Ime in priimek: **Datum:**

Semafor, konkurenčnost, razvrščanje

Namen vaje je:

- Spoznavanje problema konkurenčnosti, razvrščanja, ter mehanizma semafor

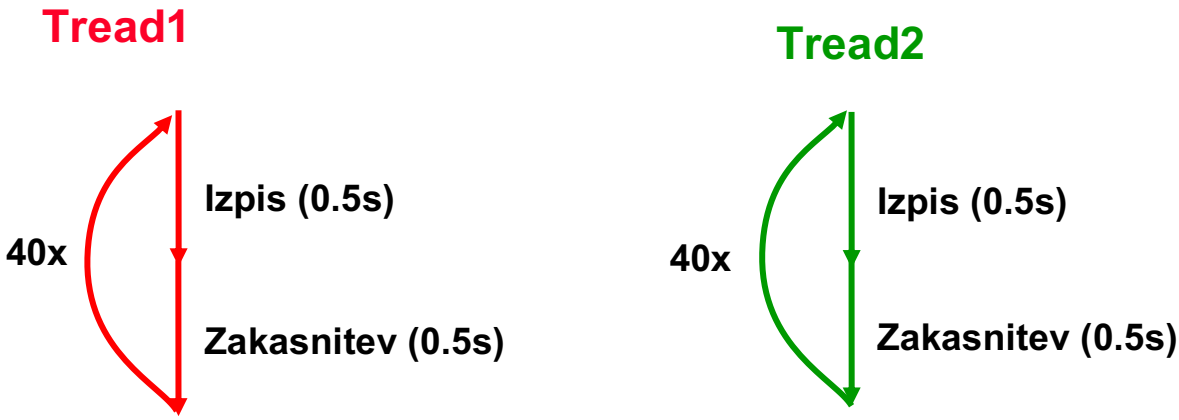
GRADIVA: https://github.com/matejkrenLTFE/POKS2022

1.VAJA

Kreirajte dve niti, ki počasi izpisujeta 40 vrstic besedila. Izpis :

```
Tread 1 i=1 :  11111111111111111111
Tread 2 i=1 :  22222222222222222222
```

Med izpisi posameznih znakov dodajte zakasnitev s for zanko in procesiranje (zakasnitev) – for zanka z matematiko ali taskDelay().



1. V pripravljeni funkciji kreirajte in aktivirajte dve niti (*pthread_create()*)
2. Niti naj izvajata po 40 zank: izpis teksta (ime niti, ponovitev, *Tekst*) + zakasnitev
3. *Tekst* sestavlja izpis 60 enakih znakov v skupnem trajanju 0,5s
4. Dodajte zakasnitev okrog 0,5s realizirano z matematiko v *for* zanki
5. Preizkusite program in med izvajanjem opazujte stanje niti (*ps -elL*)

Opišite in razložite dogajanje :

2.Vaja

1. Zmanjšajte prioriteto druge niti (*nice(3))*
2. Preizkusite program in opazujte prioriteto niti in obremenitev CPU (*ps -elL*)

nice - change process priority

nice() adds inc to the nice value for the calling process. (A higher nice value means a low priority.) Only the superuser may specify a negative increment, or priority increase. The range is from -20 to 19 (least favorable).

Opišite in razložite dogajanje :

3.Vaja

1. Postavite globalno spremenljivko *Semafor* in pripravite funkcije
2. Izpis posamezne vrstice zaščitite s semaforjem *Semafor*

Inicializacija :

```
Sem1 = 1;  /* Inicialization */
```

Uporaba :

```
while(Sem1 == 0);  /* Wait while Semaphore is not free */
Sem1 = 0;          /* Acquire Semaphore */

/* kritično področje */

Sem1 = 1;          /* Release Semaphore */
```

Opišite in razložite dogajanje:

4.Vaja

1. Uporabite POSIX neimenovani semafor: *sem_init(), sem_wait(), sem_post().*

Overview of POSIX

POSIX semaphores allow processes and threads to synchronise their actions. A semaphore is an integer whose value is never allowed to fall below zero. Two operations can be performed on semaphores: increment the semaphore value by one (sem_post()); and decrement the semaphore value by one (sem_wait()). If the value of a semaphore is currently zero, then a sem_wait() operation will block until the value becomes greater than zero. Before being used, an unnamed semaphore must be initialised using sem_init().

```
#include <semaphore.h>
```

```
//  sem_t semid;
//  sem_init(&semid,1,1);
//  sem_wait(&semid);
//  sem_post(&semid);
```

ZGLED:

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

void fun1();

int main ()
{
  pthread_t thread1, thread2;
  char *msg1 = "1";
  char *msg2 = "2";

  pthread_create ( &thread1, NULL, (void *) fun1, (void*) msg1);
  pthread_join (thread1, NULL);
  printf("Konec  \n");
  return (1);
}

void fun1(void * ptr)
{
  char *msg;
  msg = (char *) ptr;
  int i,j;

  for (i=0; i< 40; i++)
  {
    printf("Thread %s i=%d ", msg, i );
    for (j=0;j<60; j++)
    {
      printf("%s", msg);
    }
    printf("\n");
  }
}
```

Opišite in razložite dogajanje :

Komentar, zapiski vaje: