

# Ćwiczenie PSI-01. Realizacja modelu kolejkowego komunikacji (klient-serwer) przez łącze szeregowe

PRZEMYSŁOWE SIECI INFORMATYCZNE

LABORATORIUM SYSTEMÓW STEROWNIA PRZEMYSŁOWEGO I AUTOMATYKI BUDYNKÓW

KATEDRA AUTOMATYKI NAPĘDU I URZĄDZEŃ PRZEMYSŁOWYCH  
WWW.KANIUP.AGH.EDU.PL

AKADEMIA GÓRNICZO-HUTNICZA  
WWW.AGH.EDU.PL

<b>Temat:</b>	Realizacja modelu kolejkowego komunikacji (klient-serwer) przez łącze szeregowe
<b>Narzędzia:</b>	Kompilator języka C, linker, debugger, symulator serwer transmisji szeregowej
<b>Wymagane umiejętności:</b>	Podstawowa znajomość architektury systemu UNIX programowania w języku C

## Wstęp.

Obsługa transmisji szeregowej wymaga użycia standardowych funkcji języka C takich jak dla obsługi plików, czyli

- open()
- read()
- write()
- close()

Dodatkowo funkcją specyficzną do odczytu danych z urządzenia szeregowego (/dev/ser) jest funkcja `dev_read`.

### **dev\_read**

odczyt danych z urządzenia szeregowego

#### **postać funkcji:**

```
#include <sys/dev.h>
int dev_read( int fd,
              void *buf,
              unsigned n,
              unsigned min,
              unsigned time,
              unsigned timeout;
              pid_t proxy;
              int *armed );
```

#### **opis funkcji:**

Funkcja `dev_read()` czyta do  $n$  bajtów danych z urządzenia szeregowego wskazywanego przez deskryptor  $fd$  do bufora wskazywanego przez  $buf$ .

Funkcja `dev_read()` jest alternatywą dla funkcji `read()` dla urządzeń szeregowych. Udostępnia dodatkowe argumenty umożliwiające obsługę czasów (`timeout`) i depozytów wyzwalanych w przypadku zdarzeń w urządzeniu szeregowym. Te dodatkowe argumenty ułatwiają i optymalizują obsługę urządzeń szeregowych.

## Ćwiczenie PSI-01. Realizacja modelu kolejkowego komunikacji (klient-serwer) przez łącze szeregowo

### Przykład użycia funkcji `dev_read`:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/dev.h>
#include <sys/proxy.h>
#include <sys/kernel.h>

unsigned char buf[100];

void main()
{
    pid_t proxy, pid;
    int fd, armed, n, N = 10;

    fd = open( "/dev/ser1", O_RDONLY );
    /*
     * Przeczytaj do 'N' bajtów z 'fd'
     * używając (min, time, timeout) o wartościach (1, 0, 0)
     */
    n = dev_read( fd, &buf, N, 1, 0, 0, 0, 0 );

    /*
     * Przykład użycia depozytów do powiadamiania
     * o obecności nowych danych bez potrzeby
     * odpytywania.
     */

    proxy = qnx_proxy_attach( 0, 0, 0, -1 );

    /*
     * "Uzbrojenie" pierwszego depozytu
     */
    n = dev_read( fd, &buf, 0, 1, 0, 0, proxy, 0 );

    for( ;; ) {
        /*
         * czekaj na depozyt lub wiadomość
         */
        pid = Receive( 0, &buf, sizeof( buf ) );
        if( pid == proxy ) {
            armed = 0;
            while( !armed ) {
                /*
                 * Kontynuuj aż do ostatniego depozytu
                 */
                n = dev_read( fd, &buf, N, 1, 0, 0,
                               proxy, &armed );

                /*
                 * tu obsługa 'n' bajtów w 'buf'
                 */
            }
        }
        else {
            /* obsługa innych depozytów lub wiadomości */
        }
    }
}
```

# Ćwiczenie PSI-01. Realizacja modelu kolejkowego komunikacji (klient-serwer) przez łącze szeregowe

## Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z obsługą komunikacji szeregowej (w modelu klient-serwer) na poziomie protokołu tekstowego.

## Program ćwiczenia.

### Zadanie:

Zrealizować program klienta komunikującego się przy pomocy łącza szeregowego z serwerem, który realizuje zdefiniowany niżej protokół.

### Protokół:

Komunikacja polega na wysłaniu wiadomości do serwera i odbieraniu odpowiedzi (potwierżeń). Wiadomości i odpowiedzi kodowane są w postaci ramek. Ramka zawiera adres urządzenia, do którego jest adresowany komunikat i tylko przez to urządzenie jest odbierana. W jednej ramce może wystąpić jeden rozkaz odczytu lub zapisu rejestru w urządzeniu, przy czym format przesyłanych danych (zarówno wysyłanych jak i odbieranych) zależy od typu rejestru.

Każda wysłana wiadomość jest potwierdzana

- odpowiedzią pozytywną realizowaną jako „echo” jeśli wiadomość realizująca rozkaz zapisu (Wx) została odebrana poprawnie
- odpowiedzią pozytywną realizowaną jako przesył danych jeśli wiadomość realizująca rozkaz odczytu (Rx) została odebrana poprawnie
- odpowiedzią negatywną (NO) w przypadku wystąpienia błędu sumy kontrolnej.

Wiadomość nie jest potwierdzana jeśli:

- podany zostanie błędny adres
- wiadomość nie zostanie odebrana w całości (brak znaku końca).

### Ramka:

	Znak startu	Adres	Rozkaz	Dane	Suma kontr.	Znak końca
Opis	':'	Zakres 01-0f heksadecymalnie	Wx, Rx lub NO	Tekst	Modulo 256 heksadecymalnie	'\n'
Długość	1 bajt	2 bajty	2 bajty	N bajtów	2 bajty	1 bajt

### Rozkazy:

Wx - zapis danej do rejestru x (adres heksadecymalny z zakresu 0-f)

Rx - odczyt danej z rejestru x (adres heksadecymalny z zakresu 0-f)

NO - odpowiedź negatywna.

## Ćwiczenie PSI-01. Realizacja modelu kolejkowego komunikacji (klient-serwer) przez łącze szeregowe

### Rejestry:

adres 0x0-0x5: zmiennoprzecinkowe

format: liczba zmiennoprzecinkowa z kropką dziesiętną i znakiem

adres 0x6-0xa: tekstowe o długości 16 bajtów

format: tekst o długości maksymalnie 16 znaków

adres 0xb-0xf: jednobajtowe

format: zapis heksadecymalny

### Przykłady:

1. Zapisz dane „12345” do rejestru „0” w urządzeniu o adresie „01”

- wiadomość wysłana: :01W01234521

- wiadomość odebrana: :01W01234521

2. Odczytaj dane z adresu z rejestru „0” w urządzeniu o adresie „02”

- wiadomość wysłana: :02R01E

- wiadomość odebrana: :02R012.3416

3. W przypadku błędu sumy kontrolnej w komunikacji z urządzeniem a dresie „01”

- wiadomość odebrana :01N019