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.

Višedretvenost

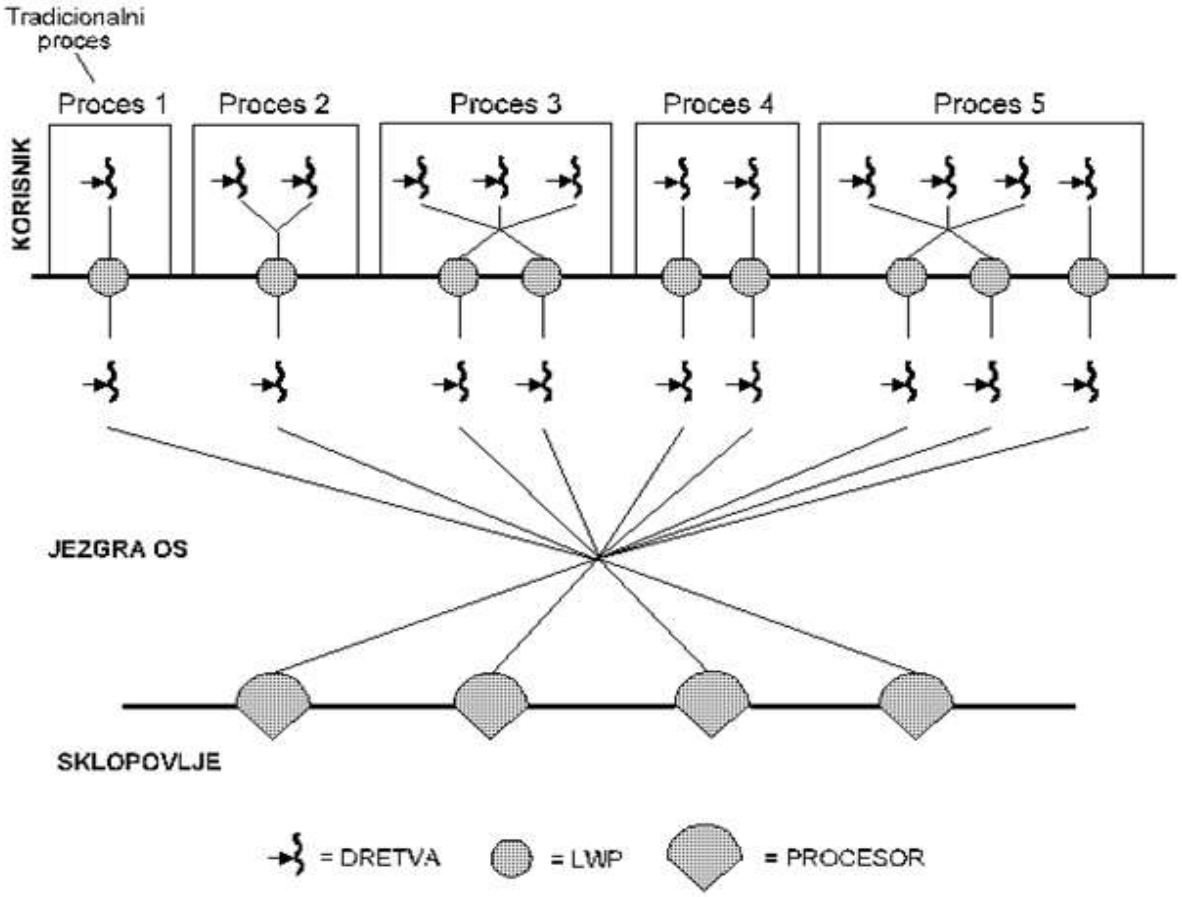
Sadržaj

- Funkcije za rukovanje dretvama
- Stvaranje dretvi
- Završetak rada dretve
- Win32
- Dekkerov postupak međusobnog isključivanja

Pojam višedretvenosti (1)

- Povijest višedretvenog programiranja počinje 60-tih, dok se njihova implementacija na UNIX sustavima pojavljuje sredinom 80-tih godina, a na ostalim sustavima nešto kasnije.
- Ideja višedretvenog programiranja jest u tome da se program sastoji od više jedinica koje se samostalno mogu izvoditi.
- Programer ne mora brinuti o redoslijedu njihova izvođenja, već to obavlja sam operacijski sustav.
- Štoviše, ukoliko je to višeprocesorski sustav, onda se neke jedinice-dretve mogu izvoditi istovremeno. Komunikacija među dretvama je jednostavna i brža u odnosu na komunikaciju među procesima, jer se obavlja preko zajedničkog adresnog prostora, te se može obaviti bez uplitanja operacijskog sustava.

POJAM VIŠEDRETVENOSTI (2)



Slika 1.: Arhitektura višedretvenog sustava Solaris 2.4

Pojam višedretvenosti (3)

- Vidljivost dretvi jest samo unutar procesa, čiji su oni dio i čija sredstva dijele (adresni prostor, otvorene datoteke, ...).
- Sljedeća stanja su jedinstvena svakoj dretvi:
  - identifikacijski broj dretve,
  - registarsko stanje uključujući programsko brojilo i kazaljku stoga,
  - stog,
  - signalna maska,
  - prioriteti i
  - privatni prostor same dretve.

Pojam višedretvenosti (4)

- Upravljanje dretvi, odnosno, osiguravanje da se sve dretve izvode, obavlja s pomoću dretvene biblioteke *libthread*.
- Upravljanje se obavlja na korisničkoj razini, a ne na razini operacijskog sustava.
- Veza između dretve i jezgre su tzv. *laki* (*lightweight*) procesi, za koje se u daljnjem tekstu koristi oznaka LWP.
- LWP jest osnovna dretva upravljanja na razini jezgre sustava. LWP se, sa strane programera, može predočiti jednostavno kao virtualni procesor.

Pojam višedretvenosti (5)

- Arhitektura višedretvenog sustava *Solaris 2.4* prikazana je na slici.
- Iz slike je vidljivo da standardni UNIX procesi imaju samo jedan LWP, odnosno, jednu dretvu.
- Također, svaka dretva ne mora imati vlastiti LWP, već ih više njih može koristiti iste LWP-ove, odnosno, razlikujemo dvije skupine dretvi: *vezane* (engl. *bound threads*) i *nevezane* (engl. *unbound threads*).
- Prve, *vezane*, su takve dretve kojima je pridjeljen vlastiti LWP koji izvodi samo njene instrukcije, dok druge, *nevezane*, nisu vezane za vlastiti LWP, već se mogu izvoditi na bilo kojem LWP-u koji se nalazi na raspolaganju procesu, a nije vezan za jednu dretvu.

Pojam višedretvenosti (6)

- Vezane dretve zauzimaju nešto više sredstava sustava, ali su zbog izbjegavanja učestalih promjena dretvi na LWP-u, brže.
- Kod drugih operacijskih sustava status dretvi je drugačiji, a na neki način bi se mogao shvatiti kao na gornjoj slici za procese 1 i 4, tj. svaku dretvu raspoređuje OS.

Funkcije za rukovanje dretvama

- Iako *Solaris* ima vlastite funkcije za rukovanje dretvama u nastavku su objašnjene funkcije po POSIX standardu koje izgledaju vrlo slično.
  - POSIX je zajednički naziv za skupinu IEEE standarda kojima se definira sučelje za programiranje (API) softvera usklađenog s različitim izvedenicama operacijskog sustava UNIX. Službena oznaka je IEEE 1003, a naziv međunarodnog standarda je ISO/IEC 9945. Standardi su nastali iz projekta koji je počeo oko 1985. Naziv POSIX predložio je Richard Stallman, a naknadno je izvedeno značenje (eng. backronym) Portable Operating System Interface ("prenosivo sučelje operacijskog sustava"), pri čemu X predstavlja UNIX.
- Pogledati man threads.

Stvaranje dretvi (1)

- Sve dretve, osim prve, inicijalne, koja nastaje stvaranjem procesa, nastaju pozivom **pthread_create**:

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

- ***thread*** je kazaljka na mjesto u memoriji gdje se sprema id novostvorene dretve i za razliku od *Solaris*-ove funkcije za stvaranje dretve ovaj parametar nesmije biti NULL.

Stvaranje dretvi (2)

- ***attr*** je adresa strukture koja sadrži podatke o atributima s kojima se želi stvoriti dretvu. Ako se za attr postavi *NULL* onda se uzimaju pretpostavljene vrijednosti (dovoljno dobre za lab. vježbe).
- ***start_routine*** predstavlja pokazivač na početnu funkciju koju će novostvorena dretva imati kao početnu (npr. kao što glavna dretva ima funkciju *main*).
- ***arg*** je adresa parametra koji se prenosi dretvi (može biti NULL ako se ništa ne prenosi). Pošto se može prenijeti samo jedan parametar u slučaju potrebe prijenosa više parametara oni se pohranjuju u strukturu te se šalje pokazivač na tu struktru.

Završetak rada dretve (1)

- Normalan završetak dretve jest njen izlazak iz prve, inicijalne funkcije, ili pozivom funkcije ***pthread_exit***:
- ***int pthread_exit(void *status);***
- ***status*** je kazaljka na stanje sa kojim dretva završava.

Završetak rada dretve (2)

- Dretva čeka na završetak druge dretve pozivom funkcije **pthread_join**:
- **int pthread_join(pthread_t cekana_dr, void **stanje);**
- **cekana_dr** je identifikacijski broj dretve na čiji se kraj čeka.
- **stanje** je kazaljka na kazaljku izlaznog statusa dočekane dretve.
- Funkcija **pthread_join** zaustavlja izvođenje pozivajuće dretve sve dok određena dretva ne završi sa radom. Nakon ispravnog završetka funkcija vraća nulu.

Završetak rada dretve (3)

- Normalni završetak višedretvenog programa zbiva se kada sve dretve završe s radom, odnosno, kada prva, početna dretva izađe iz prve funkcije (*main*). Prijevremeni završetak zbiva se pozivom funkcije *exit* od strane bilo koje dretve, ili pak nekim vanjskim signalom (SIGKILL, SIGSEGV, SIGINT, SIGTERM, ...).
- Primjer (primjer_dretve.c)

FUNKCIJA pthread_self

- Identifikacijski broj dretve moguće je dobiti pozivom funkcije **pthread_self**:
- **pthread_t pthread_self(void);**
- *Napomene*
  - Prilikom prevođenja potrebno je postaviti zastavicu -D_REENTRANT koja ukazuje na to da se koriste višedretvene inačice upotrijebljenih funkcija, ako takve postoje, te zastavicu -*lpthread* (npr. **gcc -D_REENTRANT -lpthread prvi.c -o prvi**).

POSIX dretve

- Stranice (manual) POSIX dretvi u kojima su detaljno opisane funkcije za rad s dretvama *pthread*: pthread, pthread_create, pthread_exit, pthread_detach, pthread_join, pthread_mutex_init, pthread_mutex_lock, pthread_mutex_unlock, pthread_mutex_destroy, pthread_cond_init, pthread_cond_wait, pthread_cond_signal, sem_init, sem_wait, sem_post, sem_destroy…

Win32

- Stvaranje procesa pod Win32 obavlja se funkcijom CreateProcess(). Primjer.
- Zajednička memorija ostvaruje se pomoću funkcija CreateFileMapping i MapViewOfFile. Primjer
- Stvaranje dretvi pod Win32 obavlja se funkcijom CreateThread(). Primjer.

Zadaća 3.

- Ostvariti program koji simulira tijek rezervacije stolova u nekom restoranu. Program na početku treba stvoriti određeni broj dretvi koji se zadaje u naredbenom retku. Svaka dretva/proces nakon isteka jedne sekunde provjerava ima li slobodnih stolova te slučajno odabire jedan od njih. Nakon odabira dretva ulazi u kritični odsječak te ponovo provjerava je li odabrani stol slobodan. Ako jest, označava stol zauzetim i izlazi iz kritičnog odsječka. U oba slučaja, nakon obavljene operacije ispisuje trenutno stanje svih stolova te podatke o obavljenoj rezervaciji. Prilikom ispisa za svaki stol mora biti vidljivo je li slobodan ili broj dretve/procesa koja je taj stol rezervirala. Broj stolova se također zadaje u naredbenom retku. Svaka dretva ponavlja isti postupak sve dok više nema slobodnih stolova. Program završava kada sve dretve završe.

Upute

- Primjer ispisa: (3 dretve, 5 stolova)

Dretva 1: odabirem stol 2
Dretva 2: odabirem stol 2
Dretva 3: odabirem stol 5
Dretva 2: rezerviram stol 2, stanje:
-2---
Dretva 1: neuspjela rezervacija stola 2, stanje:
-2---
Dretva 3: rezerviram stol 5, stanje:
-2--3

itd.

- Zaštitu kritičnog odsječka postupka rezervacije stola ostvariti koristeći Lamportov algoritam međusobnog isključivanja.

Lamportov algoritam

- Zajedničke varijable: **ULAZ[0..n-1]**, **BROJ[0..n-1]** sve početno postavljeno u nulu

```
uđi_u_kritični_odsječak(i)
   ULAZ[i] = 1
   BROJ[i] = max ( BROJ[0], ..., BROJ[n-1] ) + 1
   ULAZ[i] = 0

   za j = 0 do n-1 čini
      dok je ULAZ[j] <> 0 čini
         ništa
      dok je BROJ[j] <> 0 && (BROJ[j] < BROJ[i] || (BROJ[j] == BROJ[i] && j < i))
čini
         ništa

izađi_iz_kritičnog_odsječka(i)
   BROJ[i] = 0
```

Literatura

- Korisni linkovi:
- http://linux.die.net/
- http://www.inf.pucrs.br/~pinho/shared_memory_library.htm

Kraj