Wojskowa Akademia Techniczna

Wydział Elektroniki

Języki C/C++ w zastosowaniach sieciowych

Instrukcja laboratoryjna - zadania laboratoryjne

v.1.0

Opracował:
dr inż. Jarosław Krygier
dr inż. Krzysztof Maślanka
dr inż. Sebastian Szwaczyk

Warszawa 2020

Spis treści

Przygotowanie do ćwiczeń laboratoryjnych:

1. Pobrać, zainstalować i zapoznać się z oprogramowaniem Eclipse.
2. Zapoznać się z materiałami z wykładów - samodzielnie przećwiczyć zadania dodatkowe.
3. Zapoznać się z podstawowymi poleceniami systemu pomocy Linux.

1. Ćwiczenie 1 (8 godz.)

Celem ćwiczenia jest odświeżenie wiedzy i utrwalenie umiejętności programowania w języku. W celu zaliczenia ćwiczenia 1, należy zaliczyć poszczególne zagadnienia. Ocena: zal/nzal.

Zadanie 1. Uruchomienie projektu

Zadanie ma na celu skompilowanie i uruchomienie najprostszego możliwego programu.

1. Utworzyć nowy projekt dla języka C w IDE Eclipse typu "Hello world ANSI C Project".
2. Przeanalizować strukturę programu i dokonać modyfikacji tak, aby program wyświetlał komunikat "*Witaj mój drogi stwórco!*".
3. Skompilować i uruchomić oprogramowanie.
4. Uruchomić skompilowany program z terminala systemu operacyjnego.

Zadanie 2. Deklaracja zmiennych

Zadanie ma na celu utrwalenie sposobu deklaracji oraz użycia zmiennych w języku C.

1. Utworzyć nowy projekt dla języka C w IDE Eclipse typu "Hello world ANSI C Project".
2. Zadeklarować zmienne typu: *int*, *char*, *unsigned char, float, double*.
3. Przypisać wartość poszczególnym zmiennym.
4. Wyświetlić wartość poszczególnych zmiennych korzystając z funkcji *printf()*.
5. Korzystając z funkcji standardowych języka C odczytać i wyświetlić zarezerwowany rozmiar pamięci dla poszczególnych zmiennych.
6. Zadeklarować stałą typu *int* i przypisać jej wartość.
7. Zmienić wartość przypisaną stałej.

Zadanie 3. Pętle

Zadanie ma na celu utrwalenie sposobu użycia podstawowych pętli w języku C.

1. Utworzyć nowy projekt dla języka C w IDE Eclipse typu "Hello world ANSI C Project".
2. Zadeklarować zmienne typu *int* i przypisać im wartości *0* i *100* oraz wyświetlić na standardowym wyjściu.
3. Korzystając z pętli *for* zwiększyć dwudziestokrotnie jedną ze zmiennych o *1*, drugą w tym czasie dwudziestokrotnie zmniejszyć o *2*.
4. Wyświetlić wartości zmiennych.
5. Korzystając z pętli *while* zwiększyć dwudziestokrotnie wartość jednej ze zmiennych o *2*, drugiej również dwudziestokrotnie o *2*.
6. Wyświetlić wartości zmiennych.

Zadanie 4. Wskaźniki

Zadanie ma na celu utrwalenie sposobu użycia zmiennych wskaźnikowych.

1. Utworzyć nowy projekt dla języka C w IDE Eclipse typu "Hello world ANSI C Project".
2. Zadeklarować zmienne typu *int, char, unsigned char*.
3. Zadeklarować zmienne wskaźnikowe, które będą przetrzymywały adresy do wcześniej zadeklarowanych zmiennych.
4. Przypisać dowolne wartości zadeklarowanym zmiennym (nie dotyczy zmiennych wskaźnikowych).

5. Wskazać adresy odpowiednich zmiennych dla zmiennych wskaźnikowych.
6. Wyświetlić zarezerwowany rozmiar pamięci dla zmiennych i zmiennych wskaźnikowych a także wartość tych zmiennych.
7. Przypisać poszczególnym zmiennym wartość przez ich wskaźnik.
8. Ponownie wyświetlić rozmiar i wartość zmiennych i zmiennych wskaźnikowych.

Zadanie 5. Funkcje

Celem zadania jest utrwalenie zasad tworzenia funkcji w języku C oraz korzystania z plików nagłówkowych. Zadanie polega na napisaniu programu pozwalającego na wykorzystaniu funkcji sumowania, która będzie w różny sposób otrzymywać wartości i prezentować wynik.

1. Utworzyć nowy projekt dla języka C w IDE Eclipse typu "Hello world ANSI C Project".
2. W pliku z funkcją główną (*main()*) zadeklarować i zdefiniować funkcję *suma()*, która będzie wymagała dwóch argumentów o wartościach całkowitych i będzie wyliczała i wyświetlała ich sumę.
3. W funkcji *main()* zadeklarować po jednej zmiennej globalnej i lokalnej typu *int* i przypisać im wartość. Wartości tych zmiennych wykorzystać do przekazania jako argumenty funkcji *suma()*. Wywołać funkcję Suma().
4. Uruchomić program.
5. Na bazie funkcji *suma()* utworzyć funkcję *suma2()* z dwoma argumentami, w której zadeklarować zmienną lokalną o takiej samej nazwie co istniejąca, zadeklarowana zmienna w funkcji *main()*. Funkcja *suma2()* ma obliczać sumę obu swoich argumentów oraz wartości zmiennej lokalnej i wyświetlać wynik działania.
6. Wywołać funkcję *suma2()* w funkcji *main()* i uruchomić program.
7. W funkcji *main()*, zadeklarować zmienną wskaźnikową, która będzie wskazywać na utworzoną wcześniej zmienną lokalną.
8. Utworzyć funkcję *suma3()*, do której dzięki argumentom zmienna globalna zostanie przekazana przez wartość, a zmienna lokalna przez wskaźnik. Wynik obliczeń dwóch parametrów funkcja *suma3()* ma zwrócić do programu przez wartość, a sumę obu parametrów oraz wartości zmiennej lokalnej przez wskaźnik dostarczony jako argument.
9. Wywołać funkcję *suma3()* w funkcji *main()*, wyświetlić wyniki obliczeń i uruchomić program.

Zadanie 6. Dynamiczna alokacja i zwalnianie pamięci

1. Zadanie ma na celu utrwalenie sposobu rezerwacji i zwalniania pamięci w języku C. Zbudować program, który będzie buforował liczby całkowite
2. Utworzyć nowy projekt dla języka C w IDE Eclipse typu "Hello world ANSI C Project".
3. Zadeklarować zmienną wskaźnikową o nazwie "wsk_do_bufora" do typu całkowitego bez znaku.
4. Zadeklarować zmienną o nazwie "liczba_elementow_bufora" typu całkowitego bez znaku. Przypisać tej zmiennej dowolną wartość z przedziału <5,10>.
5. Zaalokować pamięć do przechowywania elementów typu całkowitego bez znaku w liczbie "liczba_elementow_bufora" z wykorzystaniem funkcji malloc().
6. Przypisać poszczególnym elementom bufora kolejne wartości poczynając od 100
7. Wyświetlić rozmiary zmiennych " wsk_do_bufora" i "liczba_elementow_bufora".
8. Wyświetlić zawartość elementów bufora.
9. Zwolnić zaalokowaną pamięć, wykorzystując funkcję free().

10. Wyświetlić zawartość tablicy.

Zadanie 7. Struktury

Celem zadania jest utrwalenie podstawowych operacji na strukturach danych w języku C.

1. Utworzyć nowy projekt dla języka C w IDE Eclipse typu "Hello world ANSI C Project".
2. Zdefiniować strukturę dwuelementową, w której jeden z elementów jest wskaźnikiem na zmienną typu *unsigned char*, drugi typu *int*.
3. W funkcji głównej programu zadeklarować zmienną o nazwie *zmienna_strukt* typu struktury wcześniej zdefiniowanej struktury oraz wskaźnik na tą zmienną.
4. Przypisać dowaloną wartość do zmiennej typu *int* stanowiącej element struktury - wykorzystując zadeklarowaną zmienną *zmienna_strukt* i wyświetlić wartość tego elementu na standardowym wyjściu.
5. Przypisać wartość NULL do zmiennej wskaźnikowej typu *unsignet char* stanowiącej element struktury - wykorzystując zadeklarowaną zmienną *zmienna_strukt* i wyświetlić wartość tego elementu na standardowym wyjściu.
6. Zmienić wartość elementów struktury przez wskaźnik do zmiennej *zmienna_strukt* i wyświetlić na standardowym wyjściu.

Zadanie 8. Pliki nagłówkowe

Celem zadania jest utrwalenie sposobu wykorzystywania plików nagłówkowych podczas pisania oprogramowania w języku C. Zadanie polega na napisaniu programu, który będzie wykorzystywał funkcje napisane w zadaniu 5. Funkcje te będą umieszczone w zewnętrznym pliku nagłówkowym.

1. Utworzyć nowy projekt dla języka C w IDE Eclipse typu "Hello world ANSI C Project".
2. Dodać do projektu plik nagłówkowy (funkcje.h), do którego należy przenieść deklaracje wszystkich funkcji z zadania 6. Definicje funkcji przenieść do pliku funkcje.c.
3. W pliku nagłówkowym zdefiniować strukturę, która będzie wykorzystana do przekazania wartości liczb do zsumowania do poszczególnych funkcji. Zmodyfikować funkcje tak, aby możliwe było przekazanie parametrów przez strukturę.
4. Uruchomić program wykorzystujący funkcje z pliku nagłówkowego.

Zadanie 9. Pola bitowe

Celem zadania jest utrwalenie sposobu wykorzystania pól bitowych w języku C.

1. Utworzyć nowy projekt dla języka C w IDE Eclipse typu "Hello world ANSI C Project".
2. Zdefiniować i zadeklarować zmienną z ośmioma flagami (polami bitowymi) o długości 1b.
3. Zdefiniować i zadeklarować zmienną z trzema flagami o długości 1b.
4. Zdefiniować i zadeklarować zmienną z dwoma polami bitowymi o długości 1b i jednym o długości 4b.
5. Wyświetlić rozmiar wszystkich zmiennych.
6. Zapisać wartości do poszczególnych flag.
7. Wyświetlić wartości poszczególnych pól bitowych jak też całej zmiennej.

Zadanie 10. Forma zapisu danych

Celem zadania jest utrwalenie mechanizmów zapisu liczby w pamięci.

1. Utworzyć nowy projekt dla języka C w IDE Eclipse typu "Hello world ANSI C Project".

2. Zadeklarować zmienną typu *int* i przypisać jej wartość 0x11223344 (4 bajty).
3. Utworzyć strukturę zawierającą cztery pola typu *unsigned char* (4 bajty) i zadeklarować dla niej zmienną.
4. Skopiować wartość zadeklarowanej zmiennej do struktury (4 bajty), wykorzystując funkcję *memcpy()*.
5. Wyświetlić poszczególne pola struktury.
6. Sprawdzić jaka jest forma zapisu zmiennej w systemie operacyjnym i zweryfikować z wartościami wypisanymi z poszczególnych pól struktury.

Zadanie 11. Listy wiązane

Celem zadania jest wykorzystanie list wiązanych w języku C. Należy napisać program, który będzie buforował nieznaną liczbę par wartości całkowitej i rzeczywistej.

1. Utworzyć nowy projekt dla języka C w IDE Eclipse typu "Hello world ANSI C Project".
2. W pliku nagłówkowym przygotować strukturę, która będzie wykorzystana do przechowywania elementów bufora w liście wiązanej.
3. Zadeklarować zmienne pozwalające na powiązanie elementów listy, czyli dla: dowiązania nowego elementu, wskazania poprzedniego elementu i wskazania pierwszego elementu (punktu zakotwiczenia).
4. Przygotować i przypisać wartości dla zmiennych pierwszego elementu na liście.
5. Dodać do listy elementy, tak aby zawierała 10 par wartości.
6. Wyświetlić zawartość poszczególnych elementów listy.
7. Usunąć piąty i siódmy element z listy (pamiętać o zapewnieniu spójności listy).
8. Wyświetlić zawartość poszczególnych elementów listy.

2. Ćwiczenie 2 (8 godz.)

Celem zadania jest napisanie programu, który będzie odbierał ramki z interfejsu sieciowego (obsługa bufora statycznego), sprawdzał ich zawartość (obsługa protokołów telekomunikacyjnych), a dane z wybranych ramek zapisywał do dalszej analizy w liście wiązanej (buforze dynamicznym). Ocena: liczbowa.

Przygotowanie do ćwiczenia:

Korzystając z kodu załączonego na końcu zadania przygotować i uruchomić program, który będzie odbierał wszystkie ramki Ethernet z wybranego interfejsu sieciowego i wyświetlał ich zawartość na standardowym wyjściu w postaci szesnastkowej.

Realizacja ćwiczenia:

1. (Ocena: 3) W otrzymanym programie dokonać modyfikacji tak, aby wyświetlać odebrane pakiety określonego typu (typ odbieranego pakietu dla każdego studenta określa prowadzący zajęcia):
    1) IP
    2) IPv6
    3) TCP
    4) UDP
    5) ARP
    6) ICMP
    7) ICMPv6

    W standardowym wyjściu (konsoli) ma być wyświetlona informacja o typie (pole typu w ramce Ethernet) przenoszonych przez każdą odebraną ramkę Ethernet danych, a zawartość odebranego datagramu (np. nagłówek ICMP + dane, nagłówek IP + dane, nagłówek TCP + dane, itp.) ma być wyświetlana w standardowym wyjściu w postaci ciągu liczb HEX, z oznaczeniem nazw poszczególnych pól nagłówka datagramu.
2. (Ocena: 4) Wykonać pkt. 1 oraz przygotować plik nagłówkowy zawierający struktury danych niezbędne do przechowywania informacji z zadanego nagłówka i pola danych. Uzupełnić zmienne strukturalne odebranymi danymi.
3. (Ocena: 4.5) Wykonać pkt. 1 i 2 oraz przygotować strukturę do utworzenia listy wiązanej. Załadować zawartość odebranych 20 datagramów do listy wiązanej. Wyświetlić w standardowym wyjściu zawartość poszczególnych pól nagłówków każdego z 20 pakietów załadowanych do listy (nazwa pola, wartość HEX, komentarz dot. wykorzystania tej wartości), po czym wyczyścić listę (zwolnić pamięć całej listy). Powtarzać te czynności (załadowanie 20 datagramów, wyświetlenie zawartości ich pól po załadowaniu) do momentu zatrzymania programu przez użytkownika wybraną kombinacją klawiszy (nie stosować standardowych sygnałów przerywających funkcjonowanie programu).
4. (Ocena: 5) Wykonać pkt. 1, 2 i 3 oraz uzupełnić program do wysłania pakietów w trybie FIFO z bufora dynamicznego (listy wiązanej) na wskazany adres IP (pakiety wtedy nie są kasowane jak w punkcie 3). Można skorzystać z dołączonego programu nadajnika.

```
/*
 ============================================================================
 Name        : EthRecv.c
 Author      : Krzysztof Maslanka
 Version     : v1.0
 Copyright   : ISL WEL WAT
 Description : Odbiornik z gniazdm surowym
 ============================================================================
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/socket.h>
#include <linux/if_ether.h>

int main(void) {
	printf("Uruchamiam odbieranie ramek Ethernet.\n"); /* prints  */

	//Utworzenie bufora dla odbieranych ramek Ethernet
	unsigned char *buffer = (void*) malloc(ETH_FRAME_LEN);

	//Otwarcie gniazda pozwalającego na odbiór wszystkich ramek Ethernet
	int iEthSockHandl = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
	//Kontrola czy gniazdo zostało otwarte poprawnie, w przypadku bledu wyświetlenie komunikatu.
	if (iEthSockHandl<0)
		printf("Problem z otwarciem gniazda : %s!\n", strerror(errno));

	//Zmienna do przechowywania rozmiaru odebranych danych
	int iDataLen = 0;

	//Pętla nieskończona do odbierania ramek Ethernet
	while (1) {

		//Odebranie ramki z utworzonego wcześniej gniazda i zapisanie jej do bufora
		iDataLen = recvfrom(iEthSockHandl, buffer, ETH_FRAME_LEN, 0, NULL, NULL);

		//Kontrola czy nie było bledu podczas odbierania ramki
		if (iDataLen == -1)
			printf("Nie moge odebrac ramki: %s! \n", strerror(errno));
		else { //jeśli ramka odebrana poprawnie wyświetlenie jej zawartości
			printf("\nOdebrano ramke Ethernet o rozmiarze: %d [B]\n", iDataLen);
		}
	}

	return EXIT_SUCCESS;
}

/*
 ============================================================================
 Name        : jcpp_lab_gniazdo_RAW_nad.c
 Author      : J. Krygier
 Version     : v1.0
 Copyright   : ISL WEL WAT
 Description : Nadajnik z gniazdem surowym
 ============================================================================
 */
//UWAGA: program wymaga uprawnien administratora
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h> //socket(), sendto()
#include <arpa/inet.h> //IPPROTO_RAW
#include <unistd.h>//sleep
#include <netinet/ip.h> //nagl. ip
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <net/if.h>
```

```
unsigned short csum();

int main(void) {
	puts("Prosty nadajnik RAW");
	int gniazdo_raw;
	unsigned char datagram[1500];
	struct sockaddr_in adres_gniazda_celu;
	int rozmiar_danych;
	int one = 1;
	const int *val = &one;
	unsigned char dane[1480];

	//serowanie datagramu
	memset(datagram, 0, 1500);

	struct iphdr *iph = (struct iphdr*) datagram;

	// trzeba zbudowac naglowek IP samemu dla tego gniazda
	gniazdo_raw = socket(PF_INET, SOCK_RAW, IPPROTO_RAW);
	if (gniazdo_raw == -1) {
		//nie moge utworzyc gniaza
		perror("Blad:");
		exit(1);
	}

	// ustawimy interfejs do wysylania
	struct ifreq ifr;
	memset(&ifr, 0, sizeof(ifr)); //wyzerujemy strukture
	snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "lo"); // wpiszemy nazwe interfejsu 'lo'- interfejs lokalny
	setsockopt(gniazdo_raw, SOL_SOCKET, SO_BINDTODEVICE, (void*) &ifr, sizeof(ifr)); // ustwimy odpowiednia opcje w
gniezdzie

	//skompletujemy adres gniazda docelowego, jesli naglowek budowany adresu nie trzeba ustawiac
	adres_gniazda_celu.sin_family = AF_INET;
	//adres_gniazda_celu.sin_port = htons(8080); //nie ma znaczenia jesli ustawiamy w naglowku IP oddzielnie
	adres_gniazda_celu.sin_addr.s_addr = inet_addr("127.0.0.2"); //adres docelowy


	if (setsockopt(gniazdo_raw, IPPROTO_IP, IP_HDRINCL, val, sizeof(one)) < 0) {
		perror("Blad ustawienia opcji IP_HDRINCL");
		exit(0);
	}
	//uzupelnienie naglowka ipv4
	iph->version = 4;
	iph->ihl = 5;
	iph->tos = 0xe0;
	iph->tot_len = sizeof(struct iphdr) + sizeof(dane);
	iph->id = htonl(54321); //jakis identyfikator
	iph->frag_off = 0;
	iph->ttl = 255;
	iph->protocol = IPPROTO_ICMP;
	iph->check = 0; //tymczasowo
	iph->saddr = inet_addr("127.0.0.1");
	iph->daddr = adres_gniazda_celu.sin_addr.s_addr;

	//suma kontrolna wlasna funkcja
	iph->check = csum((unsigned short *) datagram, iph->tot_len); // powinien system tez obliczyc

	//ustawimy dane jakies przykladowe dane
#if 1
	memset(dane, 1, 1480);
	memset(dane, 0xff, 2); // ustawimy 2 bajty
	memset(dane + 2, 0xff, 2); // ustawimy kolejne
	dane[4] = 0x0;
	dane[5] = 0xa;
#endif

	memcpy(datagram + sizeof(struct iphdr), dane, sizeof(dane));

	//wylistowanie datagramu
	int i;
	int j=0;
	for (i = 0; i < 1500; i++) {
		printf("%.2x ", datagram[i]);
```

```
                j++;
                if (0 == j%20){ // wypisywanie po 20 oktetów
                        printf ("\n");
                }
        }
        printf("\n");

        rozmiar_danych = sendto(gniazdo_raw, datagram, sizeof(datagram), 0,
                        (struct sockaddr*) &adres_gniazda_celu, sizeof(adres_gniazda_celu));
        if (rozmiar_danych < 0) {
                perror("Blad transmisji");
        } else {
                printf("Wyslano : %d [B]\n", rozmiar_danych);
        }
        return EXIT_SUCCESS;
}

unsigned short csum(unsigned short *ptr, int nbytes) {
        register long sum;
        unsigned short oddbyte;
        register short answer;

        sum = 0;
        while (nbytes > 1) {
                sum += *ptr++;
                nbytes -= 2;
        }
        if (nbytes == 1) {
                oddbyte = 0;
                *((u_char*) &oddbyte) = *(u_char*) ptr;
                sum += oddbyte;
        }

        sum = (sum >> 16) + (sum & 0xffff);
        sum = sum + (sum >> 16);
        answer = (short) ~sum;

        return (answer);
}
```

3. Ćwiczenie 3 (8 godz.)

Celem zadania jest napisanie programów generujących pakiety danych do sieci oraz programów typu klient-serwer z własnym protokołem aplikacyjnym, które umożliwią przesyłanie na serwer danych oraz ich interpretację na serwerze w wybranych przez użytkownika momentach.

Przygotowanie do ćwiczenia:

Przygotować projekt protokołu wymiany danych między klientem i serwerem opisanym w części "Realizacja ćwiczenia". Projekt protokołu powinien zawierać:

1) graficzny format PDU (jednostki danych protokołu) protokołu (z polami nagłówka i danych),
2) diagram sekwencji (wymiana komunikatów między klientem i serwerem),
3) algorytmy działania klienta i serwera.

Realizacja ćwiczenia:

1. (Ocena 3) Napisać program klienta i serwera wykorzystujący gniazda RAW. Komunikacja między klientem i serwerem powinna być realizowana za pomocą pakietów IPv4 przenoszących jednostki PDU własnego protokołu. Nagłówki protokołu IPv4 powinny być uzupełniane w programie. Pole „protocol" należy wybrać z wartości nieprzypisanych (https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml), tj. 144 + *numer na liście studentów*.

   Własne jednostki PDU powinny się składać z

   1. Nagłówka
   2. Pola danych

   Nagłówek własnych jednostek PDU powinien zawierać następujące pola:

   1. Wersja własnego protokołu (przypisać standardowo 1)
   2. Kod operacji
   3. Typ przenoszonych danych
   4. Długość nagłówka
   5. Rozmiar pola danych
   6. Opcje (dołączane opcjonalne, maksymalnie 3 opcje o różnej długości)

   Opcje powinny się składać z:

   1. Kod opcji
   2. Długość opcji
   3. Dane opcji (zmienny rozmiar)

   Operacje klienta:

   1. Przygotować i przesłać w polu danych do serwera
      i. adres MAC aktywnego interfejsu oraz
      ii. adres IP aktywnego interfejsu
      iii. 10 liczb podanych przez użytkownika
   2. Przygotować i przesłać w opcjach
      i. adres MAC aktywnego interfejsu oraz
      ii. adres IP aktywnego interfejsu
      iii. 10 liczb podanych przez użytkownika

3. Przygotować następujące Typy przenoszonych danych
    i. Typ 1: Przenoszony jest tylko adres MAC
    ii. Typ 2: Przenoszony jest tylko adres IP
    iii. Typ 3: przenoszone są tylko liczby
4. Przygotować następujące Kody operacji
    i. Kod 1: brak operacji
    ii. Kod 2: dodawanie
    iii. Kod 3: odejmowanie
    iv. Kod 4: mnożenia
    v. Kod 5: dzielenie
    vi. Kod 7: poprawny odbiór
5. Przyjąć i wypisać potwierdzenie otrzymania danych przez serwer
6. W przypadku braku potwierdzenia wysłać dane ponownie (aż do skutku)

Operacje serwera:

1. Odebrać dane od klienta
2. Wypisać w konsoli adres MAC i IP klienta (o ile są przekazane)
3. Zapisać przekazane liczby (o ile są przekazane) do bufora (listy wiązanej)
4. Wykonać operację na liczbach wskazanych przez klienta (o ile są przekazane) i wypisać jej wynik w konsoli.
5. Gdy przesłano liczby, przekazać do klienta potwierdzenie zawierające:
    i. W nagłówku
        1. Typ przenoszonych danych:
            a. Typ 4: potwierdzenie
        2. Kod operacji:
            a. Kod 6: wynik operacji

6. Gdy przesłano liczby i pozostałe dane, przekazać do klienta potwierdzenie zawierające:
    i. W nagłówku
        1. Typ przenoszonych danych:
            a. Typ 4: potwierdzenie
        2. Kod operacji:
            a. Kod 6: wynik operacji
7. Gdy przesłano tylko pozostałe dane, przekazać do klienta potwierdzenie zawierające:
    i. W nagłówku
        3. Typ przenoszonych danych:
            a. Typ 4: potwierdzenie
        4. Kod operacji:
            a. Kod 7: dane odebrano poprawnie

2. (Ocena: 4) Wykonać pkt. 1 z wykorzystaniem gniazd datagramowych (nr portu docelowego: 1024+*numer na liście studentów*). W kliencie umożliwić przesyłanie maski podsieci, a w serwerze dodatkowo dokonać kalkulacji adresu IP sieci klienta wraz z jej wypisaniem w konsoli. W

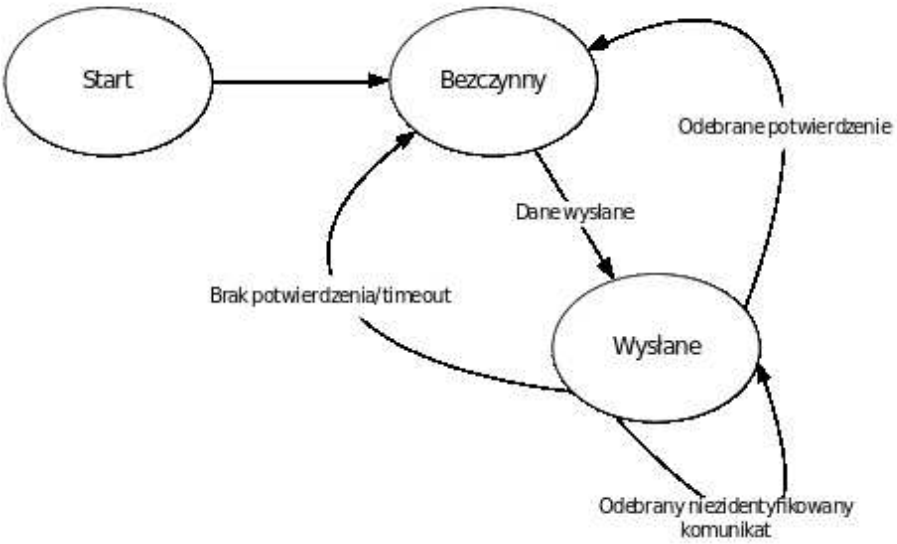
przypadku przesłania maski, serwer powinien w potwierdzeniu zwrócić do klienta obliczony adres sieci. Samodzielnie dodać wymagane pola PDU w celu realizacji dodatkowych fukcji.

3. (Ocena: 5) Opracować własny protokół wg załączonego diagramu stanów. Do przenoszenia jednostek PDU wykorzystać gniazdo datagramowe. Podczas opracowania protokołu wykorzystać następujące informacje, które mają być wymieniane między klientem i serwerem (przygotować własny nagłówek kontrolny - strukturę):
   - typ przenoszonych danych,
   - typ operacji,
   - rozmiar pola danych w B

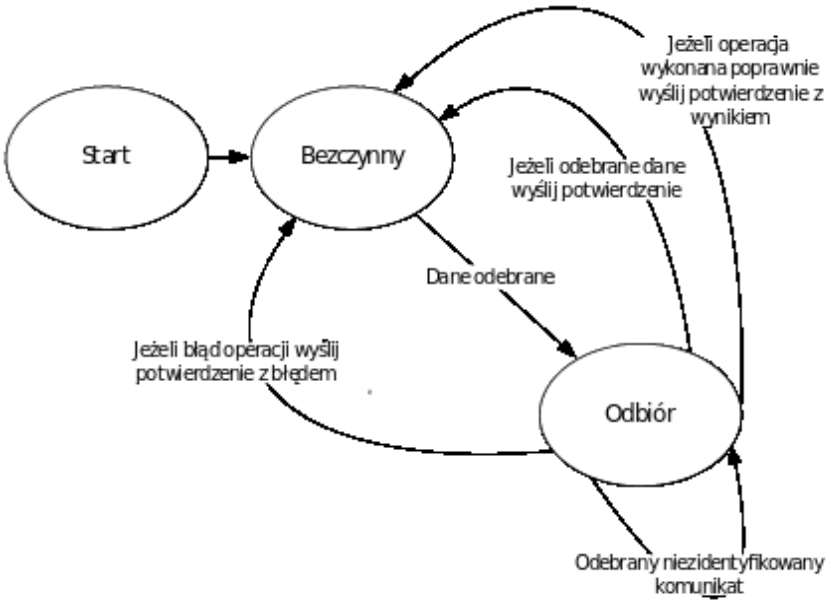
   Przygotować strukturę PDU zawierającą:
   - nagłówek (jak wyżej)
   - pola danych - (liczba pól: 4 + *numer na liście studentów*, rozmiar pola 8b).

   Przygotować strukturę do listy wiązanej:
   (następny, poprzedni, pierwszy element, zmienna typu *int*).
   Odebrane dane z pola danych (liczby 8b) zapisywać jako sumę w zmiennej typu *int* struktury.
   Przygotowane struktury z kolejnych odebranych danych zapisywać do listy wiązanej.
   Maksymalny rozmiar listy z zapisanymi wartościami wynosi 10 elementów. Po przekroczeniu rozmiaru ma być zapisywane 10 najnowszych elementów.
   Po odebraniu każdego kolejnego elementu w serwerze należy wyświetlić zawartość całej listy.
4. Poza przesyłaniem danych do serwera może zostać wysłane żądanie wykonania operacji (odpowiedni kod typu operacji w nagłówku) na zapisanych danych. Typy operacji są następujące:
   - wykonać sumę danych, wynik przesłać do klienta,
   - wykonać odejmowanie przesłanych danych, wynik przesłać do klienta,
   - wykonać sortowanie danych w porządku rosnącym, wynik przesłać do klienta jako listę kolejnych pozycji w polu danych (maksymalnie *numer w dzienniku* pozycji, w przypadku większej liczby przesłać informację o błędzie).
   - wykonać sortowanie danych w porządku malejącym, wynik przesłać do klienta jako listę kolejnych pozycji w polu danych (maksymalnie *numer w dzienniku* pozycji, w przypadku większej liczby przesłać informację o błędzie).

   Po wykonaniu operacji lista zapisanych danych jest czyszczona, a kolejne odbierane przez serwer dane są zapisywane do pustej listy. Wynik operacji po przesłaniu do klienta ma być przez niego wyświetlony na standardowym wyjściu.



Rys. 1. Diagram maszyny stanów klienta



Rys. 1. Diagram maszyny stanów serwera