ОСуРВ лабораторијска вежба 7

Међу-процесна комуникација

верзија 1.0

Одсек РТ-РК

24. 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 sa 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 садржи кодове који би одговарали дугмићима који су притиснути на joypad-у.

За комуникацију ових порука између два RPi користиће се `ZeroMQ` библиотека у C-у. `ZeroMQ` је веома широко коришћена 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 горепоменута RPi-ја.

2.1 Контролни RPi

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

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

2.2 Моторни RPi

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

```
cd ~/Public/OSuRV_Labs
git pull
cd L7_IPC/SW/Driver/gpio_ctrl/
make
make start
```

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

```
cd ~/Public/OSuRV_Labs/L7_IPC/SW/App/2_zeroMQ_IPC/
./waf configure
./waf build
./build/wiper_node
```

2.3 Тестирање

Притискати дугмиће на joypad-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` покретати апликације ради тестирања.

3 Објашњење кода

3.1 Чвор издавач

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

```
#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 прављењем `ZeroMQ` контекста, отварањем издавачеве утичнице (енгл. Socket) и повезивање исте:

```
void* context = zmq_ctx_new();
if(!context){
    perror("Failed to create ZeroMQ context");
    return 1;
}
void* publisher = zmq_socket(context, ZMQ_PUB);
if(!publisher){
```

```
    perror("Failed to create ZeroMQ PUB socket");
    zmq_ctx_destroy(context);
    return 1;
}
if(zmq_bind(publisher, ZMQ_ENDPOINT) != 0){
    perror("Failed to bind ZeroMQ PUB socket");
    zmq_close(publisher);
    zmq_ctx_destroy(context);
    return 1;
}
```

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

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

3.2 Чвор претплатник

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

```
#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 прављењем, отварањем утичнице, повезивањем и претплаћивањем на издавача:

```
void* context = zmq_ctx_new();
if(!context){
    perror("Failed to create ZeroMQ context");
    return 1;
}
void* subscriber = zmq_socket(context, ZMQ_SUB);
if(!subscriber){
    perror("Failed to create ZeroMQ socket");
    zmq_ctx_destroy(context);
    return 1;
}
if(zmq_connect(subscriber, ZMQ_ENDPOINT) != 0){
    perror("Failed to connect ZeroMQ socket");
    zmq_close(subscriber);
    zmq_ctx_destroy(context);
    return 1;
}
if(zmq_setsockopt(subscriber, ZMQ_SUBSCRIBE, "", 0) != 0){
    perror("Failed to set ZMQ_SUBSCRIBE");
    zmq_close(subscriber);
    zmq_ctx_destroy(context);
    return 1;
}
```

У бесконачној петљи се примају поруке:

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

Ако је резултат већи од 0 (у овом случају тачно бајтова колико је порука) порука је примљена и на основу вредности дугмића се помера мотор брисача.

4 Задатак

4.1 Нови чвор издавач

Реализовати `routine_node.c`. Механизам слања порука је идентичан као и код `joy_node.c`, али уместо очитавања 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.