8.10 Familija *exec* funkcija

Postoji 6 razlicitih exec funkcija, kada kažemo exec mislimo na bilo koju od ovih funkcija.

```
int execl(const char *pathname, const char *arg0, ... /* (char *)0 */ );

int execv(const char *pathname, char *const argv []);

int execle(const char *pathname, const char *arg0, ...
           /* (char *)0,  char *const envp[] */ );

int execve(const char *pathname, char *const argv[], char *const envp []);

int execlp(const char *filename, const char *arg0, ... /* (char *)0 */ );

int execvp(const char *filename, char *const argv []);
```

Funkcija exec zamenjuje proces iz koga je pozvana, izvršavanjem programa čije je ime navedeno kao prvi argument te funkcije. Kad neki proces pozove exec, kernel odbacuje njegov kod segment, data segment, stek. Pronalazi se odgovarajući izvršni fajl koji je prvi argument exec funkcije, kernel čita njegove segmente, pravi nove za naš proces i vrši pripreme za nastavljanje novog procesa. Kod koji se exec-uje nasleđuje puno stvari od prethodnog procesa, id procesa, id roditeljskog procesa, radni direktorijum, itd.

TABELA
Prve 4 uzimaju argument `pathname` dok poslednje 2 uzimaju argument `filename`. U slučaju da `filename` sadrži karakter '/', uzima se kao `pathname`. Inače se izvršni fajl traži u direktorijumima koji se nalaze u PATH promenljivoj. Najčešće biramo funkciju iz ove dve grupacije na sledeći način:

1) ako izvršavamo neki program koji smo mi napisali koristimo neku od prve 4 funkcije
2) ako izvršavamo neku ugrađenu komandu koristimo neku od posledje 2 funkcije

echo $PATH

Druga razlika je u prenosu argumenata. Funkcije execl, execlp i execle zahtevaju da se svaki od argumenata prenese kao poseban argument. Kraj se označava null pokazivačem, tj. sa *(char *)0*. Za ostale 3 funkcije mora da se napravi niz pokazivača na argumente, i adresa ovog niza je argument ovim trima funkcijama.

Treća razlika je u prosleđivanju environment liste promenljivih programu. Funkcije koje se završavaju sa 'e', omogućavaju nam da prenesemo pokazivač na niz pokazivača na environment stringove. Ostale 4 funkcije koriste environ promenljivu.

Teško je zapamtiti šta koja funkcija od ovih radi. Slova u imenima funkcija mogu pomoći. Slovo ‘p’ označava da funkcija uzima argument `filename` i koristi PATH promenljivu da nađe izvršni fajl. Slovo ‘l’ označava da funkcija uzima listu argumenata, a slovo ‘v’ označava da je argument vektor argv[]. Slovo ‘e’ oznacava da funkcija uzima envp[] niz umesto da koristi trenutni environment.
TABELA

U većini UNIX implementacija, samo je jedna od ovih 6 funkcija, *execve*, sistemski poziv. Ostalih 5 su bibliotečke funkcije koje koriste ovaj sistemski poziv.
TABELA – veza 6 exec funkcija

(PRIMER exec)
Funkcija exec se jednom poziva i ne vraća se. Dakle, nema povratka u kod iz koga je pozvan (povratak je jedino moguć ako exec nije uspeo).

Kod drugog exec poziva prijavljena je greška, jer u direktorijumima koji su u PATH-u nije nađen program args. Ako prebacimo program args u tekući direktorijum, na većini računara bi trebalo da radi i drugi exec poziv (jer se direktorijum . često nalazi u PATH-u).

8.11 Promena korisničkih i grup IDova

Kada su našim programima potrebne dodatne privilegije da bi pristupili resursima kojima trenutno nemaju pristup, potrebno je da promene korisnički ili grup ID tako da pristup bude omogućen. Slično, nekada je potrebno oduzeti određene privilegije da bi se onemogućio pristup pojedinim resursima.

Dobra je praksa da se programima obezbedi minimum privilegija da bi ostvarili zacrtane zadatke. Na taj način smanjuje se broj sigurnosnih rupa koje zlonamerni korisnik može iskoristiti.

Dve funkcije – TABELA

Komanda *passwd* koristi *setuid*. Odmah se sa setuid postavlja effective uid na običnog korisnika, on unese staru šifru, novu šifru, pa ponovo novu šifru. Za ovo nam ne trebaju povećane privilegije. Kada treba da piše, postavljaju mu se prava root korisnika, pa se odradi taj deo posla. Na ovaj način se značajno povećava sigurnost, jer se mali deo koda obavlja sa povećanim privilegijama. U principu, operativni sistem se postavlja u viši nivo privilegija, izvršava niz instrukcija koje odgovaraju zahtevima procesa (proces nema direktnu kontrolu nad tim instrukcijama), zatim se sistemu vraćaju privlegije pozivajućeg procesa i on predaje kontrolu tom procesu.

8.13 Funkcija *system*

Ukoliko je argument ove funkcije null pokazivač, onda funkcija vraca ne-nula vrednost ukoliko je komandni processor dostupan. Ovo omogućava proveru da li je system funkcija podržana na datom sistemu. Na svim UNIX sistemima ova funkcija je podržana.

Ova funkcija je implementirana pozivima fork, exec, waitpid. Postoje 3 vrste povratne vrednosti:

- Ukoliko dođe do greške pri pozivu fork ili waitpid vrati grešku, system vraća -1 i postavlja vrednost u errno.
- Ukoliko dođe do greške pri exec, povratna vrednost je 127.
- Ukoliko se sve 3 funkcije uspešno izvrše, povratna vrednost iz system je status prekida shell-a.

(PRIMER system)
Za shell se kaže da je *interaktivan* jer prikazuje odgovarajući znak ($) i reaguje na unete komande

korisnika. Koristimo –c da bi shell uzeo sledeći argument komandne linije, umesto da čita iz fajla ili sa standardnog ulaza (probati komandu u terminalu *sh -c date*). Shell parsira string koji je terminiran sa null u odvojene argumente komandne linije za komandu koja se izvršava.

Ukoliko ne koristimo shell da izvršimo komandu, već pokušavamo sami da je izvršimo, to otežava situaciju. Prvo, morali bi da pozovemo execlp umesto execl, da bi koristili promenljivu PATH. Drugo, morali bi smo da podelimo C string terminiran sa null u odvojene argumente komandne linije za poziv execlp. Druga opcija je korišćenje funkcije *system.*

Pozivamo funkciju _exit umesto exit da bi smo sprečili da se U/I baferi, koji su preneti od strane roditelja detetu, isprazne u telu deteta.

(Primer) Izmeniti primer **fork** sa prethodnog dvočasa tako da se na kraju umesto *return 0* poziva *_exit (0)*. Kada se program pokrene sa *./fork > tmp* većina poruka se ne ispisuje na ekran jer _exit ne vrši pražnjenje bafera.

Još jedna prednost funkcije system u odnosu na direktan poziv fork i exec, je da obrađuje sve greške i signale.

8.16 Vremena procesa

UNIX održava dve vrednosti vremena:

1. Kalendarsko vreme. Broj sekundi od Epohe, 00:00:00 1.1.1970. Primitivni sistemski tip koji čuva ovu vrednost je **time_t**.
2. Procesno vreme. Meri resurse centralnog procesora koje koristi proces. Ovo vreme se meri u otkucajima sata, kojih ima 50, 60 ili 100 u sekundi. Primitivni sistemski tip **clock_t** čuva ove vrednosti.

UNIX čuva 3 vrednosti za proces:

- ukupno vreme ((wall) clock time)
- korisničko procesorsko vreme
- sistemsko procesorsko vreme

Ukupno vreme je vreme potrebno da se izvrši proces, i njegovo vreme zavisi od ostalih procesa koji se izvršavaju na sistemu. Da bi što pravilnije izmerili ovo vreme, merenje treba da radimo bez drugih aktivnih procesa na sistemu.

Korisničko procesorsko vreme je procesorsko vreme dodeljeno korisničkim instrukcijama (prolazak kroz niz u petlji, izračunavanje kvadrata broja, itd. su primeri ovog vremena). Sistemsko procesorsko vreme je procesorsko vreme dodeljeno kernelu dok izvršava naredbe tog procesa (ovo vreme se odnosi na izvršavanje sistemskih poziva od strane kernela, obično na instrukcije za koje nemamo dovoljno prava u korisničkom režimu, kao što su pisanje i čitanje po disku). To je vreme koje se utroši za izvršavanje read, write i drugih sistemskih poziva. U oba slučaja meri se vreme koje proces provede obavljajući instrukcije, u prvom slučaju u korisničkom režimu rada a u drugom slučaju u kernel režimu rada. Zbir prethodnih dvaju vremena često se naziva **procesorsko vreme**.

Pomoću *time* komande možemo dobiti ova 3 vremena u shellu.

time sleep 2

Svaki process moze pozvati **times** funkciju da dobije ova 3 vremena za sebe i bilo koju decu koja su završila izvršavanje. NAPOMENA: razlikovati funkciju *time* od funkcije *times*!

```
struct tms {
     clock_t  tms_utime;  /* user CPU time */
     clock_t  tms_stime;  /* system CPU time */
     clock_t  tms_cutime; /* user CPU time, terminated children */
     clock_t  tms_cstime; /* system CPU time, terminated children */
   };
```

Struktura ne sadrži polje za ukupno vreme. To vreme se vraća kao povratna vrednost funkcije. Ova vrednost se meri od nekog proizvoljnog trenutka u prošlosti, pa ne možemo koristiti njenu apsolutnu vrednost; zato koristimo relativnu vrednost. Npr, pozivamo times i čuvamo povratnu vrednost. U kasnijem trenutku pozivamo times ponovo i oduzimamo 2 vrednosti. Razlika je ukupno vreme.

(PRIMER times)

Dva polja koja se odnose na decu procese, sadrže podatke samo o deci na koju smo čekali. Sve *clock_t* vrednosti (broj tikova) vraćene od ove funkcije se pretvaraju u sekunde pomoću funkcije *sysconf*.

Zadatak 1. Napisati program koji implementira shell. Program prikazuje na ekranu znak % i čeka unošenje komandi. Svaka komanda se izvršava pomoću kombinacije fork/exec. Komande koje su podržane su one koje se sastoje od jedne reči (dozvoljena je komanda *ls* ali nije dozvoljena *ls -l*).

Zadatak 2. Napisati program zbir.c koji sabira sve (cele) brojeve koji mu se prenose kao argumenti komandne linije. Napisati drugi program koji u svakom redu učitava niz brojeva koje treba sabrati, i poziva prethodno napisani program za svaki red. Jedan red unosa se odnosi na jedan poziv programa, drugi red na sledeći poziv, itd. (Savet: pošto se ne zna koliko je brojeva u liniji, koristiti verziju exec funkcije sa slovom 'v' u imenu; pogodna funkcija za ovaj zadatak je *strtok*, ona izdvaja nisku po nisku iz pročitane linije).

Zadatak 3. Napisati program koji koji kao argument komandne linije prima ceo broj *n* i formira nizove celih brojeva takve da se u prvom nizu nalaze brojevi *1...n* a u drugom njihovi kvadrati *1...n^2.* Izračunati euklidsko rastojanje između ova dva vektora (koren kvadrata razlika elemenata vektora) i prikazati koliko je počev od prve funkcije pa do zadnje u main-u potrošeno: ukupnog vremena, korisničkog procesorskog vremena i sistemskog procesorskog vremena.