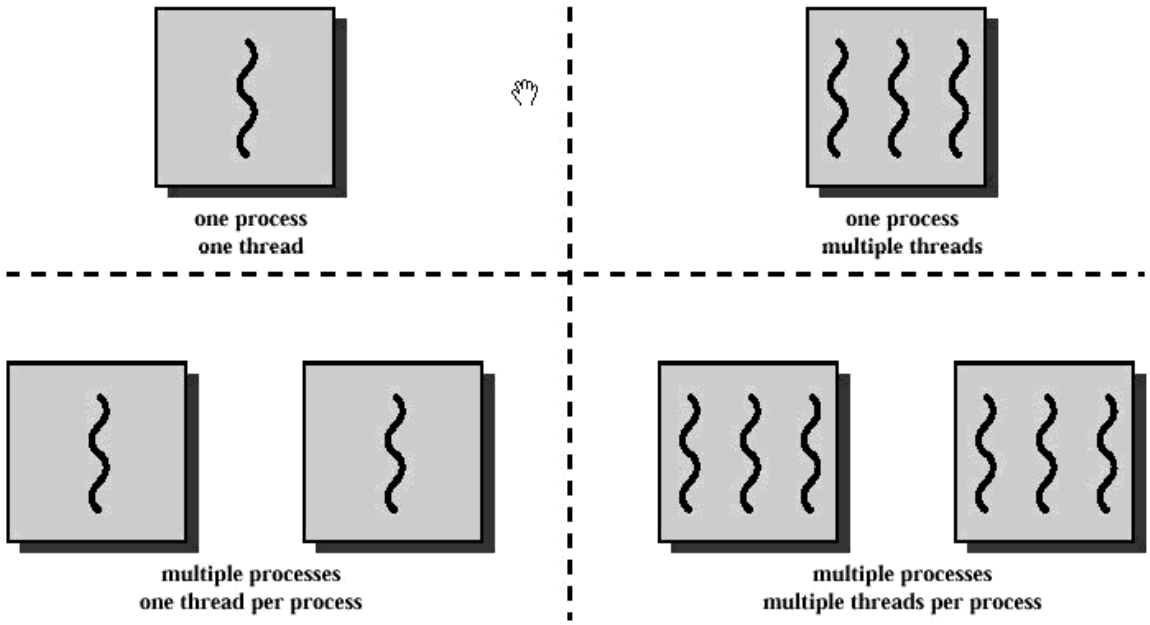
11. NITI

Operativni sistem izvršava više procesa u isto vreme. Svaki proces dobija po deo procesorskog vremena i to nam stvara utisak da se oni izvršavaju paralelno. Često svaki od procesa izvršava jednu radnju u jednom trenutku. Sa više niti u okviru jednog procesa možemo postići da jedan proces izvršava više radnji u jednom trenutku, svaka nit posebnu radnju. Niti dele deskriptore i memoriju procesa u ovkiru koga su pokrenute. Korišćenje niti može biti korisno zbog:

1. Jednostavnosti koda,
2. Ako umesto niti koristimo više procesa značajno je teža sinhronizacija,
3. Na višeprocesorskim sistemima dobijamo poboljšane performanse.



Da bi smo kompajlirali program u C-u koji koristi niti potrebno je da u toku kompilacije kompajleru prosledimo **–pthread** opciju. Biblioteka koju uključujemo da bi smo koristili niti je **pthread.h**. Identifikator niti je tip **pthread_t.** Identifikatori niti su garantovano različiti samo u okviru istog procesa, tj. različiti procesi na sistemu mogu imati iste identifikatore niti. Ovaj tip je na većini sistema broj, ali može biti i struktura, tako da za poređenje dva identifikatora niti ne možemo da koristimo operator == već funkciju:

int pthread_equal(pthread_t tid1, pthread_t tid2);

Da bi smo dobili informaciju o identifikatoru trenutne niti koristimo funkciju

pthread_t pthread_self(void);

Da bi smo kreirali novu nit koristimo funkciju

*int pthread_create(pthread_t *tidp, const pthread_attr_t *attr, void *(*start_rtn)(void), void *arg);*

Identifikator nove niti se smešta u promenljivu *tidp*. U *attr* smeštamo razna podešavanja vezana za samu nit, *start_rtn* je funkcija od koje nit kreće sa izvršavanjem, a *arg* argument koji šaljemo funkciji. Ukoliko ova funkcija vrati *vrednost različitu od 0*, onda je to kod greške do koje je došlo. Na osovnu vrednosti ovog koda funkcija *strerror* generiše smislenu poruku o greški (uključiti zaglavlje *string.h*). Sam program se naziva *glavna nit.*

NAPOMENA: Ne treba koristiti promenljivu errno kada se proverava da li je došlo do greške pri radu sa nitima.

Primer 1 – u primeru se na **lošiji** način čeka na završetak niti pomoću naredbe *sleep*

Primer 1a – modifikovan prethodni primer pokazuje ***kako ne treba*** **koristiti zajednički argument za više niti.**

Izlazak iz niti je moguć na tri načina

1. Kada nit završi svoj zadatak
2. Kada je druga nit iz istog procesa ugasi
3. Pozivom pthread_exit funkcije.

pthread_exit:

*void pthread_exit(void *rval_ptr);*

Čekanje niti da završi zadatak kao i dobijanje vrednosti koju je nit vratila moguće je uz pomoć pthread_join funkcije:

*int pthread_join(pthread_t thread, void **rval_ptr);*

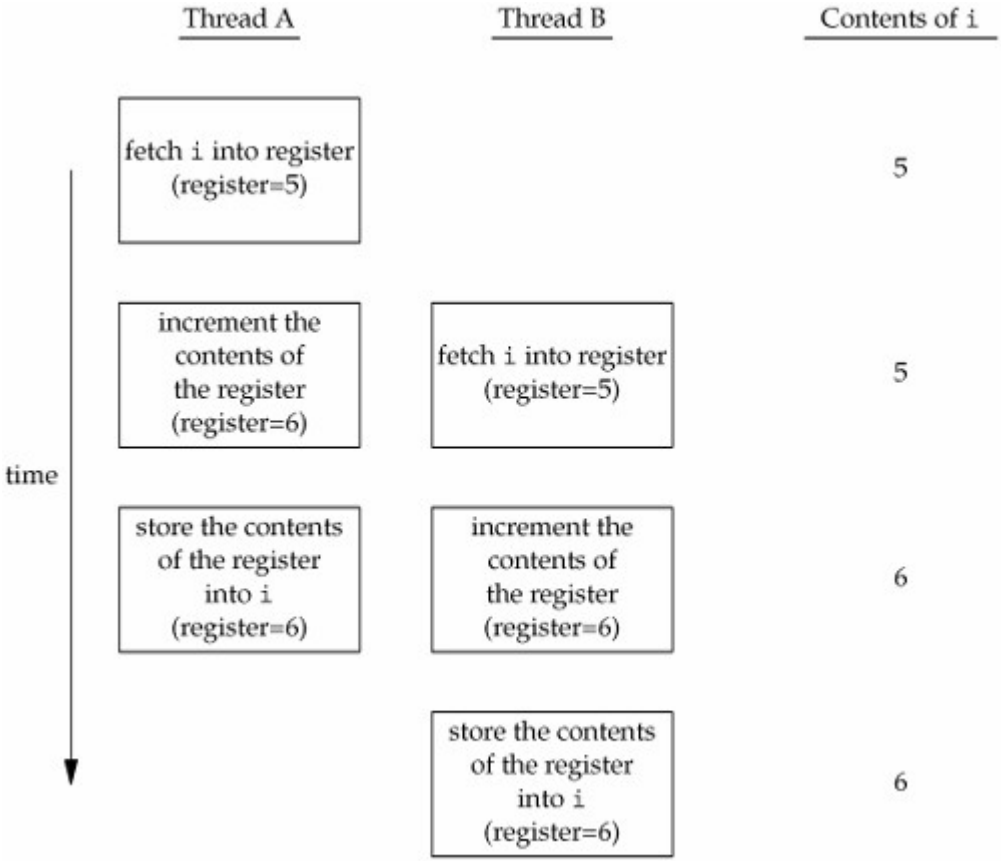
Primer 2 - primer demonstrira korišćenje funkcije za čekanje na niti, ali na **lošiji** način.

Primer 2a – ispravljene greške iz prethodnog primera (**dobro** čekanje na niti).

Sinhronizacija niti

Izmena vrednosti jedne promenljive nije atomična operacija. Izmena vrednosti promenljive može da se sastoji iz više koraka:

1. Učitaj vrednost iz memorije u registar,
2. Izmeni vrednost u registru,
3. Upiši novu vrednost nazad u memoriju.



Primer 3. - primer pravilnog čekanja na niti ali i različitih vrednosti rezultata u zavisnosti od instance procesa.

Za sinhronizaciju niti koristimo mutekse (MUTual EXclusion). Za inicijalizaciju i brisanje muteksa koristimo funkcije:

*int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);*

*int pthread_mutex_destroy(pthread_mutex_t *mutex);*

Za zaključavanje i otključavanje muteksa koristimo funkcije:

*int pthread_mutex_lock(pthread_mutex_t *mutex);*

*int pthread_mutex_trylock(pthread_mutex_t *mutex);*

*int pthread_mutex_unlock(pthread_mutex_t *mutex);*

Muteksima ograničavamo delove koda (na primer menjanje vrednosti jedne promenljive, tako da vrednost promenljive može da menja samo jedna nit u jednom trenutku). Ukoliko je neki muteks zaključan iz jedne niti, svaki put kada pokuša druga nit da ga zaključa, ona se blokira sve dok prva nit ne otključa mutex. U tom trenutku sve blokirane niti zbog tog muteksa se odblokiraju i samo ona koja se prva pokrene će zaključati mutex, dok će ostale opet da se blokiraju.

Primer 4. Primer koji koristi mutekse da zaključa promenljivu koja se menja.

Moguće greške pri radu sa nitima:

-svaki put se alocira novi prostor za 4. argument *pthread_create* a pre čekanja se ovaj prostor oslobađa. Prostor treba osloboditi tek pošto se sačeka na završetak rada niti.

Zadaci

1. Napisati program koji množi *kvadratne* matrice koje su smeštene u fajlu čije se ime prenosi kao argument komandne linije. U fajlu se nalazi prvo dimenzija matrice pa elementi prve matrice pa elementi druge matrice. Čitanje se može obaviti pomoću funkcija standardne biblioteke. Pretpostaviti da dimenzija neće biti veća od 10 (matrice se mogu alocirati statički). Za svaku poziciju (i,j) matrice koja predstavlja proizvod kreira se posebna nit koja izračunava proizvod i-te vrste i j-te kolone. Funkcija koju poziva ova nit prima pokazivač na strukturu koja ima polja: vrsta i kolona. Elementi i dimenzija matrice mogu biti globalne promenljive. Obezbediti da se čeka na završetak svih niti. Npr. za ulazni fajl:

3

1 0 0

1 1 1

0 1 2

1 1 1

0 0 1

3 0 1

množenjem treba da se dobije:

1 1 1

4 1 3

6 0 3

2. Napisati program koji učitava 2 broja *x, y* sa standardnog ulaza i svaki od njih prosleđuje kao argument posebnoj niti. Prva nit izračunava *fib(x)* a druga *fib(y)* gde je *fib(n)* n-ti član Fibonačijevog niza. Članovi niza se izračunavaju rekurzivno, dakle na neefikasan način, sa ciljem da program duže traje. Cilj je da se na procesoru koji ima bar 2 jezgra, iskoriste 2 jezgra u isto vreme, tj. pokretanjem naredbe *top* za vreme izvršavanja programa (u drugoj komandnoj liniji) trebalo bi uočiti da program u koloni %CPU ima vrednost oko 200. Testirati na vrednostima *x* i y između 40 i 45. Ubiti proces ukoliko se sam ne završi komandom *kill pid.*

3. Ime datoteke se prosleđuje kao argument komandne linije. U datoteci je zadat broj n (n<=100), a zatim i 2n brojeva, prvih n brojeva su elementi prvog vektora, drugih n brojeva drugog. Izračunati skalarni proizvod data dva vektora

$$(a_1, a_2, a_3,...)*(b_1, b_2, b_3,\ldots)=a_1b_1 + a_2b_2 + a_3b_3 +\ldots.$$

Svaki proizvod a_kb_k treba da računa posebna nit i da ga doda ukupnom zbiru. Različite niti treba da se izvršavaju istovremeno. Ograničiti da promenljivu koja predstavlja ukupni zbir ne mogu da modifikuju dve niti istovremeno.