Multiprocesorski sistemi

Pthreads

Marko Mišić, Andrija Bošnjaković

MS1MPS, RI5MS, IR4MPS, SI4MPS

2011/2012.

Zasnovano na:

Blaise Barney, Lawrence Livermore National Laboratory

POSIX Threads Programming

https://computing.llnl.gov/tutorials/pthreads/

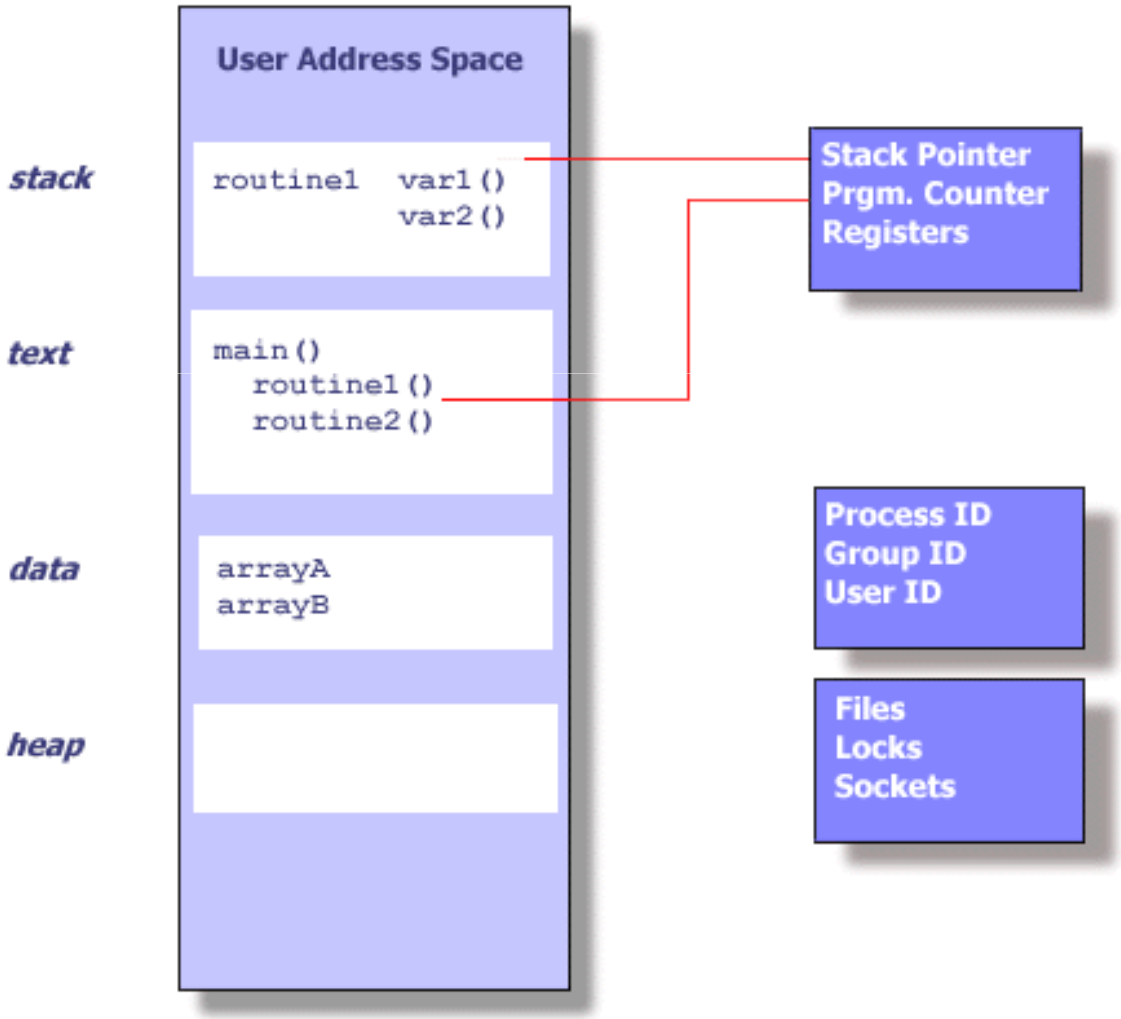
Niti (engl. threads)

- Tehnički, nit je nezavisni tok instrukcija kojeg OS može raspoređivati za izvršavanje
  - Opisno: "potprogram koji može biti izvršavan nezavisno od svog glavnog programa u istom adresnom prostoru"
- Ako glavni program sadrži više ovakvih potprograma koje OS rasporedi da budu izvršavani u paraleli i/ili nezavisno, onda može biti opisan kao program sa više niti (engl. *multi-threaded program*)
- Da bi se razumeo koncept niti, potrebno dobro razumeti koncept procesa
  - Procesi na UNIX-olikim operativnim sistemima su dobar primer

UNIX implementacija procesa (1)

- OS stvara proces
- Ovaj postupak zahteva primetne "režijske troškove" (engl. *overhead*)
- Proces sadrži sledeće informacije:
  - Process ID, process group ID, user ID, and group ID
  - Okruženje (promenljive, radni direktorijum)
  - Programski kod koji se izvršava unutar tog procesa
  - Kontekst procesa
    (registri, stek, dinamička memorija, deskriptori fajlova)
  - Još dosta stvari…
    (rukovaoci signala, .so, međuprocesna komunikacija)

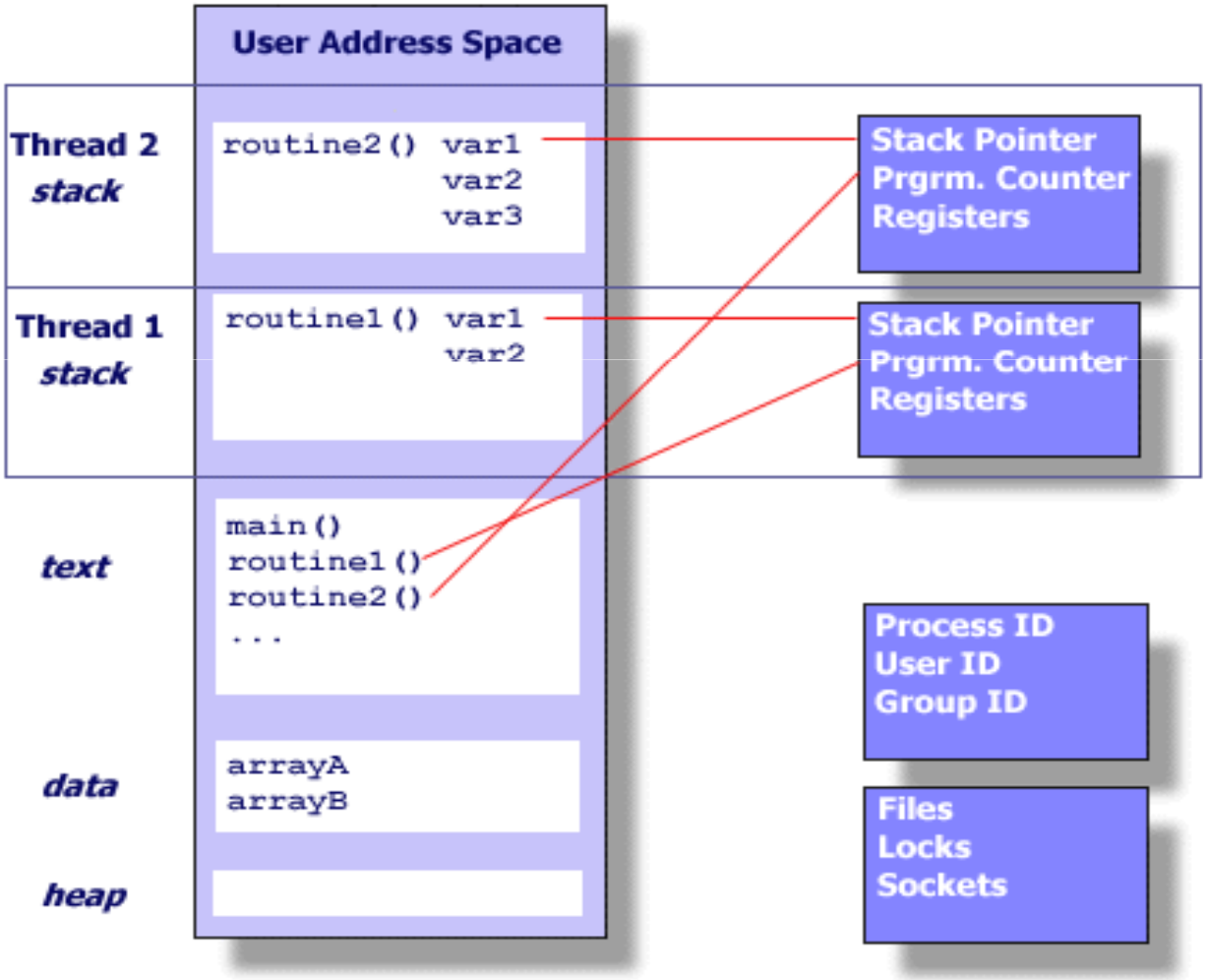
UNIX implementacija procesa (2)



UNIX implementacija niti (1)

- Niti postoje unutar procesa i koriste navedene resurse
  - Dupliciraju se samo resursi neophodni za nezavisno izvršavanje!
- Vreme života niti zavisi od vremena života roditeljskog procesa
- Moguće je izvršavati ih nezavisno zato što imaju svoju kopiju neophodnih podataka:
  - SP, IP, PSW
  - Programski dostupni registri
  - Parametri raspoređivanja (vrsta raspoređivanja i prioritet)
  - Signali
  - Lokalni podaci niti
    - Kod Pthreads, svi lokalni podaci iz funkcije koju nit izvršava su takođe privatni podaci te niti

UNIX implementacija niti (2)



Prednosti (UNIX) niti

- Postoje unutar procesa i koriste njegove resurse
- Imaju svoj nezavisni redosled izvršavanja
  - Dok god postoji roditeljski proces i dozvoljava operativni sistem
- Repliciraju samo esencijalne resurse
  - One koji su potrebni za nezavisno raspoređivanje
- Manji režijski troškovi stvaranja i uništavanja
  - Niti su "lake" (engl. *lightweight)*
  - Većina režijskih troškova se ostvari prilikom stvaranja roditeljskog procesa
- Ostatak resursa deli zajednički sa ostalim nitima istog procesa

Nedostaci (UNIX) niti

- Zbog deljenja resursa,
  ono što jedna nit uradi nad deljenim resursima
  će biti vidljivo u svim ostalim nitima tog procesa
  - Na primer, ukoliko jedna nit zatvori datoteku,
    to će videti sve niti unutar procesa
  - Ako jedna izvrši korupciju memorije,
    to će uticati na sve niti
- Mogućnost istovremenog pristupa podacima
  zahteva eksplicitnu sinhronizaciju u programskom kodu
  - Sva čitanja iz istih podatka i
    sva pisanja po istim podacima

Pthreads

- Postojanje različitih verzija niti na raznim sistemima komplikuje prebacivanje programskog koda sa jedne platforme na drugu
- Standardizacija: IEEE POSIX 1003.1c standard (1995)
- Implementacije niti koje teže ovom standardu su POSIX niti ili Pthreads
- Najčešća situacija danas je da na nekom sistemu postoje i sopstvena realizacija niti i podrška za Pthreads
- Pthreads su skup C tipova podatka i funkcija, opisanih u **`pthread.h`** i realizovanih u pratećoj biblioteci
- Postoji više verzija standarda
  - Verzija standarda i stepen poštovanja standarda različiti su od implementacije do implementacije

Zašto koristiti (POSIX) niti? (1)

- Prvenstveno zbog unapređenja performansi
- Rad sa nitima zahteva manje režijske troškove nego rad sa procesima (stvaranje i uništavanje)
- Okvirno poređenje: 50000 poziva funkcija za stvaranje, mereno uz pomoć UNIX komande **`time`**

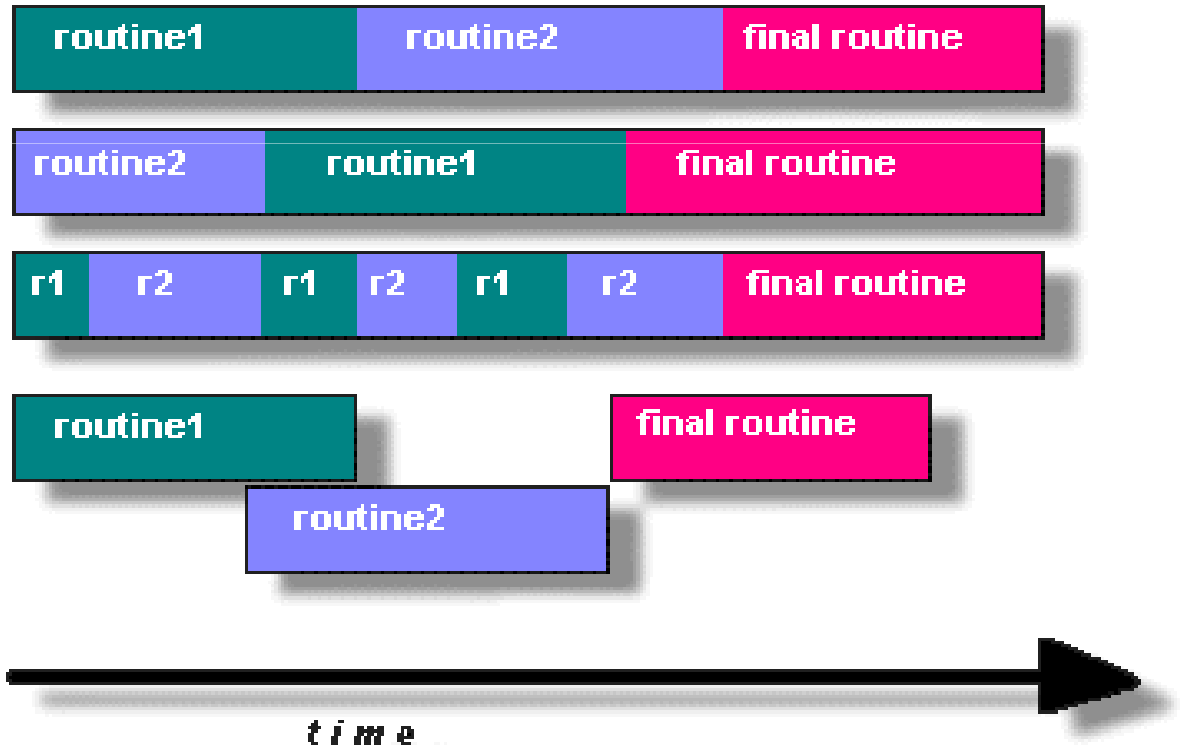
Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
AMD 2.4 GHz Opteron (8cpus/node)	17.6	2.2	15.7	1.4	0.3	1.3
IBM 1.9 GHz POWER5 p5-575 (8cpus/node)	64.2	30.7	27.6	1.7	0.6	1.1
IBM 1.5 GHz POWER4 (8cpus/node)	104.5	48.6	47.2	2.1	1.0	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2.0	1.2	0.6

Zašto koristiti (POSIX) niti? (2)

- Zbog deljenja resursa i istog adresnog prostora, komunikacija između niti je mnogo efikasnija od komunikacije između procesa
- Mogućnost preklapanja obrade i I/O operacija – dok jedna nit čeka na sporu periferiju, ostale niti intenzivno koriste procesor(e)
- Bitniji delovi programa mogu dobiti viši prioritet
- Rukovanje asinhronim događajima (npr. serveri)
- Dobro rešenje za jednoprocesorske i SMP mašine

Kada ima smisla koristiti (POSIX) niti

- Kad god je moguće paralelizovati postojeći program

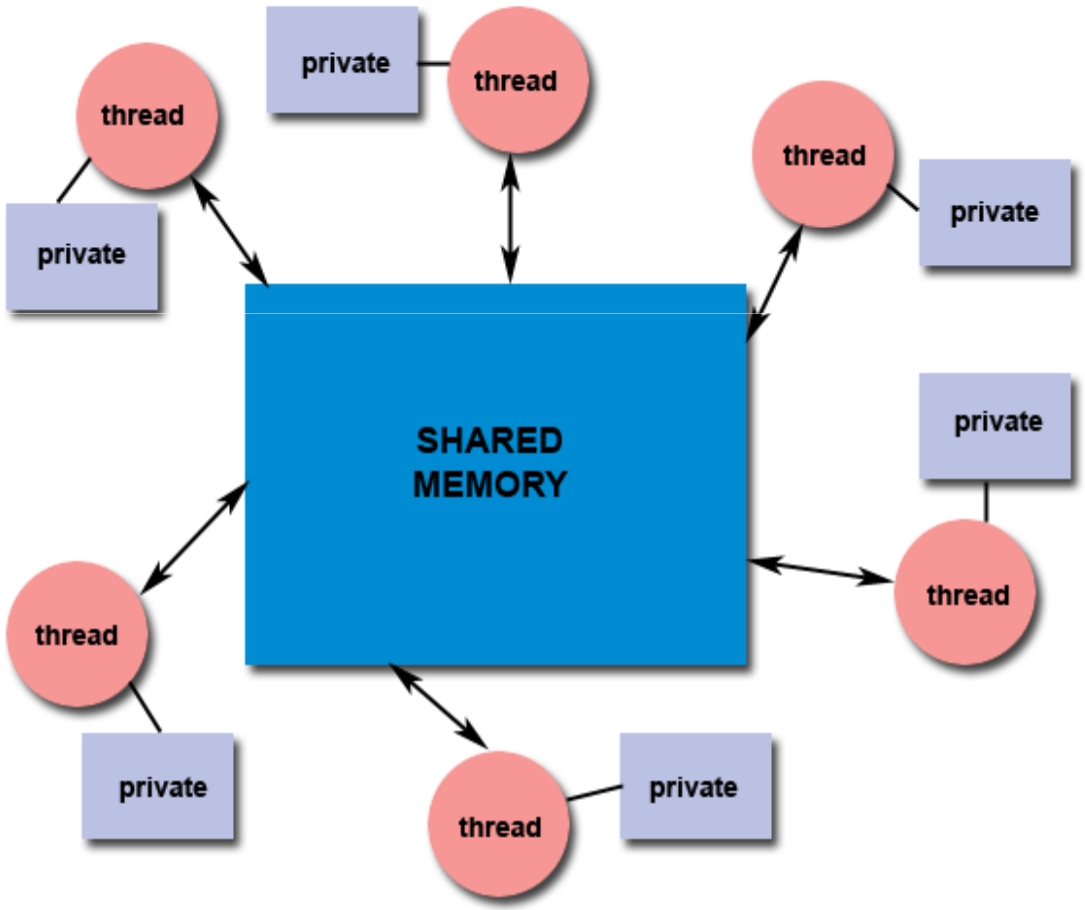


Najčešći međusobni odnosi niti unutar višenitnog programa

- **Pipeline**
  - Posao je podeljen na manje celine, koje mogu biti odvojeno završene (logički su nezavisne)
- **Manager/worker**
  - Jedna nit dodeljuje poslove ostalima (glavna nit reguliše I/O operacije i raspoređuje posao po ostalim nitima)
    - Static worker pool
    - Dynamic worker pool
- **Peer**
  - Slično kao manager/worker
  - Glavna nit se pridružuje poslu kad reši I/O

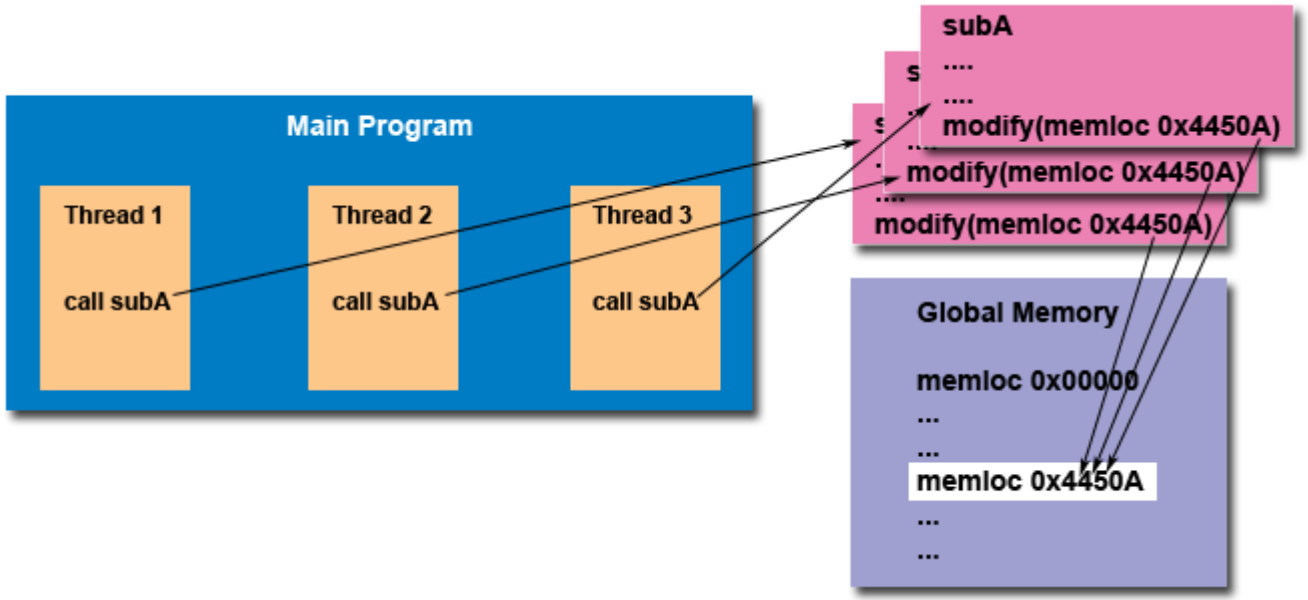
Model deljene memorije

- Sve niti imaju pristup do iste memorije (globalne, deljene)
- Pored toga, svaka nit ima svoje privatne podatke (privatnu memoriju)



Bezbednost niti (engl. thread-safeness)

- Programer je odgovoran za:
  - Zaštitu integriteta deljenih podataka
  - Nesmetan pristup deljenim podacima
  - Pravilnu upotrebu postojećeg programskog koda za koji nije poznato da li je *thread-safe*



Pthreads API (1)

- Tipovi podataka i funkcije mogu biti podeljeni u nekoliko grupa, od kojih su najvažnije sledeće četiri:
  - Upravljanje nitima
    - Stvaranje, uništavanje, rastavljanje, spajanje
  - Sinhronizacija putem međusobnog isključivanja (engl. *mutex*)
    - Stvaranje, uništavanje, zaključavanje, otključavanje
  - Uslovne promenljive
    - Stvaranje, uništavanje, čekanje, signalizacija
  - Sinhronizacioni mehanizmi višeg nivoa
    - Barijere, brave za čitanje i upis, itd.

Pthreads API (2)

- Sve navedene funkcionalnosti imaju odgovarajuće tipove podataka i funkcije za inicijalizaciju i manipulaciju objektima
- Sve funkcije rade nad objektima (nitima, bravama...)
  - Objekti su netransparentni
    - Ne može se direktno pristupati njihovim poljima
  - Stvaranje i promena atributa objekata se vrši isključivo pozivanjem API funkcija
- Svi kreirani i inicijalizovani objekti se moraju uništiti kada više nisu potrebni

Konvencija imenovanja i upotreba

- Svi identifikatori počinju sa `pthread_`

Routine Prefix	Functional Group
`pthread_`	Threads themselves and miscellaneous subroutines
`pthread_attr_`	Thread attributes objects
`pthread_mutex_`	Mutexes
`pthread_mutexattr_`	Mutex attributes objects.
`pthread_cond_`	Condition variables
`pthread_condattr_`	Condition attributes objects
`pthread_key_`	Thread-specific data keys

- Upotreba: `#include<pthread.h>`
- Prevođenje: `gcc/g++ -pthread` …

Stvaranje i uništavanje

- pthread_create(
  thread,attr,start_routine,arg
)
- pthread_exit (status)
- pthread_attr_init (attr)
- pthread_attr_destroy (attr)

Stvaranje niti (1)

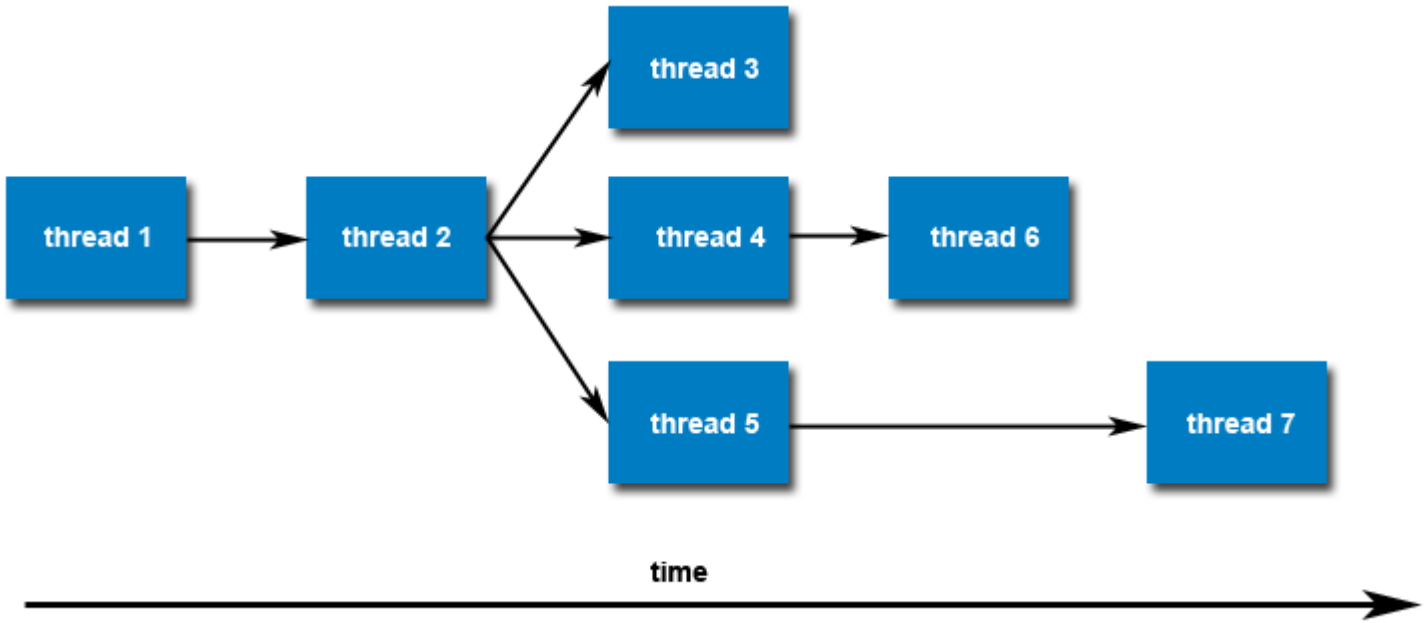
- `pthread_create(`
  `thread,attr,start_routine,arg`
  `)`
- Prvi parametar predstavlja netransparentni objekat koji predstavlja "ručku" (engl. *handle*) niti unutar programa
- Atributi su objekti kojima se može uticati na:
  - Politiku raspoređivanja i prioritiranja
  - Da li je nit *detached* ili *joinable*
  - Eksplicitno upravljanje stekom
- Atributi se stvaraju i uništavaju sa:
  - `pthread_attr_init (attr)`
  - `pthread_attr_destroy (attr)`

Stvaranje niti (2)

- Funkcija nad kojom se kreira nit mora imati sledeće zaglavlje:
  - `void* funkcija (void*);`
- Četvrti argument `pthread_create` funkcije predstavlja pokazivač na strukturu pomoću koje se prenose podaci u telo niti
- Ključna pitanja:
  - Da li nit dobija procesor odmah po stvaranju?
  - Od čega zavisi kada nit dobija procesor?
  - Do kada će nekoj niti biti dozvoljeno izvršavanje?

Stvaranje niti (3)

- Jedna nit može kreirati proizvoljan broj drugih niti
  - Ukupan broj niti ograničen samo od strane OS-a
  - Ne postoji hijerahijski odnos među nitima
    - Sve niti su međusobno jednake (engl. *peers*)



Završetak rada niti (1)

- Svaka nit može da završi svoj rad na nekoliko načina:
  - Završetkom funkcije nad kojom je pokrenuta nit
  - Pozivom `pthread_exit` funkcije
    - Bez obzira da li posao niti završen ili ne
  - Nit je opozvana od strane druge niti pozivom `pthread_cancel` funkcije
  - Ceo proces je prekinut,
    zbog poziva funkcijama `exec` ili `exit`
  - Zbog završetka funkcije `main`
  - Ako se završi funkcija `main`,
    a nije eksplicitno pozvala funkciju `pthread_exit`

Završetak rada niti (2)

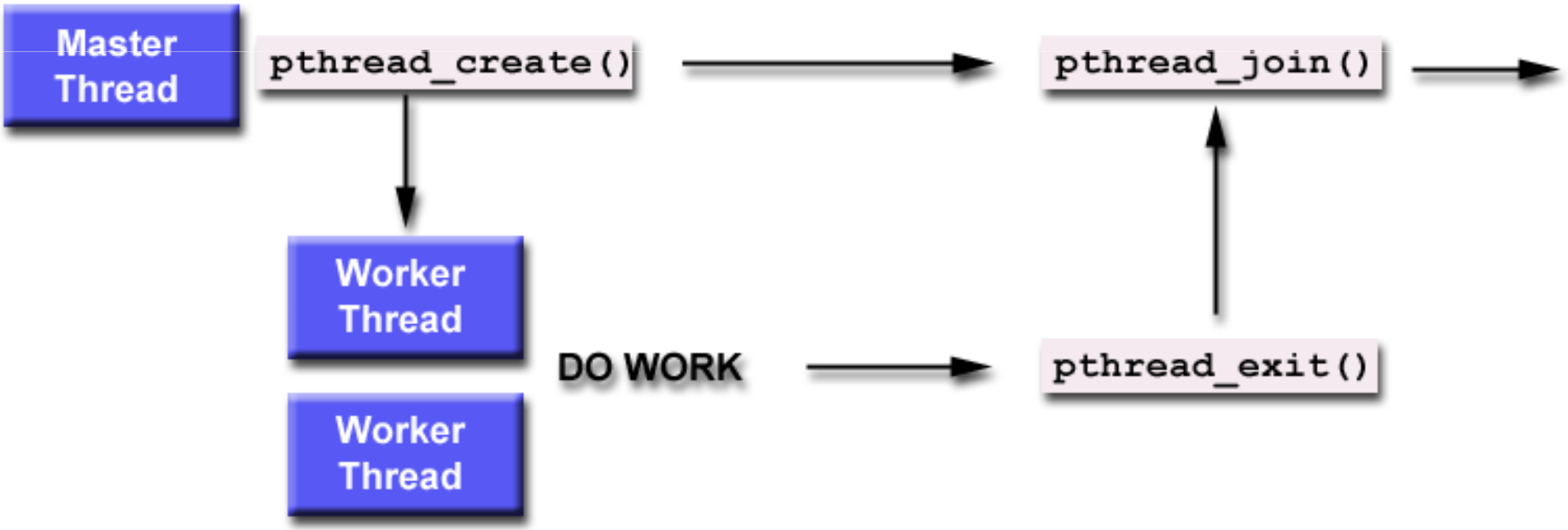
- `pthread_exit` funkcija može da vrati opcionalni status prilikom završetka niti
  - Taj status se može dohvatiti `pthread_join` funkcijom
- Ukoliko se `pthread_exit` pozove iz `main` funkcije, ona će se blokirati dok god i poslednja stvorena nit ne završi svoje izvršavanje
  - U suprotnom završetak `main` funkcije nasilno prekida izvršavanje svih stvorenih niti
- Funkcija `pthread_exit` ne zatvara datoteke otvorene unutar tela niti

Prosleđivanje podataka nitima

- Postoji više načina
  - preko globalnih promenljivih
    (npr. promenljive deklarisane na nivou fajla)
  - preko pokazivača na strukturu sa podacima
- Potrebno je uvek voditi računa
  da nit kontrolisano upisuje podatke
  - svoje bez sinhronizacije
  - deljene isključivo sinhronizovano

Rastavljanje i spajanje niti

- `pthread_join (threadid,status)`
- `pthread_detach (threadid)`
- `pthread_attr_setdetachstate (attr,detachstate)`
- `pthread_attr_getdetachstate (attr,detachstate)`



Ostale funkcije

- Upravljanje stekom
  - `pthread_attr_getstacksize (attr, stacksize)`
  - `pthread_attr_setstacksize (attr, stacksize)`
  - `pthread_attr_getstackaddr (attr, stackaddr)`
  - `pthread_attr_setstackaddr (attr, stackaddr)`
- Razno
  - `pthread_self ()`
  - `pthread_equal (thread1,thread2)`
  - `pthread_once (once_control, init_routine)`
  - `pthread_yield ()`

Brava (engl. lock, mutex lock, mutex)

- Služi za zaštitu pristupa do deljenog resursa, kad je potrebno da u jednom trenutku samo jedna nit može koristiti deljeni resurs
- Česta upotreba je ažuriranje vrednosti deljenih podataka
- Postoje dva načina zaključavanja brave
  - Blokirajući
  - Neblokirajući
- Ako više od jedne niti čeka na otključavanje brave, po otključavanju *scheduler* (slučajno) odlučuje koja će dobiti procesor
  - Uzimajući prioritete niti u obzir

Funkcije za rad sa bravama

- Stvaranje i uništavanje
  - `pthread_mutex_init (mutex,attr)`
  - `pthread_mutex_destroy (mutex)`
  - `pthread_mutexattr_init (attr)`
  - `pthread_mutexattr_destroy (attr)`
  - Može se izvršiti i statička inicijalizacija sa podrazumevanim parametrima sa `PTHREAD_MUTEX_INITIALIZER` makroom
- Zaključavanje i otključavanje
  - `pthread_mutex_lock (mutex)`
  - `pthread_mutex_trylock (mutex)`
  - `pthread_mutex_unlock (mutex)`

Vrste brava

- Podržane su tri vrste brava
  koje se mogu podesiti odgovarajućim atributima
- Normalna brava (*normal mutex*)
  - Ako nit koja je već zaključala bravu
    pokuša to da uradi još jednom, desiće se *deadlock*
- Rekurzivna brava (*recursive mutex*)
  - Dozvoljava jednoj niti da istu bravu zaključa više puta
  - Upotreba u rekurzivnim funkcijama
- Brava sa proverom na grešku (*error-checking mutex*)
  - Brava može da se zaključa samo jednom
  - Ukoliko jedna nit to pokuša još jednom,
    funkcija za zaključavanje će vratiti grešku
  - Izbegava se *deadlock*

Uobičajeni scenario upotrebe brave

- Stvaranje i inicijalizacija brave
- Nekoliko niti pokušava da zaključa bravu
- Samo jedna nit uspeva
- Nit koja je zaključala bravu
  obavlja željeni deo posla
- Po obavljenom poslu, ta nit otključava bravu
- Sledeća nit zaključava bravu
  i ponavlja opisani postupak
- Kad više ne bude potrebna, bravu treba uništiti

Brava sa čekanjem (engl. spinlock)

- Namena slična kao kod obične brave
- Upotreba kad je procenjeno izgubljeno vreme kod promene konteksta zbog uspavljivanja niti pri neuspešnom zaključavanju
  veća od procenjenog izgubljenog vremena na čekanje da se brava otključa
- Nema mnogo smisla za jednoprocesorske sisteme
  - SMP, manycore, GPU
- Funkcije imaju prefiks `pthread_spin_`

Uslovne promenljive

- Služe za praćenje događaja u povezanim nitima, a ne za međusobno isključivanje niti
  - Služe za sinhronizaciju baziranu na vrednostima deljenih podataka
- Bez uslovnih promenljivih, nit bi morala da (stalno) ispituje da li je određeni uslov zadovoljen
- Ovakav pristup nepotrebno troši vreme i resurse
- Uslovne promenljive **moraju biti** korišćene u sprezi sa bravama

Funkcije za rad sa uslovnim promenljivama

- Stvaranje i uništavanje
  - `pthread_cond_init (condition,attr)`
  - `pthread_cond_destroy (condition)`
  - `pthread_condattr_init (attr)`
  - `pthread_condattr_destroy (attr)`
- Čekanje i obaveštavanje
  - `pthread_cond_wait (condition,mutex)`
  - `pthread_cond_signal (condition)`
  - `pthread_cond_broadcast (condition)`
- **Napomena:** `cond_wait` i `cond_signal` nemaju veze sa funkcijama `wait` (P) i `signal` (V) kod semafora

Operacije na uslovnim promenljivama

- `pthread_cond_wait` bezuslovno blokira pozivajuću nit na uslovnoj promenljivoj
  - Pozivalac prethodno mora da zaključa pridružena brava
  - Prilikom blokiranja, brava se otključava
  - Kada nit primi signal, brava se automatski ponovo zaključava
- Funkcija `pthread_cond_signal` budi jednu od niti blokiranu na uslovnoj promenljivoj
- Funkcija `pthread_cond_broadcast` budi sve niti blokirane na uslovnoj promenljivoj
  - Obe operacije moraju biti pozvane dok je pridružena brava zaključana
  - Nakon poziva bravu treba otključati
  - Poziv bilo koje od ovih operacija na praznoj uslovnoj promenljivoj izaziva logičku grešku

Uobičajeni scenario upotrebe uslovne promenljive (1)

- Glavni program (glavna nit)
  - Definiše podatke koji zahtevaju sinhronizaciju (npr. postavlja brojač na 0)
  - Deklariše i inicijalizuje uslovnu promenljivu na jedan od dva moguća načina
    - Statički, u deklarativnom delu programa
      `pthread_cond_t uslov = PTHREAD_COND_INITIALIZER;`
    - Dinamički, u izvršnom delu programa
      `pthread_cond_init(&uslov, &atributi_uslova);`
  - Deklariše i inicijalizuje logički pridruženu bravu
  - Stvara niti koje će obavljati posao (npr. nit A i nit B)

Uobičajeni scenario upotrebe uslovne promenljive (2)

- Nit A
  - Obavlja posao dok ne dođe do tačke gde neki uslov mora biti ispunjen (npr. brojač mora dostići određenu vrednost)
  - Zaključa pridruženu bravu i proveri ispunjenost uslova (npr. vrednost brojača)
  - Ako uslov nije ispunjen, poziva `pthread_cond_wait()` i čeka na obaveštenje od niti B
    Važno: ovaj poziv bezuslovno blokira nit A i otključava pridruženu bravu, kako bi nit B mogla da promeni uslov i time se izbegao deadlock
  - Kad dobije obaveštenje od niti B, nit A postaje spremna za izvršavanje i nastavlja sa radom vezanim za uslov nakon što dobije procesor
    Važno: nit je po nastavku rada u stanju identičnom kao pre poziva `pthread_cond_wait()` i **ponovo** ima zaključanu pridruženu bravu, a to znači da **ne treba** zaključavati pridruženu bravu **ponovo**
  - Po završenoj upotrebi zajedničkih promenljivih od kojih zavisi uslov, otključava bravu i nastavlja sa radom nevezanim za deljenu promenljivu

Uobičajeni scenario upotrebe uslovne promenljive (3)

- Nit B
  - Obavlja svoj posao nevezan za uslov
  - Zaključava pridruženu bravu
  - Menja podatke od kojih zavisi ispunjenost uslova na koji čeka nit A (npr. promena brojača)
  - Ako je ovom promenom ispunjen uslov koji čeka nit A, obavesti nit A da je došlo do ispunjenja uslova
  - Otključava pridruženu bravu i nastavlja sa nevezanim poslom
  - Ni u kom slučaju se ne blokira unutar kritične sekcije
- Glavni program (glavna nit)
  - Čeka na završavanje niti A i B
  - Uništava bravu sa `pthread_mutex_destroy()`
  - Uništava deljenu promenljivu sa `pthread_cond_destroy()`

Uzroci čestih grešaka

- Šta tačno radi `pthread_cond_wait`?
  - Čeka na obaveštenje od `pthread_cond_signal`, i uspavljuje (blokira) pozivajuću nit sve do tada
  - Po buđenju, nit nastavlja sa radom kao da nije ni spavala
  - **Važno:** Nema potrebe dodatno otključavati/zaključavati bravu
- Šta tačno radi `pthread_cond_signal`?
  - Bezuslovno šalje obaveštenje **prvoj** niti blokiranoj na datoj uslovnoj promenljivoj
- Šta tačno radi `pthread_cond_broadcast`?
  - Bezuslovno šalje obaveštenje **svim** nitima blokiranim na datoj uslovnoj promenljivoj
- Poslednje dve funkcije nikada ne blokiraju pozivajuću nit i nemaju efekta ako nema makar jedne blokirane niti u trenutku poziva, jer se njihov efekat nigde ne pamti

Sinhronizacija na barijeri

- Kod određenih paralelnih izračunavanja, potrebno je da se niti sretnu u određenoj tački pre nego što nastave svoje izvršavanje
  - Problem se može rešiti korišćenjem brava
- Uobičajeno rešenje je korišćenje barijere
  - Tip `pthread_barrier_t`
  - Prilikom pozivanja funkcije za čekanje, pozivajuća nit se blokira
- Barijere nisu implementirane na svim sistemima

Funkcije za rad sa barijerom

- Stvaranje i uništavanje
  - `pthread_barrier_init (barrier,attr,count)`
  - `pthread_barrier_destroy (barrier)`
  - `pthread_barrierattr_init (attr)`
  - `pthread_barrierattr_destroy (attr)`
- Sinhronizacija na barijeri
  - `pthread_barrier_wait (barrier)`

RW brava (engl. read-write lock)

- Brave štite kritične sekcije u kodu
- U pojedinim primenama
  niti većinom čitaju podatke, a tek ponekad pišu
  - Brava dozvoljava samo jednoj niti da uđe u sekciju
  - Ponekad, suviše restriktivan pristup
- Sinhronizacioni konstrukt višeg nivoa
  koji omogućava višestruki pristup podatku
  radi čitanja su brave za čitanje i upis

Scenario upotrebe

- Nit čita podatak:
  - Pokušava da uđe u kritičnu sekciju
  - Ako nema nikoga, dopušta se ulazak
  - Ako je unutar kritične sekcije nit koja piše, blokira se
  - Ako se u kritičnoj sekciji nalazi druga nit koja čita, dopušta se ulazak
- Nit piše podatak:
  - Pokušava da uđe u kritičnu sekciju
  - Ako nema nikoga, dopušta se ulazak
  - Ako je unutar kritične sekcije nit koja čita ili piše, blokira se
- Kome dati prednost prilikom pristupa RW bravi?

Funkcije za rad sa RW bravama

- Stvaranje i uništavanje
  - `pthread_rwlock_init (rwlock,attr,count)`
  - `pthread_rwlock_destroy (rwlock)`
  - `pthread_rwlockattr_init (attr)`
  - `pthread_rwlockattr_destroy (attr)`
- Zaključavanje i otključavanje
  - `pthread_rwlock_rdlock (rwlock)`
  - `pthread_rwlock_rdtrylock (rwlock)`
  - `pthread_rwlock_wrlock (rwlock)`
  - `pthread_rwlock_wrtrylock (rwlock)`
  - `pthread_rwlock_unlock (rwlock)`