Ime in priimek: **Datum:**

Krožni izravnalnik (Message queues)

Namen vaje je:

- Spoznavanje problema asinhronega pošiljanja sporočil med procesi preko vrst oziroma krožnega izravnalnika.
- Vrsta kot polje (array) in funkcije OS

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

1.VAJA

Kreirajte dve niti (*Thread1* in *Thread2*). Nit *Thread1* naj niti *Thread2* pošilja sporočilo (tekst) preko krožnega izravnalnika (vrsta FIFO). *Thread1* naj izvaja periodično zakasnitev in oddajo teksta v krožni izravnalnik in ob oddaji tudi izpis na terminal. *Thread2* naj po začetni nekaj sekundni zakasnitvi periodično preverja zasedenost izravnalnika in bere ter izpisuje sprejeto vsebino na terminal.

Najprej vrsto sprogramirajte kot polje (array) s pomočjo spodnjega ogrodja. Preizkusite program, opazujte izpis na konzolo, stanje niti (*ps -elL*) in vsebino vrste (*cat /dev/mqueue/test_queue*).

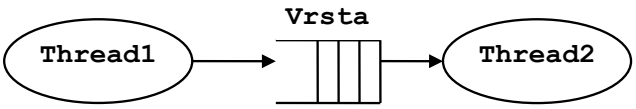
Vrsta kot *array* v C prog.jeziku:

1. V *main()* funkciji kreirajte in aktivirajte dve niti *Thread1* in *Thread2*.
2. Niti naj imata izpis začetka in konca izvajanja.
3. Pripravite globalno vrsto (polje, kazalec vpisa, kazalec branja)
4. *Thread1* naj ciklično (15 x) izvaja zakasnitev in vpis vrstice teksta v vrsto ter izpis na terminal.
5. *Thread2* naj ima začetno zakasnitev 5 sekund (*sleep()*), nato naj ciklično izvaja zakasnitev, branje iz vrste in izpis na terminal.
6. Dodajte kontrole za preverjanje polne vrste in lepo zaključitev.
7. Preizkusite program.
8. Med delovanjem preverite stanje niti (*ps -elL*) in vrste.

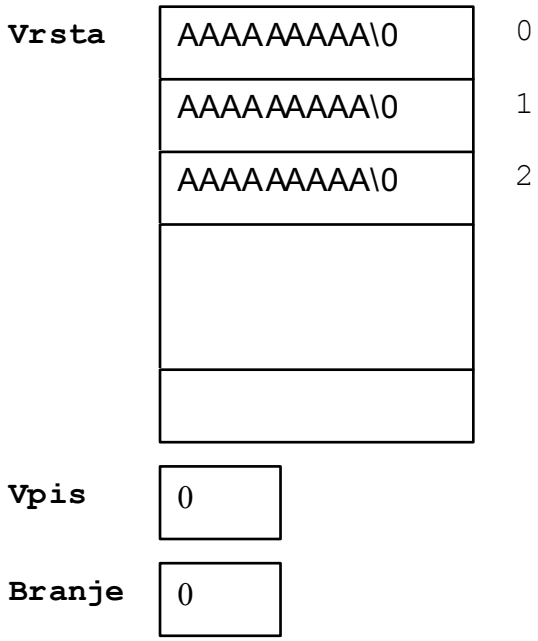
Opišite in razložite dogajanje :

Navodilo:

Krožni izravnalnik oziroma vrsta omogoča asinhrono izmenjavo sporočil med procesi (nitmi) preko začasnega pomnilnika. Thread1 naj vpisuje v vrsto, Thread2 pa bere iz nje.



Realizacija vrste programirano kot polje (array) v C programskem jeziku.




```
#define MAX_MSGS 10
#define MAX_MSG_LEN 100

char Vrsta[MAX_MSGS][MAX_MSG_LEN];

int Vpis=0;
int Branje=0;
```

Thread1 - sender

- Inicializacija
- 15 krat
  - Vpis teksta v Vrsto "AAA...."
  - Izpis oddanega
  - Increment index Vpis
  - Zakasnitev (1s)
- na koncu odda tekst "XX..."

Thread2 - receiver

- Inicializacija vrste
- Začetna zakasnitev (5 s)
- Branje iz Vrste (če ni prazna)
- Izpis sprejetega
- Ob sprejemu "X" prekinitev
- Increment index Branje
- Zakasnitev (0,5s)

Za kopiranje stringov uporabite funkcijo strcpy():

```
char * strcpy ( char * destination, const char * source );
```

Zgled nal1.c**:**

```
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#define MAX_MSGS 10
#define MAX_MSG_LEN 100
void Fun1 (void);

char Vrsta[MAX_MSGS][MAX_MSG_LEN];
int Vpis = 0;
int Branje = 0;

int main ()
{
  pthread_t thread1, thread2;
  pthread_create ( &thread1, NULL, (void *) Fun1, NULL);
  pthread_join (thread1, NULL);
  printf("Konec  \n");
}

void Fun1(void)
{
  char MESSAGE [MAX_MSG_LEN] = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\0";
  char MESSAGEX [MAX_MSG_LEN] = "XXXXXXXXXXXXXXXXX\0";
  int i;

  Vpis = 0;
  Branje = 0;
  printf("Thread1 started - sender \n");
  for (i=0; i<15; i++)
  {
    sleep(1);
    while ( ( (Vpis + 1) % MAX_MSGS)  == Branje);  /* cakaj - vrsta polna*/
    strcpy(Vrsta[Vpis], MESSAGE);
    printf("Thread1 sending %d: %s \n",i, (char *) & MESSAGE);
    Vpis = (Vpis + 1) % MAX_MSGS;
    MESSAGE[0]++;
  }
  printf("Thread1 sending %d: %s \n",i, (char *) & MESSAGEX);
  strcpy(Vrsta[Vpis], MESSAGEX);
  Vpis = (Vpis + 1) % MAX_MSGS;
  printf("Thread1 ended \n");
}

void Fun2(void)
{
  char msgBuff[MAX_MSG_LEN];  /* local received message */

  printf("Thread2 started - receiver \n");
  sleep(5);
  while(1)
  {
    while (Vpis == Branje);  /* cakaj - vrsta prazna*/
    //strcpy();
    printf("Thread2 received: %s Branje=%d Vpis=%d\n",msgBuff, Branje, Vpis);
    //Branje = (Branje + 1) % MAX_MSGS;
    if (msgBuff[0] == 'X')
    {

      printf("Thread2 ended \n");
```

2.VAJA

Krožni izravnalnik realizirajte z uporabo funkcij operacijskega sistema LINUX – POSIX message queues.

1. Sprogramirajte dva programa (procesa s skupnim common.h, zgled spodaj): *Sender* in *Receiver*
2. Uporabite POSIX vrsto (Message queues) funkcije: *mq_open(), mq_send(), mq_receive(), mq_close()*
3. Realizirajte prenos 15 sporočil iz taska *Sender* v task *Receiver*.
4. Preizkusite kodo. Najprej morate startati sprejemnik, ki inicializira vrsto, nato oddajnik.

Opišite in razložite dogajanje :

Dodatek A: Spremenite pogoje tako, da bo med izvajanjem vrsta nekaj časa polna in vpis ne bo mogoč.

Opišite in razložite dogajanje :

Dodatek B: Med izvajanjem preverite stanje taskov in trenutno zasedenost vrste

```
mq_getattr(mq, &attr);
```

Opišite in razložite dogajanje :

Dodatek C: Spremenite prioriteto sporočilom pri oddaji.

Opišite in razložite dogajanje :

Komentar, zapiski vaje:

Overview of POSIX message queues

http://linux.die.net/man/7/mq_overview

POSIX message queues allow processes to exchange data in the form of messages.
Message queues are created and opened using ***mq_open();*** this function returns a message queue descriptor (***mqd_t***), which is used to refer to the open message queue in later calls. Each message queue is identified by a name of the form /somename;
Messages are transferred to and from a queue using ***mq_send()*** and ***mq_receive()***. When a process has finished using the queue, it closes it using ***mq_close()***, and when the queue is no longer required, it can be deleted using ***mq_unlink()***. Queue attributes can be retrieved and (in some cases) modified using ***mq_getattr()*** and ***mq_setattr()***. A process can request asynchronous notification of the arrival of a message on a previously empty queue using ***mq_notify().***

mq_open() creates a new POSIX message queue or opens an existing queue. The queue is identified by name.

```
mqd_t mq_open(const char *name, int oflag);
mqd_t mq_open(const char *name,int oflag, mode_t mode, struct mq_attr *attr);
```

mq_send() adds the message pointed to by msg_ptr to the message queue referred to by the descriptor mqdes. The msg_len argument specifies the length of the message.

```
int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned msg_prio);
```

mq_receive() removes the oldest message with the highest priority from the message queue referred to by the descriptor mqdes, and places it in the buffer pointed to by msg_ptr

```
ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,unsigned *msg_prio);
```

mq_getattr() and **mq_setattr()** respectively retrieve and modify attributes of the message queue referred to by the descriptor mqdes.

```
int mq_getattr(mqd_t mqdes, struct mq_attr *attr);
int mq_setattr(mqd_t mqdes, struct mq_attr *newattr, struct mq_attr *oldattr);

struct mq_attr {
    long mq_flags;       /* Flags: 0 or O_NONBLOCK */
    long mq_maxmsg;      /* Max. # of messages on queue */
    long mq_msgsize;     /* Max. message size (bytes) */
    long mq_curmsgs;     /* # of messages currently in queue */
};
```

mq_close() closes the message queue descriptor mqdes

mq_unlink() removes the specified message queue name. The message queue name is removed immediately.

Mounting the message queue file system

On Linux, message queues are created in a virtual file system. This file system can be mounted (by the superuser) using the following commands:

```
# mkdir /dev/mqueue
# mount -t mqueue none /dev/mqueue
```

After the file system has been mounted, the message queues on the system can be viewed and manipulated using the commands usually used for files (e.g., ls(1) and rm(1), **cat**).

Za prevajanje je potrebna opcija –lrt : **`gcc Receiver.c -o Receiver -lrt`**

ZGLED **common.h**

```
#define QUEUE_NAME  "/test_queue"
#define MAX_SIZE    128
#define MAX_MSGS    10
#define MSG_STOP    "XXXX\0"
```

ZGLED **receiver.c**

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <mqueue.h>
#include "common.h"

int main()
{
    mqd_t mq;
    struct mq_attr attr;
    char buffer[MAX_SIZE + 1];

    /* initialize the queue attributes */
    attr.mq_flags = 0;
    attr.mq_maxmsg = MAX_MSGS;
    attr.mq_msgsize = MAX_SIZE;
    attr.mq_curmsgs = 0;

    /* create the message queue */
    mq = mq_open(QUEUE_NAME, O_CREAT | O_RDONLY, 0644, &attr);
    //sleep();
    while(1)
    {
        ssize_t bytes_read;
        bytes_read = mq_receive(mq, buffer, MAX_SIZE, NULL); /* receive
message */
        //printf();
        //if(buffer[1] == MSG_STOP[1])
    }
    mq_close(mq);   /* cleanup */
    mq_unlink(QUEUE_NAME);
    return 0;
}
```

ZGLED **sender.c**

```
#include "common.h"
int main()
{
    mqd_t mq;
    char MESSAGE[MAX_SIZE]="AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\0";
    /* open the mail queue */
    mq = mq_open(QUEUE_NAME, O_WRONLY);

    for (i=0;i<15;i++)
    {
      mq_send(mq, MESSAGE, MAX_SIZE, 0); /* send the message */
      //printf();
      //sleep();
    }
    mq_send(mq, MSG_STOP, MAX_SIZE, 0);
    mq_close(mq);
    return 0;
}
```