Uvod u distribuirane informacijske sustave

Procesi i niti

Sadržaj

- Procesi
- Niti
- Vrste
- Implementacija
- Višenitni klijenti
- Višenitni serveri
- Koncept virtualizacije

Procesi i niti

- Proces (OS) *= program in execution*
- Nedistribuirani sustavi: Operacijski sustav se brine o zakazivanju procesa *(process scheduling)*
- Distribuirani sustavi: problema ima više
- Više procesa i niti - efikasnost klijent server sustava
- Dozvoljava klijentima i serverima da se realiziraju na način da se komunikacija i lokalna obrada mogu preklapati
- Viša razina izvedbe

Procesi i niti

- Su pristup koji omogućava da se više stvari obavi paralelno
- Omogućavaju da se programi (prividno) izvršavaju simultano
- Na jednoprocesorskim mašinama to je samo privid
- Višeprocesorske mašine – bolja iskorištenost resursa

Zašto trebamo procese i niti

- Problem: blokirajuća izvedba:
  - Program treba obrađivati podatke za vrijeme unosa (spreadsheet)
  - Server treba izvršavati druge klijente dok čeka kraj učitavanja sa diska
- Rješenje:
  - Više procesa
  - Više niti
  - Jedan proces + event driven programiranje

Procesi i Niti

- Procesi – grade distribuirani sustav
- Granulacija koju nude procesi nije dovoljno učinkovita
- Niti nude finiju granulaciju
- Procesi i niti u nedistribuiranom sustavu
- Procesi i niti u distribuiranom sustavu – klijent server

Proces

- Proces je program u izvršenju
- Da bi izvršio program OS kreira niz virtualnih procesora
- OS prati procese pomoću tablice procesa
- Tablica procesa: čuva podatke o CPU registrima, memorijske mape, otvoreni dokumenti, informacije računanja privilegije...
- OS ulaže mnogo napora u osiguravanje da dva procesa ne mogu maliciozno utjecati jedan na drugoga
- Transparentnost konkurentnosti

Stvaranje procesa (OS)

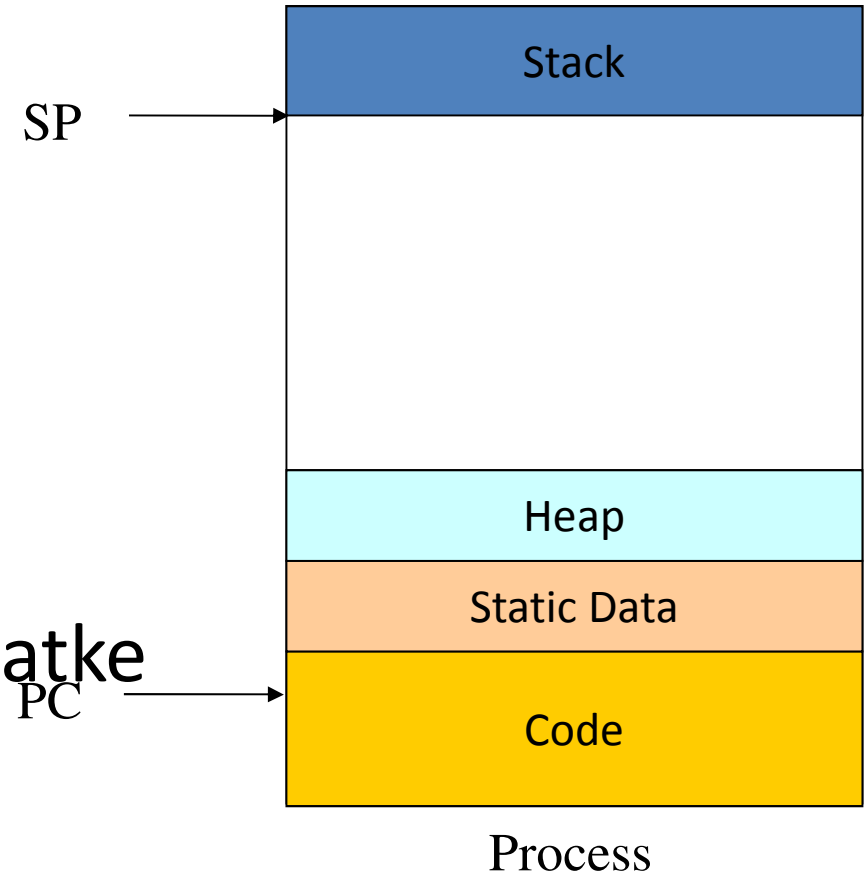
- Kreiranje neovisnog adresnog prostora
- Alokacija
- Kopiranje koda
- Inicijaliziranje stoga

- Prebacivanje procesa znači pohraniti
  - CPU kontekst (registri, program counter, stack pointer)
  - Registre jedinice za upravljanje memorijom (MMU)
  - Poništiti adrese za prevođenje cachea (TLB T*ranslation lookaside buffer*)

- Ako OS podržava više procesa nego što može stati u memoriju - uključiti i hard disk u priču

Proces u memoriji

- Kontekst izvršavanja
  - Program counter (PC)
  - Stack pointer (SP)
  - Data registers
- Izvršni kod
- Podaci
- Stog za privremene podatke

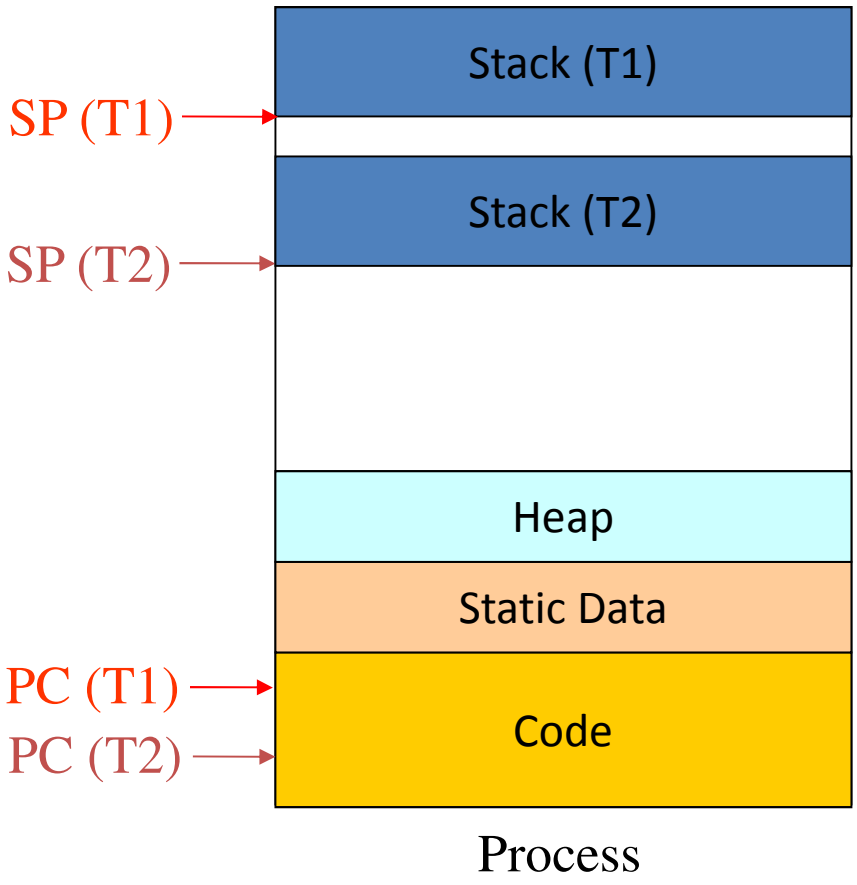


Niti (*Threadovi*)

- Nit neovisno izvršava svoj dio koda
- Ali nije nužno osigurati toliku razinu transparentnosti konkurentnosti
- Samo CPU kontekst i dodatne informacije o niti (mutex varijabla da je nit blokirana)
- Zaštita podataka jedne niti od drugih niti je na odgovornosti programera
- Izvedba višenitnog programa ne smije biti lošija od jednonitne
- Zahtjeva dodatne napore programera oji trebaju osigurati zaštitu niti
- „proper design and keeping things simple” bi trebalo biti dovoljno

Što je Thread?

- Kontekst izvršavanja
  - Program counter (PC)
  - Stack pointer (SP)
  - Data registers

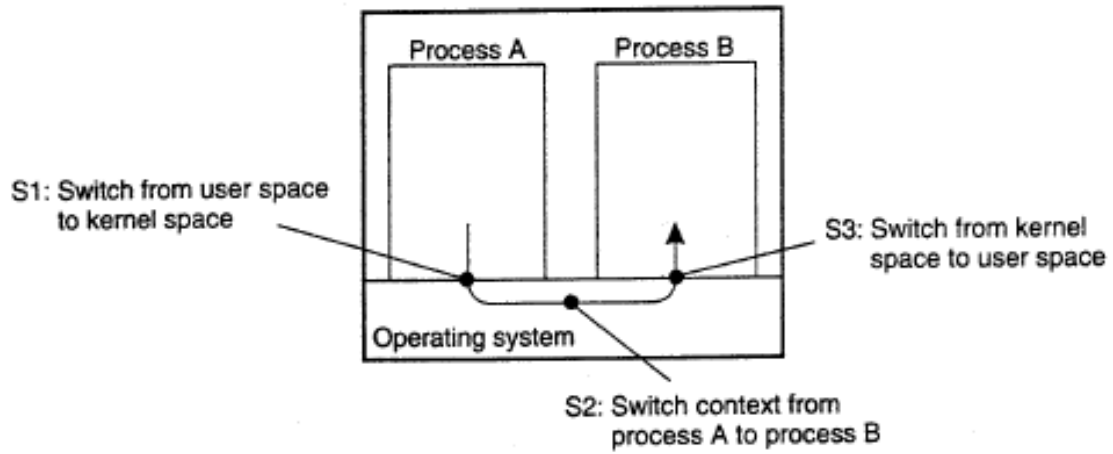


Procesi u nedistribuiranom sustavu

- Velike aplikacije sastavljene od programa koji surađuju, odvojenih procesa
- IPC – *Inter process communication* mehanizam
  - imenovane cijevi
  - redovi poruka
  - dijeljene memorije
- Komunikacija zahtjeva promjenu konteksta

Više-procesne aplikacije

- Komunikacija među procesima zahtijeva intervenciju kernela
- User space <-> kernel space
- Bolji pristup bio bi koristiti niti



- Još jedan razlog za niti: modeliranje aplikacije kao kolekcije surađujućih niti

Višenitne aplikacije

- Granulacija procesa nije dovoljna
- Unutar istog procesa – više niti
- Manji problem prebacivanja konteksta
- Programer vodi računa o izolaciji niti

Niti u nedistribuiranom sustavu

- Npr. proračunske tablice (Excel) – unos treba biti paralelan s obradom
- Unos podatka treba trigerirati proračune
- Imati barem 2 niti:
  - Jedna za unos
  - Jedna za proračun
  - Dodatno – jedna za pohranu podataka...
- Na višeprocesorskim računalima – paralelno izvršavanje niti

Primjer kreiranja niti (Java)

```
final List list;                          // some sort unsorted list of objects
// A Thread class for sorting a List in the background
class Sorter extends Thread {
    List l;
    public Sorter(List l) { this.l = l; }                  // constructor
    public void run() { Collections.sort(l); }             // Thread body
}

// Create a Sorter Thread
Thread sorter new Sorter(list);
// Start running the thread; the new thread starts running the run method above
// while the original thread continues with the next instructions
sorter.start();

System.out.println("I'm the original thread");
```

(Java in a Nutshell, Flanagan)

Primjer kreiranja niti (c)

```
#include <pthread.h>
#include <stdio.h>
void *threadFunc(void *arg) /* This is our thread function.  It is like main(), but for a thread*/
{
        char *str;
        int i = 0;
        str=(char*)arg;
        while(i < 110 ){
        // some useful  code
        }
        return NULL;
}

int main(void){
   pthread_t pth;   //this is our thread identifier
   int i = 0;
  pthread_create(&pth,NULL,threadFunc,"foo");
        while(i < 100){
        //main thread code
        }
printf("main waiting for thread to terminate...\n");
pthread_join(pth,NULL);
return 0;
}
```

Pristup niti globalnoj varijabli

```
Int gIndex; // globalna
    varijabla
//main
While (gIndex<100){
…//citamo vrijednost
}
```

```
//thread
for(i = 0; i < max; i ++) {
...
gIndex = i; //postavljanje
    vrijednosti
...
}
```

Da li je postavljanje integer vrijednosti atomarna operacija?

Varijabla gIndex mora biti deklarirana kao *volatile*

Pristup niti globalnoj varijabli

```
Char* Error ;//globalna var

//Nit1
If (!ok){
Error="Thread 1 error"
}

//Nit2
If(something_wrong){
Error="Thread 2 error"
}
```

- Varijabla ne može istovremeno imati dvije vrijednosti
- Ne možemo se osigurati da se greška u Niti 1 neće dogoditi istovremeno kada i greška u Niti 2

- Treba "zaključati" pristup varijabli

Thread safety

- Koncept programiranja višenitnih aplikacija
- Kod je “Thread safe” ako manipulira dijeljenim resursima na način koji osigurava sigurno izvršavanje svih niti
- Postoji više strategija koje osiguravaju sigurno izvršavanje niti

Semafori

- Varijabla koja na jednostavan način omogućava kontrolu pristupa više procesa ili niti dijeljenom resursu
- Vode računa o broju slobodnih resurasa
- Kada proces zatraži pristup resursu:
  - Ako postoji slobodan resurs: broj slobodnih resursa se umanji za 1
  - Ako su svi resursi zauzeti, stavlja ga u red čekanja

Mutex

- Mutual exclusion
- Varijabla koja ima smisao ključa nekog resursa
- Binarni semafor: da li je resurs slobodan ili zauzet

- Zanimljiv primjer sinkronizacije niti semaforima i mutexima:

- http://www.youtube.com/watch?v=pqO6tKN2lc4

Implementacija niti

- Thread package (posix - pthreads, java.lang.Thread)
- Pristupi implementaciji niti
  - User level thread
    - Thread library
    - Administracija u korisničkom adresnom prostoru
    - Kernel nije svjestan da se proces odvija kao multithread
    - OS ne mora podržavati thread, potrebna je samo thread biblioteka
    - mana: blokirajući pozivi blokiraju sve niti
  - Kernel level thread

- Kernel level thread
  - Kernel se brine o prebacivanju niti
  - Nema blokirajućih poziva
  - Prebacivanje konteksta niti može postati jednako „skupo”kao prebacivanje konteksta procesa
- Hibrid user-level i kernel-level niti: LightWeight process (LWP)
  - Izvršava se u kontekstu procesa,
  - Jedan proces može sadržavati više LWP

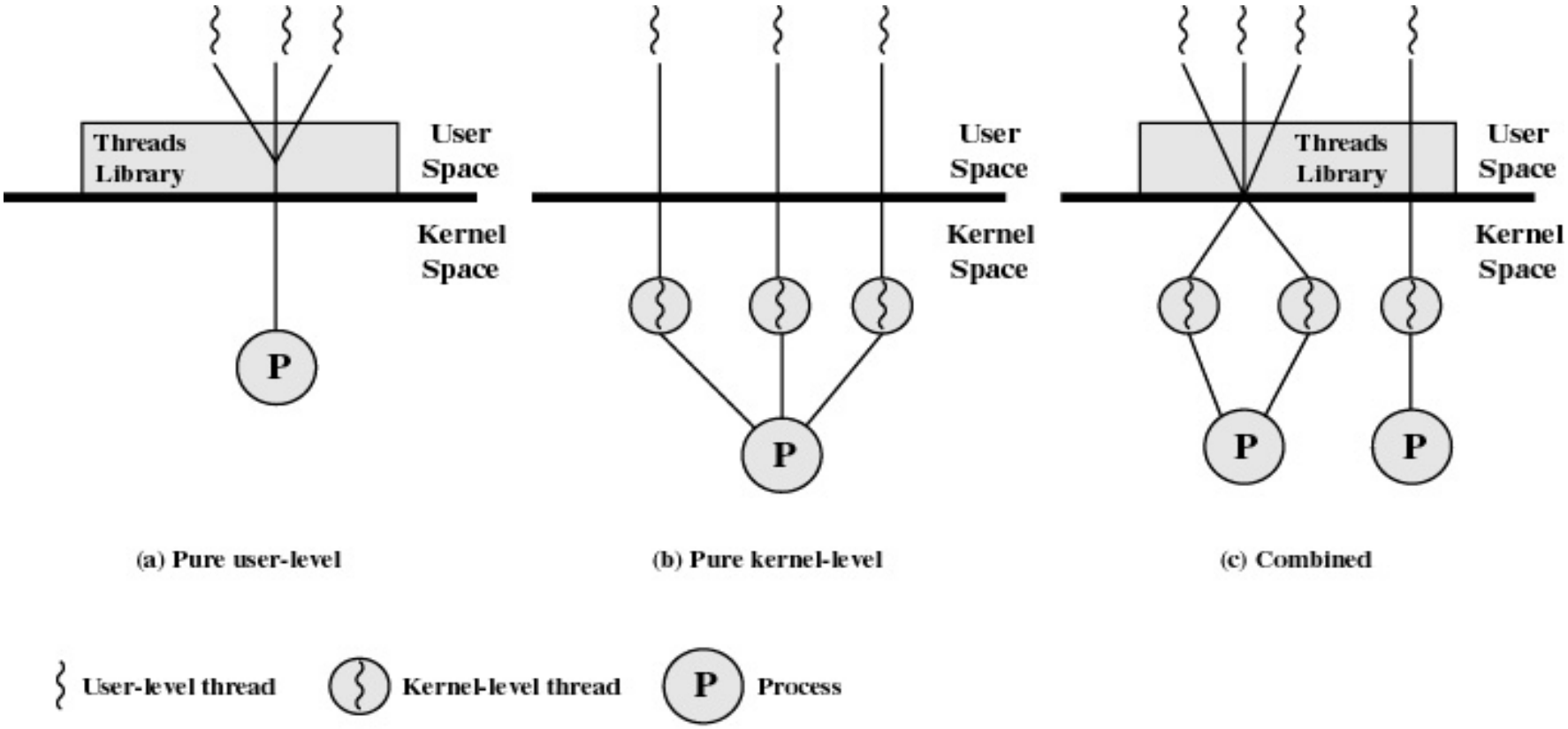
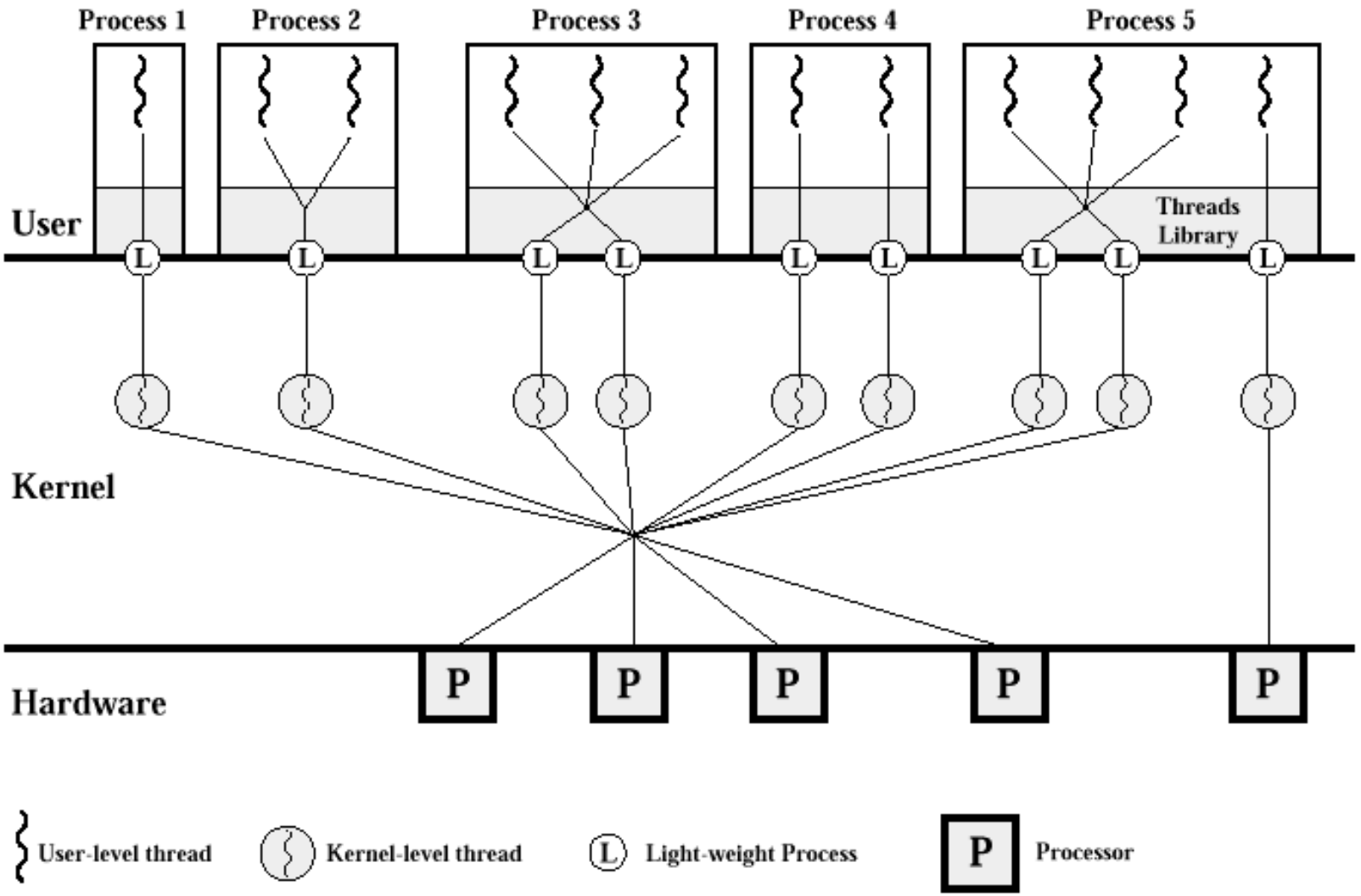


Figure 4.6 User-Level and Kernel-Level Threads


Process 1
Process 2
Process 3
Process 4
Process 5
User
Threads Library
L
Kernel
Hardware
P
User-level thread
Kernel-level thread
Light-weight Process
Processor

Višenitni klijenti

- Način skrivanja čekanja u komunikaciji: započeti komunikaciju i odmah započeti s nečim drugim
- Npr. web browser:
  - uspostavi TCP/IP konekciju, dohvati HTML, dok dohvaća dodatne resurse (slike, ikone, skripte ..) HTML se prikazuje
  - Paralelno može otvoriti više konekcija
  - Web serveri mogu biti replicirani na više adresa – web browser može dohvaćati podatke paralelno - balansiranje opterećenja.

Višenitni serveri

- Glavna upotreba višenitnih servisa – višenitni serveri
- Pojednostavljuje organizaciju koda
- Koristi paralelizam (višeprocesorske mašine)
- File server: poslužuje datoteke
- Dispatcher thread i worker thread
- Dispatcher prima zahtjeve
- Worker dohvaća i poslužuje lokalne datoteke
- Dok se datoteka učitava sa diska, nit je blokirana i druge niti mogu koristiti procesor

Višenitni serveri

- Jednonitni server – čeka dok se učitaju podaci sa diska i većinu vremena je nekoristan
- Ali …
- Event driven programiranje
- Proces može pamtiti stanja zahtjeva umjesto da ih obrađuje (finite state machine).
- Poruka može biti:
  - novi zahtjev: započinje se obrada i zapisuje stanje u tablicu
  - Odgovor od započetog posla: obrađuju se podaci i šalju zahtjevatelju
- Nije model sekvencijalnog procesa (kao kod prva dva primjera)
- Jako učinkoviti: nema promjene konteksta
- Jako teško programirati

Trade-offs

Model	**Characteristics**
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Event driven (Finite state machine)	Parallelism, nonblocking system calls

Virtualizacija

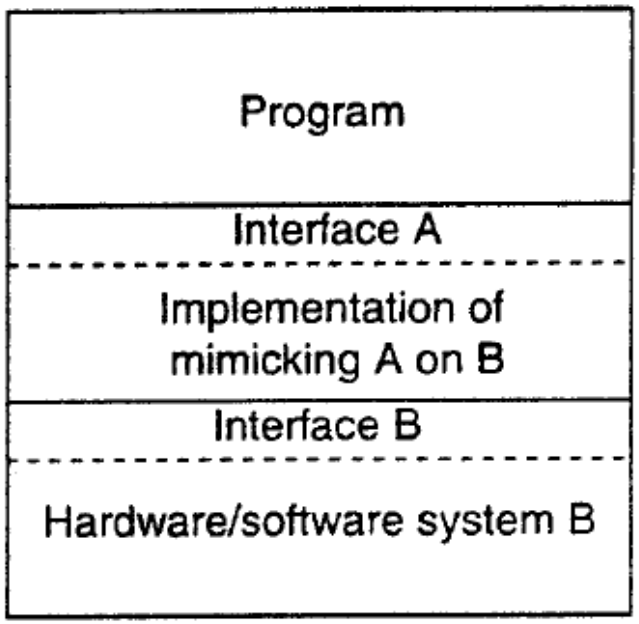
- Niti i procesi daju privid postojanja više resursa (više procesora)
- Ovaj se pristup može proširiti i na ostale resurse – virtualizacija resursa
- Brzi razvoj softwarea i hardwarea – aplikacija gotovo uvijek nadživi operacijski sustav i hardver koji je podržava
- Stvoriti virtualno okruženje na novom sustavu na kojem će takva aplikacija raditi

Virtualizacija

- Virtualizacija je kreiranje virtualne verzije nečega, poput hardwarske platforme (npr. mobitel) operacijskog sustava (cygwin), uređaja za pohranu (virtualni disk) ili mrežnih resursa
  - Olakšava administraciju velikog broja servera ili resursa
  - Potpomaže skalabilnost i bolje iskorištavanje resurasa
  - Pogon iza cloud computing i utility computing
  - Koristi se odavno (npt 70-tih IBM-ova mainframe računala koriste ovu tehniku)

Uloga virtualizacije u ditribuiranom sustavu

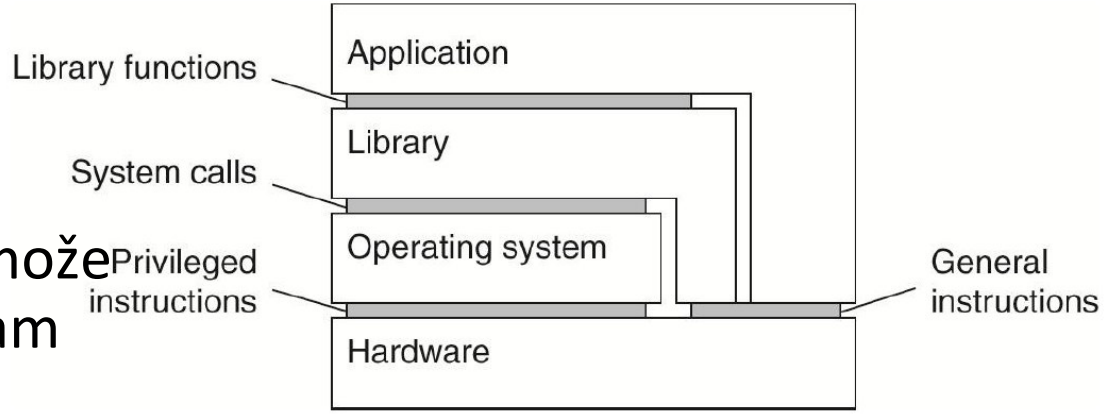
- Virtualizacija se bavi proširivanjem ili zamjenom postojećeg sučelja sa tako da imitira ponašanje nekog drugog sustava



Arhitekture virtualnih mašina

Razine sučelja:

1. Između hardwarea i softwarea koje sadrži strojne naredbe koje može pozvati bilo koji program



1. Između hardwarea i softwarea koje sadrži strojne naredbe koje može pozvati samo privilegirani program (npr. OS)
2. Sučelje koje se sastoji od sistemskih poziva koje nudi operacijski sustav
3. Sučelje koje se sastoji od poziva biblioteka – API (često se i sistemski pozivi skrivaju iza API)

Arhitekture virtualnih mašina

- Virtualizacija se može implementirati na više razina:
  - Process Virtual Machine: abstraktni skup naredbi koji se koristi za izvršavanje aplikacije, npr. Java runtime ili Windows emulator (Wine)
  - *Virtual Machine Monitor*: sloj koji u potpunosti skriva postojeći hardware, ali nudi potpuno novi skup naredbi na istom hardweru kao sučelje. Ovo omogućava imati više operacijskih sustava na jednoj plsatformi npr.: Vmware, VirtualBox, Xen, VirtualPC, Parallels etc.

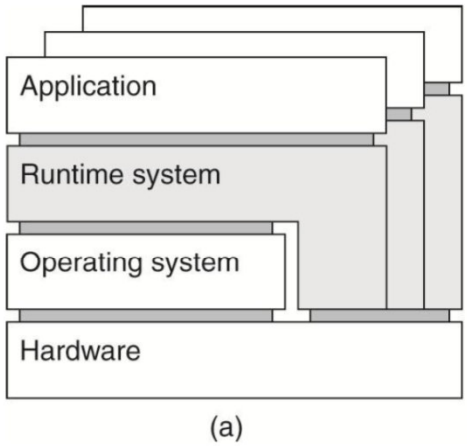


Figure 3-7. (a) A process virtual machine, with multiple instances of (application, runtime) combinations.

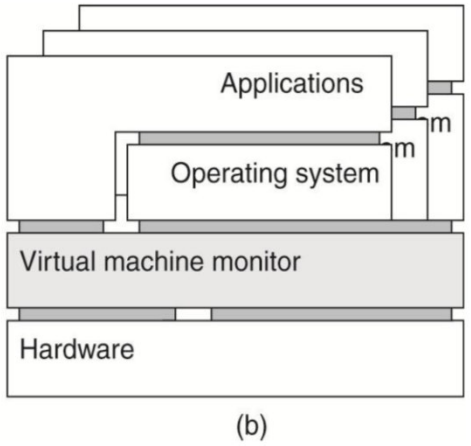


Figure 3-7. (b) A virtual machine monitor, with multiple instances of (applications, operating system) combinations.

Sljedeći put

- Klijenti
- Serveri
- Migracija koda
- Komunikacija