također čitati odnosno pisati od tog mjesta. Za razliku od toga, ako dijete zatvori svoj opisnik datoteke, to nema veze s roditeljevim opisnikom datoteke.
  - Vrijeme izvođenja procesa djeteta je postavljeno na nula.

SUSTAVSKI POZIV *fork* (4)

- Dijete se može inicijalizirati novim programom (poziv ***exec***) ili izvoditi poseban dio već prisutnog programa, dok roditelj može čekati da dijete završi ili paralelno raditi nešto drugo. Osnovni oblik upotrebe sustavskog poziva ***fork*** izgleda ovako:

```
if (fork() == 0) {
   posao procesa djeteta
   exit(0);
}
nastavak rada procesa roditelja (ili ništa);
wait(NULL);
```

Plavo - izvodi proces roditelj, **zeleno** - izvode oba procesa (provjera povratne vrijednosti fork()-a), **crveno** - izvodi proces djeteta.

SUSTAVSKI POZIVI *execv*

- **int execv(const char *_program_, char **_args_);**
- Obično proces dijete sa sistemskim pozivom *execve* ili nekim sličnim poziva neki novi program. Npr. kada korisnik u komandnoj ljusci utipka npr. sort, ljuska sa forkom klonira proces i u procesu dijetetu izvrši sort. Razlog ovog postupka u 2 koraka je da bi se djetetu omogućila manipulacija sa file deskriptorima nakon fork-a, a prije izvršavanja nekog drugog programa, kako bi se mogle izvršiti redirekcije standardnih ulaza i izlaza.

SUSTAVSKI POZIV *CreateProcess*

- U Windowsima (pritom se misli na WindowsAPI), koristimo API poziv ***CreateProcess***, koji obavlja i kreiranje novog procesa i učitavanje novog programa u novokreirani proces.

SUSTAVSKI POZIVI **exit**, **wait** i **getpid** (1)

- **void exit(int status) ;**
- Poziv ***exit*** završava izvođenje procesa koji poziva tu funkciju. Prije završetka, uredno se zatvaraju sve otvorene datoteke. Ne vraća nikakvu vrijednost jer iza njega nema nastavka procesa. Za ***status*** se obično stavlja 0 ako proces normalno završava, a 1 inače. Roditelj procesa koji završava pozivom ***exit*** prima njegov ***status*** preko sustavskog poziva ***wait***.

SUSTAVSKI POZIVI **exit**, **wait** i **getpid** (2)

- **int wait(int *statusp) ;**
- Ovaj sustavski poziv čeka da neki od procesa djece završi (ili bude zaustavljen za vrijeme praćenja), s tim da mu se ne može reći koji proces treba čekati. Funkcija vraća identifikacijski broj procesa djeteta koji je završio i sprema njegov status (16 bitova) u cijeli broj na koji pokazuje ***statusp***, osim ako je taj argument NULL. U tom slučaju se status završenog procesa gubi. U slučaju greške (djece nema, ili je čekanje prekinuto primitkom signala) rezultat je 1.

SUSTAVSKI POZIVI **exit**, **wait** i **getpid** (3)

- Postoje tri načina kako može završiti proces: pozivom ***exit***, primitkom signala ili padom sustava (nestanak napajanja ili slično). Na koji je način proces završio možemo pročitati iz statusa na koji pokazuje *statusp* osim ako se radi o trećem slučaju (vidi **man wait**).
- Ako proces roditelj završi prije svog procesa djeteta, djetetu se dodjeljuje novi roditelj - proces ***init*** s identifikacijskim brojem 1. ***init*** je važan prilikom pokretanja sustava, a u kasnijem radu većinom izvodi ***wait*** i tako "prikuplja izgubljenu djecu" kada završe.

SUSTAVSKI POZIVI **exit**, **wait** i **getpid** (4)

- Ako proces dijete završi, a roditelj ga ne čeka sa ***wait***, on postaje proces-zombi (*zombie*). Otpuštaju se njegovi segmenti u radnom spremniku, ali se zadržavaju njegovi podaci u tablici procesa. Oni su potrebni sve dok roditelj ne izvede ***wait*** kada proces-zombi nestaje. Ako roditelj završi, a da nije pozvao ***wait***, proces-zombi dobiva novog roditelja (***init***) koji će ga prikupiti sa *wait*.
- **pid_t getpid() ;**
  - Poziv ***getpid*** vraća identifikacijski broj procesa (PID).

Pokretanje paralelnih procesa

- U ovoj vježbi trebat će pokrenuti više procesa tako da rade paralelno. To se može izvesti s dvije petlje. U prvoj se stvaraju procesi djeca pozivom ***fork***-a svako dijete poziva odgovarajuću funkciju. Iza poziva funkcije treba se nalaziti ***exit*** jer samo roditelj nastavlja izvršavanje petlje. Nakon izlaska iz prve petlje, roditelj poziva ***wait*** toliko puta koliko je procesa djece stvorio.

```
for (i = 0; i < N; i++)
  switch (fork()) {
  case 0:
    funkcija koja obavlja posao djeteta i
    exit(0);
  case -1:
    ispis poruke o nemogućnosti stvaranja procesa;
  default:
    nastavak posla roditelja;}
while (i--) wait (NULL);
```

ZAJEDNIČKI ADRESNI PROSTOR (1)

- Nakon stvaranja novog procesa sa ***fork***, procesi roditelj i dijete dijele segment s podacima koji se sastoji od stranica.
- Sve dok je stranica nepromjenjena oba procesa je mogu čitati.
- Ali čim jedan proces pokuša pisati na stranicu procesi dobiva odvojene kopije podataka.
- Tada niti globalne varijable nisu zajedničke za sve procese, pa ako jedan proces promjeni neku varijablu drugi to neće primijetiti.
- To je jedan od razloga za korištenje zajedničkog spremnika. Varijable koja trebaju biti zajedničke za sve procese moraju se nalaziti u zajedničkom spremniku kojeg prethodno treba zauzeti.

ZAJEDNIČKI ADRESNI PROSTOR (2)

- Zajednički spremnički prostor je najbrži način komunikacije među procesima.
- Isti spremnik je priključen adresnim prostorima dva ili više procesa.
- Čim je nešto upisano u zajednički spremnik, istog trenutka je dostupno svim procesima koji imaju priključen taj dio zajedničkog spremnika na svoj adresni prostor.
- Za sinkronizaciju čitanja i pisanja u zajednički spremnik mogu se upotrijebiti semafori, poruke ili posebni algoritmi.

ZAJEDNIČKI ADRESNI PROSTOR (3)

- Blok zajedničkog spremnika se kraće naziva segment.
- Može biti više zajedničkih segmenata koji su zajednički za različite kombinacije aktivnih procesa.
- Svaki proces može pristupiti k više segmenata.
- Segment je prvo stvoren izvan adresnog prostora bilo kojeg procesa, a svaki proces koji želi pristupiti segmentu izvršava sustavski poziv kojim ga veže na svoj adresni prostor.
- Broj segmenata je određen sklopovskim ograničenjima, a veličina segmenta može također biti ograničena.

Sustavski pozivi za stvaranje i rad sa zajedničkim spremnikom (1)

- **typedef key_t int;**
- **int shmget(key_t key, int size, int flags) ;**
- Ovaj sustavski poziv pretvara ključ (***key***) nekog segmenta zajedničkog spremnika u njegov identifikacijski broj ili stvara novi segment. Novi segment duljine barem ***size*** bajtova će biti stvoren ako se kao ključ upotrijebi **IPC_PRIVATE**. U devet najnižih bitova ***flags*** se stavljaju dozvole pristupa (na primjer, oktalni broj 0600 znači da korisnik može čitati i pisati, a grupa i ostali ne mogu). ***shmget*** vraća identifikacijski broj segmenta koji je potreban u ***shmat*** ili -1 u slučaju greške.

Sustavski pozivi za stvaranje i rad sa zajedničkim spremnikom (2)

- Proces veže segment na svoj adresni prostor sa ***shmat***:
- **char *shmat(int segid, char *addr, int flags) ;**
- Ako segment treba vezati na određenu adresu treba je staviti u ***addr***, a ako je ***addr*** jednako NULL jezgra će sama odabrati adresu (moguće ako se kasnije ne koristi dinamičko zauzimanje spremnika s ***malloc*** ili slično). ***flags*** također najčešće može biti 0. ***segid*** je identifikacijski broj segmenta dobijen pozivom ***shmget***. ***shmat*** vraća kazaljku na zajednički adresni prostor duljine tražene u ***shmget*** ili -1 ako dođe do greške. Dohvaćanje i spremanje podataka u segmente obavlja se na uobičajen način.

Sustavski pozivi za stvaranje i rad sa zajedničkim spremnikom (3)

- Segment se može otpustiti sustavskim pozivom *shmdt*:
- **int shmdt(char *addr) ;**
- Zajednički spremnički prostor ostaje nedirnut i može mu se opet pristupiti tako da se ponovno veže na adresni prostor procesa, iako je moguće da pri tome dobije drugu adresu u njegovom adresnom prostoru. ***addr*** je adresa segmenta dobivena pozivom ***shmat***.

Sustavski pozivi za stvaranje i rad sa zajedničkim spremnikom (4)

- Uništavanje segmenta zajedničke memorije izvodi se sustavskim pozivom ***shmctl***:
- **int shmctl(int segid, int cmd, struct shmid_ds *sbuf) ;**
- Za uništavanje segmenta treba za ***segid*** staviti identifikacijski broj dobiven sa ***shmget***, ***cmd*** treba biti **IPC_RMID**, a *sbuf* može biti NULL. Greška je uništiti segment koji nije otpušten iz adresnog prostora svih procesa koji su ga koristili. ***shmctl***, kao i ***shmdt*** vraća 0 ako je sve u redu, a -1 u slučaju greške. (Detaljnije o ovim pozivima u: man shmget, man shmop, man shmctl)

Struktura programa sa paralelnim procesima i zajedničkim spremnikom

definiranje kazaljki na zajedničke varijable

proces **k**

početak

proces koji koristi zajedničke varijable

...

kraj

...

glavni program

početak

zauzimanje zajedničke memorije

pokretanje paralelnih procesa

oslobađanje zauzete zajedničke memorije

kraj

- **VAŽNO**: Varijablama u zajedničkom spremniku se nužno pristupa korištenjem kazaljki.

Zadatak

- Napišite program koji demonstrira korištenje zajedničke memorije, te uključuje mogućnost prijevremenog izlaska iz programa (ctrl+C). S tim da prekidna rutina briše zauzete sustavske resurse (semafore i zajedničku memoriju) prije no što program završi.
- Rješenje (**p3.c**)

Zadaća 2. (1)

- Dekkerov postupak međusobnog isključivanja
- Ostvariti sustav paralelnih procesa/dretvi. Struktura procesa/dretvi dana je sljedećim pseudokodom:

```
proces proc(i)          /* i  [0..n-1] */
    za k = 1 do 5 čini
    uđi u kritični odsječak
    za m = 1 do 5 čini
        ispiši (i, k, m)
    izađi iz kritičnog odsječka
kraj.
```

Zadaća 2. (2)

- Međusobno isključivanje ostvariti za dva procesa/dretve međusobnim isključivanjem po Dekkerovom algoritmu.

DEKKEROV ALGORITAM

```
zajedničke varijable: PRAVO, ZASTAVICA[0..1]
funkcija uđi_u_kritični_odsječak(i,j) {
    ZASTAVICA[i] = 1   //p1 želi ući
    dok je ZASTAVICA[j]<>0 čini {     //sve dok p2 želi ući
        ako je PRAVO=j onda {         //ako p2 ima prednost
            ZASTAVICA[i] = 0          //p1 odustaje
            dok je PRAVO=j čini {     //čeka prednost
                ništa
                }
                ZASTAVICA[i] = 1      //pa javi da želi ući
                }
            }
        } kraj.
        funkcija izađi_iz_kritičnog_odsječka(i,j) {
            PRAVO = j   //prednost dajemo p2
            ZASTAVICA[i] = 0   //izlaz
        } kraj.
```

Upute

- Ako se program rješava s procesima tada treba zajedničke varijable tako organizirati da se prostor za njih zauzme odjednom i podijeli među njima. Ovo je nužno zbog ograničenog broja segmenata i velikog broja korisnika.
- Ovisno o opterećenju računala i broju procesa koji se pokreću, a da bi se vidjele razlike prilikom izvođenja programa može biti potrebno usporiti izvršavanje sa:
- sleep(1);
- nakon ispisi (**i**, **k**, **m**).

Literatura

- Korisni linkovi:
- W.Richard Stevens: Advanced Programming the Unix Environment (Chapter 8)(advanced-programming-the-unix-environment.pdf)
- Dekker's Algorithm (os_lec_09_concurrency_sw.pdf)
- Primjeri sa vježbi (10.2.Lab)

Kraj