2. Procesi i dretve u UNIX-u

2.1. Općenito o procesima i dretvama

Osnovni pojmovi: program, proces, dretva

- poglavlje u skripti: 4.1. Osnovni pojmovi – program, proces, dretva

Pri pokretanju *programa* operacijski sustav (OS) stvara *proces*:

- rezervira mjesto u memoriji za proces (adresni prostor za procesa)
- stvara opisnik procesa, opisnike datoteka koje proces koristi, ... (u memoriji OS-a)
- stvara početnu dretvu procesa (i njen opisnik)
- dretva može krenuti s radom

Adresni prostor procesa sastoji se od:

1. segmenta s instrukcijama (program)
2. segmenta s podacima (program)
3. gomile
4. stoga

Adresni prostor procesa dohvatljiv je svim dretvama procesa!

Jedinstveno za svaku dretvu:

- identifikacijski broj dretve
- kontekst (stanje registara procesora)
- kazaljka stoga i sam stog
- prioritet
- *razne zastavice (npr. signalna maska)*
- *privatni prostor svake dretve*

U adresnom prostoru OS-a:

- segment sustavskih podataka procesa:
  - opisnik procesa, opisnici dretvi
  - opisnici naprava, otvorenih datoteka, . . .
  - međuspremnici

Stvaranje novog procesa je zahtjevnija operacija od stvaranja nove dretve i traži više sredstava od sustava (npr. više memorije). Stoga se uglavnom preporuča korištenje stvaranja više dretvi unutar istog procesa. Iznimke su uglavnom radi sigurnosti, kad se želi sakriti podatke među dretvama (npr. u web pregledniku se ne želi dopustiti da jedna stranica ("tab") može doći do podataka s druge).

2.2. Pokretanje više dretvi u UNIX okruženju

Stvaranje dretvi prema POSIX sučelju

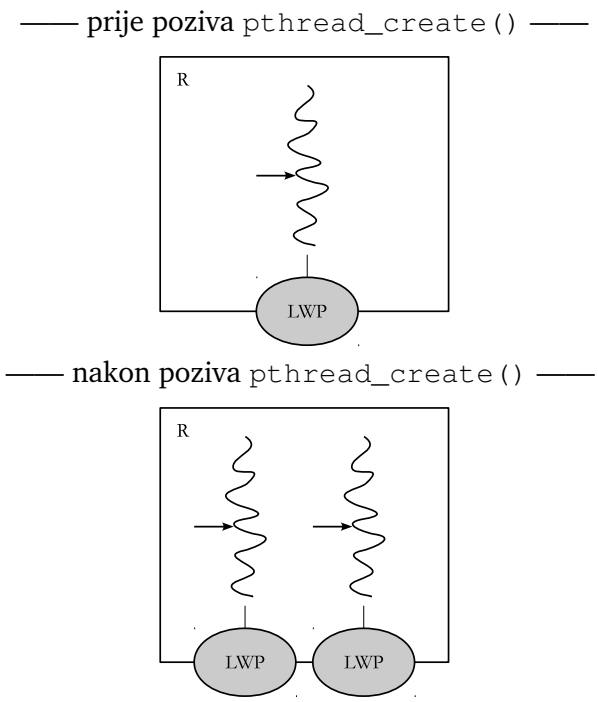
POSIX (engl. *The Portable Operating System Interface*) nastoji omogućiti prenosivost programa na razini izvorna koda - da se program napisan za jedan operacijski sustav (i okruženje) može jednostavno prevesti i pokretati na nekom drugom.

POSIX sučelje za stvaranje nove dretve je funkcija `pthread_create` s parametrima:

1. `pthread_t *opisnik` – mjesto za spremanje opisnika dretve
2. `pthread_attr_t *svojstva` – parametri za novu dretvu (može NULL)
3. `void *(*početna_funckija)(void *)` – početna funkcija za novu dretvu
4. `void *pparam` – parametar koji se šalje u početnu funkciju

Npr. `pthread_create (&opisnik, NULL, dretvena_funkcija, &podaci)`

Grafički prikaz:



Slika 2.1. Prije i poslije poziva `pthread_create`

Primjer programa koji stvara više dretvi:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <unistd.h>
5
6  void *dretva ( void *rbr )
7  {
8          int i, *d = rbr;
9          for ( i = 1; i <= 5; i++ ) {
10                 printf ( "Dretva %d, i=%d\n", *d, i );
11                 sleep (1);
12         }
13         return NULL;
14 }
15 int main ()
16 {
17         int i, j, BR[3];
18         pthread_t t[3];
19
20         for ( i = 0; i < 3; i++) {
21                 BR[i] = i;
22                 if ( pthread_create ( &t[i], NULL, dretva, &BR[i] ) ) {
23                         printf ( "Ne mogu stvoriti novu dretvu!\n" );
24                         exit (1);
25                 }
26         }
27         for ( j = 0; j < 3; j++)
28                 pthread_join ( t[j], NULL ); //čekaj kraj dretve t[j]
29
30         return 0;
31 }
32 // gcc -lpthread ... ili na Linuxu gcc -pthread ...
```

Prijenos parametara u dretvu

- kada treba svim dretvama dati istu vrijednost:
  - preko globalne varijable
  - parametar = kazaljka na istu strukturu
- kada treba svakoj dretvi dati njenu zasebnu vrijednost:
  - za svaku dretvu napraviti zasebnu strukturu podataka i kazaljka na nju poslati dretvi
  - sve drukčije je KRIVO !
    - npr. umjesto `&RBR[i]` staviti `&i` je KRIVO
      - glavna dretva i stvorene dretve rade paralelno, a glavna mijenja varijablu `i`

Završetak rada dretve

- izlaskom iz početne funkcije (navedene u `pthread_create`)
- pozivom `pthread_exit`
- završetkom rada početne dretve (dretve koja kreće s funkcijom `main`) završava proces – i sve ostale dretve tada budu prekinute! (vrijedi za C/C++, ali ne za Python)

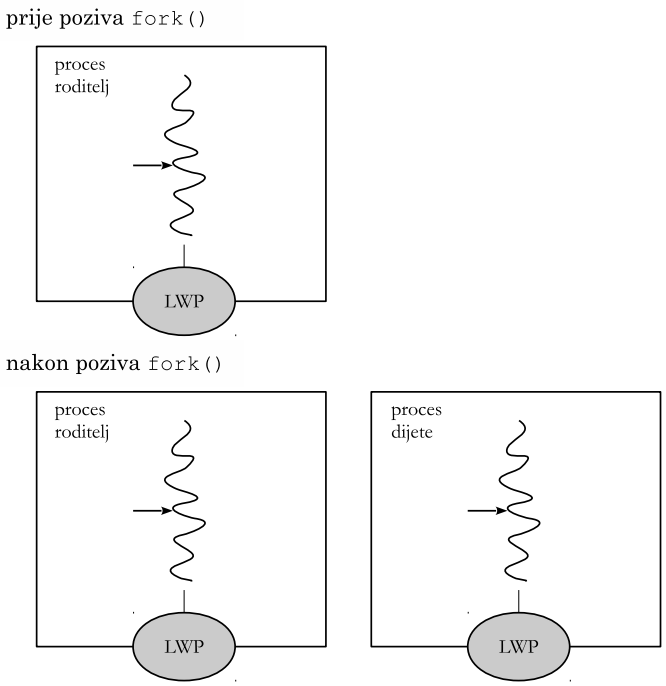
2.3. Pokretanje više procesa u UNIX okruženju s pozivom `fork()`

- proces *roditelj* stvara proces *dijete*
- dijete dobiva *kopiju* trenutnog adresnog prostora roditelja
  - podaci NISU zajednički (logički gledano, o implementaciji u 8. poglavlju)
  - u C-u: globalne varijable imaju istu vrijednost u oba procesa samo dok se ne napravi promjena u jednom od procesa – ta je promjena vidljiva samo u tom procesu, nije u onom drugome (tamo je i dalje ona stara vrijednost)
- prilikom međuprocesne komunikacije upliće se OS

Sučelje `fork()`

- prototip: `int fork (void);`
- u funkciju ulazi jedan proces, izlaze dva
- dijete kao povratnu vrijednost dobiva 0, a proces roditelj dobiva PID djeteta
- PID – jedinstveni identifikacijski broj procesa (svaki proces ima svoj PID)
- PID roditelja != PID djeteta
- u slučaju greške povratna vrijednost je -1 i novi proces nije stvoren
- greška: sustav nema dovoljno resursa za stvaranje nova procesa
- proces dijete je kopija procesa roditelja
- sredstva sustava su jedino što procesi dijele:
  - npr. otvorene datoteke, dijeljeni spremnik, semafori, redovi poruka

Grafički prikaz fork-a:



Slika 2.2. Prije i poslije poziva `fork`

Uobičajeni kraći način korištenja `fork`-a:

```
if ( fork() == 0 ) //oba procesa uspoređuju povratnu vrijednost s nulom!
{
    posao_procesa_djeteta;
    exit (0); // završava proces, izlazni status = 0 (uspješno završen)
}
nastavak rada procesa roditelja;
wait ( NULL ); //zaustavlja roditelja do proces dijete ne završi
```

Ili pravilnije, s ispitivanjem povratne vrijednosti:

```
pid_djeteta = fork ();
switch ( pid_djeteta )
{
case -1:
    printf("Ne mogu stvoriti novi proces!\n");
    exit(1);
case 0:
    Posao djeteta;
    exit(0);
default:
    Posao roditelja za novostvoreni proces;
}
nastavak rada roditelja;
wait ( NULL );
```

Primjer programa koji stvara više procesa:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define BR_PROCESA 3
#define BR_PROLAZA 5

void Proces ( int i )
{
   int j;

   for ( j = 1; j <= BR_PROLAZA; j++ ) {
      printf ( "D: Proces:%2d, prolaz:%2d (pid=%d, ppid=%d)\n",
               i, j, (int) getpid(), (int) getppid() );
      sleep (1);
   }
}
int main ( void )
{
   int i, pid;

   for ( i = 0; i < BR_PROCESA; i++ ) {
      switch ( pid = fork() ) {
      case -1:  printf ( "R: Ne mogu stvoriti novi proces!\n" );
                exit (1);
      case 0:   Proces (i);
                exit (0);
      default:  printf ( "R: Stvorio proces %d\n", pid );
                break;
      }
   }
   for ( i=0; i < BR_PROCESA; i++)
      wait(NULL);
   return 0;
}
```

Završetak rada procesa

- proces završava ("svojevoljno", bez grešaka ili signala):
  - izlaskom iz početne funkcije (`main`)
  - pozivom funkcije `exit(izlazni_status)` (npr. `exit(0)` ako je program napravio sve ili npr. `exit(1)` ako je bila kakva greška)
- proces roditelj može čekati na završetak procesa djeteta s funkcijom `wait(&status)`
  - ukoliko samo želimo čekati kraj nekog procesa djeteta (prvog koji završi) a status nas ne zanima može se koristiti poziv `wait(NULL)`
  - ako želimo status, onda argument treba biti kazaljka na cijeli broj preko kojeg ćemo dohvatiti status (npr. `int status; wait(&status);`)
  - ako želimo čekati na točno određni proces, treba koristiti funkciju `waitpid(pid, &status, 0)` (`man waitpid` za upute)
- svaki proces u operacijskom sustavu ima svog roditelja
  - iznimka su neki početni procesi
  - npr. pogledati stupce PID i PPID (id procesa i id procesa roditelja) u ispisu naredbe `ps -Al`
- proces roditelj može završiti i prije procesa djeteta
  - u tom slučaju dijete dobije novog roditelja (jednog od onih početnih)
  - kaže se da proces dijete tada postaje "zombie"
  - najčešće takvo ponašanje nije poželjno, osim kada se pokreću neki procesi "u pozadini" (npr. servisi), kada se želi te procese odijeliti od njihova roditelja (npr. ljuske) i datoteka koje je od njega naslijedio (stdin, stdout, stderr, ...)

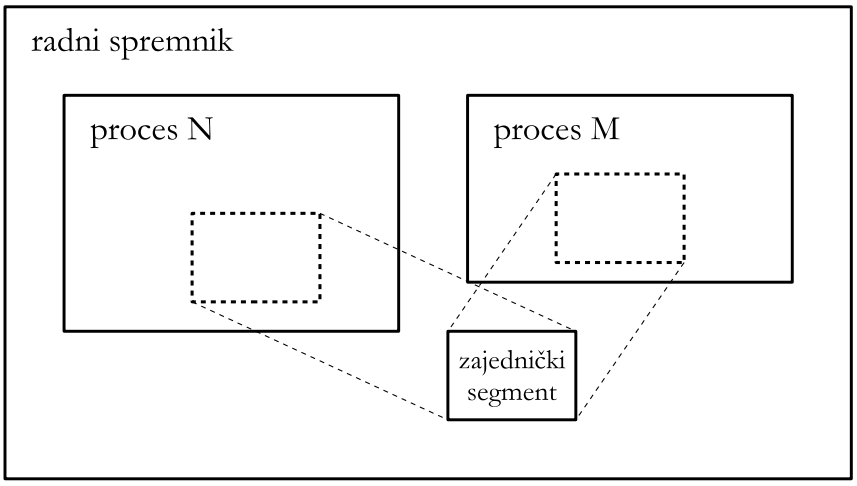
2.4. Korištenje zajedničkog spremnika

Što je to i kako koristiti "zajednički spremnik"?

- kod dretvi istog procesa
  - "zajednički spremnik" je zapravo cijeli adresni prostor procesa
  - najjednostavnije je koristiti globalne varijable
- kod dretvi različitih procesa
  - zajednički spremnik treba stvoriti preko sučelja OS-a:
    1. stvoriti segment zajedničkog spremnika i povezati ga s procesom
    2. globalne varijable (kazaljke) usmjeriti na taj segment
    3. stvoriti nove procese s fork
    4. globalne varijable = kazaljke ne mijenjati – mijenjati/koristiti sadržaj na koji one pokazuju, a koji je u zajedničkom spremniku!

U nastavku pod "zajednički spremnik" smatramo zajednički spremnik za komunikaciju dretvi različitih procesa.

Zajednički spremnik za procese



Slika 2.3. Zajednički spremnik (idejno ostvarenje)

Funkcije za rad sa ZS u *UNIX*-u (najjednostavnije):

- inicijalizacija (početni proces):
  1. `shmget` – dohvati segment zajedničkog spremnika (postojeći ili stvori novi)
  2. `shmat` – poveži segment na proces, adresu segmenta u globalnu varijablu
  3. pokreni druge procese (`fork`)
- korištenje segmenta (rad svih procesa)
- kraj rada (zadnji "živi" proces, npr. nakon `wait`-a):
  1. `shmdt` – odvoji segment od procesa
  2. `shmctl` – obriši segment (shmctl može i druge stvari raditi)