Instytut Teleinformatyki

Wydział Inżynierii Elektrycznej i Komputerowej
Politechnika Krakowska

programowanie usług sieciowych

„Protokół SCTP”

laboratorium: 04
system operacyjny: Linux

Kraków, 2009

Spis treści

1. Wiadomości wstępne

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące **protokołu SCTP** oraz programowania gniazd z jego wykorzystaniem. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

1.1. Tematyka laboratorium

Tematyką laboratorium jest programowanie aplikacji klient-serwer w oparciu o protokół SCTP oraz sprawdzenie jego właściwości. SCTP (ang. *Stram Control Transmission Protocol)* jest protokołem warstwy transportowej modelu ISO/OSI opracowanym przez organizację standaryzacyjną IETF. Początkowym założeniem projektowym SCTP było umożliwienie efektywnej transmisji wiadomości sygnalizacyjnych PSTN (ang. *Public Switched Telephone Network*) przez sieci wykorzystujące protokół IP.
W trakcie trwania prac nad SCTP zdecydowano, że będzie on protokołem ogólnego przeznaczenia. SCTP posiada szereg cech zapożyczonych z niezawodnego, zorientowanego połączeniowo protokołu TCP oraz bezpołączeniowego, niezapewniającego niezawodności UDP.
Protokół SCTP oferuje m.in.:

- wsparcie dla hostów z wieloma interfejsami (tzw. *multihoming*),
- wsparcie dla komunikacji za pomocą wielu strumieni w jednym połączeniu (asocjacji),
- niezawodny transport danych z selektywnymi potwierdzeniami,
- przesyłanie komunikatów (podobnie jak w przypadku UDP, ale z zapewnieniem niezawodności),
- mechanizmy kontroli przepływu danych i unikania przeciążeń,
- możliwość wymiany komunikatów bez zachowania kolejności,
- wykrywanie MTU ścieżki,
- mechanizm zabezpieczający przed atakami DoS (ang. *Denial of Service*).

1.2. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi:

- budowy modelu ISO/OSI
- obsługi programu tcpdump
- stosowania gniazd w systemie Linux [1]
- atakami typu SYN Flood [10]
- TCP SYN cookies [8, 9]

Należy także zapoznać się z zagadnieniami dotyczącymi **protokołu SCTP**: [1 – 7]

- budowa nagłówków SCTP
- ustanawianie połączenia (*4-way handshake*)
- strumienie i asocjacje SCTP
- mechanizmy zapewniające niezawodną transmisję danych
- funkcje gniazd SCTP
- opcje `SCTP_INITMSG` oraz `SCTP_STATUS`

Literatura:

[1] W.R. Stevens, *„Programowanie Usług Sieciowych”*, *„API: gniazda i XTI”*.
[2] IETF (http://www.ietf.org/), RFC 2960, *„Stream Control Transmission Protocol”*
[3] IETF, RFC 3257, *„Stream Control Transmission Protocol Applicability Statement”*
[4] IETF, RFC 3286, *„An Introduction to the Stream Control Transmission Protocol”*
[5] IETF, RFC 3309, *„Stream Control Transmission Protocol Checksum Change”*
[6] Internet Society (http://www.isoc.org/briefings/017/), *„Why is SCTP needed given TCP and UDP are widely available?”*
[7] IBM (http://www.ibm.com/developerworks/linux/library/l-sctp/), *„Better networking with SCTP”*
[8] D. J. Bernstein, *„TCP SYN Cookies”*, http://cr.yp.to/syncookies.html
[9] LWN.net, *„Improving syncookies”*, http://lwn.net/Articles/277146/
[10] CERT (http://www.cert.org/), Advisory CA-1996-21, *„TCP SYN Flooding and IP Spoofing Attacks”*

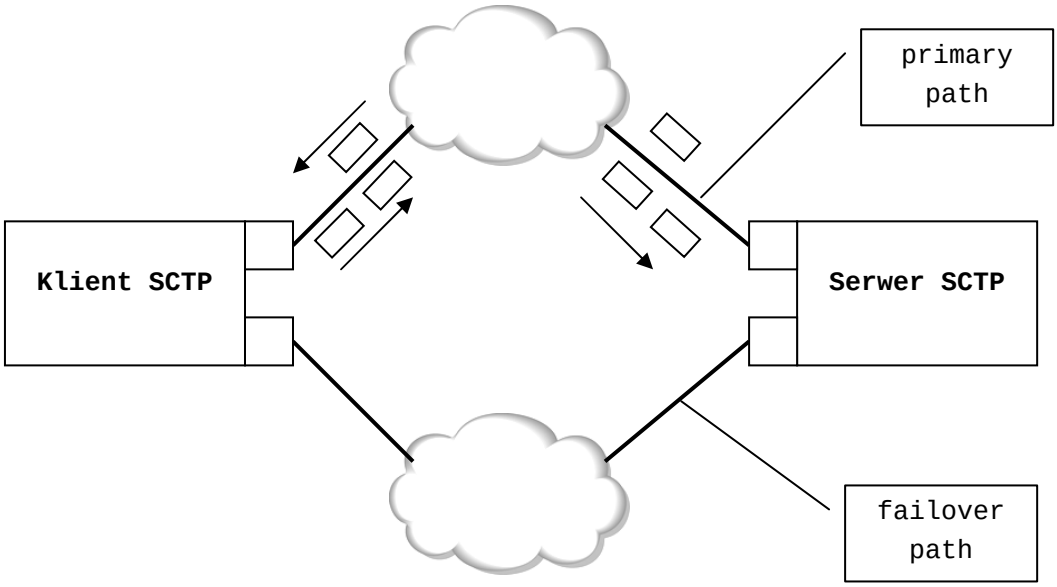
1.3. Opis laboratorium

1.3.1. Protokół SCTP

SCTP jest niezawodnym, zorientowanym połączeniowo protokołem warstwy transportowej modelu ISO/OSI. Jego implementacja w jądrze systemu Linux bazuje na *RFC 2960* oraz *RFC 3309*. W stosunku do protokołów TCP i UDP, SCTP posiada kilka nowych właściwości i usprawnień.

Właściwości protokołu SCTP:

1. SCTP jest protokołem połączeniowym. Aby mogła nastąpić komunikacja, konieczne jest ustanowienie połączenia. Połączenie między dwoma stacjami sieciowymi nazywa się asocjacją i jest zestawiane poprzez wymianę czterech wiadomości (*4-way handshake*).
2. Podczas ustanawiania asocjacji, SCTP wykorzystuje mechanizm *cookies, co* zabezpiecza serwer przed atakami typu DoS.
3. SCTP wspiera wyłącznie transmisję typu *unicast*, tzn. transmisję pomiędzy dokładnie dwoma węzłami sieciowymi (każdy węzeł może być reprezentowany przez wiele adresów IP).
4. Protokół SCTP zapewnia wsparcie dla stacji sieciowych z wieloma interfejsami (ang. *multihoming*). Podczas ustanawiania asocjacji stacje sieciowe wymieniają listy adresów IP, które będą mogły zostać wykorzystane do komunikacji. Jeden adres IP - określany jako podstawowy (ang. *primary*) – wykorzystywany jest do transmisji wszystkich danych użytkownika. Pozostałe adresy mogą zostać wykorzystane w przypadku, gdy podstawowy adres stanie się nieosiągalny lub w przypadku retransmisji danych. W obecnej formie protokół SCTP nie zakłada wykorzystania *multihomingu* w celu równoważenia obciążenia (ang. *load balancing*). Wykorzystanie wielu adresów IP (skonfigurowanych na wielu interfejsach fizycznych) zwiększa natomiast prawdopodobieństwo przetrwania asocjacji w przypadku awarii połączenia lub awarii interfejsu sieciowego.
   Protokół SCTP monitoruje stan połączeń sieciowych i w przypadku awarii jednego połączenia, wysyła dane alternatywną trasą. Innymi słowy, jeżeli transmisja między dwoma stacjami sieciowymi może odbywać się z wykorzystaniem wielu niezależnych tras, to asocjacja staje się tolerancyjna na awarie sieci.
   Multihoming zwiększa niezawodność, a co za tym idzie dostępność usług.



Rys. 1. Asocjacja SCTP wykorzystująca dwie niezależne sieci.

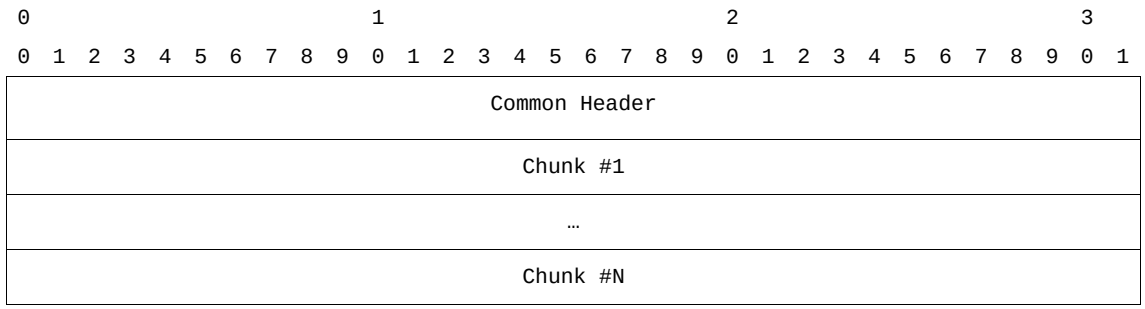
5. Protokół SCTP dopuszcza istnienie wielu strumieni w ramach jednej asocjacji (ang. *multistreaming*). Strumień jest niezależnym, **jednokierunkowym** kanałem komunikacyjnym. Każda stacja sieciowa przypisuje strumieniowi unikalny numer. Numer ten ma znaczenie **lokalne**, tzn. dwie stacje komunikujące się za pomocą asocjacji SCTP mogą utworzyć strumienie o takim samym numerze. Interpretacja numerów strumieni, czy określenie zależności między nimi należy do aplikacji. Aplikacja może na przykład traktować dwa strumienie o tym samym numerze (wychodzący i przychodzący) jako jeden strumień dwukierunkowy. Podczas ustanawiania asocjacji, aplikacja określa liczbę strumieni wychodzących - OS (ang. *outbound streams*) oraz maksymalną liczbę strumieni przychodzących - MIS (ang. maximum *inbound streams*), którą jest w stanie zaakceptować. Po poprawnym ustanowieniu asocjacji, numer strumienia dla każdej ze stacji może należeć do zakresu od 0 do min(liczba OS stacji lokalnej, liczba MIS stacji zdalnej) - 1.
6. Dane przesyłane są w postaci komunikatów (jak datagramy UDP), nie w postaci strumienia bajtów (jak w przypadku TCP). Jeżeli klient przesyła do serwera wiadomość o rozmiarze 100 bajtów, to serwer odbierze dokładnie 100 bajtów - za pomocą jednej niepodzielnej operacji. Zatem w strumieniu transmitowane są komunikaty o określonych rozmiarach, a aplikacja nie musi wprowadzać rekordów określających granice poszczególnych wiadomości.
   W razie potrzeby SCTP przeprowadza fragmentację wysyłanych wiadomości. Aplikacja pełniąca rolę odbiorcy może odebrać tylko pełną wiadomość – po defragmentacji. Nie jest możliwa sytuacja, w której aplikacja odczytuje wiadomość we fragmentach.
7. Komunikaty transportowane są z zapewnieniem niezawodności i kontroli przepływu. SCTP wykrywa pakiety zduplikowane oraz pakiety, w których dane uległy przekłamaniu. Integralność danych jest zapewniona dzięki sumie kontrolnej obliczanej za pomocą algorytmu CRC32C (pole sumy kontrolnej zajmuje 32 bity, w TCP 16 bitów).
8. SCTP wykorzystuje mechanizm selektywnych potwierdzeń i retransmisji. W przypadku protokołu TCP selektywne potwierdzenia są rozszerzeniem protokołu, natomiast każda implementacja SCTP musi zapewniać dla nich wsparcie.
9. Transmisja komunikatów może odbywać się:
   a. z zachowaniem kolejności komunikatów w strumieniu (aplikacja nie może odebrać komunikatu N strumienia S, jeżeli do stacji sieciowej nie dotarł choćby jeden komunikat o numerze mniejszym od N i należący do strumienia S),
   b. bez zachowania kolejności. Właściwość ta może zostać wykorzystana przez aplikacje, dla których kolejność komunikatów nie ma znaczenia (komunikaty mogą być przetwarzane niezależne od siebie).
10. Strumienie są niezależnymi kanałami transmisyjnymi. Zakłócenie przesyłania danych w jednym strumieniu nie wpływa na inne. Przykład: Aplikacja komunikuje się za pomocą 2 strumieni. Komunikaty są transmitowane z zachowaniem

kolejności względem strumienia, w którym są przesyłane (punkt 8.a). Jeżeli w strumieniu pierwszym pakiet N uległ zagubieniu, to aplikacja nie może odebrać i przetworzyć kolejnych pakietów (N+1, N+2, …), nawet jeżeli zostały one odebrane przez stację sieciową i są przechowywane w buforze. Przetwarzanie pakietów zostało wstrzymane do czasu retransmisji pakietu N. Tym samym został zablokowany strumień pierwszy, podczas gdy dane należące do strumienia drugiego mogą być odebrane i przetwarzane bez opóźnień. Sytuacja opisana w tym punkcie nosi nazwę *head-of-line blocking.* Protokół TCP jest podatny na tego typu problemy - z uwagi na fakt istnienia tylko jednego kanału komunikacyjnego pomiędzy nadawcą i odbiorcą.

11. SCTP nie zezwala na połączenia pół-zamknięte (ang. *half-closed*). Jeżeli dany host zainicjuje zamknięcie asocjacji, to zdalna stacja musi również uczestniczyć w poprawnym jej zamknięciu:
    a. zakończyć przyjmowanie danych przez interfejs między użytkownikiem a implementacją SFTP (takim interfejsem są gniazda),
    b. wysłać wiadomości z kolejki danych oczekujących na wysłanie,
    c. oczekiwać na potwierdzenia wysłanych danych, ale nie akceptować nowych,
    d. zamknąć asocjację.
12. W protokole SCTP nie występuje stan TIME_WAIT (w porównaniu z TCP). Związane jest to z następującymi faktami:
    a. Zamknięcie asocjacji przebiega inaczej niż w przypadku zamknięcia połączenia TCP (punkt 11).
    b. Dzięki polu *Verification Tag* wspólnego nagłówka pakietu SCTP, nie istnieje problem związany z pakietami poprzedniej inkarnacji asocjacji. Każda ze stron spodziewa się określonej wartości pola *Veryfication Tag* w pakietach otrzymanych od partnera komunikacji. Pakiety z nieprawidłową wartością są odrzucane.

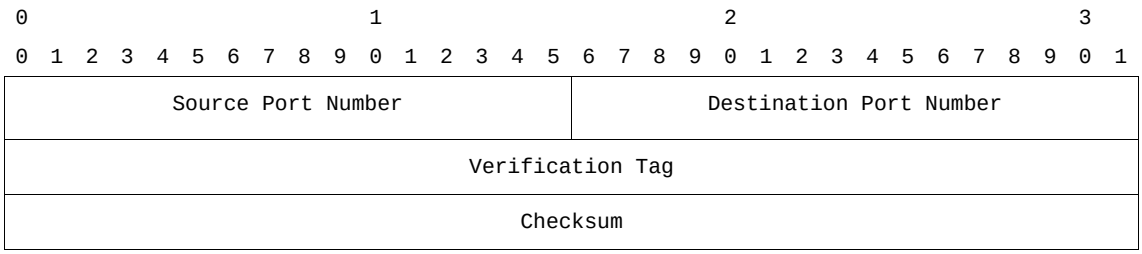
1.3.2. Budowa pakietu SCTP

Pakiet SCTP składa się ze wspólnego nagłówka (ang. *Common Header*) oraz jednej lub więcej jednostek *Chunk* opatrzonych własnym nagłówkiem.
Budowę wspólnego nagłówka przedstawia poniższy rysunek:



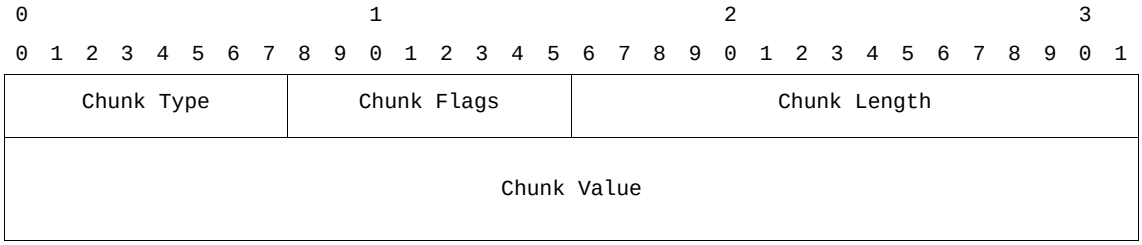
Rys. 2. Budowa pakietu SCTP[2].

Poniżej przedstawiona jest postać wspólnego nagłówka SCTP:



Rys. 3. Postać nagłówka *Common Header*[2].

Zastosowanie pola *Verification Tag* opisane zostało w rozdziale 1.3.1, punkt 12. Kolejny rysunek prezentuje budowę jednostek *Chunk*:



Rys. 4. Nagłówek jednostek *chunk*[2].

- *Chunk Type* – identyfikuje typ informacji przenoszonej w polu *Chunk Value.* Najczęściej spotykane wartości przedstawia poniższa tabela:

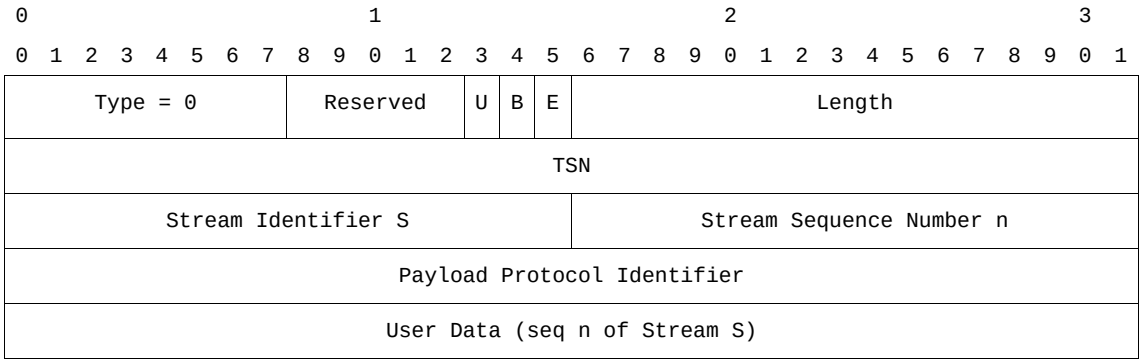
ID	Chunk Type	ID	Chunk Type
0	Payload Data (DATA)	7	Shutdown (SHUTDOWN)
1	Initiation (INIT)	8	Shutdown Acknowledgement (SHUTDOWN ACK)
2	Initiation Acknowledgement (INIT ACK)	9	Operation Error (ERROR)
3	Selective Acknowledgement (SACK)	10	State Cookie (COOKIE ECHO)
4	Heartbeat Request (HEARTBEAT)	11	Cookie Acknowledgement (COOKIE ACK)
5	Heartbeat Acknowledgement (HEARTBEAT ACK)	14	Shutdown Complete (SHUTDOWN COMPLETE)
6	Abort (ABORT)		

- *Chunk Flags* – zawartość tego pola zależy od typu jednostki *chunk*.
- *Chunk Length* – rozmiar jednostki chunk w bajtach. Do rozmiaru wliczane są pola: *Chunk Type*, *Chunk Flags*, *Chunk Length* oraz *Chunk Value*, natomiast padding (jeżeli występuje) nie jest wliczany. Rozmiar jednostek *chunk* musi być wielokrotnością 4 bajtów. Jeżeli jednostka *chunk* nie spełnia tego założenia,

nadawca musi dodać padding (wyzerowane bajty, maksimum 3). Padding jest ignorowany przez odbiorcę.

- *Chunk Value* (zmiennej długości) – właściwa informacja transmitowana w jednostce *chunk*. Zastosowanie i format tego pola zależy od typu jednostki *chunk*. Pole *Chunk Value* jednostek zawierających informacje kontrolne SCTP dodatkowo składa się z:
    - nagłówka, którego format zależy od typu jednostki *chunk*,
    - opcjonalnych parametrów w formie TLV (ang. *Type-Length-Value*).

Poniżej przedstawiona jest postać jednostki ***DATA chunk*** pozwalającej na przesyłanie danych użytkownika:



Rys. 5. Format jednostki *DATA chunk*[2].

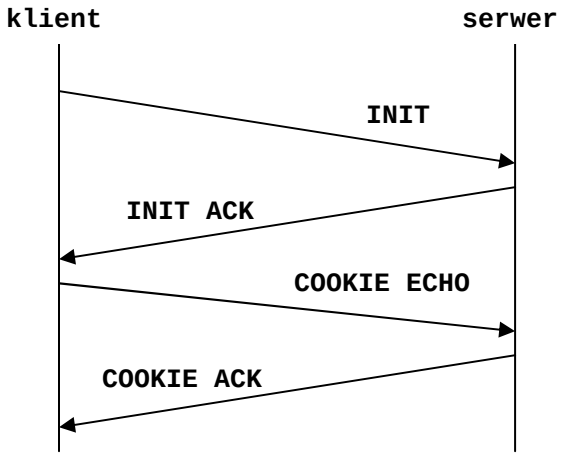
- *(U)nordered bit* – jeżeli ustawiony na 1, to odbiorca powinien zignorować wartość pola *Stream Sequence Number* i przekazać dane do wyższych warstw bez gwarancji zachowania kolejności.
- *(B)eginning bit* – jeżeli ma wartość 1, wskazuje, że jest to pierwszy fragment wiadomości użytkownika.
- *(E)nding bit* - jeżeli ma wartość 1, wskazuje, że jest to ostatni fragment wiadomości użytkownika.
  Wiadomość nie poddana fragmentacji powinna mieć bity B oraz E ustawione na wartość 1. Fragment nie będący pierwszym ani ostatnim, powinien mieć oba bity ustawione na 0.
- *Length* – rozmiar *DATA chunk* w bajtach licząc od pola *Type* do końca danych użytkownika (*User Data*).
- *TSN* (*Transmission Sequence Number*) - umożliwia detekcję zduplikowanych oraz zgubionych pakietów. Potwierdzenia (ang. *acknowledgements*) wysyłane przez odbiorcę bazują na *TSN*. SCTP przypisuje TSN do komunikatu lub jego fragmentu (dzięki TSN możliwe jest odtworzenie komunikatu z fragmentów). TSN jest niezależny od *Stream Sequence Number*.
- *Stream Identifier* – identyfikuje strumień, do którego należą dane.
- *Stream Sequence Number* – numer sekwencyjny danych w strumieniu identyfikowanym przez *Stream Identifier*. Numer sekwencyjny może przyjmo-

wać wartości od 0 do 65535. *SSN* jest używany przez odbiorcę do określenia kolejności komunikatów w danym strumieniu. Jeżeli komunikat poddawany jest fragmentacji, to każdy fragment komunikatu będzie posiadał ten sam numer sekwencyjny *Stream Sequence Number*.

- *Payload Protocol Identifier* – identyfikuje protokół wyższej warstwy (obecnie brak zastosowań; pole wyzerowane).
- *User Data* - dane użytkownika. Rozmiar danych musi być wielokrotnością 4 bajtów. Jeżeli warunek ten nie jest spełniony, nadawca musi dodać padding (wyzerowane bajty, maksimum 3). Padding nie jest uwzględniany przez wartość pola *Length*.

1.3.3. Ustanawianie asocjacji SCTP

SCTP jest protokołem połączeniowym – definiuje procedury popranego ustanowienia oraz zamknięcia asocjacji. Ustanowienie asocjacji odbywa się za pomocą minimum 4 pakietów SCTP, stąd nazwa: *4-way handshake*.



Rys. 6. *4-way handshake* protokołu SCTP.

Przebieg *4-way handshake*:

1. Klient wysyła pakiet SCTP z *INIT chunk*. Za pomocą jednostki INIT klient ustala wartość pola *Verification Tag* jakiej spodziewa się we wszystkich pakietach odebranych od serwera. Ponadto klient powiadamia serwer o rozmiarze bufora jaki zarezerwował dla danych, określa liczbę strumieni wychodzących oraz maksymalną liczbę strumieni przychodzących na jaką zezwala. INIT chunk zawiera początkową wartość pola TSN i może zawierać opcjonalne parametry (np. listę adresów IPv4, IPv6 lub nazwę domenową hosta).
2. Serwer potwierdza otrzymanie INIT wysyłając pakiet SCTP z jednostką INIT ACK. INIT ACK zawiera podobny zestaw parametrów jak INIT (początkowy

TSN, liczba strumieni, opcjonalna lista adresów serwera), ale ponadto zawiera parametr *State Cookie. State Cookie* powinno zawierać:

a. informacje potrzebne do ustanowienia asocjacji,
b. MAC (ang. *Message Authentication Code*),
c. czas utworzenia oraz czas życia *cookie* (okres przez jaki jest ważne).

Po otrzymaniu pakietu z jednostką INIT, serwer alokuje strukturę TCB (ang. *Transmission Control Block*). TCB zawiera informacje z otrzymanego INIT oraz jednostki INIT ACK, która zostanie wysłana w odpowiedzi do klienta. W TCB, serwer ustawia aktualny czas oraz okres ważności *cookie*. Następnie, z TCB wyodrębniany jest pewien podzbiór informacji (wystarczający do odtworzenia asocjacji) i na jego podstawie (oraz sekretnego klucza) obliczany jest MAC.
Do obliczenia MAC wykorzystywana jest jednokierunkowa, kryptograficzna funkcja haszująca i sekretny klucz, który powinien być pseudolosową wartością. Klucz powinien być regularnie zmieniany (czas utworzenia *cookie* w parametrze *State Cookie* może być użyty do określenia, który klucz ma zostać użyty do weryfikacji MAC). Wyodrębniony podzbiór informacji oraz MAC stanowią wartość *cookie* parametru *State Cookie* w jednostce INIT ACK. Po wysłaniu pakietu SCTP z INIT ACK, serwer usuwa strukturę TCB (wszystkie zasoby zawierające informacje o asocjacji). Zapobiega to atakom DoS, które wyczerpują zasoby pamięciowe serwera.

3. W odpowiedzi na INIT ACK serwera, klient przesyła jednostkę COOKIE ECHO, która zawiera *cookie* z parametru *State Cookie* otrzymanego od serwera. W pakiecie SCTP z COOKIE ECHO mogą być przesyłane dane użytkownika.
4. Po otrzymaniu COOKIE ECHO serwer podejmuje następujące działania:
    a. Oblicza MAC na podstawie podzbioru informacji w *cookie* otrzymanym od klienta oraz sekretnego klucza (czas utworzenia *cookie* może zostać użyty do określenia, który klucz należy użyć).
    b. Porównuje obliczony MAC z otrzymanym od klienta w *cookie*. Jeżeli występuje różnica, pakiet SCTP jest odrzucany (klient nie jest o tym powiadamiany).
    c. W przypadku zgodności MAC, serwer porównuje czas utworzenia *cookie* z aktualnym czasem. Jeżeli czas jaki upłynął od utworzenia jest dłuższy od czasu życia (ang. *lifespan,* domyślnie 60 sekund) w otrzymanym *cookie*, to pakiet SCTP powinien być odrzucony (klient jest powiadamiany za pomocą pakietu z jednostką ERROR).
    d. Jeżeli *cookie* przeszło walidację, serwer tworzy asocjację (alokuje zasoby na podstawie informacji w *cookie*) i odpowiada wysyłając pakiet z jednostką COOKIE ACK. W pakiecie tym mogą być przesyłane dane użytkownika.

1.3.4. Modele programowania gniazd SCTP

Istnieją dwa zasadnicze typy gniazd SCTP:

1. gniazda typu jeden-do-jeden (ang. *one-to-one*),
2. gniazda typu jeden-do-wielu (ang. *one-to-many*).

Gniazdo typu *one-to-one* umożliwia utworzenie tylko jednej asocjacji. Gniazda tego typu ułatwiają przekształcenie programów wykorzystujących protokół TCP, ponieważ pozwalają na posługiwanie się tym samym API (funkcje operujące na gniazdach: *connect()*, *bind()*, *listen()*, *accept()*, *send()*). Podczas migracji do protokołu SCTP trzeba pamiętać o następujących różnicach:

a. SCTP transmituje dane jako komunikaty, nie jako strumień bajtów,
b. opcje TCP przekazywane do funkcji `setsockopt()` należy zastąpić odpowiednimi opcjami SCTP,
c. protokół SCTP nie zapewnia wsparcia dla połączeń pół-zamkniętych (tworzenie takich połączeń w przypadku TCP umożliwia funkcja `shutdown()`).

W celu utworzenia gniazda typu *one-to-one* należy wywołać funkcję `socket()` z parametrami:

```
int      sockfd;
sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP); //lub AF_INET6
```

Gniazdo typu *one-to-many* może utworzyć wiele asocjacji. Ponadto:

a. Gniazda tego typu uniemożliwiają użycie funkcji `send()` oraz `write()`. Zamiast tych funkcji należy wykorzystać `sendto()`, `sendmsg()` lub `sctp_sendmsg()`.
b. Każde wywołanie `sendto()`, `sendmsg()` lub `sctp_sendmsg()` z adresem, dla którego asocjacja nie została ustanowiona powoduje próbę aktywnego ustanowienia asocjacji (ang. *active open*). Mechanizm ten działa nawet w przypadku, gdy proces wywołał dla gniazda funkcję `listen()`.
c. Właściwość opisana powyżej pozwala na ustanowienie asocjacji bez wywołania funkcji `connect()`. To z kolei umożliwia przesyłanie danych użytkownika podczas ustanawiania asocjacji za pomocą *4-way handshake* (w 3 i 4 pakiecie SCTP).

W celu utworzenia gniazda typu *one-to-many* należy wywołać funkcję `socket()` z parametrami:

```
int      sockfd;
sockfd = socket(AF_INET, SOCK_SEQPACKET, IPPROTO_SCTP);
                //lub AF_INET6
```


Do komunikacji za pomocą gniazd typu *one-to-many* wykorzystuje się najczęściej funkcje `sctp_sendmsg()` oraz `sctp_recvmsg()`. W stosunku do `sendmsg()` i `recvmsg()` nie istnieje konieczność utworzenia i wypełnienia odpowiednich struktur danych – wszystkie potrzebne informacje przekazywane są bezpośrednio za pomocą argumentów wywołania funkcji.

sctp_sendmsg

```
#include <sys/socket.h>
#include <netinet/sctp.h>

int sctp_sendmsg(int sd, const void *msg, size_t len,
                 struct sockaddr *to, socklen_t tolen, uint32_t ppid,
                 uint32_t flags, uint16_t stream_no, uint32_t timetolive,
                 uint32_t context);
```

Parametr	Opis
`sd`	deskryptor gniazda
`*msg`	wskaźnik do bufora, z którego wysyła się komunikat
`len`	rozmiar komunikatu w bajtach
`*to`	wskaźnik na strukturę zawierająca adres, pod który ma zostać wysłany komunikat
`tolen`	rozmiar struktury gniazdowej
`ppid`	*Payload Protocol Identifier* jednostki DATA (dla danych użytkownika przyjmuje wartość zero)
`flags`	opcjonalne flagi (np.: `SCTP_UNODRERED` – transmisja bez zachowania kolejności komunikatów)
`stream_no`	numer strumienia, którym przesłany będzie komunikat
`timetolive`	czas ważności wiadomości w milisekundach - nadawca zaniecha dalszych prób wysłania komunikatu po jego upływie
`context`	wartość, która powiązana będzie z komunikatem jeżeli wysyłanie zakończy się niepowodzeniem

sctp_recvmsg

```
#include <sys/socket.h>
#include <netinet/sctp.h>

int sctp_recvmsg(int sd, const void *msg, size_t len,
                 struct sockaddr *from, socklen_t *fromlen,
                 struct sctp_sndrcvinfo *sinfo, int msg_flags);
```


Parametr	Opis
sd	deskryptor gniazda
*msg	wskaźnik do bufora, do którego zapisany zostanie komunikat
len	rozmiar bufora w bajtach
*from	wskaźnik do struktury adresowej wypełnianej przez funkcję sctp_recvmsg() adresem źródłowym komunikatu
*fromlen	wskaźnik do rozmiaru struktury gniazdowej
*sinfo	wskaźnik do struktury sctp_sndrcvinfo wypełnianej przez funkcję po otrzymaniu komunikatu
*msg_flags	wskaźnik do flag wypełnianych przez funkcję po otrzymaniu komunikatu

Poniżej przedstawiona jest struktura sctp_sndrcvinfo wypełniana przez funkcję sctp_recvmsg() po otrzymaniu komunikatu:

```
#include <netinet/sctp.h>

struct sctp_sndrcvinfo {
        __u16 sinfo_stream;
        __u16 sinfo_ssn;
        __u16 sinfo_flags;
        __u32 sinfo_ppid;
        __u32 sinfo_context;
        __u32 sinfo_timetolive;
        __u32 sinfo_tsn;
        __u32 sinfo_cumtsn;
        sctp_assoc_t sinfo_assoc_id;
};
```

Najważniejsze pola to: sinfo_stream (numer strumienia), sinfo_ssn (*Stream Sequence Number*) oraz identyfikator asocjacji - sinfo_assoc_id.

1.4. Cel laboratorium

Celem laboratorium jest poznanie podstawowych właściwości protokołu SCTP. Ponadto zostanie przećwiczone tworzenie i korzystanie z gniazd typu *jeden-do-jeden* oraz *jeden-do-wielu*. Podczas realizacji tego laboratorium zapoznasz się z:

- budową pakietu SCTP,
- z techniką tworzenia gniazd dla protokołu SCTP,
- z możliwościami i ograniczeniami gniazd typu *jeden-do-jeden* i *jeden-do-wielu*,
- sposobem transmisji danych przez protokół SCTP.

2. Przebieg laboratorium

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

2.1. Przygotowanie laboratorium

Programy opisane w zadaniach 2 oraz 3 wymagają biblioteki odpowiedzialnej za dostęp do implementacji protokołu SCTP w jądrze systemu Linux.
Przed przystąpieniem do realizacji zadań należy upewnić się, że w systemie obecna jest biblioteka SCTP wraz z odpowiednimi plikami nagłówkowymi. W przypadku dystrybucji *Ubuntu* lub *Debian* wystarczy sprawdzić, czy zainstalowane są pakiety: *libsctp1*, *libsctp-dev*.

2.2. Zadanie 1. Komunikacja z wykorzystaniem interfejsu one-to-one

Celem zadania jest zaimplementowanie programów klienta oraz serwera echo. Komunikacja powinna odbywać się za pomocą gniazd typu *one-to-one*, tzn. pomiędzy gniazdem a asocjacją powinna istnieć relacja 1-do-1.

Zadaniem klienta jest przesyłanie danych wprowadzanych z klawiatury do serwera.
Program klienta należy zaimplementować w oparciu o poniższe założenia:

1. Adres IP i numeru portu serwera są argumentami wywołania programu.
2. Klient w pętli oczekuje na dane do przesłania.
3. Dane (ciąg znaków) pobierane są ze standardowego wejścia i przesyłane do serwera po wciśnięciu ENTER.
4. Wciśnięcie ENTER bez uprzedniego wprowadzenia danych powoduje zamknięcie asocjacji i przerwanie działania programu.
5. Po wysłaniu danych, klient oczekuje na odpowiedź serwera i wypisuje ją na standardowe wyjście.

Serwer powinien zostać zaimplementowany w oparciu o model iteracyjny, przyjmować dane z dowolnego interfejsu, wypisywać je na standardowe wyjście i odsyłać w niezmienionej formie. Poprawne zamknięcie asocjacji ma zakończyć działanie serwera.
Programy mają zostać napisane za pomocą API kompatybilnego z protokołem TCP.

W celu uruchomienia programów należy wykonać następujące czynności:

1. Skompilować programy źródłowe do postaci binarnej:
   ```
   $ gcc –o client1 client1.c
   $ gcc –o server1 server1.c
   ```
2. Uruchomić sniffer z odpowiednimi opcjami (np. tcpdump).
3. Uruchomić serwer STCP podając wybrany przez siebie numer portu:
   ```
   $ ./server1 <numer portu>
   ```
4. Uruchomić klienta podając adres i numer portu, na którym nasłuchuje serwer:
   ```
   $ ./client1 <adres IP> <numer portu>
   ```
5. Za pomocą sniffera zaobserwować *4-way handshake*. Szczególną uwagę należy zwrócić na numery jakie przesyłane są w polach *Initiate Tag* oraz *Initial TSN* jednostek INIT, INIT-ACK.
6. Proszę odpowiedzieć na następujące pytania:
   a. Czy dane użytkownika mogą być transmitowane podczas ustanawiania asocjacji?
   b. Za pomocą ilu strumieni transmitowane są dane użytkownika?
   c. Jak podczas transmisji kolejnych komunikatów zmienia się *TSN* (*Transmission Sequence Number*)?

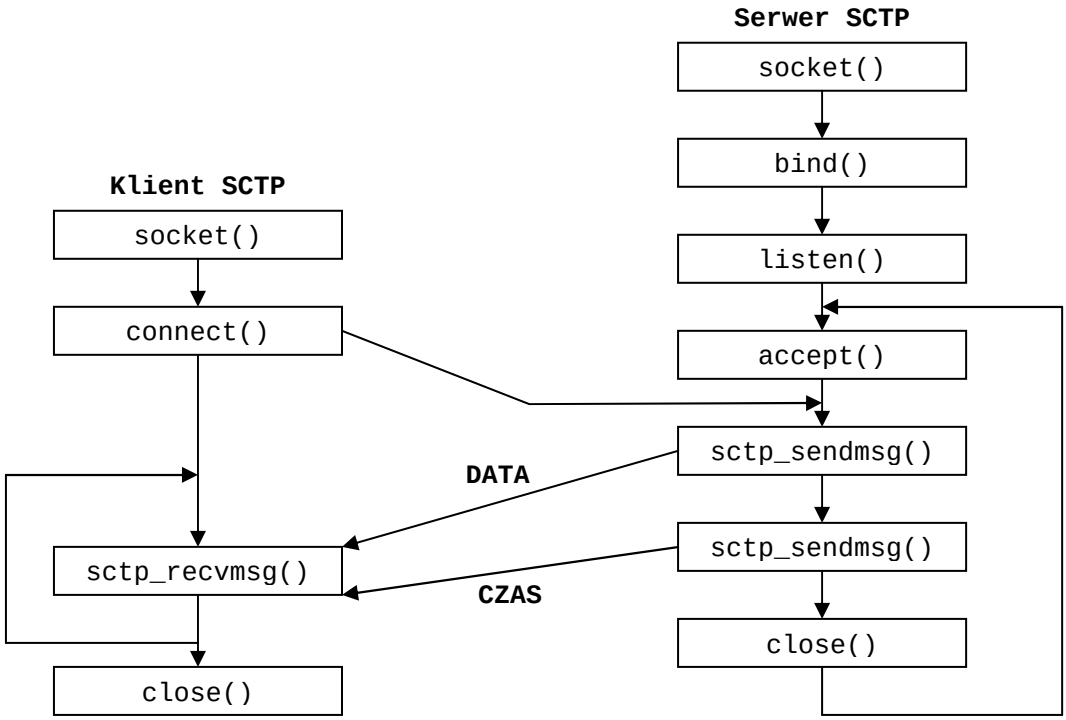
2.3. Zadanie 2. Komunikacja z wykorzystaniem wielu strumieni

Zadanie polega na zaimplementowaniu programów klienta i serwera datetime. Komunikacja powinna odbywać się za pomocą gniazd typu *one-to-one*.
Argumentami wywołania programu klienta mają być adres IP oraz numerem portu serwera. Przed ustanowieniem asocjacji z serwerem proszę określić liczbę strumieni wychodzących (3), maksymalną liczbę strumieni przychodzących (4) oraz maksymalną liczbę prób nawiązania asocjacji (5) (funkcja `setsockopt()`, opcja `SCTP_INITMSG`).
Po ustanowieniu asocjacji klient powinien:

a. wypisać informacje z nią związane: ID i stan asocjacji, liczbę strumieni wychodzących i przychodzących (funkcja `getsockopt()`, `opcja SCTP_STATUS`).
b. Odebrać aktualną datę oraz czas i zakończyć działanie. Data oraz czas mają być przesłane oddzielnymi strumieniami.

Serwer powinien zostać zaimplementowany w oparciu o model iteracyjny, przyjmować maksymalnie dwa strumienie i tworzyć dwa strumienie wychodzące. Aktualną datę i czas można uzyskać za pomocą funkcji: `time()`, `localtime()` oraz `strftime()` (ostatnia służy do formatowania danych). Po obsłużeniu klienta serwer ma oczekiwać na kolejne połączenie.

Programy mają zostać napisane z użyciem API specyficznego dla protokołu SCTP (funkcje `sctp_sendmsg()` oraz `sctp_recvmsg()`) oraz z wykorzystaniem funkcji `connect()` i `accept()`.



Rys. 7. Schemat komunikacji klient-serwer dla gniazd typu *one-to-one*.

W celu uruchomienia programów należy wykonać następujące czynności:

1. Skompilować programy źródłowe do postaci binarnej:
   ```
   $ gcc –o client2 client2.c -lsctp
   $ gcc –o server2 server2.c -lsctp
   ```
2. Uruchomić sniffer z odpowiednimi opcjami.
3. Uruchomić serwer STCP podając wybrany przez siebie numer portu:
   ```
   $ ./server2 <numer portu>
   ```
4. Uruchomić klienta podając adres i numer portu, na którym nasłuchuje serwer:
   ```
   $ ./client2 <adres IP> <numer portu>
   ```
5. Za pomocą sniffera zaobserwować postać pakietów SCTP, a w szczególności:
   a. wartości określające liczbę strumieni wychodzących i maksymalną liczbę strumieni przychodzących w jednostkach INIT oraz INIT ACK,
   b. wartości pól *Stream Identifier* oraz *Stream Sequence Number* w jednostkach DATA.
6. Proszę eksperymentować z maksymalną liczbą strumieni jakie może zaakceptować klient. Jaki rezultat da ustawienie tej wartości na 1?

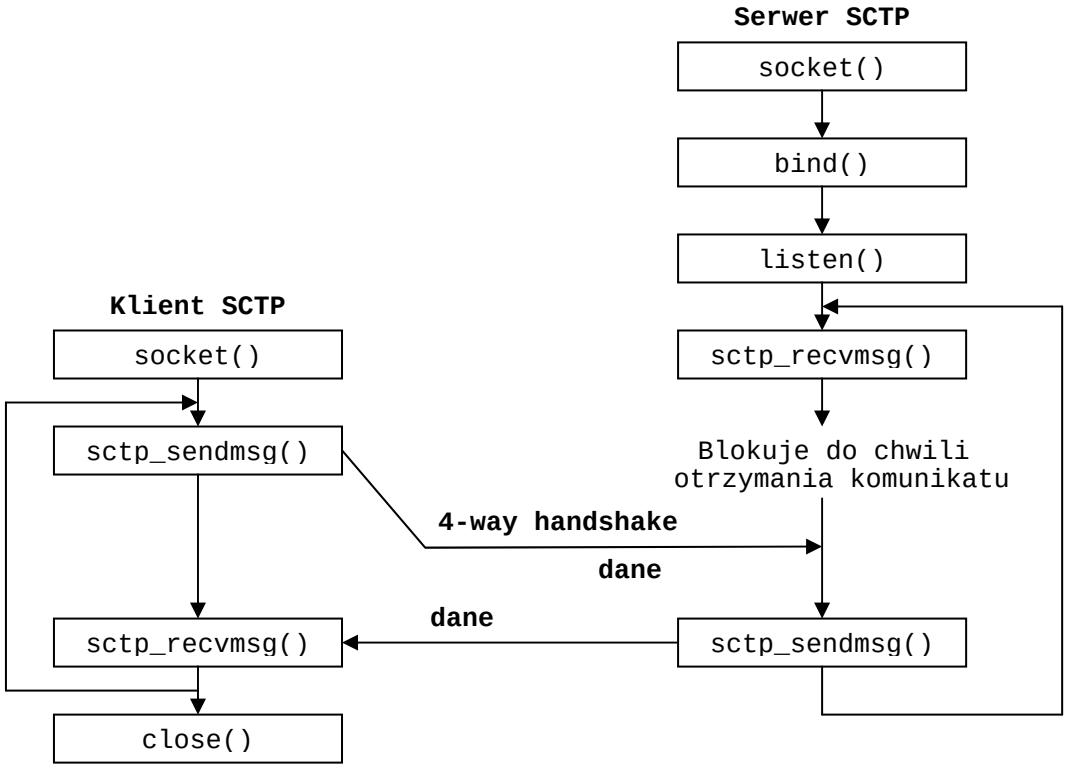
2.4. Zadanie 3. Komunikacja z wykorzystaniem interfejsu one-to-many

Celem zadania jest zaimplementowanie programów demonstrujących właściwości gniazd typu *one-to-many*.

Klient ma spełniać założenia opisane w zadaniu 1, z tą różnicą, że:

a. Powinien wypisywać numer strumienia, którym wysyłany będzie komunikat. Pierwszy komunikat ma zostać wysłany za pomocą strumienia o numerze zero.
b. Po odebraniu odpowiedzi, ma wypisać numer strumienia, którym dane nadeszły, ID asocjacji oraz *SSN*.
c. Każdy kolejny komunikat ma zostać wysłany strumieniem o numerze równym numerowi strumienia, którym poprzednio odebrano dane.

Serwer powinien zostać zaimplementowany w oparciu o model iteracyjny, przyjmować dane z dowolnego interfejsu i odsyłać w niezmienionej formie. Oprócz numeru portu proszę zdefiniować argument wywołania, który może przyjmować wartość 1 lub 0. Jeżeli wartość tego parametru wynosi zero, to serwer ma odesłać odpowiedź strumieniem o numerze strumienia, którym odebrał komunikat od klienta (zawsze będzie to 0). W przeciwnym wypadku ma zwiększyć numer o jeden i wysłać komunikat. Inkrementacja ma odbywać się w sposób cykliczny. Jeżeli serwer ma N strumieni o numerach 0, 1, …, N-1, to kolejne komunikaty będą przesyłane strumieniami: 0, 1, …, N-1, 0, 1, ….



Rys. 8. Schemat komunikacji klient-serwer dla gniazd typu *one-to-many*.

Z uwagi na sposób w jaki zadanie zostało sformułowane, komunikacja dla parametru serwera równego 1 jest możliwa przy założeniu, że:

min(*OS* klienta, *MIS* serwera) >= min(*OS* serwera, *MIS* klienta)

Mając na uwadze powyższe założenie, proszę określić w programach poprawną liczbę strumieni wychodzących (*OS*) oraz maksymalną liczbę strumieni przychodzących (*MIS*).

Program serwera powinien zostać zrealizowany w oparciu o gniazdo typu `SOCK_SEQPACKET`, a klienta w oparciu o gniazdo typu `SOCK_STREAM`.

W celu uruchomienia programów należy wykonać następujące czynności:

1. Skompilować programy źródłowe do postaci binarnej:
   ```
   $ gcc –o client3 client3.c -lsctp
   $ gcc –o server3 server3.c -lsctp
   ```
2. Uruchomić sniffer z odpowiednimi opcjami.
3. Uruchomić serwer STCP podając wybrany przez siebie numer portu oraz dodatkowy parametr (0 lub 1):
   ```
   $ ./server3 <numer portu> <parametr>
   ```
4. Uruchomić klienta podając adres i numer portu, na którym nasłuchuje serwer:
   ```
   $ ./client3 <adres IP> <numer portu>
   ```
5. Za pomocą sniffera zaobserwować postać pakietów SCTP:
   a. wartości określające liczbę strumieni wychodzących i maksymalną liczbę strumieni przychodzących w jednostkach INIT oraz INIT ACK,
   b. wartości pól *Stream Identifier* oraz *Stream Sequence Number* w jednostkach DATA.
6. Proszę zwrócić szczególną uwagę na informacje wypisywane przez program klienta w zależności od ustawienia parametru wywołania serwera.
7. Proszę uruchomić kilka instancji aplikacji klienta i połączyć się z serwerem. Jakie identyfikatory asocjacji są wypisywane podczas komunikacji z serwerem?

3. Opracowanie i sprawozdanie

Realizacja laboratorium pt. *„Protokół SCTP”* polega na wykonaniu wszystkich zadań programistycznych podanych w drugiej części tej instrukcji. Wynikiem wykonania powinno być sprawozdanie w formie wydruku papierowego dostarczonego na kolejne zajęcia licząc od daty laboratorium, kiedy zadania zostały zadane.
Sprawozdanie powinno zawierać:

- opis metodyki realizacji zadań (system operacyjny, język programowania, biblioteki, itp.),
- algorytmy wykorzystane w zadaniach (zwłaszcza, jeśli zastosowane zostały rozwiązania nietypowe),
- opisy napisanych programów wraz z opcjami,
- trudniejsze kawałki kodu, opisane tak, jak w niniejszej instrukcji,
- uwagi oceniające ćwiczenie: trudne/łatwe, nie/realizowalne, nie/wymagające wcześniejszej znajomości zagadnień (wymienić jakich),
- wskazówki dotyczące ewentualnej poprawy instrukcji celem lepszego zrozumienia sensu oraz treści zadań.

