Sistemi za rad u realnom vremenu

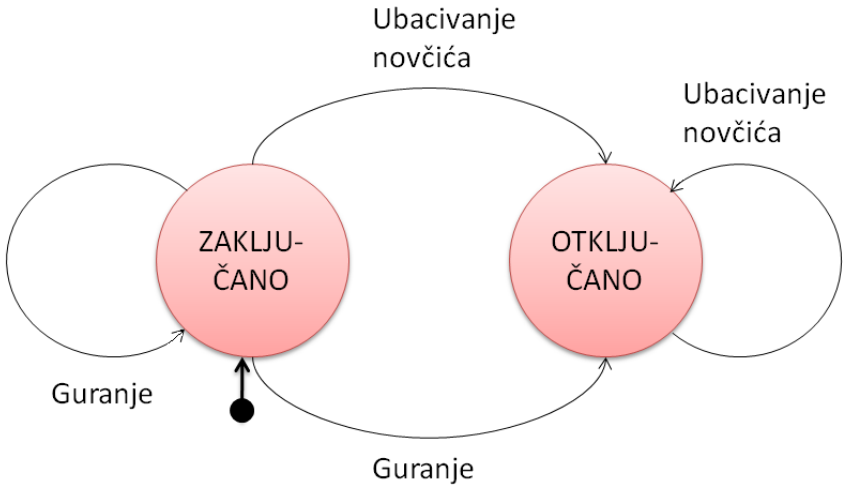
Predmetni profesor : doc. dr. Marko Tanasković
Predmetni asistent : doc. dr. Marko Tanasković
E-mail: mtanaskovic@singidunum.ac.rs

Konačni automat - definicija

- Konačni automat ili Finite State Machine često se naziva samo State Machine (automat ili mašina stanja)
- Matematički model izračunavanja koji se koristi za konstrukciju programa ili sekvencijalne logike
- Abstraktna mašina koja se može naći u tačno jednom od konačnog broja stanja
- Stanje automata se menja usled nekog događaja (prekid) ili ukoliko je određeni uslov ispunjen
- Konačni automat je definisan listom svih stanja, početnim stanjem i uslovima za prelazak sa jednog stanja na drugo

Konačni automat – jednostavan primer

Kružna rampa koja se otključava ubacivanjem novčića



Jednostavna mašina sa dva stanja

Konačni automat – jednostavan primer

Kružna rampa koja se otključava ubacivanjem novčića

Trenutno stanje	Događaj	Novo stanje	Aktivnost
Zaključano	Ubačen novčić	Otključano	Rampa se drži zaključana
	Guranje	Zaključano	
Otključano	Ubačen novčić	Otključano	Rampa se drži otključana
	Guranje	Zaključano	

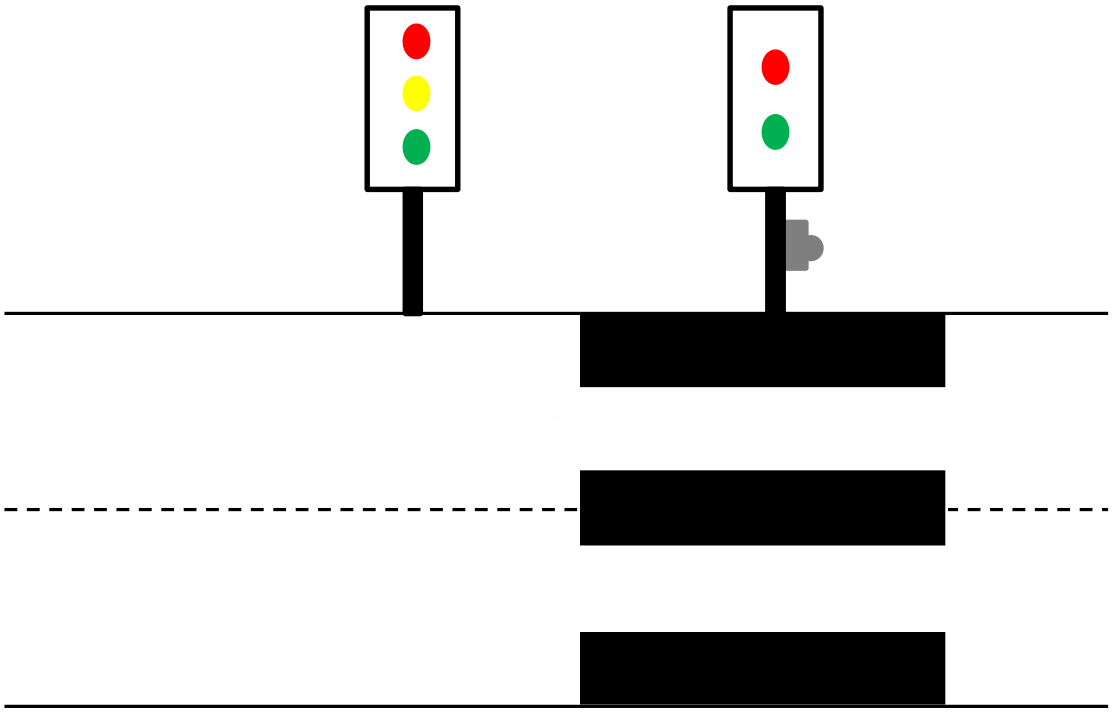
Konačni automat – primeri primene

- Primere primene konačnih automata srećemo u mnogim sistemima za rad u realnom vremenu

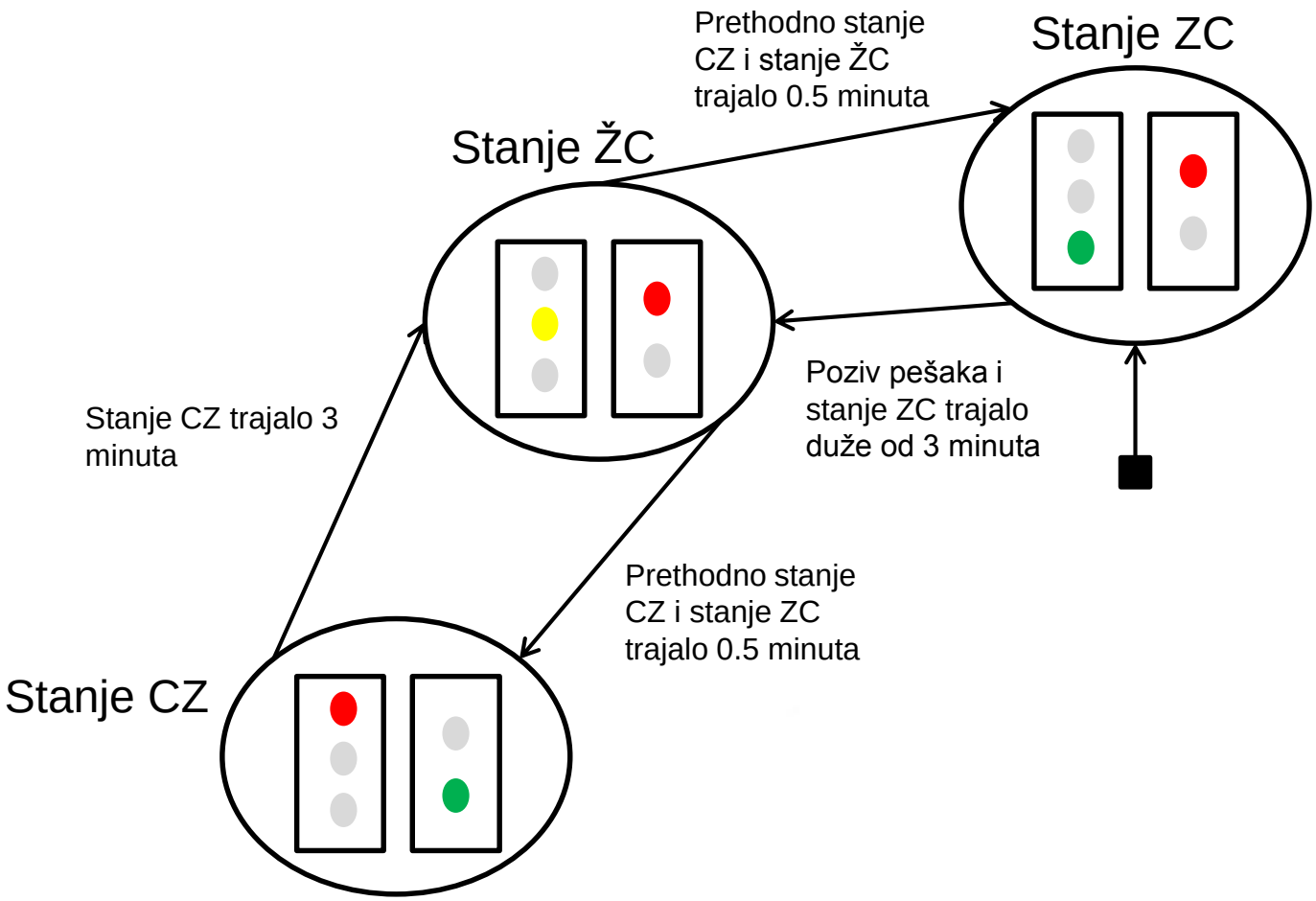
  - automati za prodaju
  - sistem za naplatu parkinga i drugi sistemi sa rampama
  - liftovi
  - semafori
  - elektronske brave sa sigurnosnim kodom
  - komunikacijski protokoli

Konačni automat – primer semafora

Napisati program za kontrolu semafora. Kada nema pešaka treba da bude zeleno za vozila i crveno za pešake. Po pritisku pešaka na taster pali se zuto za vozila, a zatim i crveno za vozila i zeleno za pešake. To traje 3 minuta i onda se pali crveno za pešake i žuto pa zeleno za vozila. Sledeći poziv pešaka biće uslišen tek pošto je zeleno svetlo za vozila trajalo duže od 3 minuta

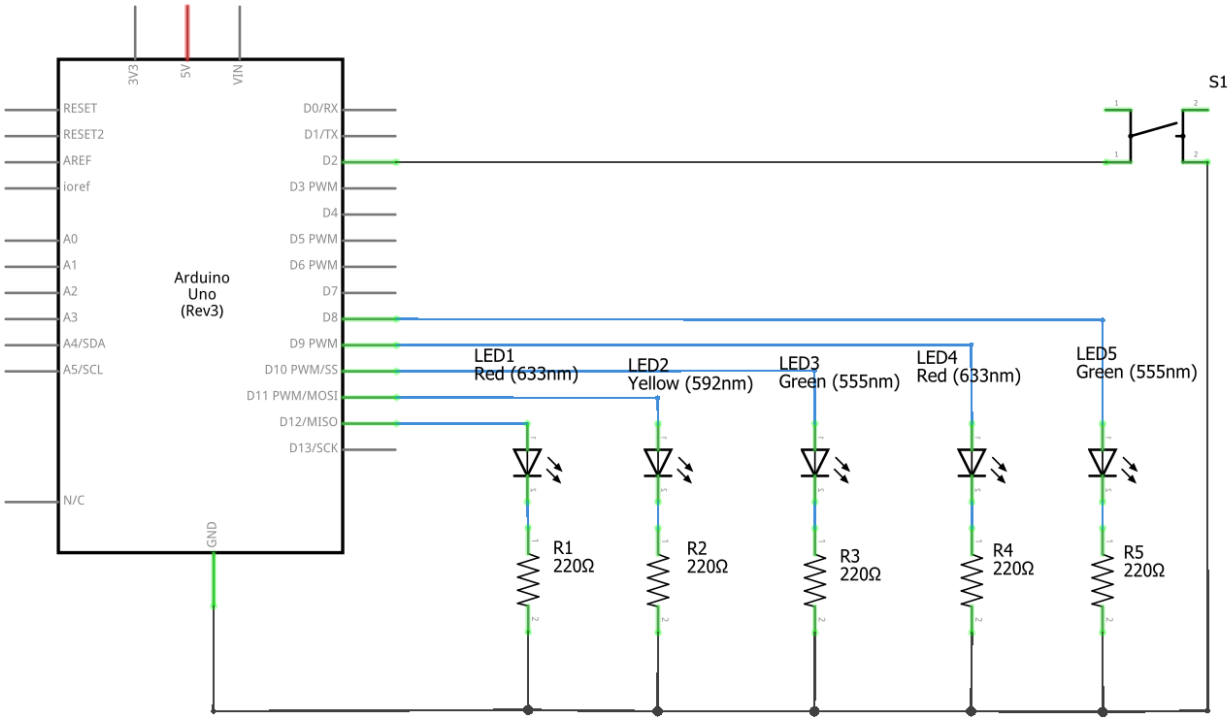


Konačni automat – primer semafora



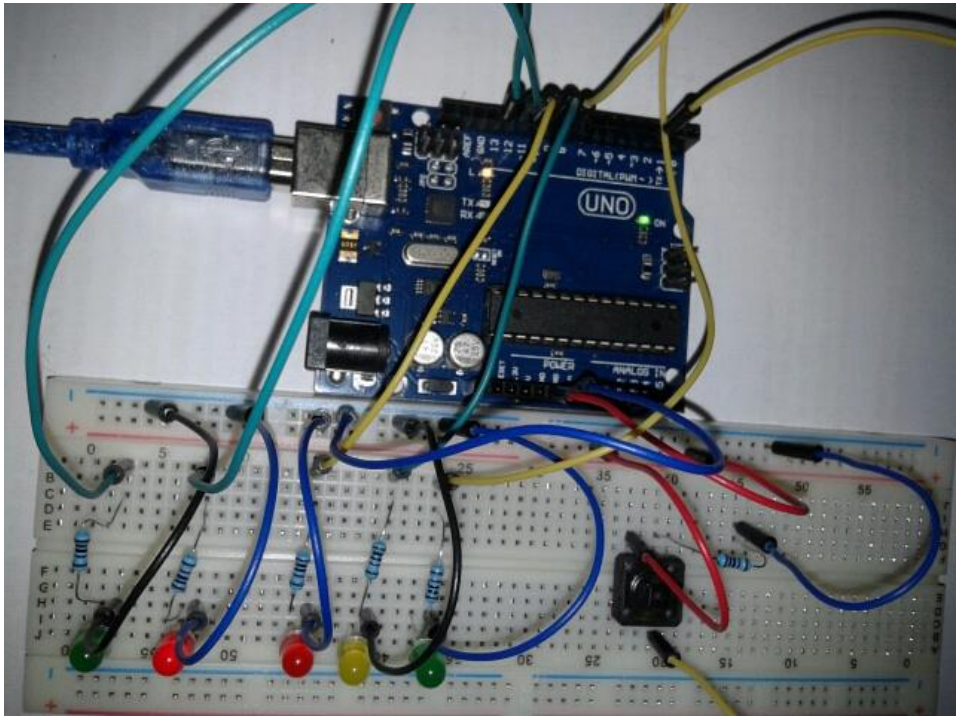
Konačni automat – primer semafora

Realizacija semafora uz pomoć Arduino platforme



Konačni automat – primer semafora

Realizacija semafora uz pomoć Arduino platforme



Konačni automat – primer semafora

Realizacija semafora uz pomoć Arduino platforme

```
#include "TimerOne.h"
const byte Pesaci_crveno = 13; // Izbrani pin za crveno svetlo za pesake
const byte Pesaci_zeleno = 12;  // Izbrani pin za zeleno svetlo za pesake
const byte Auta_crveno = 11; // Izbrani pin za crveno svetlo za automobile
const byte Auta_zuto = 10; // Izbrani pin za zuto svetlo za automobile
const byte Auta_zeleno = 9; // Izbrani pin za zeleno svetlo za automobile
const byte Poziv_pesaka = 2; // Izabrani pin za taster

const int PESACI_CRVENO_AUTA_ZELENO  = 1;
// 1 predstavlja vrednost koja oznacava ovo stanje
const int PESACI_CRVENO_AUTA_ZUTO = 2;
// 2 predstavlja vrednost koja oznacava ovo stanje
const int PESACI_ZELENO_AUTA_CRVENO = 3;
// 3 predstavlja vrednost koja oznacava ovo stanje
const int PERIOD_ZUTOG_SVETLA = 20;
// Period trajanja zutog svetla
const int PERIOD_ZELENOG_SVETLA = 80;
// Period trajanja zelenog svetla

volatile int Stanje = PESACI_CRVENO_AUTA_ZELENO;
// Promenljiva koja oznacava trenutno stanje
volatile int Sledece_stanje = PESACI_ZELENO_AUTA_CRVENO;
// Promenljiva koja oznacava sledece stanje
volatile int Brojac = 0;
// Promenljiva koja predstavlja vremenski brojac
volatile byte Zahtev_za_promenu = 0;
// Promenljiva koja predstavlja zahtev za promenu
```

Konačni automat – primer semafora

Realizacija semafora uz pomoć Arduino platforme

```
void Pusti_pesake()
{
  if((Stanje == PESACI_CRVENO_AUTA_ZELENO) ||
     (Stanje == PESACI_CRVENO_AUTA_ZUTO
      && Sledece_stanje == PESACI_CRVENO_AUTA_ZELENO))
  // Ukoliko je ukljuceno crveno za pesake i zeleno za vozila
  // ili ako je ukljuceno zuto,
  // a sledece stanje je crveno za vozila, a zeleno za pesake
  {
  Zahtev_za_promenu = 1; // Zahtev za promenu menja vrednost
  }
}
```

Konačni automat – primer semafora

Realizacija semafora uz pomoć Arduino platforme

```
void Podesavanje_vremena()
{
  if(Brojac < 800) // Proverava se vrednost brojaca
  {
    Brojac +=1; // Vrednost brojaca se povecava za jedan
  }
  if(Stanje == PESACI_CRVENO_AUTA_ZELENO
     && Zahtev_za_promenu && Brojac > PERIOD_ZELENOG_SVETLA)
  // Ukoliko je ukljuceno crveno svetlo za pesake,
  // i ukoliko je istekao period trajanja zelenog svetla
  // i ukoliko postoji zahtev za promenu
  {
    Brojac = 0; // Brojac dobija vrednost nula
    Zahtev_za_promenu = 0; // Zahtev za promenu dobija vrednost nula
    Stanje = PESACI_CRVENO_AUTA_ZUTO; // Pali se zuto za automobile
    Sledece_stanje = PESACI_ZELENO_AUTA_CRVENO;
    // Pali se zeleno za pesake a crveno za automobile
  }
  if(Stanje == PESACI_CRVENO_AUTA_ZUTO && Brojac > PERIOD_ZUTOG_SVETLA)
  // Ukoliko je ukljuceno zuto svetlo
  // i ukoliko je period trajanja zutog svetla isktekao
  {
    Brojac = 0; // Brojac dobija vrednost nula
    Stanje = Sledece_stanje; // Stanje postaje sledece stanje
  }
  if(Stanje == PESACI_ZELENO_AUTA_CRVENO && Brojac > PERIOD_ZELENOG_SVETLA)
  // Ukoliko je ukljuceno zeleno za pesake
  // i ukoliko je prosao period trajanja zelenog svetla
  {
  Brojac = 0; // Brojac dobija vrednost nula
  Stanje = PESACI_CRVENO_AUTA_ZUTO;
  // Pali se crveno pesacima, vozilima zuto svetlo
  Sledece_stanje = PESACI_CRVENO_AUTA_ZELENO;
  // Sledece stanje je pesacima crveno, a vozilima zeleno
  }
}
```

Konačni automat – primer semafora

Realizacija semafora uz pomoć Arduino platforme

```
void setup()
{
  pinMode(Pesaci_crveno, OUTPUT); //Pin diode je konfigurisan kao izlazni pin
  pinMode(Pesaci_zeleno, OUTPUT); //Pin diode je konfigurisan kao izlazni pin
  pinMode(Auta_crveno, OUTPUT); //Pin diode je konfigurisan kao izlazni pin
  pinMode(Auta_zuto, OUTPUT); //Pin diode je konfigurisan kao izlazni pin
  pinMode(Auta_zeleno, OUTPUT); //Pin diode je konfigurisan kao izlazni pin

  pinMode(Poziv_pesaka, INPUT_PULLUP); // Pin tastera je konfigurisan kao ulazni pin
  attachInterrupt(digitalPinToInterrupt(Poziv_pesaka), Pusti_pesake, FALLING);
  // Ova komanda povezuje funkciju Pusti_pesake sa prekidom
  // koji generiše silazna ivica na pinu 2 za koji je vezan taster
  Timer1.initialize(100000);
  // Inicijalizacija Timer1 i podesavanje perioda trajanja prekida na jednu sekundu
  Timer1.attachInterrupt(Podesavanje_vremena);
  // Funkciju podesavanje_vremena postavlja za funkciju prekida nakon isteka prekida
}
```

Konačni automat – primer semafora

Realizacija semafora uz pomoć Arduino platforme

```
void loop()
{
  // Na osnovu toga koje je trenutno stanje odgovarajuce diode se pale/gase
  switch(Stanje)
  {
    case PESACI_CRVENO_AUTA_ZELENO:
      digitalWrite(Pesaci_crveno, HIGH);
      digitalWrite(Pesaci_zeleno, LOW);
      digitalWrite(Auta_crveno, LOW);
      digitalWrite(Auta_zuto, LOW);
      digitalWrite(Auta_zeleno, HIGH);
    break;
    case PESACI_CRVENO_AUTA_ZUTO:
      digitalWrite(Pesaci_crveno, HIGH);
      digitalWrite(Pesaci_zeleno, LOW);
      digitalWrite(Auta_crveno, LOW);
      digitalWrite(Auta_zuto, HIGH);
      digitalWrite(Auta_zeleno, LOW);
      break;
     case PESACI_ZELENO_AUTA_CRVENO:
       digitalWrite(Pesaci_crveno, LOW);
       digitalWrite(Pesaci_zeleno, HIGH);
       digitalWrite(Auta_crveno, HIGH);
       digitalWrite(Auta_zuto, LOW);
       digitalWrite(Auta_zeleno, LOW);
     break;
   }
 }
```

Podela poslova na programske zadatke

- Rad sa prekidima vođenim vremenom i događajima može postati suviše kompleksan ukoliko postoji veliki broj prekida
- U takvoj situaciji potrebno je nezavisne upravljačke poslove podeliti na niz programskih zadataka
- Programski zadaci treba da se izvršavaju paralelno zbog bolje iskorišćenosti hardverskih resursa

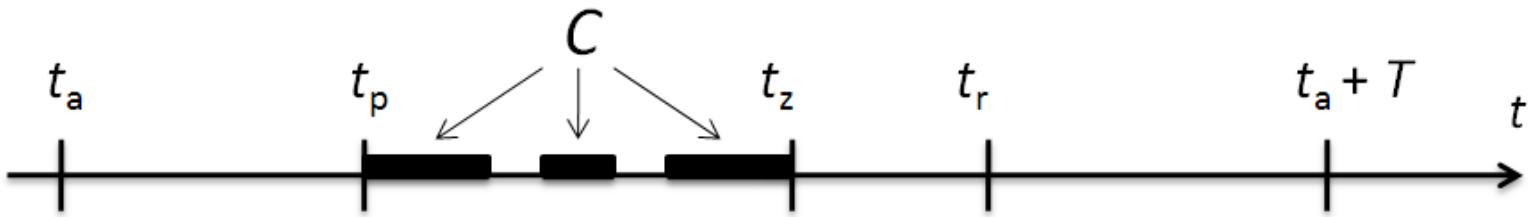
  (kada postoji jedan procesor – kvaziparalelno izvršavanje)
- Problem raspoređivanja zadataka (scheduling) i komunikacije i sinhronizacije zadataka

Raspoređivanje zadataka (scheduling)

- Svaki programski zadatak se aktivira nekim događajem u sistemu
  - Periodični zadaci se izvršavaju sa tačno određenom periodom (aktiviraju se uz pomoć signala sa tajmera)
  - Aperiodični zadaci se izvršavaju kada dođe zahtev iz okoline (aktiviraju se uz pomoć signala sa digitalnih ili analognih ulaza)
- Programski zadatak se obično ne može iszvršiti odmah po prijemu zahteva – on ulazi u red čekanja
- Algoritam za raspoređivanje (scheduling) naizmenično dodeljuje vreme zadacima koji čekaju po prioritetima

Raspoređivanje zadataka (scheduling)

- Karakteristični trenutci i vremenski intervali pri izvršavanju jednog programskog zadatka



t_a – Trenutak aktivacije zadatka
t_p – Trenutak početka izvršavanja
t_z – Trenutak završetka izvršavanja
C– Trajanje zadatka (vreme u kome mu je dodeljen procesor)
t_r – Trenutak do koga zadatak mora da se izvrši (rok izvršenja)
T – Period ponavljana aktivacije (ako je periodičan)

Raspoređivanje zadataka (scheduling)

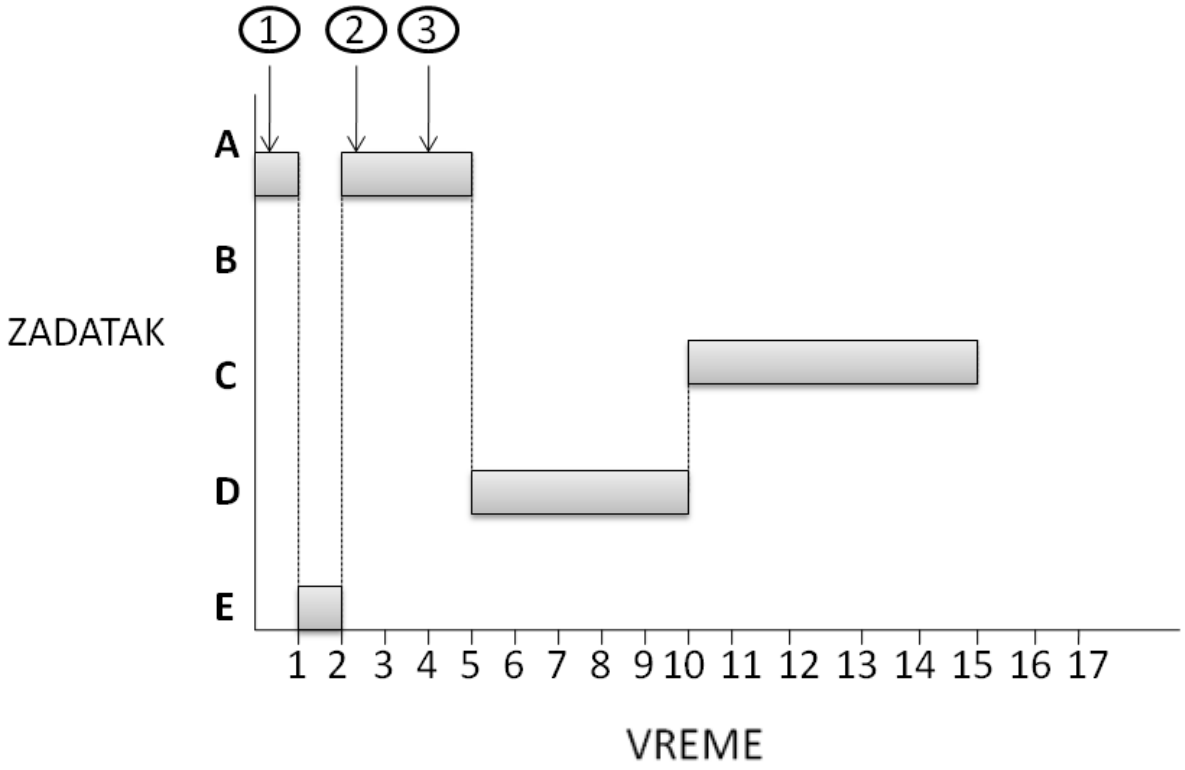
- Izvršavanje zadatka ne počinje sa trenutkom njegove aktivacije
- Sve dok se ne izvrši, zadatak se nalazi na listi zadataka koji čekaju izvršenje
- Od trenutka kada započne izvršenje do trenutka kada se završi, izvršavanje može biti prekidano radi izvršavanja nekih drugih programskih zadataka sa većim prioritetom
- Po izvršenju, programski zadatak se briše sa liste čekanja sve do njegove ponovne aktivacije nekim događajem

Podela algoritama za raspoređivanje

- Prema tome da li se dozvoljava prekidanje izvršavanja zadataka
  - preemptive (izvršavanje zadataka može biti prekidano)
  - non-preemptive (izvršavanje zadataka se ne prekida)

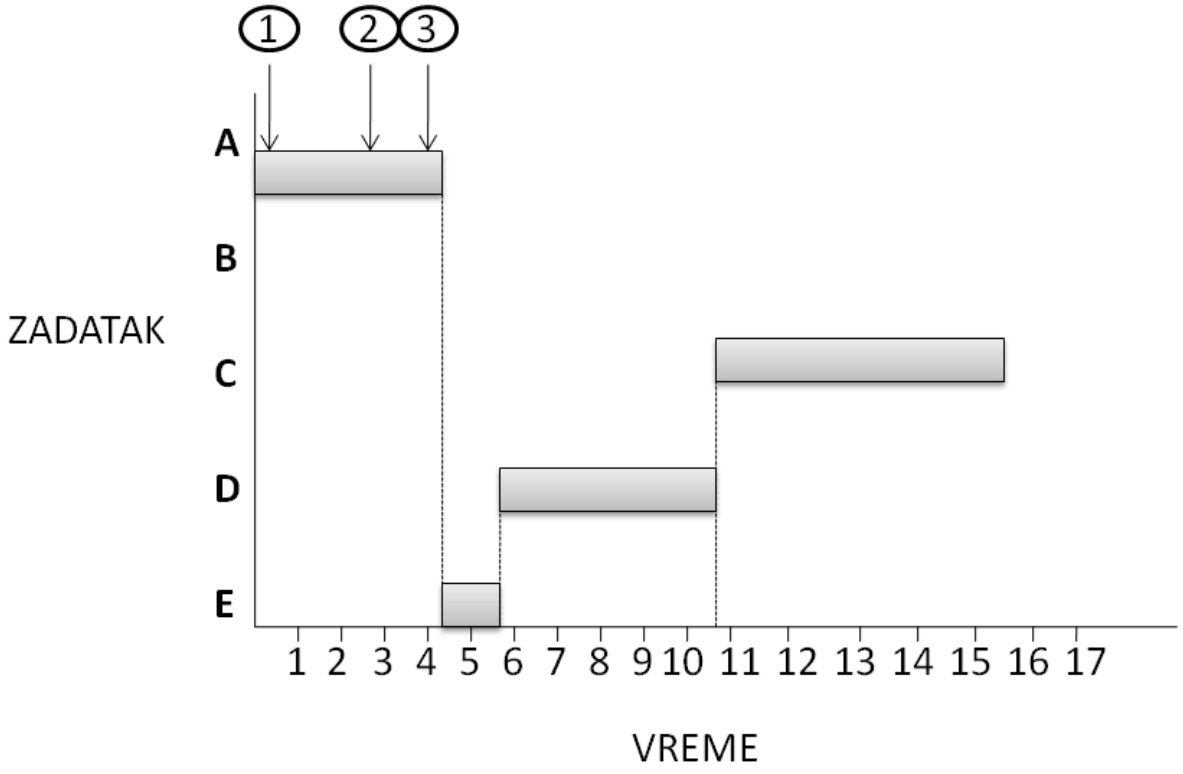
- Prema tome da li se odluke o raspoređivanju donose dinamički ili ne
  - statički (odluke su bazirane na fiksnim parametrima koji se dodeljuju zadacima pre izvršavanja)
  - dinamički (odluke su bazirane na dinamičkim parametrima koji se u toku izvršenja programa mogu menjati)

Preemptive raspoređivanje



1. Zahtev za izvršenjem zadatka E
2. Zahtev za izvršenjem zadatka C
3. Zahtev za izvršenjem zadatka D

Non-preemptive raspoređivanje



1. Zahtev za izvršenjem zadatka E
2. Zahtev za izvršenjem zadatka C
3. Zahtev za izvršenjem zadatka D

Statičko raspoređivanje

- U fazi dizajniranja softvera svakom zadatku se dodeljuje stepen prioriteta
- U toku izvršenja uvek se prvo izvršava zadatak u redu čekanja sa najvećim prioritetom
- Ukoliko postoji više zadataka sa istim prioritetom, između njih se izbor vrši po nekom drugom kriterijumu (npr. može se prvo izvršavati onaj koji je u redu čekanja duže vremena

Dodela prioriteta zadacima

- Raspoređivanje mora biti takvo da procesor stiže da izvrši sve programske zadatke u datim vremenskim rokovima

- Uslov za N periodičnih programskih zadataka

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

- Pri dodeli prioriteta mora se voditi računa da ovaj uslov uvek bude zadovoljen

Dodela prioriteta zadacima

- Mogući metod dodele prioriteta koji ispunjava uslov je dodela prioriteta prema očekivanoj učestalosti aktiviranja zadataka
- Zadaci koji treba češće da se izvršavaju dobijaju veći prioritet
- Mogući su i izuzetci od ovog pravila kod zadataka čije je izvršavanje jako važno za funkcionisanje sistema
- Prioriteti se mogu dodeljivati i prema trajanju izvršenja, tako što bi kraći zadaci imali veći prioritet ili se može koristiti kombinacija trajanja i učestalosti za dodelu prioriteta

Dinamičko raspoređivanje

- Prioriteti zadataka se menjaju dinamički u toku izvršenja programa

- Najčešće se kriterijumi definišu tako da se ostvari ispunjenje vremenskih rokova datih za izvršenje svakog od programskih zadataka

- Najčešće korišćen metod daje najveći prioritet zadatku iz liste čekanja sa najranijim trenutkom do koga zadatak mora da se izvrši.

(za izvršavanje se uvek bira zadatak koji najpre mora da se završi)

Raspoređivanje vremenski nekritičnih zadataka

- Postoje različita načela za raspoređivanje zadataka koji nisu vremenski kritični:

  - pravedno raspoređivanje vremena svim zadacima
  - favorizacija kraćih zadataka (cilj izvršiti što više zadataka)
  - tako da vreme čekanja zadataka bude što manje
  - tako da se iskorišćenost procesora maksimizuje

Raspoređivanje vremenski nekritičnih zadataka je relevantno kod računarskih sistema koji ne rade u realnom vremenu

Raspoređivanje u praksi

- U praksi se obično koristi kombinacija različitih principa
- Ne koriste se teoretski metodi raspoređivanja, već heuristički (oni koji su oprobani u praksi)
- Raspoređivanje vrši operativni sistem
- Upoznaćemo se kasnije sa raspoređivanjem kod operativnog sistema Linux

Sinhronizacija i komunikacija među zadacima

- Potrebno je da programski zadaci budu sinhronizovani i da komuniciraju jedni sa drugima
- **Sinhronizacija:** Vođenje računa o pravilima prvenstva i drugim odnosima među zadacima (npr. akcija jednog zadatka može se izvršiti tek po izvršenju neke druge akcije)
- **Komunikacija:** razmena informacija između procesa. Najčešće uz pomoć deljenih promenljivih i slanja poruka

Deljenje promenljivih

- Deljenje promenljivih među zadacima bez ograničenja može dovesti do problema usled istovremenog modifikovanja promenljive

- Primer: dva zadatka koja dele promenljivu i oba povećavaju njenu vrednost za jedan.
  - Ova komanda sadrži 3 mašinske operacije
    1.) Prepisati vrednost promenljive u neki registar
    2.) Uvećati vrednost u registru za 1
    3.) Prepisati vrednost iz registra u memorijsku lokaciju

- Prekidanje jednog od procesa između dve od ove tri mašinske komande i izvršenje identičnog koda u drugom programskom zadatku dovodi do pojave greške (rezultat će biti pogrešan)

Izbegavanje konflikata pri deljenju promenljivih

- Delovi zadataka koji pristupaju deljenim promenljivama moraju se izvršavati **bez mogućnosti da budu prekinuti**
- Ovi delovi zadataka nazivaju se **kritični delovi**
- Ovakav vid zaštite pri deljenju promenljivih se naziva **uzajamno isključivanje**

Semafori - definicija

- Konflikt u pristupu deljenoj promenljivoj se izbegava uz pomoć celobrojne promenljive čija se vrednost može menajti uz pomoć dve procedure Wait () i Signal ()

- Wait (x) smanjuje vrednost x za 1 ukoliko je ona veća od 0, inače čeka da vrednost postane veća od 0 i onda je smanjuje

- Signal (x) povećava vrednost promenljive x za 1

- Procedure Wait () i Signal () su atomske (ne mogu biti prekinute). Dva zadatka koja pozivaju proceduru Wait () za istu promenljivu ne mogu doći u konflikt pri izvršenju ove procedure.

Semafori - primer

- Semafori se mogu koristiti za zaštitu kritičnih delova koda u procesima koji dele promenljive

Zadatak 1:

Set komandi A
Wait (x)
Komande sa deljenom promenljivom
Signal (x)
Set komandi B

Zadatak 2:

Set komandi C
Wait (x)
Komande sa deljenom promenljivom
Signal (x)
Set komandi D

- Prvi zadatak koji izvrši komandu Wait (x) će blokirati izvršenje kritičnog koda drugog zadatka sve dok se ne završi izvršavanje njegovog kritičnog koda
- *x* mora biti inicijalizovano na 1

Uslovna sinhronizacija

- Uslovna sinhronizacija je neophodna kada proces želi da izvrši akciju koja može biti razumno ili bezbedno izvršena samo ukoliko je neki drugi proces prethodno već izvršio neku drugu akciju

- Primer: bafer
  - Zadatak dodavanja podataka u bafer ne može se izvršiti ukoliko je bafer već pun
  - Zadatak čitanja iz bafera ne može se izvršiti ukoliko je bafer prazan

Uslovna sinhronizacija preko semafora

- Semafori se takođe mogu koristiti i za uslovnu sinhronizaciju zadataka

Zadatak 1:

Set komandi A
Wait (x)
Komande koje se smeju izvršiti tek pošto zadatak 1 izvrši komande B

Zadatak 2:

Set komandi B
Signal (x)
Set komandi D

- Kritični deo zadatka 1 će biti blokiran sve dok se ne izvrše komande u zadatku 2 koje su preduslov za izvršavanje kritičnog dela
- *x* mora biti inicijalizovano na 0

Mane semafora

- U nekim slučajevima može doći do deadlocka (zaključavanje zadataka)
- Deadlock se dešavakada kada dva zadatka drže po jedan resurs, a svaki od njih čeka da onaj drugi oslobodi svoj resurs. U tom slučaju oba zadatka ostaju zauvek blokirana

Zadatak 1:

Set komandi A
Wait (x)
Wait (y)
Set komandi B

Zadatak 2:

Set komandi C
Wait (y)
Wait (x)
Set komandi D

Monitori

- Kod monitora se kritični delovi koda pišu u okviru procedura koje ne mogu biti prekidane
- Deljenim promenljivama može se pristupiti samo uz pomoć tih procedura koje se ne mogu prekidati
- Koriste se više nego semafori, mada se i semafori još uvek koriste u jednostavnijim softverskim projektima.