



Operativni sistemi

Niti

Prof. dr Dragan Stojanović

Katedra za računarstvo
Univerzitet u Nišu, Elektronski fakultet



Literatura

- ✿ *Operating Systems: Internals and Design Principles*, edition, W. Stallings, Pearson Education Inc., 7th – 2012, (5th -2005, 6th - 2008, 8th – 2014 , 9th – 2017)

- ✿ <http://williamstallings.com/OperatingSystems/>

- ✿ <http://williamstallings.com/OperatingSystems/OS9e-Student/>

- ✿ Poglavlje 4: Niti

Procesi i niti

❁ Koncept procesa obuhvata dve karakteristike:

❁ **Vlasništvo nad resursom**

- Proces sadrži virtuelni adresni prostor u kome smešta sliku procesa
 - **Slika procesa:** program, podaci, stek, atributi definisani u Upravljačkom bloku procesa (PCB)
- Povremeno procesu može biti dodeljeno upravljanje ili vlasništvo nad resursima (glavna memorija, UI kanali, UI uređaji, datoteke)
- OS ima f-ju zaštite da ne bi došlo do neželjenog mešanja procesa

❁ **Raspoređivanje/izvršavanje**

- **Izvršavanje procesa** prati putanju izvršenja kroz jedan ili više programa
 - Ovo izvršavanje može biti isprepletano sa drugim procesima
- OS raspoređuje procese, **dodeljuje im resurse** i startuje izvršenje

❁ Ove dve karakteristike OS tretira nezavisno

Niti (*threads*)

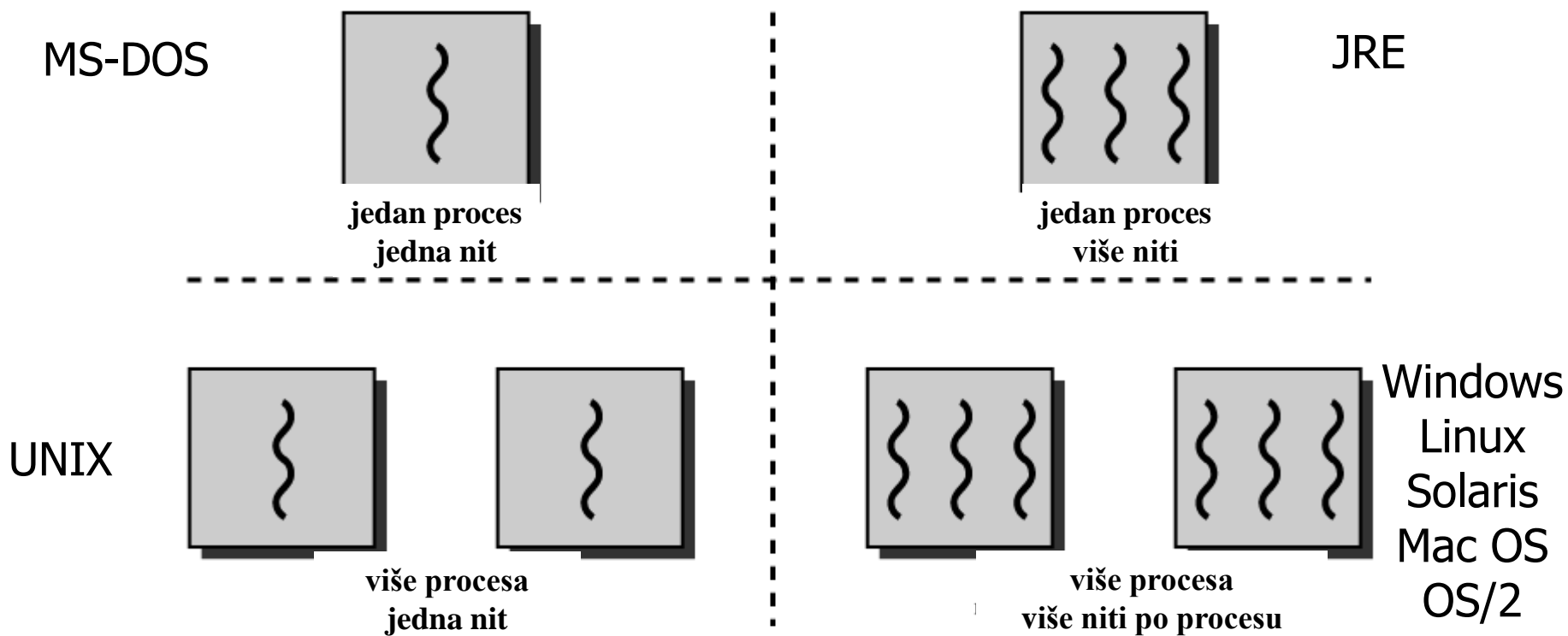
- ✿ **Proces (*task*)** predstavlja osnovnu jednicu za dodelu, vlasništvo i zaštitu u pristupu resursima
- ✿ **Nit (*thread, lightweight process*)** predstavlja entitet koji se raspoređuje i izvršava na CPU
- ✿ U tradicionalnom OS-u svaki proces ima sopstveni adresni prostor i **jednu nit izvršenja**
- ✿ U modernim OS-ima jedan proces može imati **više niti izvršenja** koje se izvode konkurentno ili paralelno
 - ▣ Sve niti pridružene jednom procesu dele program i resurse tog procesa
 - ▣ Svaka nit ima sopstveni magacin i stanje
- ✿ Nit omogućava podelu posla koji je dodeljen procesu na više niti izvršenja, tako da svaka nit preuzima deo tog posla



Višenitnost (*Multithreading*)

- ✿ **Višenitnost** je sposobnost OS-a da podrži više niti izvršenja unutar jednog procesa
- ✿ Tradicionalni pristup – jedna nit po procesu – naziva se **pristup sa jednom niti** (*single threaded*)
- ✿ Niti u OS-ima:
 - ✦ MS-DOS podržava jednu nit
 - ✦ Tradicionalni UNIX podržava više procesa, ali samo jednu nit po procesu
 - ✦ JRE je primer jednog procesa sa više niti
 - ✦ Windows, Solaris, Linux, Mac OS, OS/2 podržavaju više niti

Niti i procesi



{ = praćenje instrukcije

(Stallings, 2012)

Niti

Operativni sistemi

Proces

✿ Svakom procesu je dodeljen:

- ✦ Virtuelni adresni prostor gde se nalazi **slika procesa** – program, podaci, stek, kao i atributi definisani u Upravljačkom bloku procesa (PCB)
- ✦ Zaštićeni pristup procesoru, drugim procesima (IPC), datotekama i UI resursima (uređaji i kanali)

✿ Unutar procesa može postojati jedna ili više niti

Nit

✿ Svaka nit poseduje:

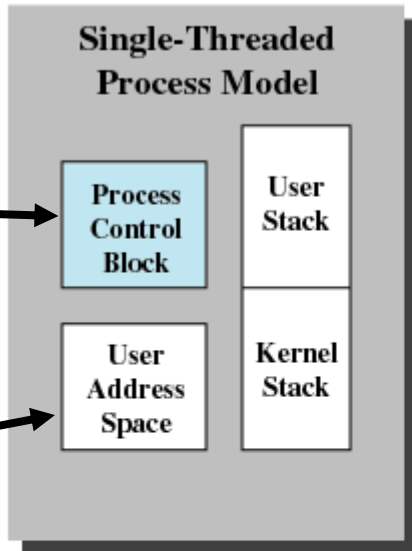
- ✦ Stanje izvršenja niti (*Izvršava se, Spremna*, itd)
- ✦ Sačuvani kontekst niti kada se ne izvršava (programski brojač, registri procesora, stek pointeri)
- ✦ Stek (magacin) izvršavanja
- ✦ Statičku memoriju za lokalne promenljive niti
- ✦ Pristup memoriji i resursima svog procesa koje deli sa svim ostalim nitima procesa

Modeli jedno-nitnog i višenitnog procesa

Model jednonitnog procesa

Upravljački
blok
procesa

Korisnički
adresni
prostor



Model višenitnog procesa

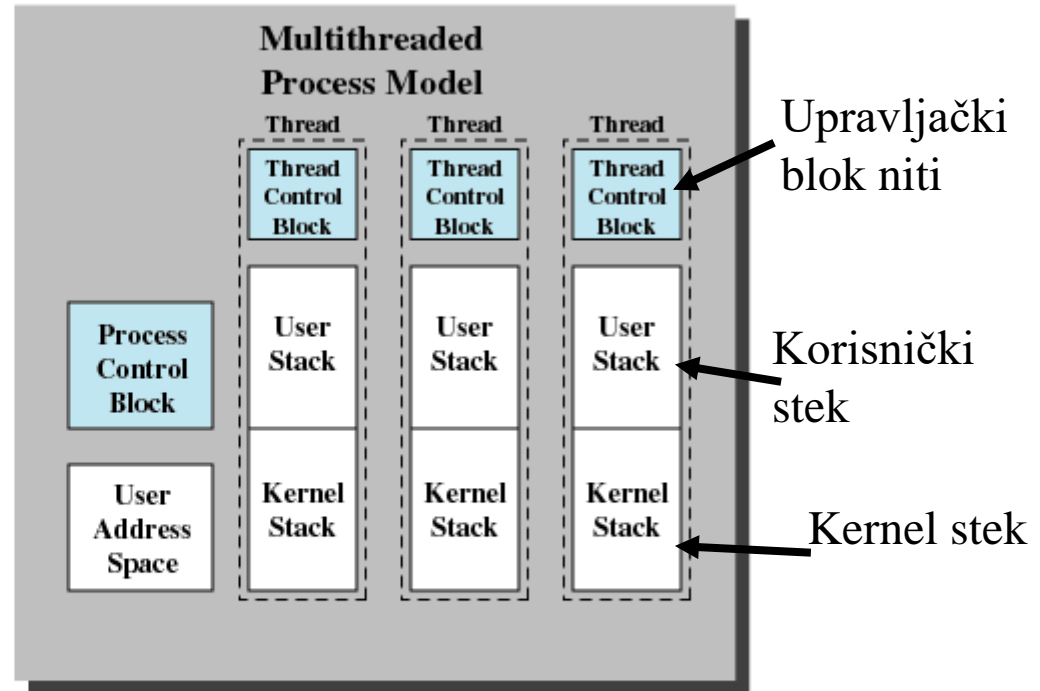


Figure 4.2 Single Threaded and Multithreaded Process Models

Koristi od niti

✚ Osnovne koristi niti se zasnivaju na dobitku u performansama

1. Potrebno je mnogo manje vremena da bi se kreirala nova nit u postojećem procesu, nego da se kreira novi proces.
2. Potrebno je mnogo manje vremena za prekid niti nego prekid procesa.
3. Potrebno je mnogo manje vremena za prebacivanje (*switch*) između dve niti unutar istog procesa nego za komutiranje procesa
4. Niti mogu komunicirati međusobno jer dele adresni prostor procesa i otvorene datoteke
 - Bez poziva OS kernela

Korišćenje niti u jednokorisničkom multiprogramskom sistemu

- ✚ Čeoni (*foreground*) i pozadinski (*background*) poslovi
 - ✚ Program za tabelarna izračunavanja – jedna nit prikazuje GUI i prihvata korisnički unos, dok druga nit izvršava korisničke komande i ažurira tabele
- ✚ Asinhrona obrada
 - ✚ *Autosave* u programu za obradu teksta
- ✚ Brzina izvršavanja
 - ✚ Više niti može da se izvršava konkurentno/paralelno, dok je jedna niti blokirana na U/I druga može da se izvršava
- ✚ Modularna struktura programa

Niti

- ✚ Neke akcije u okviru OS odnose se na sve niti jednog procesa
 - ✚ OS mora da upravlja ovim akcijama na nivou procesa.
- ✚ Primeri:
 - ✚ Suspendovanje procesa uključuje suspendovanje svih niti procesa
 - ✚ Terminiranje procesa terminira sve niti u okviru procesa
- ✚ Slično procesima
 - ✚ Niti poseduju stanja izvršenja i mogu se sinhronizovati međusobno

Stanja niti

- ✚ Osnovna stanja niti:
 - ✚ Izvršava se (*Running*)
 - ✚ Spremna (*Ready*)
 - ✚ Blokirana (*Blocked*)
- ✚ Prelazi između stanja su isti kao kod procesa
- ✚ Ne postoje suspendovana stanja jer sve niti unutar istog procesa dele isti adresni prostor
 - ✚ Suspendovanje procesa povlači suspendovanje svih niti tog procesa
- ✚ Terminiranjem procesa, terminiraju se sve niti tog procesa

Operacije nad nitima

✿ Kreiranje (umnožavanje, *spawn*)

- ✦ Kreiranjem procesa kreira se nit tog procesa. Svaka nit procesa može kreirati novu nit u okviru istog procesa, obezbeđujući pointer na funkciju (instrukcije) i argumente nove niti. Nova nit dobija sopstveni kontekst (TCB), magacine i smešta se u red spremnih

✿ Blokiranje

- ✦ Kada nit čeka na događaj (npr. U/I) blokira se pri čemu se čuvaju vrednosti korisničkih registara, programskog brojača i stek pointera, i procesor započinje izvršavanje sledeće spremne niti u istom ili drugom procesu

✿ Deblokiranje

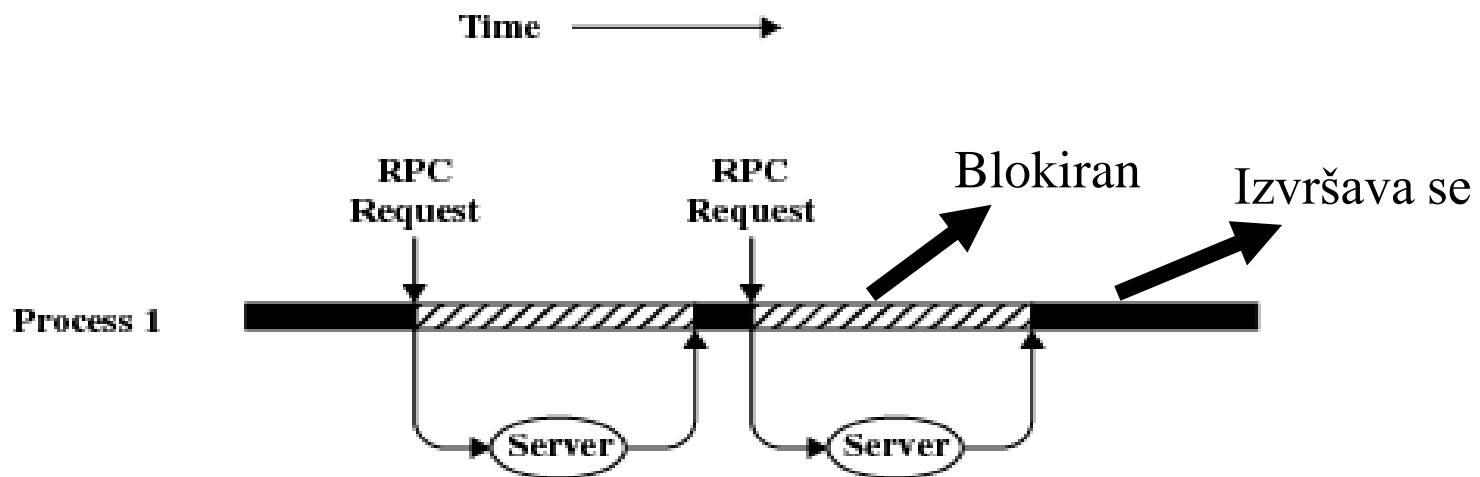
- ✦ Kada nastane događaj koji je čekala, nit se prevodi u stanje *spremna*

✿ Završetak

- ✦ Kad se niti završi, dealociraju se TCB i magacini

Primer: RPC korišćenjem jedne niti

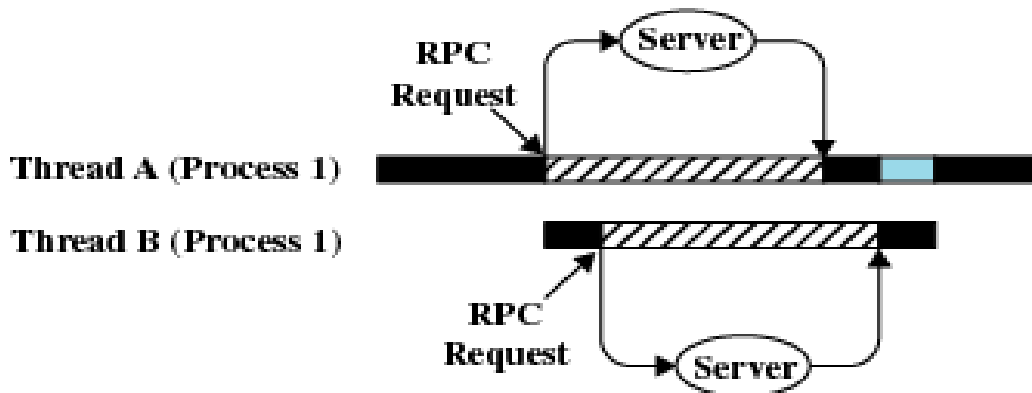
- RPC je tehnika kojom dva procesa koja se izvršavaju na različitim računarima međusobno komuniciraju koristeći mehanizam poziva procedure
- Program vrši dva poziva udaljenih procedura (*Remote Procedure Call* - RPC) na dva različita računara






(a) RPC Using Single Thread

RPC korišćenjem više niti

- Program vrši dva poziva udaljenih procedure (RPC) na dva različita računara, i za svaki poziv koristi posebnu nit
- Ostvareno je poboljšanje u performansama

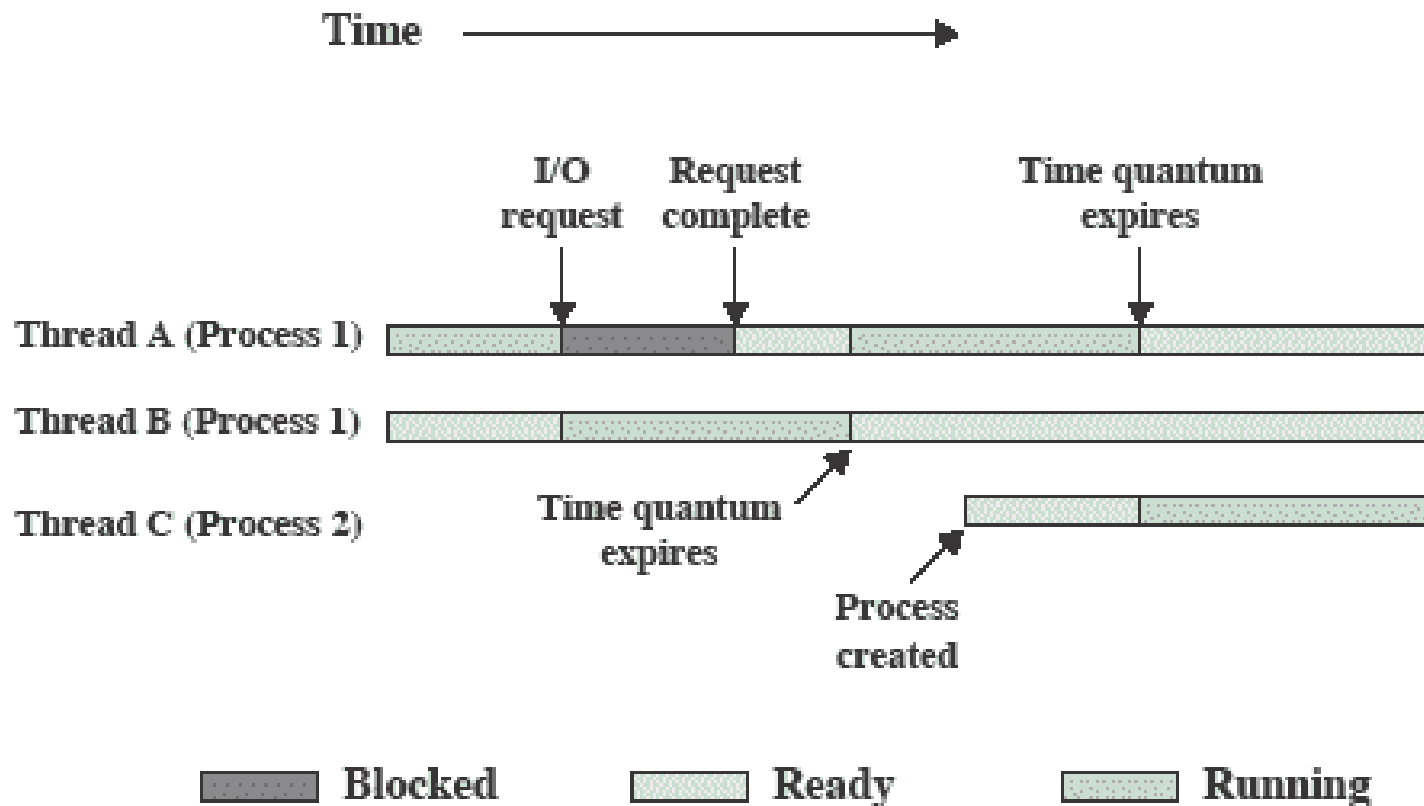


(b) RPC Using One Thread per Server (on a uniprocessor)

-  Blocked, waiting for response to RPC
-  Blocked, waiting for processor, which is in use by Thread B
-  Running

Višenitnost na jednoprocesorskom računaru

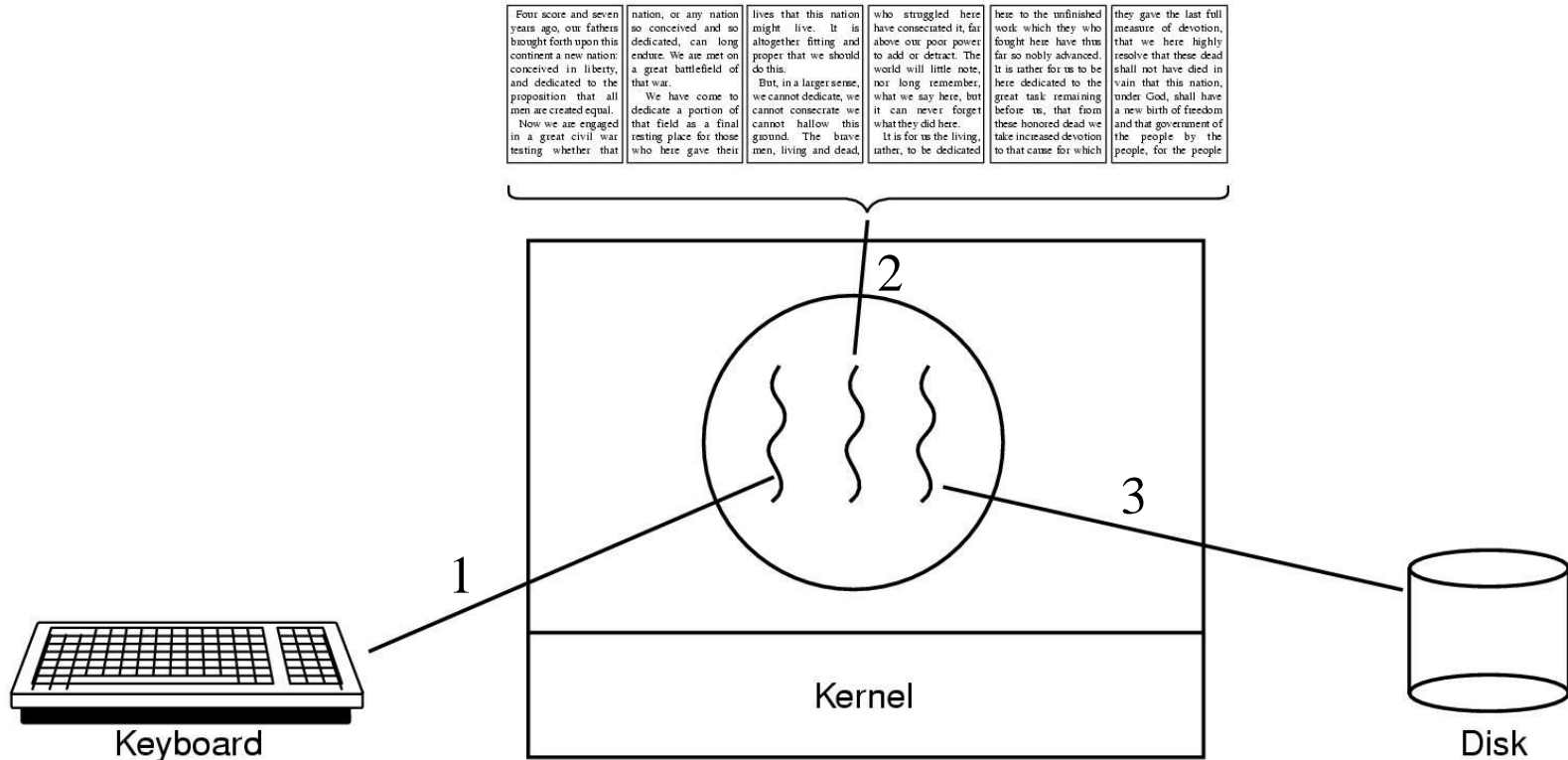
- Primer izvršenja više niti na jednoprocesorskom računaru



Izvršavanje i sinhronizacija niti

- ✿ U **jednoprocesorskom** sistemu niti jednog procesa se izvršavaju konkurentno, slično procesima kod multiprogramiranja
- ✿ U **višeprocorskom** sistemu niti jednog procesa se mogu izvršavati paralelno - na različitim procesorima
- ✿ Niti jednog procesa **nisu potpuno nezavisne**, jer se izvršavaju u jednom adresnom prostoru, što znači da dele globalne promenljive i otvorene datoteke
 - ✦ Svaka promena memorijskog resursa jedne niti utiče na druge niti koje koriste iste resurse
 - ✦ Neophodno je obezbediti sinhronizaciju više niti u pristupu istim memorijskim resursima
 - ✦ Tehnike za sinhronizaciju niti su identične tehnikama za sinhronizaciju procesa

Primer 1: Višenitni Word procesor

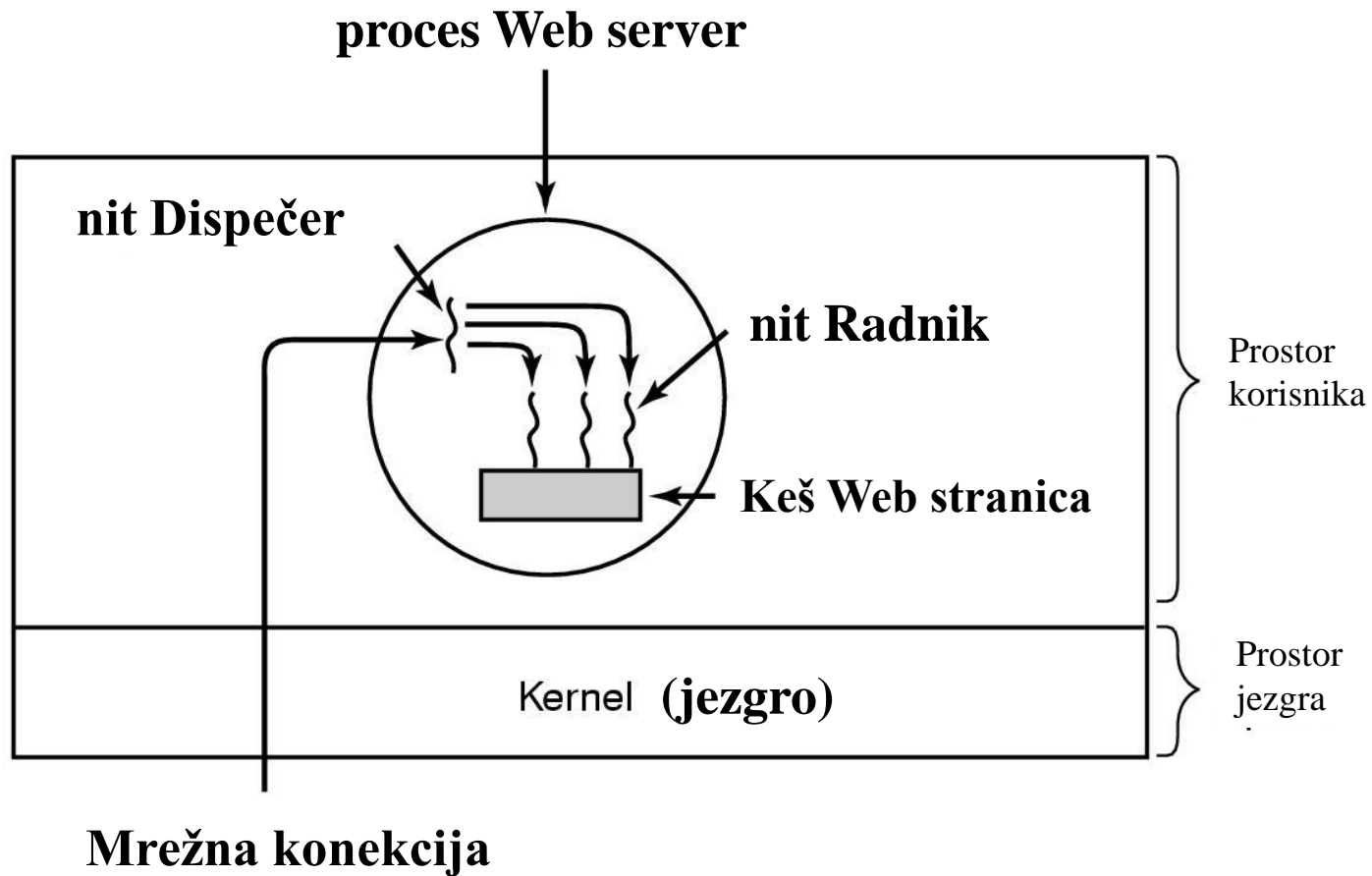


Tanenbaum, 2014

Word processor sa tri niti: 1- interakcija sa korisnikom; 2- reformatiranje dokumenta; 3 - backup dokumenta

Niti

Primer 2: Višenitni Web server



Tanenbaum, 2014

Tipovi niti

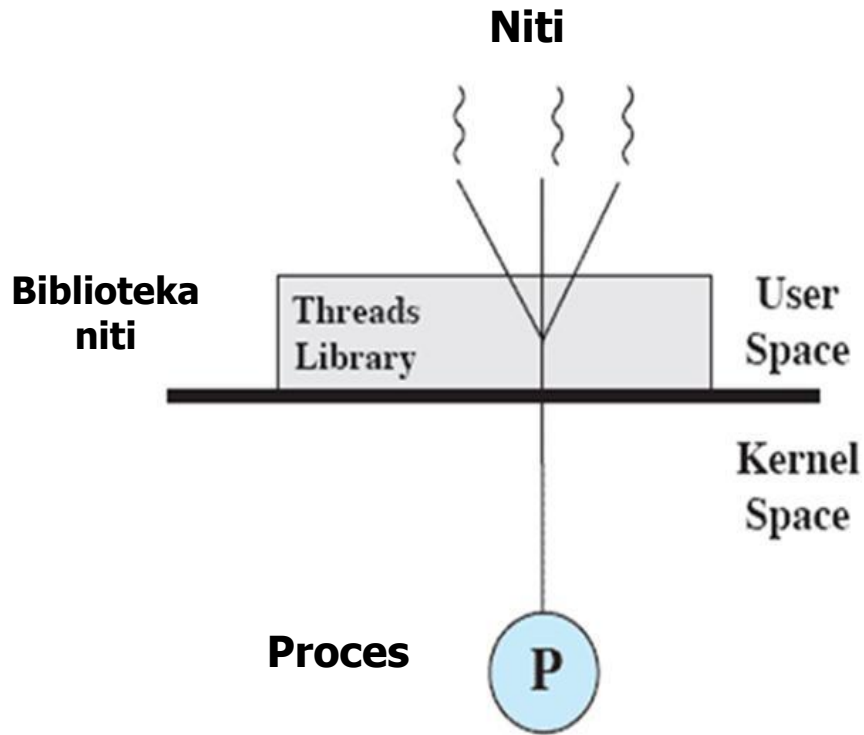
- ✿ Postoje dve kategorije implementacije niti:
 - ✦ Niti nivoa korisnika (**ULT** - *User Level Thread*)
 - ✦ Niti nivoa jezgra (**KLT** - *Kernel Level Thread*)
 - Kernelom podržane niti, *lightweight* procesi



Niti nivoa korisnika - ULT

- Kernel OS nije svestan postojanja niti
- Nitima upravlja aplikacioni program koristeći **biblioteku niti – skup funkcija za upravljanje nitima**
- Promena (komutiranje, *switching*) niti ne zahteva privilegije kernel moda, tj. nema promene moda
- Raspoređivanje niti je specifično za aplikaciju i obavlja se u okviru lokalne procedure u biblioteci niti, dakle **bez promene moda** (korisnički → kernel)
- Svaki proces može imati **sopstveni** algoritam planiranja niti
- Korišćenjem biblioteke niti bilo koja aplikacija se može implementirati kao višenitna

Niti nivoa korisnika - ULT



Podrška za niti nivoa korisnika

Biblioteka niti sadrži kod za:

- Kreiranje niti,
- Brisanje niti,
- Prenos poruka i podataka između niti,
- Raspoređivanje izvršenja niti,
- Pamćenje i restauriranje konteksta niti,...

Kontrolni blok niti (TCB) sadrži:

- Programski brojač,
- Pokazivač magacina,
- Registre procesora,
- Stanje,
- Prioritet,...

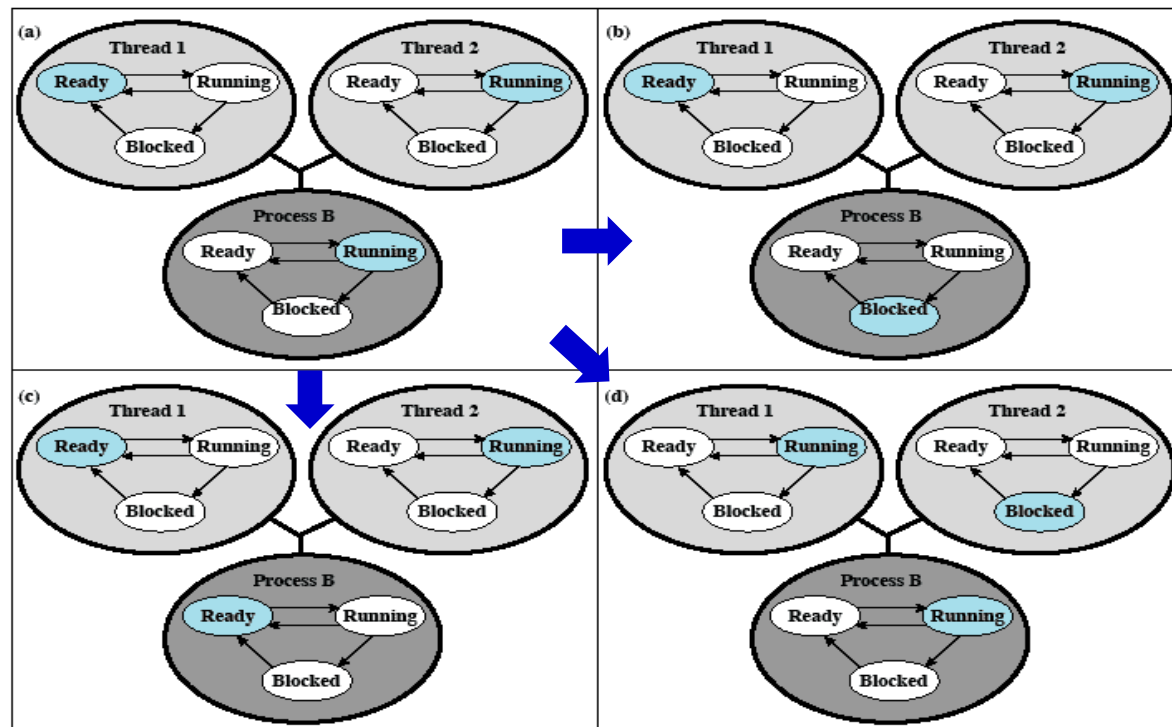


Niti nivoa korisnika - aktivnosti kernela

- ✚ Kernel ne vodi računa o aktivnostima niti, ali još uvek upravlja aktivnostima procesa
- ✚ Kada nit pozove neki sistemski poziv (pozove funkciju OS-a), ceo proces se blokira, ali za biblioteku niti ta nit je još uvek u stanju izvršenja
- ✚ Stanja niti su nezavisna od stanja procesa

Stanja ULT niti i procesa

- Proces B je u stanju *Izvršavanje* i izvršava se Nit2 (a)
- Kada se nit blokira, blokira se proces, dok je za biblioteku niti Nit2 još uvek u stanju *Izvršavanje* (b)
- Ukoliko istekne vremenski kvant proces je *Spreman*, a Nit2 i dalje u stanju *Izvršavanje* (c)
- Nit2 se *blokira* kada zahteva neku akciju od Niti1, koja prelazi u stanje *Izvršavanje*, dok je proces sve vreme u stanju *Izvršavanje* (d)



Colored state
is current state

Niti

Operativni sistemi

ULT - Prednosti i nedostaci

✚ Prednosti

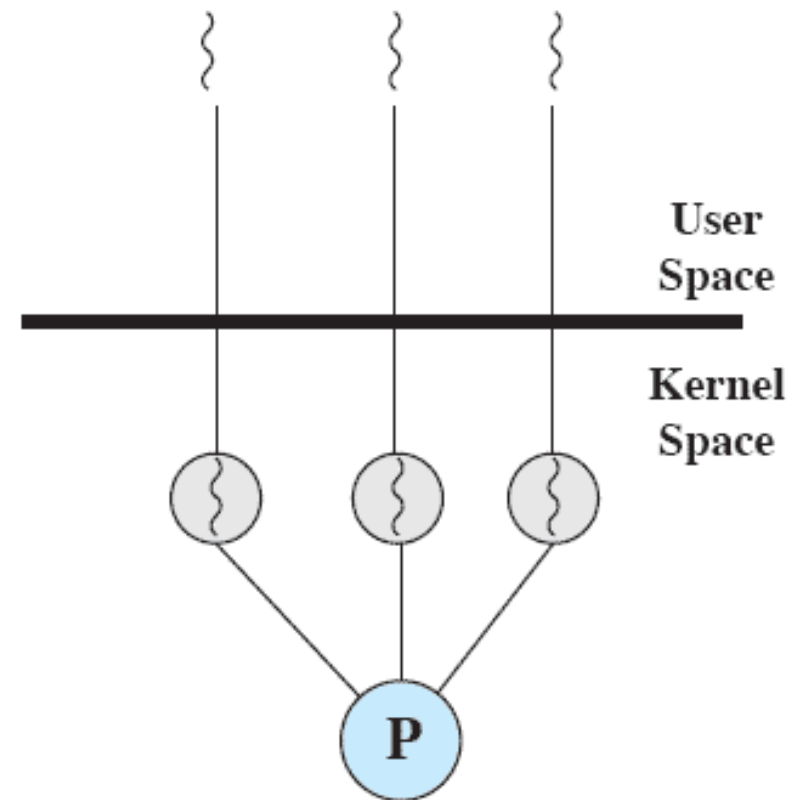
- ✚ Promena sa jedne na drugu nit ne zahteva privilegije kernel moda, nije potrebno prebacivanje moda (korisnički-kernel, kernel-korisnički), pa se brže odvija
- ✚ Raspoređivanje može biti specifičan za proces; bira se najbolji algoritam
- ✚ ULT se može izvoditi nad bilo kojim OS-om. Jedino je potrebna biblioteka niti koja se može implementirati u OS-u koji ne podržava niti

✚ Nedostaci

- ✚ Mnogi sistemski pozivi su blokirajući i kernel blokira procese. Ukoliko jedna nit pozove blokirajući sistemski poziv sve niti unutar tog procesa biće blokirane
 - Rešenje: korišćenje tehnike **omotavanja** (*jacketing*)
- ✚ Kernel može dodeljivati procesore jedino procesima
 - Niti istog procesa se ne mogu izvršavati istovremeno na različitim procesorima

Niti nivoa kernela - KLT

- Kernel upravlja nitima
- Nema biblioteke niti, ali postoji API za rad sa nitima
- Kernel održava kontekstne informacije za procese i niti
- Kernel vrši planiranje i raspoređivanje niti
- Kernel vrši promenu niti
- Primeri: Windows, Linux, Mac OS X



(b) Pure kernel-level

KLT - Prednosti i nedostaci

✚ Prednosti

- ✚ Kernel može simultano planirati više niti istog procesa na više procesora
- ✚ Blokiranje se vrši na nivou niti - kada se nit blokira, jezgro bira spremnu nit istog ili drugog procesa
- ✚ Programi kernela mogu biti višenitni

✚ Nedostaci

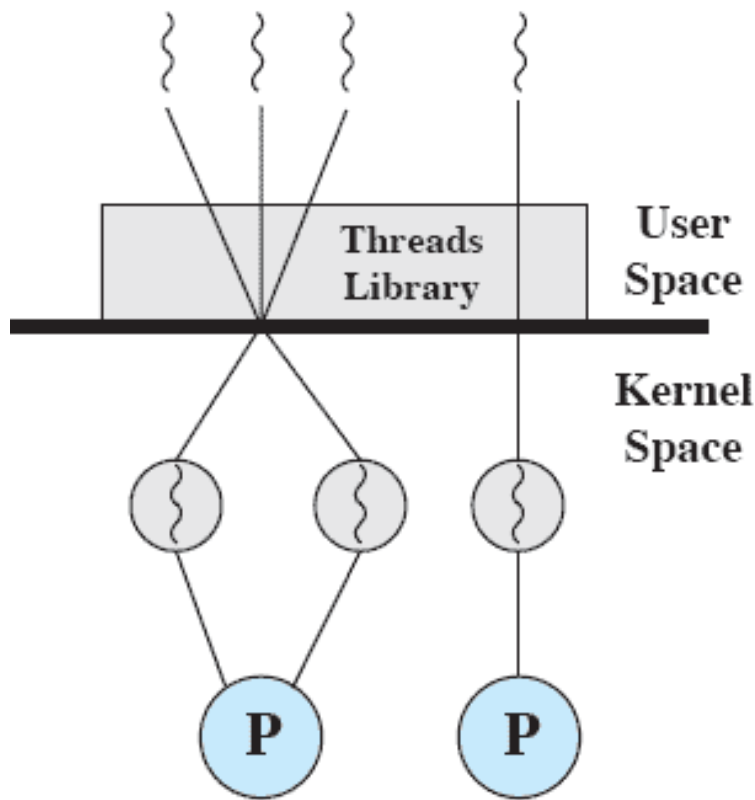
- ✚ Promena niti unutar istog procesa zahteva promenu moda (kernel mod) i angažovanje OS
- ✚ Upravljanje nitima nivoa kernela sporije nego kod ULT-a

ULT vs. KLT vs. Prosesi

- ✿ Trajanje operacija nad nitima (ULT i KLT) i procesima
 - ✚ Kreiranje prazne niti/procesa
 - ✚ Sinhronizacija niti/procesa zasnovana na signalima

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

Hibridna implementacija



(c) Combined

- ✚ Kreiranje niti obavlja se u prostoru korisnika
- ✚ Raspoređivanje i sinhronizacija niti se obavlja uglavnom na nivou aplikacije u prostoru korisnika
- ✚ Više ULT-a iz jednog procesa preslikavaju se u manji, ili jednak broj KLT-ova
- ✚ Programer može podešavati broj KLT-a za određenu aplikaciju
- ✚ Kombinuje dobre osobine ULT i KLT i minimizuje nedostatke
- ✚ Primer: Solaris

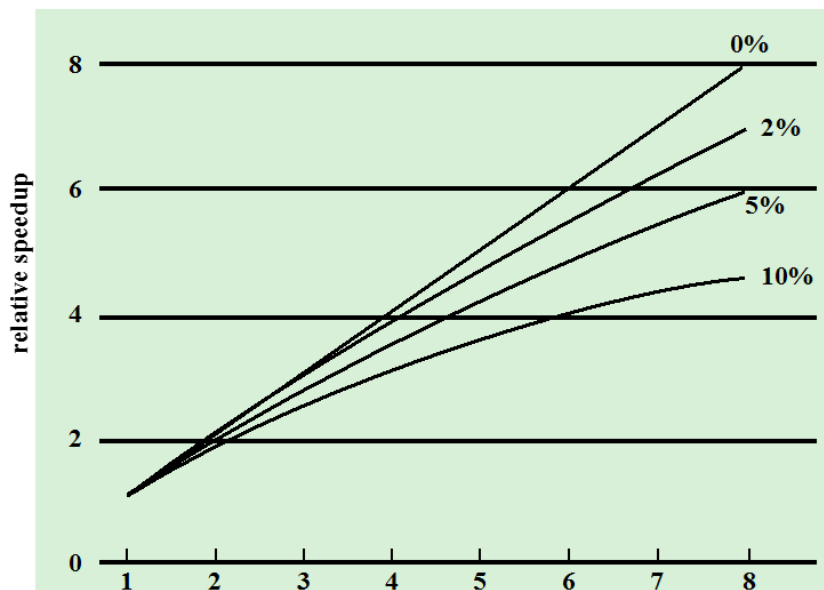
Performanse softvera na više jezgara (iz Stallings, 2012 - 7th edition)



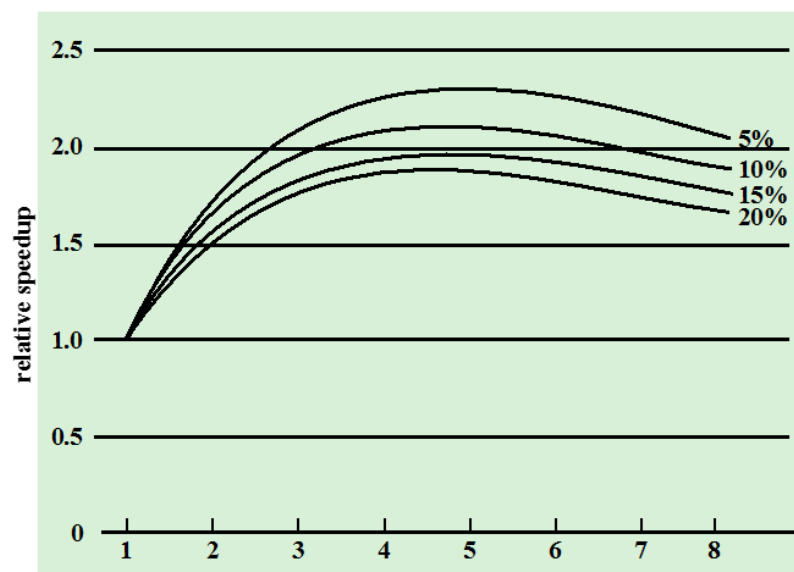
Amdahl-ov zakon

- f – procenat koda koji se može paralelizovati na N procesora

$$\text{Speedup} = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}} = \frac{1}{(1 - f) + \frac{f}{N}}$$



(a) Speedup with 0%, 2%, 5%, and 10% sequential portions



(b) Speedup with overheads



Niti u savremenim OS

- ✚ **Windows** – knjiga Stallings (Tanenbaum), praktikum
- ✚ **Solaris** – knjiga Stallings, praktikum
- ✚ **Linux/Unix** – knjiga Stallings, praktikum
 - ✚ POSIX - praktikum

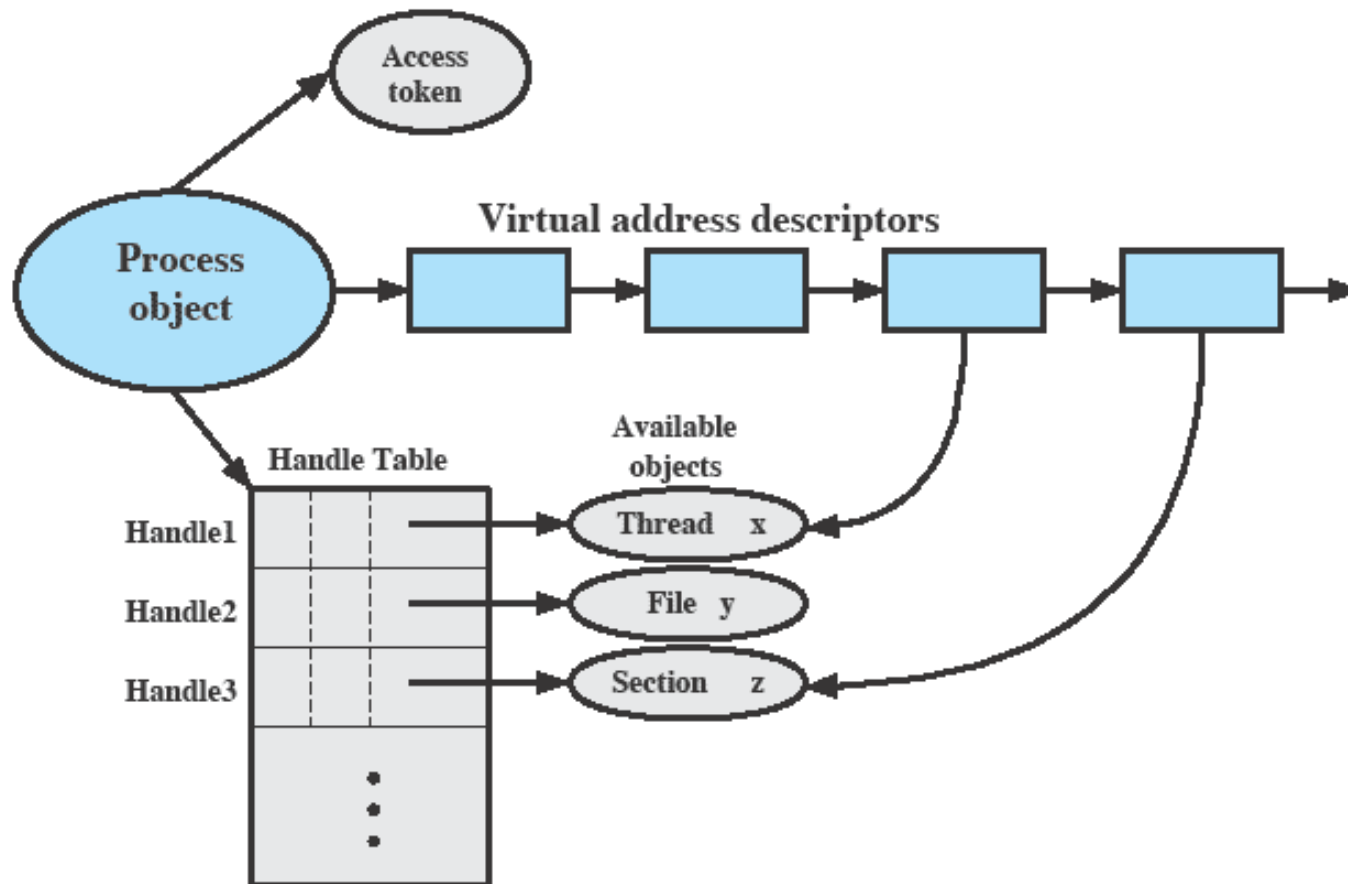
Windows procesi i niti

- ✚ Osnovni koncepti koji se koriste za upravljanje procesorom i resursima

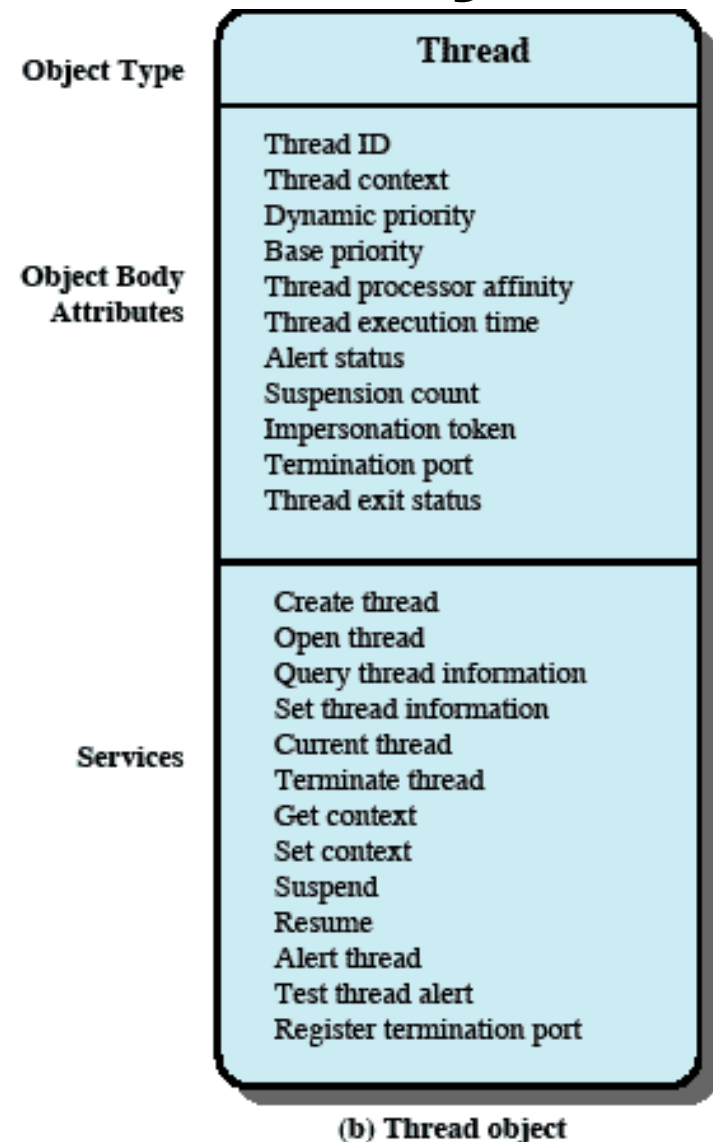
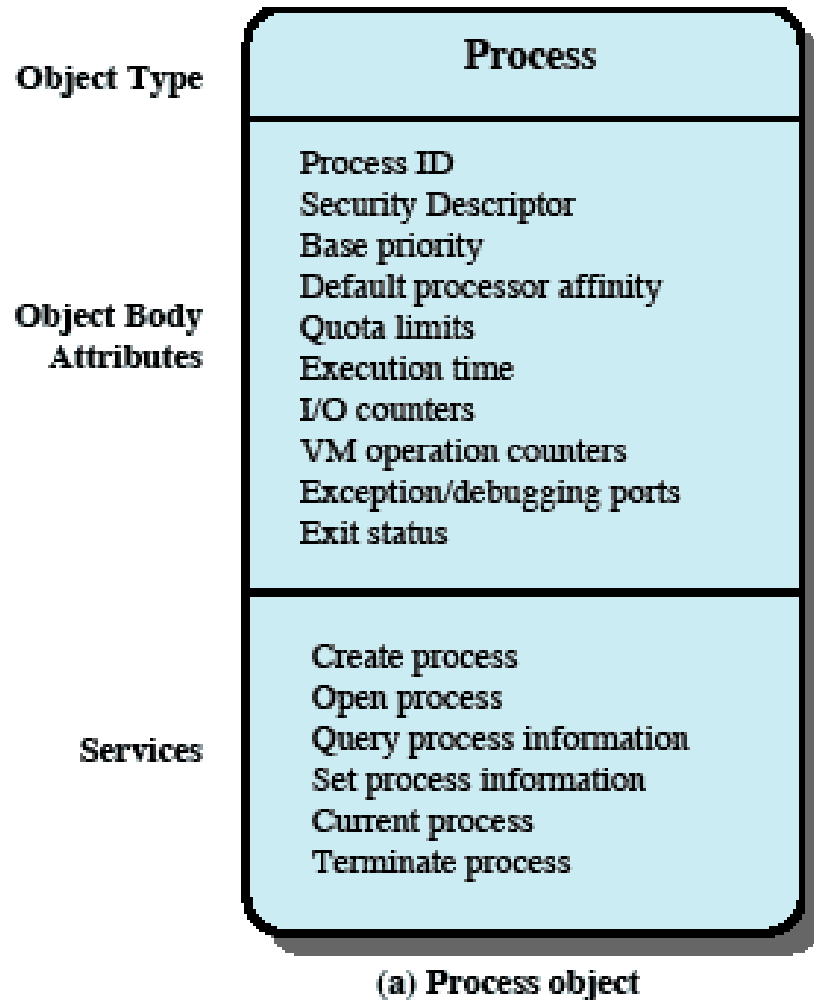
Name	Description
Job	Collection of processes that share quotas and limits
Process	Container for holding resources
Thread	Entity scheduled by the kernel
Fiber	Lightweight thread managed entirely in user space

Veza između procesa i resursa

Windows procesi i njihovi resursi



Windows Proces i Thread objekti



Stanja Windows niti

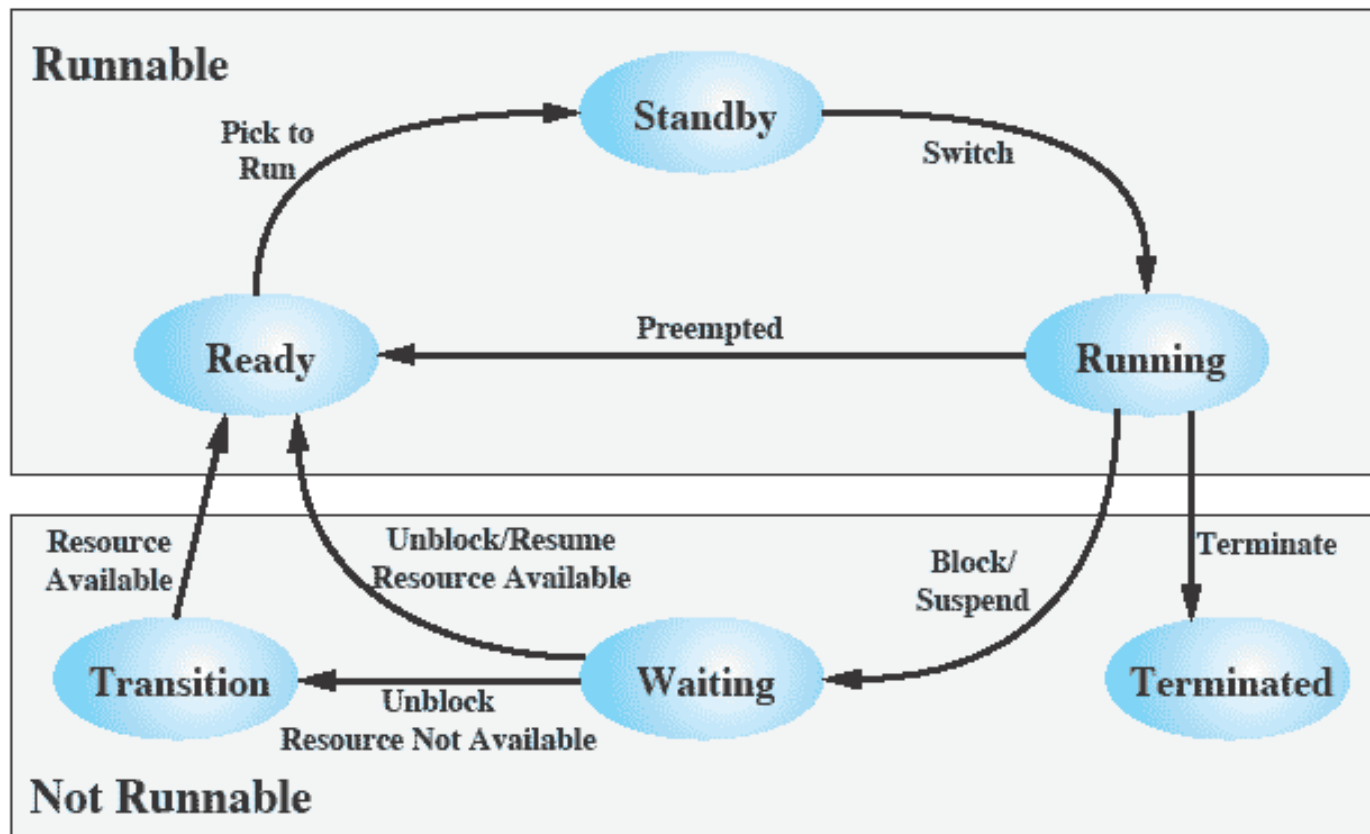


Figure 4.14 Windows Thread States



Windows API sistemski pozivi

Win32 API Function	Description
CreateProcess	Create a new process
→ CreateThread	Create a new thread in an existing process
CreateFiber	Create a new fiber
ExitProcess	Terminate current process and all its threads
→ ExitThread	Terminate this thread
ExitFiber	Terminate this fiber
SetPriorityClass	Set the priority class for a process
→ SetThreadPriority	Set the priority for one thread
CreateSemaphore	Create a new semaphore
CreateMutex	Create a new mutex
OpenSemaphore	Open an existing semaphore
OpenMutex	Open an existing mutex
WaitForSingleObject	Block on a single semaphore, mutex, etc.
WaitForMultipleObjects	Block on a set of objects whose handles are given
PulseEvent	Set an event to signaled then to nonsignaled
ReleaseMutex	Release a mutex to allow another thread to acquire it
ReleaseSemaphore	Increase the semaphore count by 1
EnterCriticalSection	Acquire the lock on a critical section
LeaveCriticalSection	Release the lock on a critical section

Neki Win32 pozivi za upravljanje procesima, nitima i fiberima

*Praktikum
Tanenbaum, 2014*

Niti

Solaris procesi i niti

- ✿ Solaris koristi 4 posebna koncepta vezana za niti
 - ✦ **Proces**: uključuje korisnički adresni prostor, stek i upravljački blok procesa
 - ✦ **Niti na korisničkom nivou**: Implementirane korišćenjem biblioteke niti u adresnom prostoru procesa; nevidljive za OS
 - ✦ **Procesi lake kategorije** (*Lightweight*): obezbeđuju mapiranje između ULT i kernel niti. Svaki LWP podržava više ULT koje mapira u jednu kernel nit.
 - ✦ **Kernel niti**: osnovni entiteti koji mogu biti raspoređeni i izvršeni na procesoru

Solaris višenitna arhitektura - primer

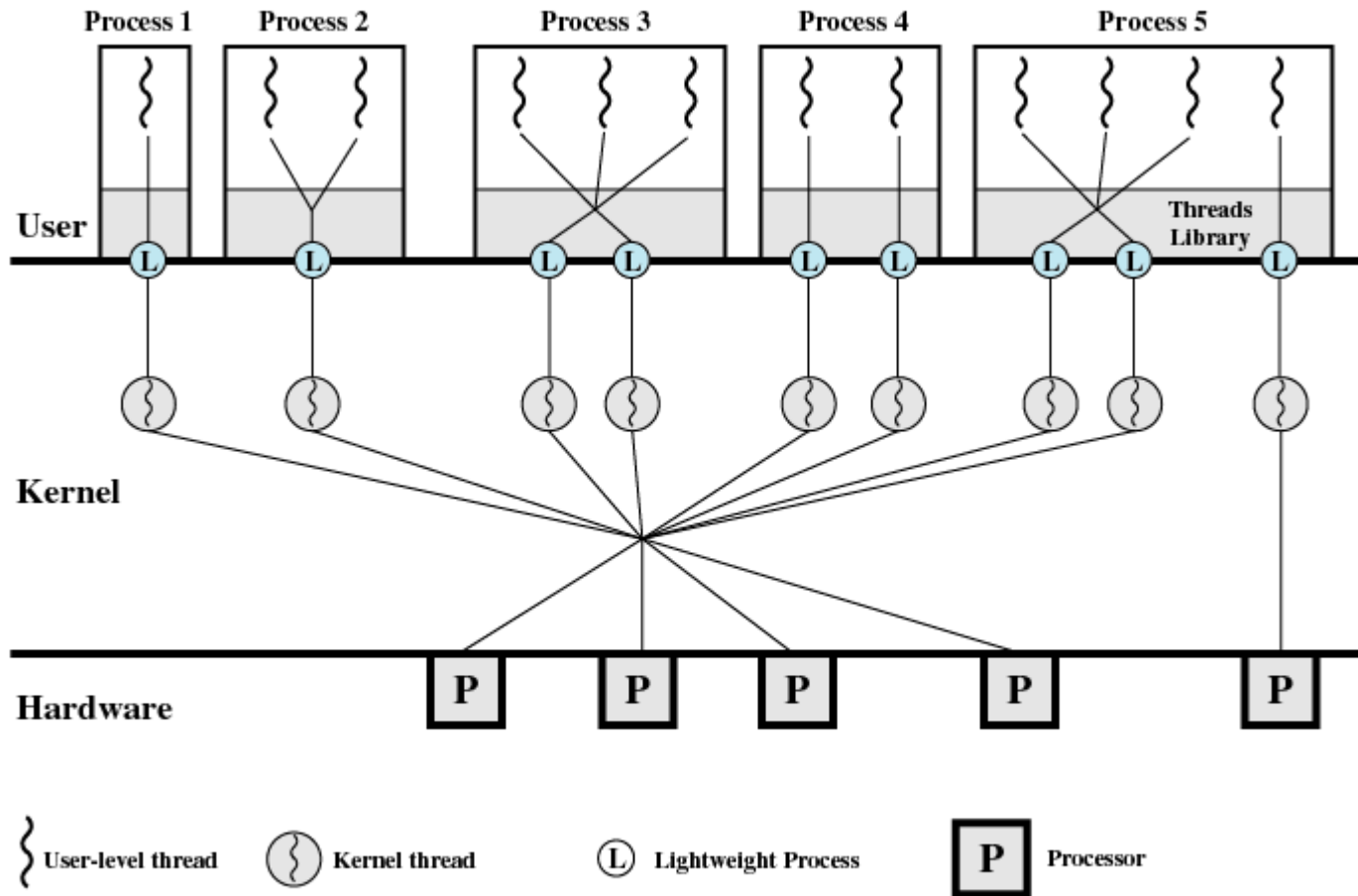
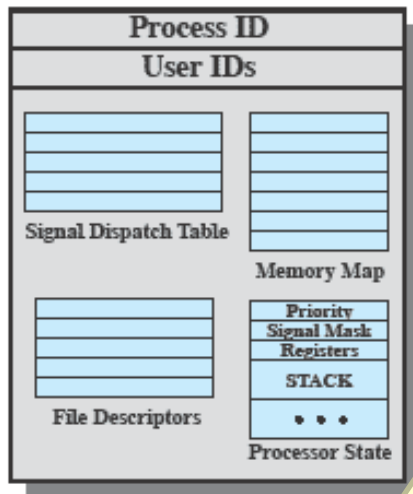


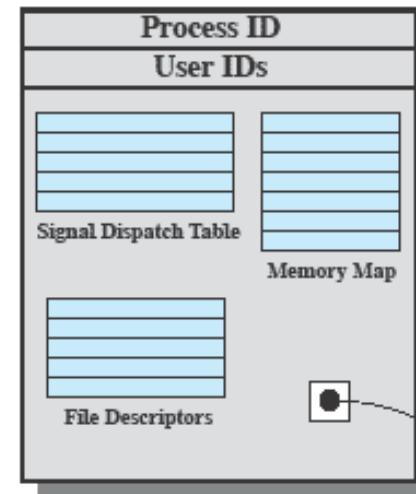
Figure 4.15 Solaris Multithreaded Architecture Example

Tradicionalni UNIX - Solaris

UNIX Process Structure



Solaris Process Structure



Solaris zamenjuje blok sa stanjem procesora listom sa LWP-ima

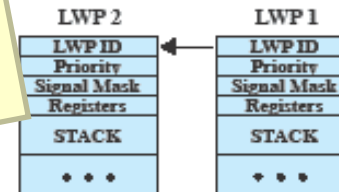


Figure 4.16 Process Structure in Traditional UNIX and Solaris [LEWI96]

UNIX/Linux niti (POSIX)

Thread call	Description
pthread_create	Create a new thread in the caller's address space
pthread_exit	Terminate the calling thread
pthread_join	Wait for a thread to terminate
pthread_mutex_init	Create a new mutex
pthread_mutex_destroy	Destroy a mutex
pthread_mutex_lock	Lock a mutex
pthread_mutex_unlock	Unlock a mutex
pthread_cond_init	Create a condition variable
pthread_cond_destroy	Destroy a condition variable
pthread_cond_wait	Wait on a condition variable
pthread_cond_signal	Release one thread waiting on a condition variable



Kreiranje niti (POSIX primer)

```
#include <stdio.h>
→ #include <pthread.h>

int glob_data = 5 ;

void *kidfunc(void *p) {
    printf ("Nova nit. Globalni podatak je: %d.\n", glob_data) ;
    printf ("Nova nit. ID procesa je: %d.\n", getpid()) ;
    glob_data = 15 ;
    printf ("Ponovo nova nit. Globalni podatak je: %d.\n", glob_data) ;
}

main ( ) {
→ pthread_t kid ;
→ pthread_create (&kid, NULL, kidfunc, NULL) ;
    printf ("Osnovna nit. Globalni podatak je: %d\n", glob_data) ;
    printf ("Osnovna nit. ID procesa je: %d.\n", getpid()) ;
    glob_data = 10 ;
→ pthread_join (kid, NULL) ;
    printf ("Kraj programa. Globalni podatak je: %d\n", glob_data) ;
}
```

Linux proces/task

- ✿ Proces ili *task*, u Linux-u je predstavljen *task_struct* strukturom podataka (PCB)
- ✿ Ova struktura sadrži informacije svrstane u sledeće kategorije:
 - ✦ Stanje
 - ✦ Informacije za raspoređivanje (prioriteti)
 - ✦ Identifikatore
 - ✦ Interprocesnu komunikaciju
 - ✦ Linkove na druge *task_struct*
 - ✦ *File* sistem
 - ✦ Memorijski adresni prostor
 - ✦ Kontekst procesa
- ✿ Nove verzije Linux, kao i Unix obezbeđuju KLT
 - ✦ Linux koristi specifično rešenje; ne pravi razliku između procesa i niti, već koristi *lightweight* procese, slično Solaris-u

Stanja Linux procesa/niti

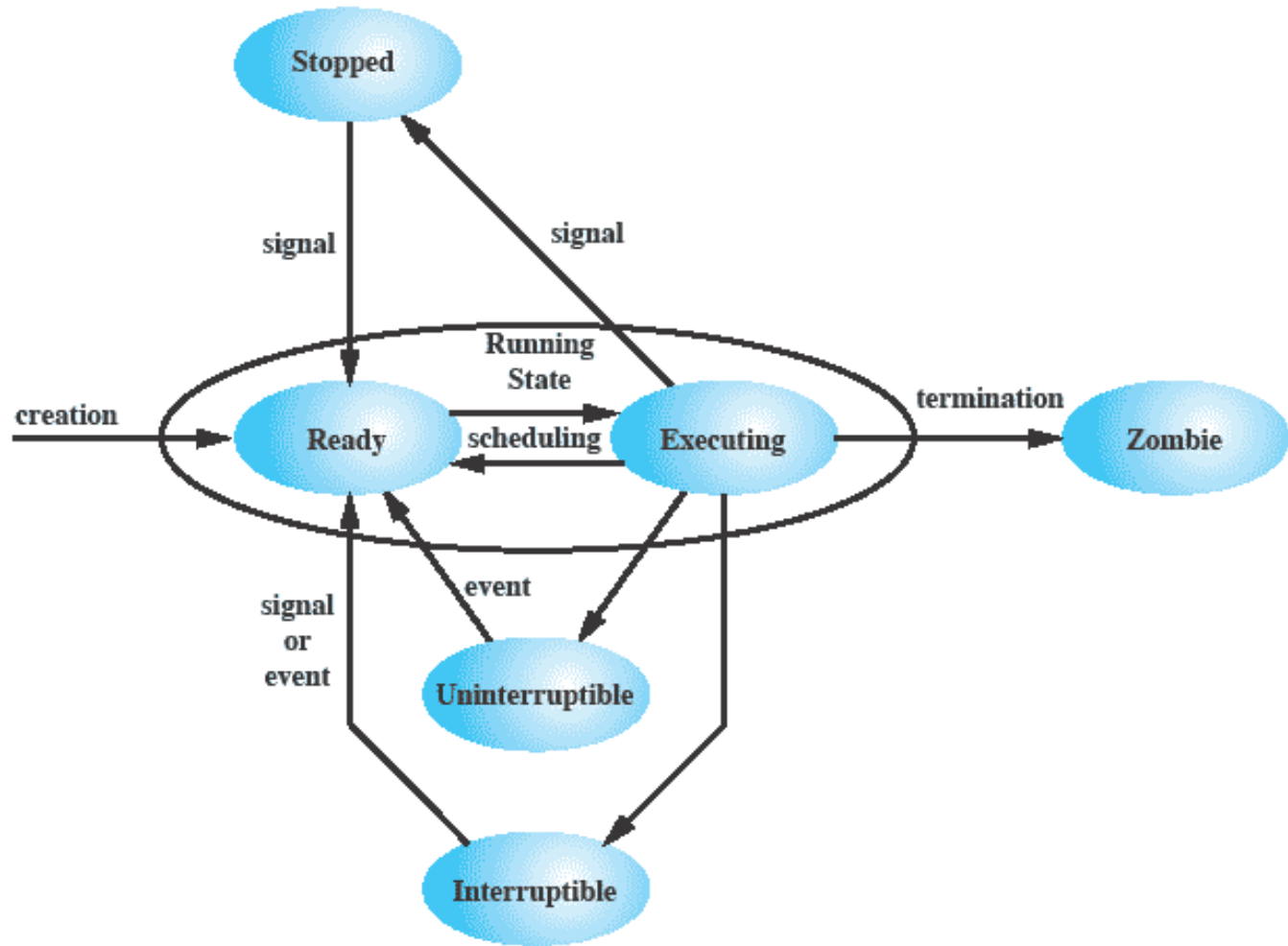


Figure 4.18 Linux Process/Thread Model

Niti



Domaći zadatak

- ✚ Poglavlje **3 Opis procesa i upravljanje**
 - ✚ 4.10 Ključni pojmovi, kontrolna pitanja i problemi

- ✚ Poglavlje **4 Niti**
 - ✚ 4.10 Ključni pojmovi, kontrolna pitanja i problemi