

# ОСуРВ лабораторијска вежба 7

## Међу-процесна комуникација

верзија 1.0

Одсек РТ-РК

23. децембар 2025.

## 1 Увод

На претходним вежбама смо се упознали са средствима комуникације између нити у оквиру истог процеса тј. истог виртуелног адресног простора. У овој вежби ће бити речи о међу-процесној комуникацији (енгл. IPC - Inter-Process Communication), на примеру дистрибуираног управљања моторима између два рачунара. RPi са hostname-ом `rpi-controls.local` ће служити као контролер, који је издавач (енгл. Publisher) командних порука, док ће други RPi са hostname-ом `rpi-motors.local` бити претплатник (енгл. Subscriber) на поруке. Поруке ће бити у облику низа од 4 бајта, које ће садржати који би одговарали дугмићима који су притиснути на јоупад-у.

За комуникацију ових порука између два RPi користиће се ZeroMQ библиотека у C-у. ZeroMQ је веома свестрана библиотека која омогућава комуникацију унутар процеса, комуникацију између процеса на једном рачунару као и дистрибуиране комуникације преко мреже путем рецимо TCP протокола. Постоје имплементације исте библиотеке у разним језицима и за разне платформе.

## 2 Радни процес

Слично као и до сада асистент или 1 студент подешава и тестира радно окружење на два горепоменућа RPi-ја.

### 2.1 Контролни RPi

На `rpi-controls.local` је потребно искомпајлирати и покренути чвор (енгл. Node) за јоупад:

```
1 cd ~/Public/OSuRV_Labs/L7_IPC/SW/App/2_zeroMQ_IPC/
2 ./waf configure
3 ./waf build
4 ./build/joy_node
```

### 2.2 Моторни RPi

На `rpi-motors.local` компајлирати и покренути драјвер за GPIO:

```

1 cd ~/Public/OSuRV_Labs
2 git pull
3 cd L7_IPC/SW/Driver/gpio_ctrl/
4 make
5 make start

```

Након драјвера компајлирати апликацију и покренути чвор за рад са DC мотором брисача:

```

1 cd ~/Public/OSuRV_Labs/L7_IPC/SW/App/2_zeroMQ_IPC/
2 ./waf configure
3 ./waf build
4 ./build/wiper_node

```

## 2.3 Тестирање

Притискати дугмиће на јоурпад-у и посматрати померање мотора као и исписе у конзоли на `rpi-controls.local` и `rpi-motors.local`.

Надаље, као и до сад, сваки ће студент на свом локалном RPi компајлирати апликацију, са `scp` копирати је на `rpi-controls.local` или `rpi-motors.local`, и преко `ssh` покренути апликације ради тестирања.

## 3 Објашњење кода

### 3.1 Чвор издавач

Главно подешавање издавача у `joy_node.c` је следећа линија:

```

1 #define ZMQ_ENDPOINT "tcp://0.0.0.0:5555"

```

где су:

- `tcp` - протокол за IPC. Опције су:
  - `tcp` - дистрибуирана комуникација.
  - `ipc` - комуникација између процеса на истом рачунару.
  - `inproc` - комуникација унутар процеса.
- `0.0.0.0` - IP или интерфејс (на пример `eth0` за LAN или `wlan0` за WiFi) на којем ће се поруке издавати. У овом случају `0.0.0.0` значи да се шаље на свим интерфејсима.
- `5555` - port на ком се поруке издају.

Код за иницијализацију почиње са креирањем ZeroMQ контекста, отварањем издавачког socket-а и везивањем истог:

```

1 void* context = zmq_ctx_new();
2 if(!context){
3     perror("Failed to create ZeroMQ context");
4     return 1;
5 }
6 void* publisher = zmq_socket(context, ZMQ_PUB);
7 if(!publisher){

```

```

8     perror("Failed to create ZeroMQ PUB socket");
9     zmq_ctx_destroy(context);
10    return 1;
11 }
12 if(zmq_bind(publisher, ZMQ_ENDPOINT) != 0){
13     perror("Failed to bind ZeroMQ PUB socket");
14     zmq_close(publisher);
15     zmq_ctx_destroy(context);
16     return 1;
17 }

```

Унутар бесконачне петље се чита стање дугмића са `/dev/input/js0` и издаје се следећим кодом:

```

1 if(zmq_send(publisher, buttons, sizeof(buttons), 0) == -1){
2     perror("Failed to publish buttons");
3 }

```

## 3.2 Чвор претплатник

Главно подешавање у `wiper_node.c` је следећа линија:

```

1 #define ZMQ_ENDPOINT "tcp://rpi-controls.local:5555"

```

Овде се подешава протокол, адреса RPi-ја који објављује поруке, и port на којем се објављују. Протокол и port су исти као и код издавача, док је адреса hostname RPi-ја са издавачем.

Иницијализација почиње креирањем контекста, отварањем socket-a, повезивањем и претплаћивањем на издавача:

```

1 void* context = zmq_ctx_new();
2 if(!context){
3     perror("Failed to create ZeroMQ context");
4     return 1;
5 }
6 void* subscriber = zmq_socket(context, ZMQ_SUB);
7 if(!subscriber){
8     perror("Failed to create ZeroMQ socket");
9     zmq_ctx_destroy(context);
10    return 1;
11 }
12 if(zmq_connect(subscriber, ZMQ_ENDPOINT) != 0){
13     perror("Failed to connect ZeroMQ socket");
14     zmq_close(subscriber);
15     zmq_ctx_destroy(context);
16     return 1;
17 }
18 if(zmq_setsockopt(subscriber, ZMQ_SUBSCRIBE, "", 0) != 0){
19     perror("Failed to set ZMQ_SUBSCRIBE");
20     zmq_close(subscriber);
21     zmq_ctx_destroy(context);
22     return 1;
23 }

```

У бесконачној петљи се врши пријем порука:

```

1 int bytes = zmq_msg_recv(&msg, subscriber, ZMQ_DONTWAIT); // Non-
   blocking receive.
2 if(bytes == N_BUTTONS){
3     memcpy(buttons, zmq_msg_data(&msg), bytes);
4 }

```

Ако је резултат већи од 0 (у овом случају тачно бајтова колико је порука) порука је примљена и на основу вредности дугмића се врши померање мотора.

## 4 Задатак

### 4.1 Нови чвор издавач

Реализовати `routine_node.c`. Механизам слања порука је идентично као и код `joy_node.c`, али уместо читавања из `/dev/input/js0` који су дугмићи притиснути на у бесконачној петљи, извршава се рутина која се састоји од слања порука са паузама између. Примера ради, пошаље се CCW командна порука, сачека се једна секунда, онда STOP порука.

### 4.2 Нови чвор претплатник

Реализовати `stepper_node.c`. Пријем порука је идентичан као и код `wiper_node.c`. Водити рачуна да је потребно користити мању паузу под `usleep()` како би се видели ефекти окретања корачног мотора.

### 4.3 Додатни задаци

Додатни задатак: Покренути контролне чворове са личног RPi. Ово захтева да се промени адреса (hostname и IP) издавача у чворовима претплатницима.

Додатни задатак: уместо низа дугмића као команду слати 1 бајт са бројем који контролише акцију мотора.

Додатни задатак: као команду слати стринг. Пратити са Wireshark-ом команде у пакетима.