POLITECHNIKA POZNAŃSKA

Laboratorium
Języki Specyfikacji i Opisu

Przesyłanie danych definiowanych za pomocą ASN.1

WYDZIAŁ
INFORMATYKI
I TELEKOMUNIKACJI

Przesyłanie danych definiowanych za pomocą ASN.1

Uwaga 1. W ćwiczeniach niezbędne są narzędzia ze środowiska pracy, które powinno być zainstalowane po pierwszych zajęciach.

Uwaga 2. Aby uzyskać dostęp do socketów w systemie Debian w narzędziu WSL w systemie Windows może być niezbędne utworzenie środowiska wirtualnego Python z przywilejem SU, a więc będzie trzeba poprzedzać polecenia poleceniem `sudo`. Należy utworzyć nowe środowisko wirtualne Python (podobnie do tworzenia środowiska wirtualnego Python z pierwszych zajęć laboratoryjnych). Jednakże tym razem należy poprzedzać polecenia poleceniem `sudo`. Następnie należy zainstalować w nim pakiet `asn1tools` (poprzedzając polecenie `pip3` poleceniem `sudo`).

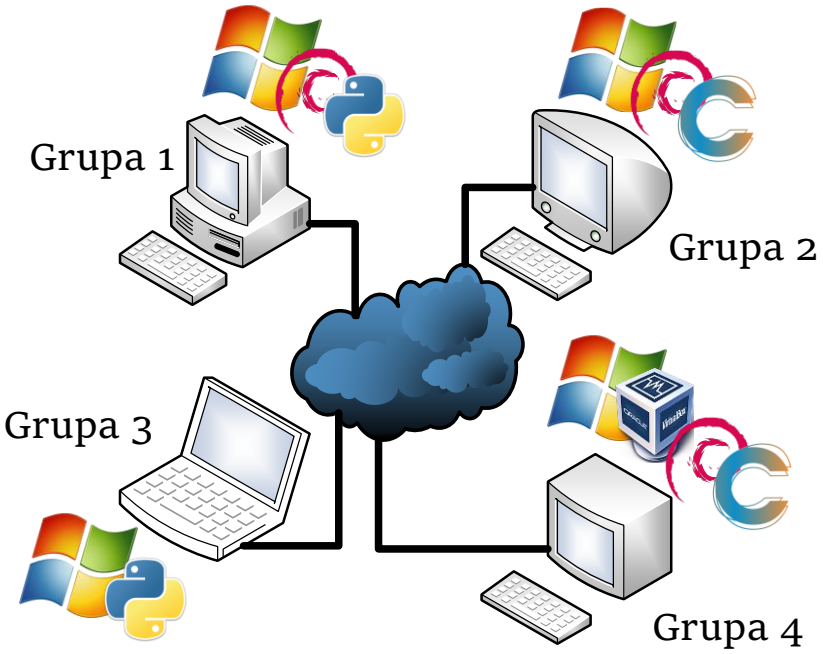
1. Specyfikacja prostej konstrukcji typu w ASN.1
    a. W ramach całej grupy ćwiczeniowej dokonać specyfikacji ASN.1 prostego typu. Definiowany typ powinien spełniać następujące wymagania:
        – znajdować się w jednym, poprawnie zbudowanym pliku z nazwą pliku tożsamą z nazwą definiowanego typu i rozszerzeniem asn1,
        – być strukturą z dwoma polami,
        – pierwsze pole jest nazwą typu łańcuch znaków ASCII,
        – drugie pole jest polem ładunkowym typu łańcuch oktetów.
    b. Zapewnić dostęp do pliku z definicją typu ASN.1 każdemu członkowi grupy ćwiczeniowej (zgranie na nośnik zewnętrzny, przesłanie pliku pocztą elektroniczną itp.).
2. Kodowanie BER
    a. Nadać przykładowe wartości w nowym typie:
        – do nazwy przypisać trzyliterowy łańcuch znaków,
        – do pola ładunkowego przypisać wartość ośmiobitową.
    b. Dokonać ręcznego kodowania BER dla tak utworzonej struktury przewidując ciąg bitów po kodowaniu. Zapamiętać przewidywany ciąg bitów (wartości heksadecymalne).

3. Podział grupy ćwiczeniowej

   a. Dokonać podziału grupy ćwiczeniowej na cztery równoliczne (z różnicą ±1 osoba) podgrupy.

   b. Dokonać wyboru/przydziału narzędzi do podgrup zgodnie ze schematem z Rys. 1:

      1. System Debian zainstalowany w podsystemie Linux w systemie Windows, w którym zainstalowano obsługę języka Python z narzędziem asn1tools,
      2. System Debian zainstalowany w podsystemie Linux w systemie Windows, w którym zainstalowano obsługę języka C z narzędziem asn1c,
      3. System Windows z zainstalowaną obsługą języka Python z narzędziem asn1tools,
      4. System Windows z VirtualBox z zainstalowaną maszyną wirtualną TASTE z obsługą języka C dla narzędzia asn1c.



Rys. 1. Schemat logiczny podziału narzędzi używanych w ćwiczeniu

   c. Sprawdzić i zapamiętać adresy IPv4, każdego z czterech komputerów i ewentualnie maszyn wirtualnych.

d. Sprawdzić działanie wybranych narzędzi zainstalowanych na każdym z komputerów.
   - W podgrupie numer 1 sprawdzić, czy nie trzeba dokonać instalacji środowiska wirtualnego Python zgodnie z Uwagą 2.

4. Tworzenie oprogramowania (zadanie w podgrupach)

A. Podgrupa 1 (Win→WSL→Debian→Python→asn1tools)
   - zbudowanie kodera i dekodera BER w języku Python z wykorzystaniem pakietu asn1tools,
   - napisanie programu serwera w języku Python wysyłającego i odbierającego zakodowane wiadomości według założeń z rozdziału Dodatki,
   - użyteczne informacje: https://pypi.org/project/asn1tools/

B. Podgrupa 2 (Win→WSL→Debian→C→asn1c)
   - zbudowanie kodera i dekodera BER w języku C z wykorzystaniem kompilatora asn1c,
   - napisanie programu serwera w języku C wysyłającego i odbierającego zakodowane wiadomości według założeń z rozdziału Dodatki,
   - użyteczne informacje: http://lionet.info/asn1c/blog/ https://github.com/vlm/asn1c

C. Podgrupa 3 (Win→Python→asn1tools)
   - zbudowanie kodera i dekodera BER w języku Python z wykorzystaniem pakietu asn1tools,
   - napisanie programu klienta w języku Python wysyłającego i odbierającego zakodowane wiadomości według założeń z rozdziału Dodatki,
   - użyteczne informacje: https://pypi.org/project/asn1tools/

D. Podgrupa 4 (Win→VirtualBox→TASTE(Debian)→C→asn1c)

- zbudowanie kodera i dekodera BER w języku C z wykorzystaniem kompilatora asn1c,
- napisanie programu klienta w języku C wysyłającego i odbierającego zakodowane wiadomości według założeń z rozdziału Dodatki,
- użyteczne informacje:
  http://lionet.info/asn1c/blog/
  https://github.com/vlm/asn1c

5. Sprawdzenie poprawności kodowania
   a. Po uruchomieniu funkcji kodujących i dekodujących w programach serwerów i klientów sprawdzić ciągi bitów po kodowaniu BER.
   b. Porównać ciągi z koderów z przewidywaniami.
6. Połączenia i transmisja
   a. Dokonać połączeń między serwerami i klientami. Należy pamiętać, aby uruchomić program serwera zanim podejmie się próbę dołączenia klienta do serwera.
   b. Sprawdzić poprawność działania programów klientów i serwerów oraz funkcji kodowania i dekodowania w różnych relacjach.
   c. Wyciągnąć wnioski.

Dodatki

A. Schemat blokowy dla serwera oprogramowanego z użyciem narzędzia asn1tools

Na Rys. 2 przedstawiono schemat blokowy dla serwera, w szczególności oprogramowanego z użyciem języka Python i narzędzia asn1tools.

W pierwszej kolejności należy skonfigurować stronę serwera podając:

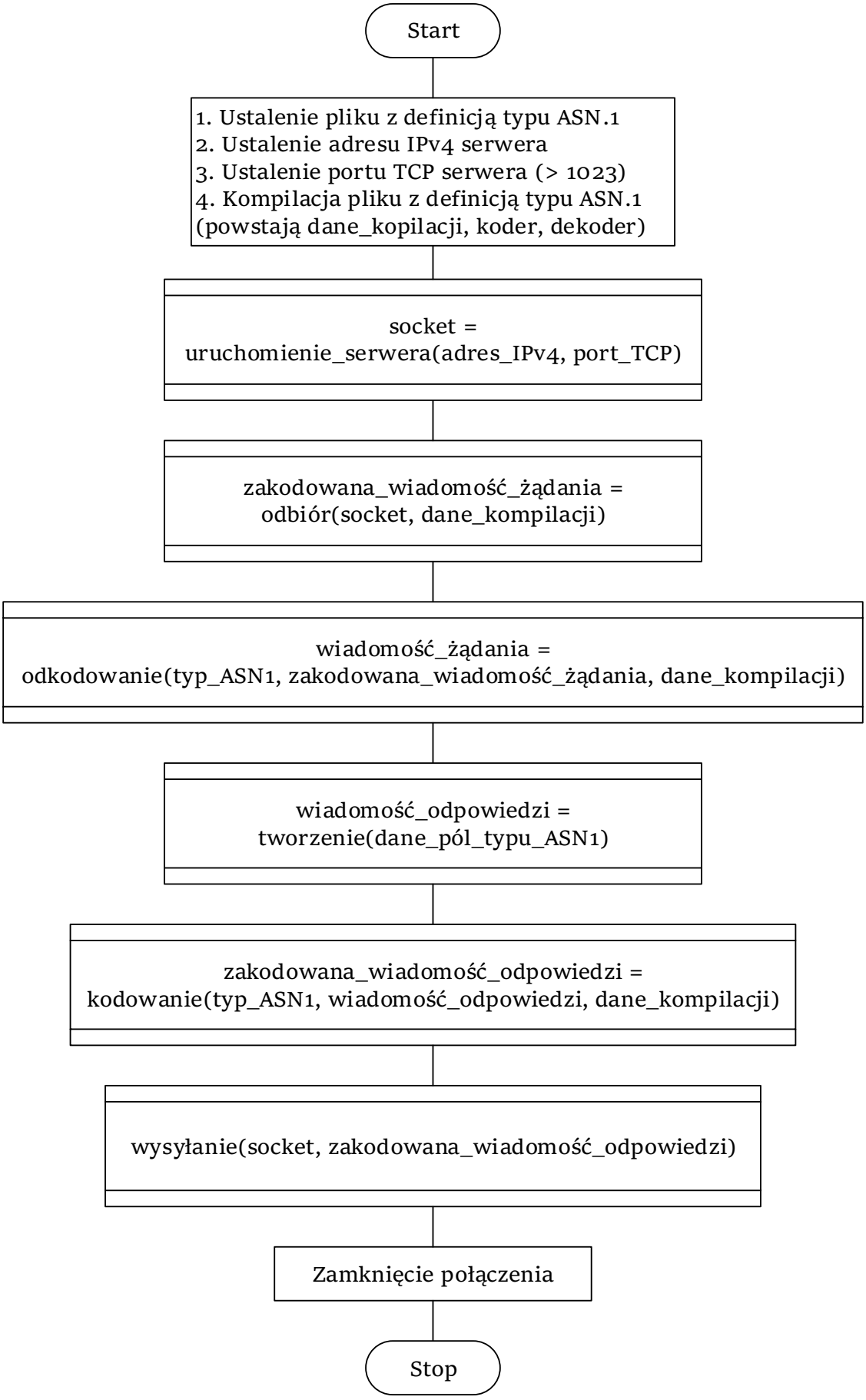
- Lokalizację pliku z definicją typu ASN.1, na podstawie której będą tworzone wiadomości wymieniane między serwerem i klientem (plik powstały według zasad podanych w punkcie 1 tej instrukcji). Uwaga: Strona serwera i strona klienta muszą posługiwać się tą samą definicją typu ASN.1.
- Adres IPv4 komputera, na którym zostanie uruchomiony program serwera. Uwaga: Adres serwera musi być znany i skonfigurowany także po stronie klienta.
- Numer portu TCP, po stronie serwera (większy niż 1023). Uwaga: Numer portu serwera musi być znany i skonfigurowany także po stronie klienta.

Dodatkowo należy skompilować plik z definicją typu korzystając z narzędzia asn1tools tworząc dane kompilacji, koder i dekoder dla nowego typu. W tym celu należy użyć funkcji:

```
asn1tools.compile_files(<lokalizacja_i_nazwa_pliku_ASN.1>)
```

Skonfigurowany serwer oczekuje na wiadomość od klienta. Klient musi mieć możliwość dołączenia się do uruchomionego serwera. W tym celu należy wykorzystać zasady oprogramowania socketów. W szczególności po stronie serwera należy wykonać sekwencję działań uruchamiających serwer i reagujących na dołączenie strony klienta. W funkcji `uruchomienie_serwera()` (patrz schemat blokowy):

- utworzyć obiekt gniazda,
- powiązać obiekt gniazda z adresem IPv4 serwera i numerem portu TCP,
- ustawić tak utworzony i powiązany obiekt gniazda w tryb nasłuchu,
- umożliwić serwerowi zaakceptowanie przychodzącego połączenia od klienta,
- zwrócić deskryptor gniazda, do którego dołączył się klient.



Rys. 2. Schemat blokowy algorytmu serwera oprogramowanego z użyciem asn1tools

Użyteczne informacje:

https://realpython.com/python-sockets/

Po dołączeniu klienta, należy umożliwić serwerowi odbieranie przesłanej przez klienta wiadomości (funkcja `odbiór()` ze schematu blokowego). Oczekiwaną przez serwer wiadomością jest ciąg bitów powstały po zakodowaniu typu ASN.1 według zasad kodowania BER. Można skorzystać z kodu przykładowej funkcji z .

```
def receiving(conn, db):
    """
    Funkcja odbierająca od klienta zakodowaną wiadomość ASN.1.

    conn - deskryptor połączenia do klienta

    db - dane, koder, dekoder po kompilacji pliku z definicją typu
    ASN.1

    encoded_message - zwracana zakodowana według zasady BER
    wiadomość ASN.1
    """
    print('Receiving message request from the client... ', end='')
    encoded_message = conn.recv(2)
    length = db.decode_length(encoded_message)
    encoded_message += conn.recv(length - 2)
    print('done.')
    print(encoded_message.hex())
    pprint(encoded_message)
    return encoded_message
```

Rys. 3. Procedura odbioru zakodowanej wiadomości w serwerze z użyciem asn1tools

Odebraną, zakodowaną wiadomość należy odkodować wykorzystując koder powstały po kompilacji pliku z typem ASN.1 (szczegóły w https://pypi.org/project/asn1tools/). Dekoder (składowa danych kompilacji) oczekuje informacji na temat typu ASN.1, który stanowi opis oczekiwanych składowych, oraz zakodowanej wiadomości, która będzie odkodowywana zgodnie z konstrukcją zapisaną w typie ASN.1. Zatem elementem funkcji odkodowującej (`odkodowywanie()` na schemacie blokowym) powinno być wywołanie funkcji dekodera w postaci:

```
message = dane_kompilacji.decode(<nazwa_typu_ASN1>,
                                 <zakodowana_wiadomość>)
```

która zwróci słownik zawierający informacje na temat wartości przesłanych w składowych typu ASN.1. Wiadomość można wyświetlić w postaci czytelnej dla użytkownika korzystając z funkcji `pprint()` (wymaga importu pakietu).

Reakcją serwera powinno być wysłanie wiadomości odpowiedzi. W pierwszej kolejności należy utworzyć wiadomość odpowiedzi (funkcja `tworzenie()` na schemacie blokowym), co polega na nadaniu wartości składowym typu ASN.1. W tym celu można skorzystać z przykładowego kodu z Rys. 4.

```
def message_compose(m_name, m_payload):
    """
    Funkcja nadająca konkretne wartości polom typu ASN.1

    Definicje pól znajdują się w pliku z definicją typu ASN.1

    parametry formalne wywołania - wartości poszczególnych pól

    message - zwracany typ ASN.1 wypełniony wartościami
    """
    message = {
     'name': m_name,
     'payload': m_payload
    }
    pprint(message)
    return message
```

Rys. 4. Procedura tworzenia wiadomości w serwerze z użyciem asn1tools

Aby wysłać wiadomość należy ją zakodować (funkcja `kodowanie()` na schemacie blokowym). W tym celu należy skorzystać z kodera, który został utworzony po kompilacji pliku z definicją typu ASN.1 (szczegóły w https://pypi.org/project/asn1tools/). Zasada wykorzystania kodera jest bardzo podobna do wcześniej przedstawionej zasady odkodowywania wiadomości. Należy jednak uwzględnić fakt, że koder wymaga informacji o kodowanej wiadomości, a w wyniku wykorzystania kodera powstaje zakodowana wiadomość. Wiadomość zakodowana koderem BER jest typu `bytes`. Można wyświetlić postać zakodowanej wiadomości korzystając z funkcji `pprint()`, a ciąg bitów (w postaci wartości heksadecymalnych) zakodowanej wiadomości można wyświetlić korzystając z metody `hex()` obiektu typu `bytes`.

Zakodowaną wiadomość odpowiedzi można wysłać (funkcja `wysyłanie()` na schemacie blokowym) do klienta wykorzystując deskryptor połączenia powstały po dołączeniu klienta według zasad wykorzystania socketów opisanych na przykład w:

https://realpython.com/python-sockets/

B. Schemat działania dla budowy serwera oprogramowanego z użyciem narzędzia asn1c

Wykorzystanie narzędzia asn1c jest bardziej skomplikowane niż narzędzia asn1tools. Wymaga wykonania zadań przygotowawczych i napisania bardziej skomplikowanego oprogramowania. Przed dalszym wykonaniem ćwiczenia należy zapoznać się szczegółowo z następującymi informacjami:

http://lionet.info/asn1c/basics.html

http://lionet.info/asn1c/documentation.html

http://lionet.info/asn1c/asn1c-usage.html

http://lionet.info/asn1c/examples.html

https://www.linuxhowtos.org/C_C++/socket.htm

Sekwencja działań jakie należy wykonać po uruchomieniu systemu Debian (WSL lub TASTE).

1. Utworzyć katalog (polecenie tworzenia katalogu `mkdir`) dla projektu kodera/dekodera.
2. Umieścić w nowo utworzonym katalogu plik z definicją typu ASN.1 (powstały w wyniku wykonania zadań opisanych w punkcie 1 tej instrukcji). Odpowiednio wykorzystać polecenia `cd` i `cp` systemu Debian. Z założenia plik powinien mieć rozszerzenie `.asn1`.
3. Będąc w katalogu z plikiem z definicją typu ASN.1 skompilować plik wykorzystując kompilator asn1c:

```
<ścieżka_do_programu_asn1c>/asn1c ./<nazwa_pliku_z_typem_ASN1>.asn1
```

4. Sprawdzić, czy w katalogu zostały umieszczone łącza do plików lub same pliki (działanie uzależnione od przełączników kompilatora asn1c – szczegóły w dokumentacji narzędzia) z rozszerzeniem `.c` i `.h`. Wśród plików powinny być między innymi pliki `<nazwa_typu_ASN1>.c` oraz `<nazwa_typu_ASN1>.h`, gdzie `<nazwa_typu_ASN1>` oznacza nazwę typu ASN.1, który jest zdefiniowany w pliku `.asn1`. Dodatkowo w katalogu powinien znajdować się plik `converter-sample.c`. Przykładowy plik zawiera funkcję `main()` i może być wykorzystany do szybkiego sprawdzenia wykorzystania kodera i dekodera, zgodnie z opisem znajdującym się w dokumencie „Quick start sheet” dostępnym pod adresem:

   http://lionet.info/asn1c/documentation.html

5. Zmienić rozszerzenie pliku `converter-sample.c` na inne lub usunąć plik z katalogu projektu.

6. Otworzyć edytor tekstów. W systemie Debian w maszynie TASTE można wykorzystać okienkowy edytor (na przykład Kate). W WSL wydać polecenie uruchamiające edytor tekstów, na przykład wydając polecenie:

```
nano <nazwa_pliku>.c
```

7. Na podstawie przykładów ze strony http://lionet.info/asn1c/examples.html napisać program kodera i zapisać go. Należy uwzględnić fakt, że przykład nie dotyczy typów składowych pól, które są używane w ćwiczeniowym typie ASN.1 (patrz punkt 1 tej instrukcji). Jak odwzorowane są odpowiednie typy ASN.1 w typy danych języka C można zobaczyć w odpowiednich plikach z rozszerzeniem `.c` i `.h` w katalogu projektu. Na przykład typ ASN.1 `BIT STRING` jest opisany w plikach `BIT_STRING.c` i `BIT_STRING.h`. Na podstawie informacji z plików opisujących poszczególne typy danych można odpowiednio zmodyfikować przykładowy program kodera, tak aby nadać wartości polom ćwiczeniowego typu ASN.1.
8. Skompilować program z pliku z rozszerzeniem `.c` do pliku wykonywalnego. Należy przestrzegać ograniczenia, że w katalogu może znajdować się tylko jeden plik z rozszerzeniem `.c`, w którym może znajdować się funkcja `main()`. Powinien być to plik z kodem kodera. Należy użyć przełącznika kompilatora `-I` ze wskazaniem bieżącego katalogu, aby kompilator języka C uwzględnił pliki nagłówkowe z bieżącego katalogu. Polecenie ma postać:

   ```
   gcc -I. -o <nazwa_kodera> *.c
   ```

   gdzie `<nazwa_kodera>` jest nazwą pliku wykonywalnego kodera.

9. Należy sprawdzić informacje zwrotne z kompilatora. Pojawiające się ostrzeżenia można w większości zignorować. Jednakże błędy uniemożliwią powstanie pliku wykonywalnego. Kompilacja musi zakończyć się bez błędów. Jeżeli błędy występują należy zapoznać się z informacjami o ich charakterze i usunąć je.
10. Po kompilacji bez błędów uruchomić plik wykonywalny kodera (jak każdy inny program wykonywalny w systemie Debian poprzedzić nazwę pliku wykonywalnego ścieżką dostępu, nawet jeżeli program wywoływany jest z tego samego katalogu).
11. Sprawdzić wynik działania programu. Wyciągnąć wnioski.
12. Na podobnej zasadzie napisać program dekodera. Zadbać o to aby w katalogu był tylko jeden plik z rozszerzeniem `.c` i funkcją `main()`.
13. Po kompilacji bez błędów, uruchomić plik wykonywalny dekodera. Sprawdzić wynika działania programu i wyciągnąć wnioski.

14. Utworzyć nowy katalog dla projektu. Umieścić w katalogu plik z definicją typu ASN.1.
15. Wykonać polecenia z punktów B.3–B.6.
16. Zredagować plik z kodem źródłowym serwera korzystając z przykładu z Rys. 5–Rys. 10.

```
#include <sys/socket.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#include <Message.h>   /* Message ASN.1 type  */

#define PORT 5000
#define BUF_SIZE 1024
```

Rys. 5. Część deklaracji programu serwera w języku C

```
int main(int argc, char** argv)
{
    int                 connfd;   // Socket descriptor

    Message_t           *message = 0; // Type to encode
    int                 bytesReceived = 0;
    char                buff[BUF_SIZE];

    char                *name_v;
    char                payload_v;

    connfd = run_server();

    memset(buff, '0', sizeof(buff));

    bytesReceived = receiving(connfd, buff);
    decoding(bytesReceived, buff);

    printf("----------END OF RECEIPT-----------------\n");
    printf("-------START OF TRANSMISSION------------\n");

    name_v = "ACK";
    payload_v = 'V';

    message = calloc(1, sizeof(Message_t)); // not malloc
    if(!message)
    {
      perror("calloc() failed!");
      exit(71); /* better, EX_OSERR */
    }

    compose_message(message, name_v, payload_v);

    encoding_sending(connfd, message);
}
```

Rys. 6. Główna funkcja programu serwera w języku C

```
int run_server()
{
    int                 listenfd, connfd; // Socket descriptors
    struct sockaddr_in  serv_addr;

    // Creating socket file descriptor
    if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {
        perror("socket failed!");
        return -1;
    }
    printf("Socket retrieve success.\n");

    memset(&serv_addr, '0', sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(PORT);

    if (bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0)
    {
        perror("bind failed!");
        return -1;
    }

    printf("Bind success.\n");

    if(listen(listenfd, 3) == -1)
    {
        printf("listen failed!\n");
        return -1;
    }

    char clntIP[100];
    int clilen;
    struct sockaddr_in cli_addr;

    connfd = accept(listenfd, (struct sockaddr *) &cli_addr, &clilen);

    inet_ntop(AF_INET, (struct in_addr *)&cli_addr.sin_addr, clntIP,
              sizeof(clntIP));
    printf("Connected from client IP: %s, port: %d\n", clntIP,
              htons(cli_addr.sin_port));

    return connfd;
}
```

Rys. 7. Funkcja uruchomienia serwera w języku C

```
int receiving(int connfd, char *buff)
{
    int i;
    int bytesReceived = 0;

    if ((bytesReceived = read(connfd, buff, BUF_SIZE)) > 0)
    {
        printf("Bytes received: %d\n", bytesReceived);
    }

    if(bytesReceived < 0)
    {
        printf("\n Read Error \n");
    }
    else
    {
        printf("Received: ");
        for (i = 0; i < bytesReceived; i++)
        {
            printf("%02X", buff[i]);
        }
        printf("\n");
    }

    return bytesReceived;
}
```

Rys. 8. Funkcja odbioru danych z socketu w języku C

```
void decoding(int bytesReceived, char *buff)
{
    Message_t       *message = 0; // Type to decode
    asn_dec_rval_t  rval;    // Decoder return value

    char            *name_v;
    char            payload_v;

    // Decode the input buffer as Message type
    rval = ber_decode(0, &asn_DEF_Message,
      (void **)&message, buff, bytesReceived);

    // Print the decoded Message type
    asn_fprint(stdout, &asn_DEF_Message, message);

    name_v = message->name.buf;
    payload_v = message->payload.buf[0];

    fprintf(stdout, "{ %s, %c }\n", name_v, payload_v);

    if(rval.code != RC_OK)
    {
      fprintf(stderr, "Broken Message encoding at byte %ld\n",
              (long)rval.consumed);
      exit(65); // better, EX_DATAERR
    }

    // Print the decoded Message type as XML
    xer_fprint(stdout, &asn_DEF_Message, message);
}
```

Rys. 9. Funkcja dekodowania odebranych danych w języku C

```c
/*
 * Funkcja zapisujaca zakodowane dane do socketu
 * Numer socketu przekazany jako app_key
 * Tu musi byc rzutowany na void *
 * Ale w funkcji write musi wrocic do int
 *
 * Zostawiono takze dane dla zapisywania do pliku
 */
static int
//write_out(const void *buffer, size_t size, void *app_key)
sending(const void *buffer, size_t size, void *app_key)
{
    int i;
    //FILE *out_fp = app_key;
    size_t wrote;

    //wrote = fwrite(buffer, 1, size, out_fp);

    wrote = write((int)app_key, buffer, size);
    printf("Sending done.\n");

    return (wrote == size) ? 0 : -1;
}


void encoding_sending(int connfd, Message_t *message)
{
    asn_enc_rval_t ec;      // Encoder return value

    ec = der_encode(&asn_DEF_Message, message, sending, (void *)connfd);

    asn_fprint(stdout, &asn_DEF_Message, message);

    if(ec.encoded == -1)
    {
        fprintf(stderr,
          "Could not encode Message (at %s)\n",
          ec.failed_type ? ec.failed_type->name : "unknown");
        exit(65); // better, EX_DATAERR
    }
    else
    {
        fprintf(stdout, "Created message with BER encoded Message\n");
    }

    // Also print the constructed Message XER encoded (XML)
    xer_fprint(stdout, &asn_DEF_Message, message);
}
```

Rys. 10. Funkcje kodowania i wysyłania danych w języku C

17. Uzupełnić kod funkcji `compose_message()` (Rys. 11) na podstawie doświadczeń z kodem kodera i dekodera.

```c
/* Funkcja tworzaca wiadomoc typu Message */
void compose_message(Message_t *message, char *name_v, char payload_v)
{
    …
}
```

Rys. 11. Funkcja do uzupełnienia

18. Po kompilacji bez błędów, uruchomić plik wykonywalny serwera. Uruchomić program i wyciągnąć wnioski.

C. Schemat blokowy dla klienta oprogramowanego z użyciem narzędzia asn1tools

Na Rys. 12 przedstawiono schemat blokowy dla klienta, w szczególności oprogramowanego z użyciem języka Python i narzędzia asn1tools. W pierwszej kolejności należy skonfigurować stronę klienta podając te same informacje na temat serwera jak w oprogramowaniu serwera. Dodatkowo należy skompilować plik z definicją typu korzystając z narzędzia asn1tools tworząc dane kompilacji, koder i dekoder dla nowego typu, według zasad opisanych w oprogramowaniu serwera.

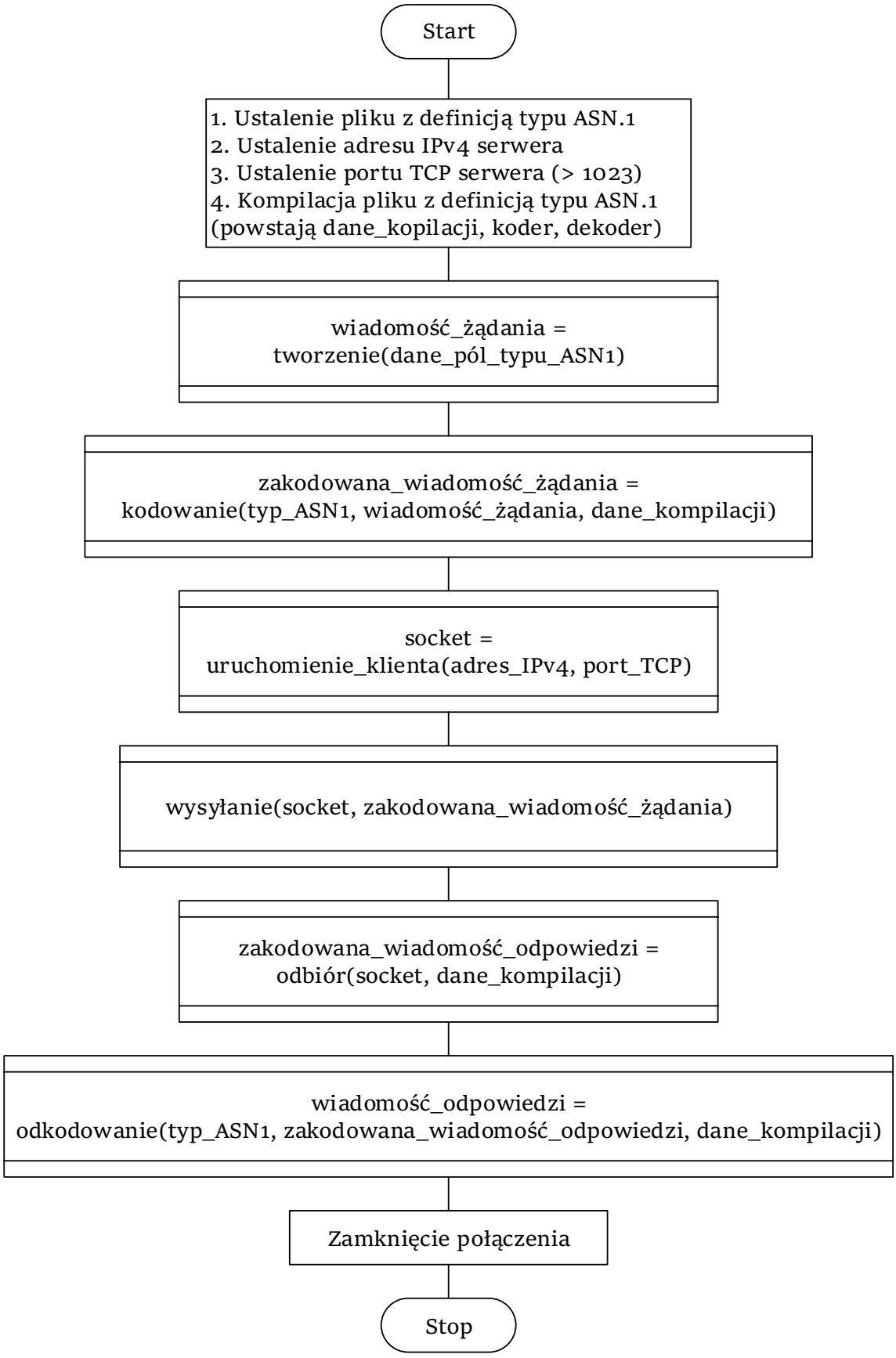
Funkcje tworzenia, kodowania, wysyłania, odbioru i odkodowywania wiadomości są bardzo podobne lub takie same jak w oprogramowaniu serwera.

Podstawowa różnica między oprogramowaniem serwera i klienta dotyczy uruchomienia klienta. Funkcja uruchomienia klienta powinna:

- utworzyć obiekt gniazda,
- połączyć klienta z serwerem uruchomionym pod adresem IPv4 i z numerem portu TCP,
- zwracać deskryptor gniazda, przez który klient komunikuje się z serwerem.

Użyteczne informacje:

https://realpython.com/python-sockets/



Rys. 12. Schemat blokowy algorytmu klienta oprogramowanego z użyciem asn1tools

D. Schemat działania dla budowy klienta oprogramowanego z użyciem narzędzia asn1c

1. Wykonać polecenia z punktów od B.1–B.15.
2. Zredagować plik z kodem źródłowym klienta korzystając z przykładów z Rys. 8, Rys. 9, Rys. 10, Rys. 13, Rys. 14. Uwaga. Należy zadbać o prawidłowy adres IPv4 serwera.
3. Uzupełnić kod funkcji `compose_message()` (Rys. 11) na podstawie doświadczeń z kodem kodera i dekodera.
4. Po kompilacji bez błędów, uruchomić plik wykonywalny serwera. Uruchomić program i wyciągnąć wnioski.

```
int main(int argc, char** argv)
{
    int                 sockfd;   // Socket descriptor

    Message_t           *message = 0; // Type to encode

    int                 bytesReceived = 0;
    char                buff[BUF_SIZE];

    char                *name_v;
    char                payload_v;

    sockfd = run_client();

    name_v = "REQ";
    payload_v = 'U';

    message = calloc(1, sizeof(Message_t)); // not malloc
    if(!message)
    {
      perror("calloc() failed!");
      exit(71); /* better, EX_OSERR */
    }

    compose_message(message, name_v, payload_v);

    encoding_sending(sockfd, message);

    printf("----------END OF TRANSMISSION-------------\n");
    printf("-----------START OF RECEIPT---------------\n");

    memset(buff, '0', sizeof(buff));

    bytesReceived = receiving(sockfd, buff);
    decoding(bytesReceived, buff);
}
```

Rys. 13. Główna funkcja programu klienta w języku C

```
int run_client()
{

    int                 sockfd = 0;
    struct sockaddr_in  serv_addr;

    char *server_address = "192.168.0.109";

    // Create a socket first
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Error: Could not create socket\n");
        return -1;
    }

    // Initialize sockaddr_in data structure
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT); // port
    serv_addr.sin_addr.s_addr = inet_addr(server_address);

    // Attempt a connection
    if(connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("\n Error : Connect failed!\n");
        return -1;
    }
    else
    {
        printf("Server at %s contacted and connected.\n", server_address);
    }

    return sockfd;
}
```

Rys. 14. Funkcja uruchomienia klienta w języku C