VEŽBA 12

Pregled

Digitalni računari (tj. njihova centralna procesorska jedinica – CPU) komuniciraju sa svojim okruženjem šaljući (upisujući) i primajući (čitajući) binarne podatke iz drugih digitalnih sistema. Uređaji sa kojima se obavlja ovakva komunikacija radi prikazivanja rezultata rada i prijema podataka radi obrade nazivaju se uređaji za ulaz/izlaz – U/I (eng. input/output – I/O). Kod složenijih sistema postoji poseban sistem potprograma koji preuzima na sebe ovu komunikaciju i obezbeđuje korisničkim programima konzistentan način pristupa U/I sistemu. Uobičajeni naziv za ovaj sistem potprograma je *drajver* (engl. driver). Dakle svaki tip uređaja sa kojim računarski sistem poseduje svoj drajver.

Još od pojave digitalnih računara jasno je da poseduju velike kapacitete obrade podataka, naročito u svakodnevnim zadacima gde je priliv novih podataka za obradu relativno spor. U takvim uslovima jedan CPU može da obavlja veći broj zadataka (obrada podataka, nadzor, upravljanje nekim uređajem) istovremeno. Naravno, ne doslovno istovremeno, nego sa stanovišta efekta istovremeno. Pri tome CPU jedan kratak interval – kvant – vremena posvećuje svakom zadatku, tj. onim zadacima koji u datom trenutku zahtevaju neku vrstu obrade. Savremeni sistemi sa više procesorskih jezgara (engl. multi-core) u stanju su i da doslovno obavljaju više zadataka istovremeno značajno podižući performanse sistema.

Istovremeno obavljanje više različitih zadataka na računaru uopšteno se naziva multitaskingom (engl. *multi-tasking*). Program pokrenut na računaru (od strane korisnika ili operativnog sistema) uobičajeno se naziva *proces*. Pošto uobičajeno istovremeno može da bude pokrenuto više procesa, ova varijanta multitaskinga naziva se multiprocesigom (engl. *multi-processing*). I sam proces može da sadrži više paralelnih niti izvršavanja i takva vrsta unutar-procesnog multi-tasking-a naziva se multitredingom (engl. *multi-threading*). Same paralelne niti zovu se tredovi (engl. *thread*).

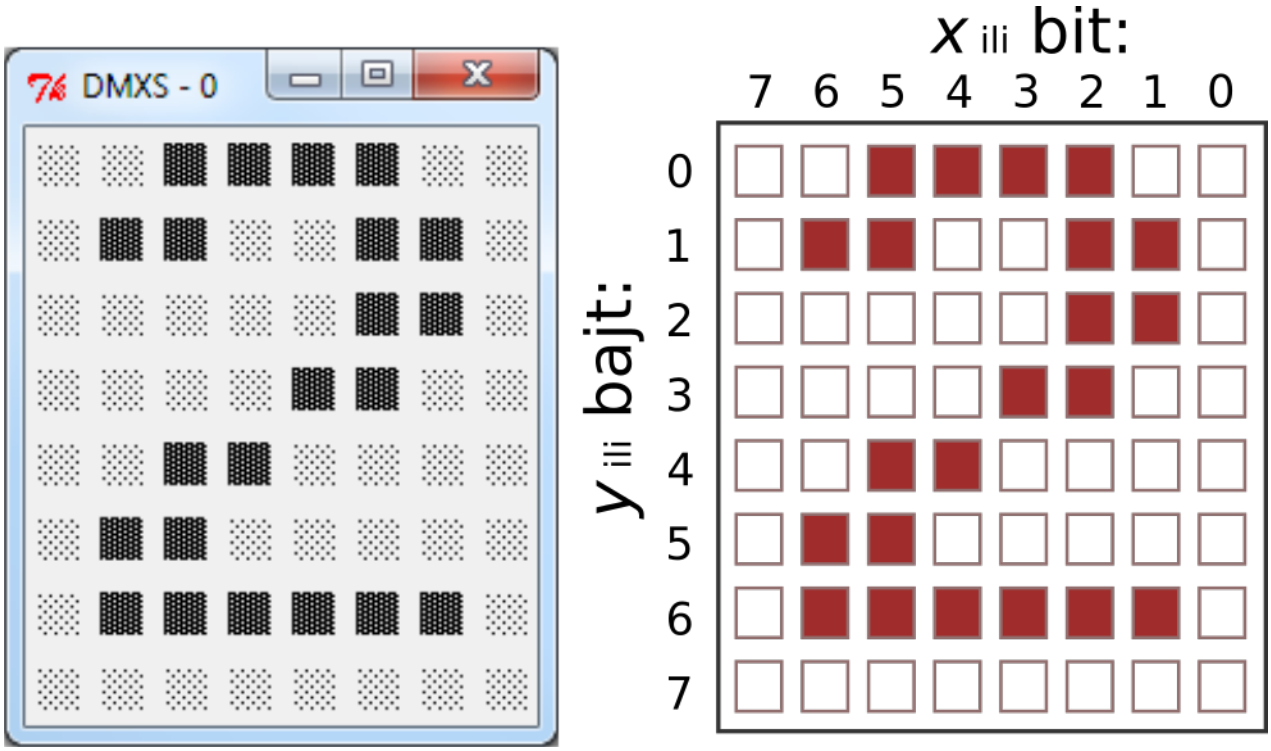
Matrični displej

Panel koji sadrži matricu ($m \times n$) elemenata od kojih svaki može po potrebe da menja svoje vizuelne karakteristike radi prikazivanja slike može se nazvati matričnim displejom. Elementi mogu i odavati svetlost (svetleti), a naprednije verzije mogu čak da odaju i svetlost raznih boja.

U ovoj vežbi koristi se simulator matričnog displeja dimenzija 8×8 . Istovremeno je moguće prikazati i kontrolisati do 10 nezavisnih displeja. Za upravljanje potrebnim brojem displeja koristi se drajver u vidu Python modula. Onako kako se pojavljuje na ekranu i šematski, simulirani matrični displej prikazan je na sledećoj slici 1.

Simulirani matrični displej funkcioniše na sledeći način: informacija prikazana na displeju fizički se skladišti u 8 odvojenih 8-bitnih registara numerisanih od 0 do 8. Numeracija bita i registara prikazana je na slici 1. Kao ilustracija dat je jedan mogući dizajn broja 2 za prikazivanje na displeju. Pristup registrima može se obaviti preko odgovarajućeg drajverskog modula sadržanog u fajlu `DMXctrl.py`[1]. Kratko uputstvo za korišćenje simulatora i drajvera dato je u dodatku ove vežbe.

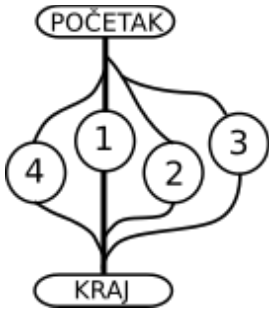
1 Ovaj fajl se može skinuti sa Internet stranice sa pripremama za ove vežbe.



Slika 1 – Simulator matričnog displeja - DMXS

Niti izvršavanja programa – tredovi

Korisnički programi po pravilu se pokreću kao jedna jedina nit izvršavanja (tred) kao što je ilustrovano na slici 2. Ta jedna nit je zadužena za formiranje novih. U nekom trenutku od te jedne niti odvojiće se sledeća, potom sledeća i taj proces se nastavlja u skladu sa zadatkom koji program obavlja. Nova nit se može odvojiti i od novonastale niti (nit 3 se odvojila od niti 2 na slici 2). Program ne može dostići svoj kraj sve dok sve njegove niti ne budu završile sa izvršavanjem, a to se šematski predstavlja kao ponovno udruživanje (engl. *join*) sa drugim nitima (na slici 1 sa osnovnom niti – 1).



Slika 2 – Ilustracija niti izvršavanja programa

Jedan program se obično razdvaja u veći broj tredova zarad obavljanja zajedničkog zadatka, što znači da će svaki tred obavljati samo deo ukupnog predviđenog posla. Ipak, pošto je zadatak jedinstven, neophodna je komunikacije između tredova. Na primer, ako jedan tred koristi rezultat rada drugog, mora se obezbediti da se rezultat ne preuzme pre nego što bude gotov. Primer otkriva jednu suštinsku osobinu multitredinga, a to je da su niti nezavisne u svom izvršavanju i ne može se ništa unapred pretpostaviti o tome kada će koji tred obavljata koji zadatak. Za njihovu saradnju,

neophodna je stroga sinhronizacija koja se ne dešava automatski, nego programer treba da vodi računa o tome. Najprostije rečeno, sinhronizacija obuhvata dve stvari:

- Mogućnost da jedan tred obavesti drugi o svom napretku,
- Mogućnost da jedan tred obustavi svoje izvršavanje dok ga drugi tred ne obavesti o ispunjenosti određenih uslova.

Ipak postoji više poznatih vrsta sinhronizacije – semafor (*semaphore*), događaj (`event`), blokada (`lock`), mutex (*mutual exclusion*), kritična sekcija (*critical section*), randevu (*rendez-vous*) i drugi. U nastavku će biti obrađeni samo događaj i blokada.

Događaj (event)

Događaj je jedinstveni objekat kojeg dele svi tredovi koji treba da se sihnronizuju pomoću njega. Tred može da:

- *Postavi* ili *obriše* događaj;
- Čeka da se događaj *desi*. Događaj se desi kada se njegovo stanje promeni iz obrisanog u postavljeno stanje. Tred je potpuno obustavljen (ne izvršava se i ne troši vreme CPU) za vreme čekanja događaja. Čekanju se može (ali nije obavezno) dodeliti i vreme isteka (engl. *timeout*). Na događaj se čeka najmanje zadato vreme isteka, nakon toga se izvršavanje treda nastavlja.

Blokada (lock)

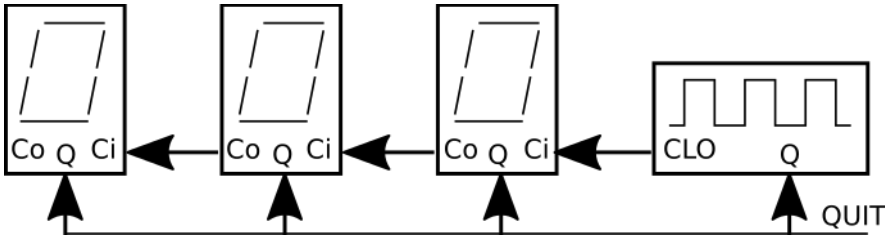
Kao i događaj, blokada je jedinstveni objekat kojeg deli svi tredovi koje treba sinhronizovati. Namena blokade nije obaveštavanje o određenom stanju, nego sprečavanje da više tredova započne istu operaciju – najčešće pristup deljenom resursu, npr. ako više tredova pristupa istom displeju radi izmene sadržaja. Blokada se može postaviti (engl. *acquire the lock*) ili ukloniti (engl. *release the lock*). Tred koji prvi postavi blokadu nastavlja izvršavanje, ostali koji pokušaju postavljanje već postavljene blokade biće obustavljeni. Jedan od obustavljenih tredova koji je pokušao da postavi blokadu moći će da nastavi sa radom kada prethodni tred blokadu ukloni. Nije poznato unapred koji od obustavljenih tredova će prvi nastaviti sa radom nakon uklanjanja blokade.

Zadaci

1. Pokrenuti jedan displej element i napisati program koji uključuje sve elemente displeja, a potom sve elemente gasi. Između svaka dva pristupa programirati pauzu od 1/3 sekunde da bi se proces vizuelno lakše pratio.
2. Date su definicije brojeva od 0 do 9 kao n-torka n-torki u fajlu `numbers.txt`[2]. Nedostaje definicija borja 2. Definisati nedostajući broj tako da odgovara obliku prikazanom na slici 1. Korišćenjem drajverskog modula napisati funkciju za ispis proizvoljne cifre na displeju i iskoristiti ga za pisanje programa koji broji 3 puta od 0 do 9 sa vremenskim intervalom od 1/3 sekunde između brojeva.
3. Napraviti trocifreni brojač korišćenjem tredova. Za ispisivanje cifara koristiti ugrađene funkcije za ispis cifre iz drajverskog modula. Brojač treba da broji sa periodom od 0,3 s. Za rešavanje zadatka koristiti tredove: po jedan tred za svaku cifru i još jedan za generisanje takt signala. Za međusobnu komunikaciju tredova koristiti događaje. Za prekid rada svih tredova

2 Ovaj fajl se može skinuti sa Internet stranice sa pripremama za ove vežbe.

koristiti poseban događaj koji šalje glavni tred. Nakon slanja događaja za kraj, glavni tred treba da sačeka završetak rada svih pokrenutih tredova pre nego što se i sam završi. Kao pomoć koristiti šemu datu na slici 3. Pravougaonike je najbolje implementirati kao objekte izvedene iz klase `threading.Thread`. Strelice predstavljaju događaje – i izvor i primaoca. Cifra najmanje težine menja stanje na svaki takt, a sledeća teža svaki put kada se prethodna menja iz cifre 9 u cifru 0. Tred prethodne cifre generiše događaj narednoj kao znak za promenu prikaza cifre.

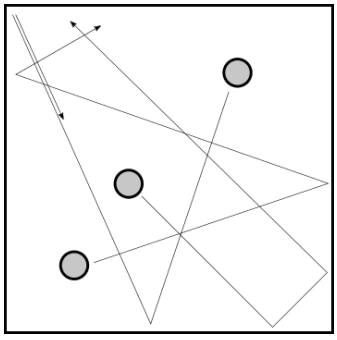


Slika 3

4. Napisati program koji simulira kretanje i odbijanje 3 nezavisne loptice od ivica displeja. Jedan element displeja predstavlja jednu lopticu. Interval vremena između dva uzastopna položaja loptice treba da bude 0,5 s. Svaka loptica kreće iz proizvoljne, ali unapred definisane pozicije i nakon toga se kreće duž dijagonalnih putanja sa fiksnim, unapred zadatim, proizvoljnim vertikalnim i horizontalnim brzinama. Ako indeks i predstavlja trenutnu poziciju, a $i+1$ narednu poziciju loptice, onda važi:

$x_{i+1} = x_i + v_x$,
$y_{i+1} = y_i + v_y$.

v_x i v_y su redom horizontalna i vertikalna brzina loptice. Prilikom udara u zid, odgovarajuća brzina menja znak (npr. udar u gornju ili donju ivicu menja znak v_y . Šematski prikaz kretanja loptica dat je na slici 4.


Slika 4

Za kontrolu pojedinačnih elemenata displeja koristiti ugrađene funkcije iz drajverskog modula. Pošto svi tredovi pristupaju istom displeju mora se osigurati da ne dođe do istovremenog pristupa više tredova. Za obezbeđivanje isključivog pristupa koristiti mehanizam lock-ovanja. Svaka jedna loptica treba da bude kontrolisana posebnim tredom, a dodatni tred treba da generiše događaje za (istovremeno) pomeranje svih loptica. Završetak programa (svih tredova) treba uraditi isto kao u prethodnom zadatku.

Dodaci

Aktiviranje simulatora matričnog displeja

Ako je fajl DMX.py učinjen izvršnim, displej se aktivira pozivanjem tog fajla kao programa:

```
./DMX.py [disp]
```

Ako nije izvršni i ne postoji mogućnost da se učini izvršnim, može se pozvati eksplicitnim pozivom Ppython runtime sistema:

```
python DMX.py [disp]
```

Opcioni parametar *disp* određuje identifikacioni broj displeja i može biti u opsegu od 0 do 9. Na osnovu tog identifikacionog broja drajver pristupa određenom displeju. Kao što se može zaključiti, istovremeno ih može biti deset. Ako se parametar ne navede, podrazumeva se vrednost 0.

Drajver matričnog displeja

Funkcije drajvera matričnog displeja sadržane su u modulu *DMXctrl* su sledeće:

- *AllLEDsOn(disp)* – Uključuje sve elemente displeja br. *disp*;
- *AllLEDsOff(disp)* – Isključuje sve elemente displeja br. *disp*;
- *SetPixel(disp, x, y)* – Uključuje element sa koordinatama *(x, y)* displeja br. *disp*. Nulte koordinate su u gornjem levom uglu;
- *ClearPixel(disp, x, y)* – Isključuje element sa koordinatama *(x, y)* displeja br. *disp*. Nulte koordinate su u gornjem levom uglu;
- *WriteLEDRow(disp, r, v)* – Postavlja *r*-ti red ($r \in [0,9]$) u stanje određeno 8-bitnim podatkom *v* na displeju br. *disp*;
- *SetDigit(disp, n)* – ovo jeIspisuje decimalnu cifru *n* ($n \in [0,9]$) na displeju br *disp*. Izgled cifara definisan je u samom simulatoru displeja;
- *AdvanceDigit(disp)* – Uvećava cifru prikazanu na displeju br. *disp* za jedan;
- *BackDigit(disp)* – Umanjuje cifru prikazanu na displeju br. *disp* za jedan.

Vremenske zadrške nevezane za sinronizaciju među tredovima

Standardni Python modul time sadrži funkciju sleep koja služi za obustavljanje izvršavanja Python programa na tačno određeno vreme:

```
time.sleep(time)
```

Ovako pozvana funkcija napraviće pauzu u izvršavanju programa tačno *time* sekundi. Vreme može biti zadato kao ceo broj ili broj u pokretnom zarezu.

Multitreding u Python-u

Ključna za multritreding u Python-u je klasa *Thread* iz modula *threading*. Uobičjeni postupak je definisanje klase koja nasleđuje klasu *Thread*. Toj klasi se dodaje konstruktor koji treba da obavi pozivanje konstruktora klase roditelja i obavi potrebne dodatne inicijalizacije – najčešće zadavanje elemenata članova i kreiranje potrebnih objekata za sinhronizaciju. Drugi ključni element

jeste metoda *run* – ta funkcija će biti funkcionalni deo treda. Tred počinje sa izvršavanjem na početku te funkcije, a završava se povratkom iz nje – dostizanjem kraja funkcije ili naredbom *return*. Tred se ne pokreće automatski, nego neki od već pokrenutih tredova (najčešće je ov glavni program) treba da kreira objekat pomenute klase i pozove njegovu metodu *start*.

Ceo program će završiti sa radom kada završe sa radom svi njegovi tredovi. Iako nije obavezno, dobra praksa je da jedan od tredova sačeka završetak rada ostalih tredova – njihovo spajanje (engl. *join*). Na taj način tred koji je sačekao završetak rada ostalih tredova može ima priliku da obavi potrebne završne operacije. Čekanje završetka rada nekog treda obavlja se pozivanjem metode *join* odgovarajućeg objekta klase *Thread*.

Sledi primer (kostur, bez funkcionalnosti) programa sa dva dodatna treda:

```
import threading

class UserThread(threading.Thread):
    def __init__(self, user_par):
        threading.__init__(self)
        self par = user_par # parametri treda

    def run(self):
        # ovde treba napisati blok koda
        # koji ce obavljati posao u
        # okviru treda

# glavni program:
# kreiranje tred objekata
th1 = UserThread(100)
th2 = UserThread(200)

# pokretanje tredova
th1.run()
th2.run()

# blok glavnog programa - glavni tred
# i glavni tred moze da obavlja nekakav posao

# na ovom mestu glavni tred ceka zavrsetak rada svih tredova
th1.join()
th2.join()
```

Događaji u Python-u

Modul threading sadrži klasu *Event*. Svaki objekat ove klase može da služi kao sinhronizacioni objekat tipa događaj. Svi tredovi koji treba da se sinhronizuju putem jednog događaja treba da se obraćaju tom jednom objektu pozivanjem njegovih metoda. Konstruktor objekta ne zahteva nikakve parametre, a novi objekat će biti u neaktivnom (obrisanom) stanju. Na raspolaganju su sledeće metode:

- *set* – postavlja događaj u aktivno stanje. Samom tranzicijom iz obrisanog u postavljeno stanje svi tredovi koji su na događaj čekali biće odblokirani (čak i u slučaju da do trenutka

aktivacije treda događaj već bude promenjen u obrisano stanje);

- *clear* – postavlja događaj u neaktivno (obrisano) stanje;
- *isSet* – vraća trenutno stanje događaja;
- *wait([timeout])* – ova metoda se odmah vraća (i ne obustavlja tred) ako je događaj postavljen. Ako je u događaj neaktivan, tred se obustavlja sve dok ne dođe do promene događaja u aktivno stanje ili vreme zadato opcionim parametrom *timeout* ne istekne. Vreme se zadaje u sekundama i može biti i ceo broj i broj u pokretnom zarezu. Ako se vreme ne zada, čeka se bez vremenskog ograničenja.

Sledeći primer prikazuje tred koji radi u beskonačnoj petlji i u svakom prolazu pređe u stanje čekanja na događaj. Da bi se obezbedilo da događaj bude sigurno obrisan pre sledećeg prolaska kroz petlju, odmah nakon čekanja događaj se briše. Ovo neće uticati na ostale tredove koji možda čekaju na događaj jer je zapravo prelazak iz obrisanog u postavljeno stanje ono što ponovo pokreće obustavljene tredove.

```
while True:
  evt.wait()
  evt.clear()
  do_work_1()  # do kraja petlje
  do_work_2()  # nastavlja se uobicajeni rad
```

Neki događaji ne služe tome da ikada budu postavljeni. Čisto vremensko čekanje se takođe može postići navođenjem argumenta metode *wait*. Ako događaj nikad nijedan tred ne postavlja, rezultat će biti vremensko čekanje u trajanju ne kraćem od navedenog:

```
while True:
  evt.wait(2.0)  # cekanje od 2 sekunde
  do_time_work_1()  # posao do kraja petlje izvrsavace
  do_time_work_2()  # se svake 2 sekunde
```

Blokade u Python-u

Python klasa pomoću koje se ostvaruju blokade je klasa *Lock* iz modula *threading*. Da bi se više tredova shinronizovalo pomoću ovakve blokade svi treba da pristupaju istom objektu klase *Lock*. Konstruktor objekta ne zahteva parametre, a nova blokada se kreira u otključanom stanju. Na raspolaganju su sledeće metode:

- *acquire([blocking])* – ova metoda pokušava da zaključa blokadu. Uspeva i vraća *True* ako blokada nije već zaključana. Ako parametar *blocking* ima vrednost *True* (podrazumeva se ako se ne navede), tred koji je metodu pozvao će biti obustavljen ako je blokada zaključana sve dok tu blokadu neki drugi tred ne otključa. Po otključavanju blokade funkcija vraća vrednost True, a tred nastavlja sa radom. Ako je pak vrednost parametra *False*, metoda će odmah vratiti vrednost *False* ako je blokada zaključana (nema obustavljanja treda), a ako nije vraća *True*;
- *release()* – ova metoda otključava blokadu. Ukoliko se pozove, a blokada nije zaključana prijavljuje se greška.

Blokadama se pristupa neposredno pre i posle blokava programa koji pristupaju deljenim resursima da bi se obezbedilo da tredovi ne pristupe određenom resursu konkurentno (istovremeno u smislu multitredinga). Konkurentno pristupanje može, a ne mora izazvati grešku, ali se svakako izbegava.

Sledi primer:

```
# deo jednog treda
self.XLock.acquire()
DMXctrl.WriteLEDRow(0, 2, 0xF3)
self.XLock.release()
```

Primedba o mestu deaktiviranja događaja (event-a)

Veći broj tredova može biti obustavljeno u stanju čekanja na događaj. Kada se događaj će se desiti onog trenutka kada na nekom mestu bude pozvana metoda `set`. Svi tredovi koji čekaju (u obustavljenom stanju) ovim stiču uslov za nastavak rada i nastaviće rad prvom prilikom. Važno je istaći da će oni nastaviti rad čak i ako se pozove metoda `clear` istog događaja. Sama tranzicija iz neaktivnog u aktivno stanje događaja je ono što će omogućiti nastavak rada svih tredova. To znači da nakon metode `set`, odmah može da se pozove i metoda `clear`, a svi tredovi koji su čekali taj događaj obustavljeni stiču uslov za nastavak rada.

Ipak treba biti oprezan: metoda `wait` koja obustavlja tredove u stanju čekanja se odmah vraća, tj. ne obustavlja tred ako je događaj u aktiviranom stanju. Ako je namera da obustavljeni tred uradi neki zadatak i ponovo pređe u stanje čekanja treba obezbediti da do sledećeg poziva metode `wait` događaj bude deaktiviran. Ako događaj deaktivira isti tred koji ga je i aktivirao neophodno je da on ne bude prekinut pre nego što će zaista uspeti da ga deaktivira, a to je bez posebnih mera nije moguće garantovati. U realnoj situaciji može se desiti da tred koji je pokrenut nakon aktiviranja događaja sledeći put pozove wait za događaj pre nego što on bude deaktiviran.

Zaključak: ako se događaj koristi za iniciranje neke jednokratne aktivnosti (triger) kod nekog treda (ili čak više njih), najbolje je deaktiviranje događaja prepustiti istom tom tredu – odmah nakon izlaska iz obustavljenog stanja. Ovako će svaki tred koji je bio u stanju čekanja (obustavljen) jednom deaktivirati događaj, ali to višestruko deaktiviranje događaja neće sprečiti aktiviranje svakog treda koji je na događaj čekao. S druge strane, događaj će sigurno biti deaktiviran pre sledećeg poziva `wait`.