PROGRAMOWANIE SYSTEMÓW CZASU RZECZYWISTEGO

LABORATORIUM

Temat: QNX Neutrino IPC native

Mariusz Rudnicki

2016

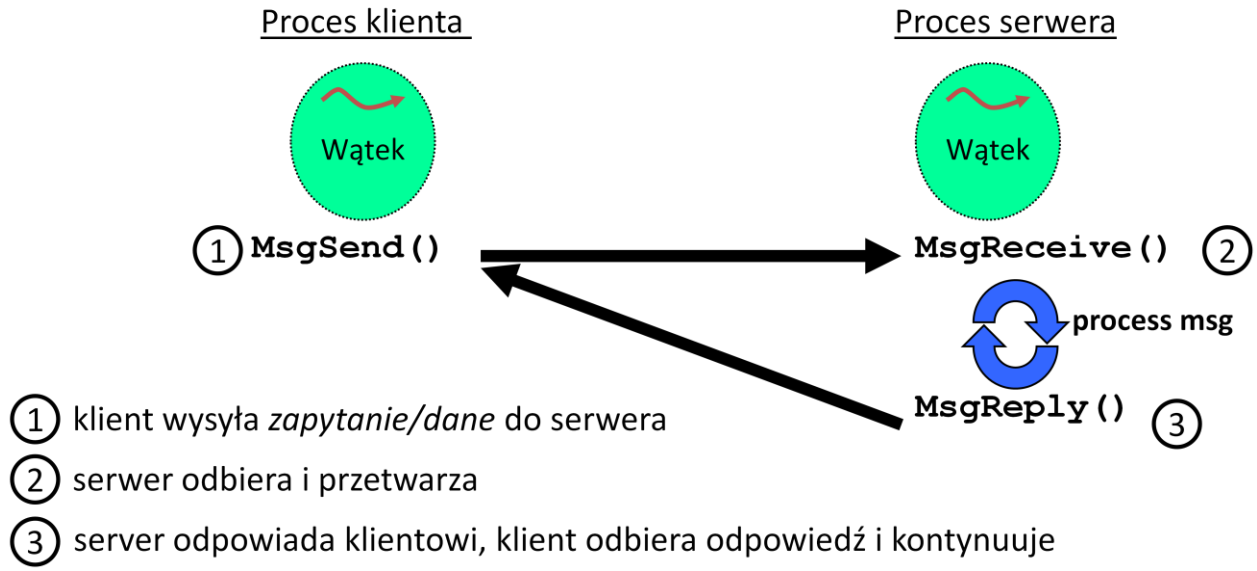
Wstęp

QNX Neutrino wspiera różnorodne mechanizmy komunikacji IPC:

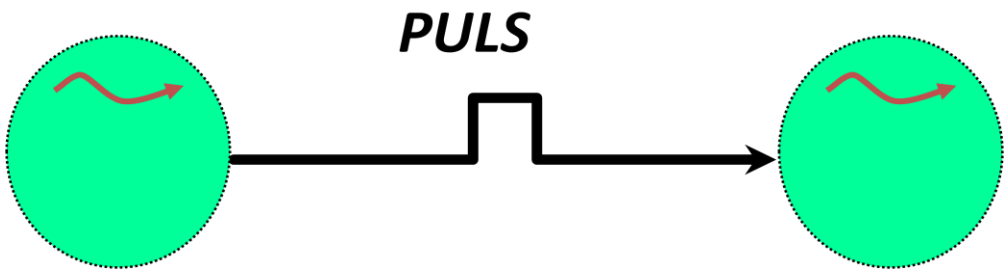
- rodzima komunikacja QNX Neutrino – (API unikalne dla QNX):
  - komunikaty QNX Neutrino;
  - pulsy QNX Neutrino;
- mechanizmy zgodne z POSIX/UNIX (API przenośne):
  - sygnały;
  - pamięć dzielona;
  - potoki (wymagany proces ***pipe***);
  - kolejki komunikatów POSIX (wymagany proces ***mqueue*** lub ***mq***);
  - gniazda TCP/IP (wymagany proces ***io-net***)

Na laboratorium zapoznamy się z rodzimą komunikacją QNX Neutrino.

Komunikacja bazująca na komunikatach QNX Neutrino oparta jest na modelu klient-serwer. Jest dwukierunkowa i synchroniczna. Oznacza to iż klient po wysłaniu komunikatu do serwera czeka na jego odpowiedź. Dopiero po otrzymaniu informacji zwrotnej może kontynuować pracę. W trakcie przesyłania komunikatów mamy do czynienia z fizycznym kopiowaniem danych pomiędzy obszarami pamięci.



Do komunikacji asynchronicznej używane są pulsy – krótkie powiadomienia.



Zalety pulsu:

- nieblokujący dla nadawcy;
- ustalony rozmiar:
  - 32-bitowa wartość,
  - 8-bitowy kod(-128 do 127);
- jednokierunkowe (nie wymagają odpowiedzi);
- szybkie i „niedrogie”

Przygotowanie platformy i środowiska IDE

Uruchom QNX Neutrino za pomocą maszyny wirtualnej. Sprawdź poleceniem ***pidin*** czy jest uruchomiony proces ***qconn***. Jeśli nie ma go na liście pracujących procesów uruchom go poleceniem ***qconn #***.Sprawdź adres IP poleceniem ***ifconfig***. Następnie uruchom środowisko QNX Momentics IDE. Wybierz przestrzeń roboczą wskazaną przez prowadzącego.

Zadanie 1.

Przeanalizuj kod źródłowy poniższego programu **server**. Odszukaj w nim fragmenty odpowiedzialne za zestawienie połączenia po stronie serwera. Wypisz odpowiednie funkcje w sprawozdaniu.

```
/*
server.c
    A QNX msg passing server.  It should receive a string from a client,
calculate a checksum on it, and reply back the client with the checksum.
    The server prints out its pid and chid so that the client can be made
aware of them.
    Using the comments below, put code in to complete the program. Look up
function arguments in the course book or the QNX documentation.
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/neutrino.h>
#include <process.h>

#include "msg_def.h"    //layout of msg's should always defined by
                        //a struct, here's it's definition
int calculate_checksum(char *text);

int main(void) {
  int chid;
  int pid;
  int rcvid;
  cksum_msg_t msg;
```

```
  int status;
  int checksum;

  setvbuf (stdout, NULL, _IOLBF, 0);  //set IO to stdout to be line
buffered

  chid = ChannelCreate(0);      //PUT CODE HERE to create a channel
  if(-1 == chid) {              //was there an error creating the channel?
    perror("ChannelCreate()"); //look up the errno code and print
    exit(EXIT_FAILURE);
  }

  pid = getpid();               //get our own pid
  //print our pid/chid so client can be told where to connect
  printf("Server's pid: %d, chid: %d\n", pid, chid);
  while(1) {
    //PUT CODE HERE to receive msg from client
    rcvid = MsgReceive(chid, &msg, sizeof(msg), NULL);
    if(rcvid == -1) {           //Was there an error receiving msg?
      perror("MsgReceive");     //look up errno code and print
      break;                    //try receiving another msg
    }

    checksum = calculate_checksum(msg.string_to_cksum);
    //PUT CODE HERE TO reply to client with checksum
    status = MsgReply(rcvid, EOK, &checksum, sizeof(checksum) );
    if(-1 == status) {
      perror("MsgReply");
    }
  }
  return 0;
}

int
calculate_checksum(char *text){
  char *c;
  int cksum = 0;

  for (c = text; *c != NULL; c++)
    cksum += *c;
  return cksum;
}
```

Na podstawie powyższego kodu zmodyfikuj kod źródłowy projektu **server** tak, aby mógł odbierać komunikaty od klienta. Po modyfikacjach uruchom program **server.**

Zadanie 2.

Przeanalizuj kod źródłowy poniższego programu **client**. Odszukaj w nim fragmenty odpowiedzialne za zestawienie połączenia po stronie klienta. Wypisz odpowiednie funkcje w sprawozdaniu.

```
/*
client.c

    A QNX msg passing client.  It's purpose is to send a string of text to
a server. The server calculates a checksum and replies back with it.  The
client expects the reply to come back as an int.

    This program program must be started with commandline args. See
if(argc != 4) below.

To complete the exercise, put in the code, as explained in the comments
below. Look up function arguments in the course book or the QNX
documentation.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/neutrino.h>
#include <sys/netmgr.h>  // #define for ND_LOCAL_NODE is in here

#include "msg_def.h"

int main(int argc, char* argv[])
{
  int coid;                 //Connection ID to server
  cksum_msg_t msg;
  int incoming_checksum;  //space for server's reply
  int status;               //status return value used for ConnectAttach and
MsgSend
  int server_pid;           //server's process ID
  int server_chid;          //server's channel ID

  setvbuf (stdout, NULL, _IOLBF, 0);

  if(4 != argc) {
    printf("This program must be started with commandline arguments, for
           example:\n\n");
    printf("   cli 482834 1 abcdefghi    \n\n");
    printf(" 1st arg(482834): server's pid\n");
    printf(" 2nd arg(1): server's chid\n");
    printf(" 3rd arg(abcdefghi): string to send to server\n"); //to make
                                it easy, let's not bother handling spaces
    exit(EXIT_FAILURE);
```

```
  }

  server_pid = atoi(argv[1]);
  server_chid = atoi(argv[2]);

  printf("attempting to establish connection with server pid: %d, chid
       %d\n", server_pid, server_chid);
  //PUT CODE HERE to establish a connection to the server's channel
  coid = ConnectAttach(ND_LOCAL_NODE, server_pid, server_chid,
                       _NTO_SIDE_CHANNEL, 0);
  if(-1 == coid) {           //was there an error attaching to server?
    perror("ConnectAttach"); //look up error code and print
    exit(EXIT_FAILURE);
  }

  msg.msg_type = CKSUM_MSG_TYPE;
  strcpy(msg.string_to_cksum, argv[3]);
  printf("Sending string: %s\n", msg.string_to_cksum);

   //PUT CODE HERE to send message to server and get the reply
  status = MsgSend(coid, &msg, sizeof(msg), &incoming_checksum,
                   sizeof(incoming_checksum) );
  if(-1 == status) {            //Was there an error sending to server?
    perror("MsgSend");
    exit(EXIT_FAILURE);
  }

  printf("received checksum=%d from server\n", incoming_checksum);
  printf("MsgSend return status: %d\n", status);

  return EXIT_SUCCESS;
}
```

Na podstawie powyższego kodu zmodyfikuj kod źródłowy projektu **client** tak, aby mógł wysyłać komunikaty do serwera. Po modyfikacjach uruchom program **client**.

Zadanie 3.

Zmień kod programu **client** tak, aby komunikat do serwera nadawany był cyklicznie.

Zastanów się, jak zmodyfikować kody powyższych programów **server** i **client**, aby odpowiedzieć na następujące pytania:

W jakim stanie jest client nim server gotowy będzie do odbierania komunikatów?

W jakim stanie jest client gdy server odbierze komunikat?

W jakim stanie jest server nim klient wyśle do niego komunikat?

Dokonaj odpowiednich modyfikacji kodu i udziel odpowiedzi na wyżej postawione pytania. Zaprezentuj wyniki prowadzącemu w perspektywie **QNX System Information.**

Zadanie 4.

Przeanalizuj kod źródłowy poniższego programu **pulse_server.**

```
/*
  pulse_server.c
    A QNX msg passing server.  It should receive a string from a client,
calculate a checksum on it, and reply back the client with the checksum.
    The server prints out its pid and chid so the client can be made aware
of them.
    Using the comments below, put code in to complete the program.  Look
up function arguments in the course book or the QNX documentation.
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/neutrino.h>
#include <process.h>

#include "msg_def.h"  //layout of msg's should always defined by a struct,
here's
                      //it's definition


int
calculate_checksum(char *text);

int
main(void) {
  int chid;
  int pid;
  int rcvid;
  msg_buf_t msg;
  int status;
  int checksum;
```

```
  setvbuf (stdout, NULL, _IOLBF, 0);   //set IO to stdout to be line
buffered

  chid = ChannelCreate(0);           //PUT CODE HERE to create a channel
  if(-1 == chid) {              //was there an error creating the channel?
    perror("ChannelCreate()");  //look up the errno code and print
    exit(EXIT_FAILURE);
  }

  pid = getpid();                //get our own pid
  //print our pid/chid so client can be told where to connect
  printf("Server's pid: %d, chid: %d\n", pid, chid);

  while(1) {
    //PUT CODE HERE to receive msg from client
    rcvid = MsgReceive(chid, &msg, sizeof(msg), NULL);
    if(rcvid == -1) {            //Was there an error receiving msg?
      perror("MsgReceive");      //look up errno code and print
      continue;                  //try receiving another msg
    }
    else if (rcvid == 0) {
          printf("a pulse was received, its code is %d, its value is
%d\n",
                 msg.pulse.code, msg.pulse.value.sival_int);
          continue;
        }

    checksum = calculate_checksum(msg.cksum.string_to_cksum);
    //PUT CODE HERE TO reply to client with checksum
    status = MsgReply(rcvid, EOK, &checksum, sizeof(checksum) );
    if(-1 == status) {
      perror("MsgReply");
    }
    return 0;
}

int
calculate_checksum(char *text)
{
  char *c;
  int cksum = 0;

  for (c = text; *c != NULL; c++)
     cksum += *c;
  return cksum;
}
```

Na podstawie powyższego kodu zmodyfikuj kod źródłowy projektu **pulse_server** tak, aby mógł odbierać pulsy do klienta. Po modyfikacjach uruchom program **pulse_server**.

Zadanie 5.

Przeanalizuj kod źródłowy poniższego programu **pulse_client.**

```
/*
  pulse_client.c

    A QNX msg passing client.  It's purpose is to send a string of text to
a server.  The server calculates a checksum and replies back with it.  The
client expects the reply to come back as an int
    This program program must be started with commandline args. See
if(argc != 4) below.
    To complete the exercise, put in the code, as explained in the
comments below. Look up function arguments in the course book or the QNX
documentation.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/neutrino.h>
#include <sys/netmgr.h>      // #define for ND_LOCAL_NODE is in here

#include "msg_def.h"

#define MY_PULSE_CODE               _PULSE_CODE_MINAVAIL
#define MY_OTHER_PULSE_CODE         _PULSE_CODE_MINAVAIL + 7

int main(int argc, char* argv[])
{
  int coid;      //Connection ID to server
  int status;    //status return value used for ConnectAttach and MsgSend
  int server_pid;         //server's process ID
  int server_chid;        //server's channel ID
  int some_value = 25;    //make up some kind of number for a value

    setvbuf (stdout, NULL, _IOLBF, 0);

  if(4 != argc) {
    printf("This program must be started with commandline arguments, for
           example:\n\n");
    printf("   cli 482834 1 abcdefghi    \n\n");
    printf(" 1st arg(482834): server's pid\n");
    printf(" 2nd arg(1): server's chid\n");
    printf(" 3rd arg(abcdefghi): string to send to server\n"); //to make
                              //it easy, let's not bother handling spaces
    exit(EXIT_FAILURE);
  }

  server_pid = atoi(argv[1]);
  server_chid = atoi(argv[2]);
```

```
  printf("attempting to establish connection with server pid: %d, chid
         %d\n", server_pid, server_chid);
  //PUT CODE HERE to establish a connection to the server's channel
  coid = ConnectAttach(ND_LOCAL_NODE, server_pid, server_chid,
                       _NTO_SIDE_CHANNEL, 0);
  if(-1 == coid) {    //was there an error attaching to server?
    perror("ConnectAttach"); //look up error code and print
    exit(EXIT_FAILURE);
  }

  status = MsgSendPulse(coid, getprio(0), MY_PULSE_CODE, 0);
  if(status == -1) {
    perror("problmem with 1st MsgSendPulse");
  }

  status = MsgSendPulse(coid, getprio(0), MY_OTHER_PULSE_CODE,
                        some_value);
  if(status == -1) {
    perror("problmem with 2nd MsgSendPulse");
  }

  return EXIT_SUCCESS;
}
```

Na podstawie powyższego kodu zmodyfikuj kod źródłowy projektu **pulse_client** tak, aby mógł wysyłać pulsy do serwera. Po modyfikacjach uruchom program **pulse_client**.

Zadanie 6.

Zmodyfikuj program **pulse_client** tak aby pulsy wysyłane były cyklicznie. Dodaj własny pulse. Sprawdź w jakim stanie jest proces **pulse_client** po wysłaniu pulsu.

Zadanie 7.

Przeanalizuj kod źródłowy poniższego programu **name_lookup_server.** Odszukaj w nim fragmenty odpowiedzialne za zestawienie połączenia po stronie serwera. Zamieść odpowiednie funkcje w sprawozdaniu.

```
/*
  name_lookup_server.c

    A QNX msg passing server.  It should receive a string from a client,
calculate a checksum on it, and reply back the client with the checksum.
    The server prints out its pid and chid so that the client can be made
aware of them.
    Using the comments below, put code in to complete the program.  Look
up function arguments in the course book or the QNX documentation.
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/neutrino.h>
#include <process.h>
#include <sys/iofunc.h>
#include <sys/dispatch.h>

#include "msg_def.h"  //layout of msg's should always defined by a struct,
                      // here's it's definition

int
calculate_checksum(char *text);

int
main(void) {
  int rcvid;
  msg_buf_t msg;
  int status;
  int checksum;
  name_attach_t* attach;

  setvbuf (stdout, NULL, _IOLBF, 0); //set IO to stdout to be line
buffered
  //PUT CODE HERE to attach a name
  attach = name_attach(NULL, SERVER_NAME, 0);
  if(NULL == attach) {        //Was there an error creating the channel?
    perror("name_attach()");  //look up the errno code and print
    exit(EXIT_FAILURE);
  }
```

```
  while(1) {
//PUT CODE HERE to receive msg from client, store the receive id in rcvid
    rcvid = MsgReceive(attach->chid, &msg, sizeof msg, NULL);
    if(rcvid == -1) {             //was there an error receiving msg?
      perror("MsgReceive");      //look up errno code and print
      continue;                    //try receiving another msg
    }
    else if(rcvid == 0) {
          printf("received pulse, code = %d\n", msg.pulse.code);
          continue;
        }
    printf("received msg: %s\n", msg.cksum.string_to_cksum);
    checksum = calculate_checksum(msg.cksum.string_to_cksum);

//PUT CODE HERE TO reply to client with checksum, store the return status
//in status
    status = MsgReply(rcvid, EOK, &checksum, sizeof checksum);
    if(-1 == status) {
      perror("MsgReply");
    }
  }
  return 0;
}


int
calculate_checksum(char *text)
{
  char *c;
  int cksum = 0;

  for (c = text; *c != NULL; c++)
    cksum += *c;
  return cksum;
}
```

Na podstawie powyższego kodu zmodyfikuj kod źródłowy projektu **name_lookup_server** tak, aby mógł odbierać komunikaty do klienta. Po modyfikacjach uruchom program **name_lookup_server.**

Zadanie 8.

Przeanalizuj kod źródłowy poniższego programu **name_lookup_client.** Odszukaj w nim fragmenty odpowiedzialne za zestawienie połączenia po stronie klienta. Wypisz odpowiednie funkcje w sprawozdaniu.

```
/*
  name_lookup_client.c

    A QNX msg passing client.  It's purpose is to send a string of text to
a server.  The server calculates a checksum and replies back with it.  The
client expects the reply to come back as an int.
    This program program must be started with commandline args. See
if(argc != 4) below
    To complete the exercise, put in the code, as explained in the
comments below. Look up function arguments in the course book or the QNX
documentation.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/neutrino.h>
#include <sys/netmgr.h>      // #define for ND_LOCAL_NODE is in here
#include <sys/iofunc.h>
#include <sys/dispatch.h>

#include "msg_def.h"


int main(int argc, char* argv[])
{
  int coid;                 //Connection ID to server
  msg_buf_t msg;
  int incoming_checksum;    //space for server's reply
  int status;               //status return value used for ConnectAttach and
                            //MsgSend

    setvbuf (stdout, NULL, _IOLBF, 0);

  if(2 != argc) {
    printf("This program must be started with commandline arguments, for
           example:\n\n");
    printf("   client abcdefghi    \n\n");
    printf(" where (abcdefghi) is string to send to server\n"); //to make
                              //it easy, let's not bother handling spaces
    exit(EXIT_FAILURE);
  }
```

```
printf("client: attempting to establish connection with server %s\n",
SERVER_NAME);
//PUT CODE HERE to establish a connection to the server's channel, store
the connection id in the variable 'coid'
  coid = name_open(SERVER_NAME, 0);
  if(-1 == coid) {    //was there an error attaching to server?
    perror("ConnectAttach"); //look up error code and print
    exit(EXIT_FAILURE);
  }

  msg.cksum.msg_type = CKSUM_MSG_TYPE;
  strcpy(msg.cksum.string_to_cksum, argv[1]);
  printf("Sending string: %s\n", msg.cksum.string_to_cksum);

  //PUT CODE HERE to send message to server and get the reply
  status = MsgSend(coid, &msg, sizeof msg, &incoming_checksum,
                   sizeof(incoming_checksum);
  if(-1 == status) {    //was there an error sending to server?
    perror("MsgSend");
    exit(EXIT_FAILURE);
  }

  printf("received checksum=%d from server\n", incoming_checksum);
  printf("MsgSend return status: %d\n", status);

  return EXIT_SUCCESS;
}
```

Na podstawie powyższego kodu zmodyfikuj kod źródłowy projektu **name_lookup_client** tak, aby mógł wysyłać komunikaty do serwera. Po modyfikacjach uruchom program **name_lookup_client.**

Zadanie 9.

Zmodyfikuj kody programów **name_lookup_client** i **name_lookup_server** tak aby komunikaty były wysyłane i odbierane cyklicznie.

Sprawozdanie.

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

Grupa: