# Lamportov postupak međusobnog isključivanja

Uz pomoć Lamportovog algoritma moguće je sinkronizirati dvije ili više dretvi, odnosno dva ili više procesa koji primjerice mogu izgledati ovako:

```
proces proc(i){          /* i  [0..n] */
   ponavljaj{
      uđi u kritični odsječak;
      K.O.;
      izađi iz kritičnog odsječka;
      N.K.O.;
   }dok je zadovoljen(uvjet);
}
```

## Lamportov algoritam:

```
zajedničke varijable: ULAZ[0..n-1], BROJ[0..n-1]

funkcija uđi_u_kritični_odsječak(i)
{
   ULAZ[i] = 1;
   BROJ[i] = max(BROJ[0],...,BROJ[n-1]) + 1;
   ULAZ[i] = 0;

   za j = 0 do n-1 čini
      dok je ULAZ[j] <> 0 čini
         ništa;
      dok je BROJ[j] <> 0 && (BROJ[j] < BROJ[i] || (BROJ[j] == BROJ[i] && j < i)) čini
         ništa;
}

funkcija izađi_iz_kritičnog_odsječka(i)
{
   BROJ[i] = 0;
}
```

### Dodatne upute:

Ako se program rješava s procesima tada treba zajedničke varijable tako organizirati da se prostor za njih zauzme odjednom i podijeli među njima. Ovo je nužno zbog ograničenog broja segmenata i velikog broja korisnika.

Ovisno o opterećenju računala i broju procesa koji se pokreću, a da bi se vidjele razlike prilikom izvođenja programa može se usporiti izvršavanje programa sa:

```
sleep(1);
```

na kraju kritičnog odsječka (K.O.).

---

**Problemi zbog izvođenja instrukcija "preko reda"**

Za ispravan rad Dekkerovog, Petersonovog i Lamportovog algoritma pretpostavlja se da se instrukcije programa izvode zadanim redoslijedom. Međutim, neki procesori, kako bi ubrzali izvođenje programa, dozvoljavaju izvođenje instrukcija i "preko reda" (engl. *out-of-order*). Takvo ponašanje može uzrokovati greške u algoritmu međusobnog isključivanja.

Primjerice, prilikom izvođenja Lampotovog algoritma može se dogoditi da se sljedeći niz instrukcija ne izvodi zadanim redoslijedom.

```
1:   ULAZ[i] = 1;
2:   najveci = max(BROJ[0],...,BROJ[n-1]);
3:   BROJ[i] = najveci + 1;
4:   ULAZ[i] = 0;
```

Procesor koji omogućuje izvođenje instrukcija "preko reda" bi mogao ustanoviti da su instrukcije u trećoj i četvrtoj liniji međusobno nezavisne i mogao bi ih izvesti proizvoljnim redosljedom, npr. liniju 4 prije linije 3 (razlog za to može biti da se element `ULAZ[i]` nalazi u priručnom spremniku, a `BROJ[i]` ne). U tom se slučaju ne radi više o Lamportovom algoritmu.

Kako bi se izbjeglo proizvoljno izvođenje instrukcija trebalo bi se jezičnom procesoru naznačiti da je pristup spornim varijablama točno određen programskim kodom.

**Jedan od načina** da se to napravi jest da se varijabla proglasi tipom podataka `atomic`:

```
#include <stdatomic.h>
atomic_int ULAZ[N], BROJ[N];
```

**Drugi način** je da se koriste atomarne operacije. Atomarne operacije, osim što će navedenu operaciju napraviti kao atomarnu (nedjeljivu), još osiguravaju ispravan redoslijed obavljanja operacija (instrukcije prije moraju biti gotovo prije, instrukcije poslije ne smiju započeti prije nego li je ova operacija gotova). Primjer s njihovim korištenjem za postavljanje i čitanje varijabli u Lamportovom algoritmu je u nastavku.

```
//umjesto: ULAZ[i] = 1;
__atomic_store_n (&ULAZ[i], 1, __ATOMIC_RELEASE);

//umjesto: ULAZ[i] = 0;
__atomic_store_n (&ULAZ[i], 0, __ATOMIC_RELEASE);

//umjesto: BROJ[i] = najveci + 1;
__atomic_store_n (&BROJ[i], najveci + 1, __ATOMIC_RELEASE);

//umjesto: while (ULAZ[j] == 1) ;
do __atomic_load (&ULAZ[j], &ulazJ, __ATOMIC_ACQUIRE);
```

```
while (ulazJ == 1);

//umjesto: while (BROJ[j] != 0 && itd) ;
do __atomic_load (&BROJ[j], &brojJ, __ATOMIC_ACQUIRE);
while (brojJ != 0 && itd);

//umjesto: BROJ[i] = 0;
__atomic_store_n (&BROJ[i], 0, __ATOMIC_RELEASE);
```

Više informacija o ovakvim operacijama na Atomic GCC te Atomic GCC: Memory model.