

Katarzyna Kuźniar, Kazimierz Lal, Tomasz Rak



ĆWICZENIA

Programowanie w **Linuksie**

To naprawdę proste!

Poznaj języki programowania stosowane w systemie Linux
Znajdź, zainstaluj i skonfiguruj niezbędne narzędzia programistyczne
Naucz się tworzyć, kompilować i testować doskonale aplikacje użytkowe



Autorzy:

Katarzyna KUŹNIAR (rozdziały 1-3).

Kazimierz LAL, Politechnika Rzeszowska, Wydział Elektrotechniki i Informatyki,
Katedra Informatyki i Automatyki, Rzeszów, Polska (rozdziały 1-3).

Tomasz RAK, Politechnika Rzeszowska, Wydział Elektrotechniki i Informatyki,
Katedra Informatyki i Automatyki, Rzeszów, Polska (rozdziały 1-3).

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktorzy prowadzący: Michał Mrowiec, Tomasz Waryszak
Projekt okładki: Maciej Pasek

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie?cwplin>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wybranych przykładów dostępne są pod adresem:

<ftp://ftp.helion.pl/przyklady/cwplin.zip>

ISBN: 978-83-246-4345-5

Copyright © Helion 2012

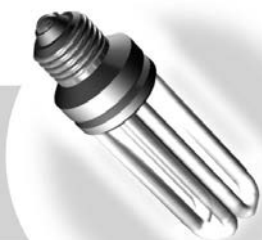
Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

Wprowadzenie	5
Rozdział 1. Języki programowania	9
Język C	11
Język C++	12
Język Java	13
Język Python	15
Język Perl	17
PHP	18
JavaScript	20
Rozdział 2. Darmowe narzędzia programistyczne	23
Edytory tekstowe	23
Kompilatory	48
Zintegrowane środowiska programistyczne	65
Rozdział 3. Zastosowania wybranych środowisk programistycznych	123
Język C	123
Język C++	152
Język Java	169
Język Python	195
Język Perl	211
Graficzny interfejs użytkownika	224
Język PHP	249
JavaScript	267
Narzędzia	280



Zastosowania wybranych środowisk programistycznych



Po wstępnym omówieniu w poprzednim rozdziale różnych środowisk programistycznych teraz przyszedł czas na pokazanie, jak używa się ich w praktyce w codziennej pracy programisty. W poniższym tekście w szczególności przedstawiono, jak ustawia się potrzebne opcje umożliwiające kompilowanie i uruchamianie programów, a także procedury instalacji koniecznych bibliotek oraz dodawania ich do projektów. Użyto do tych celów prostych przykładów oraz zadań programistycznych do samodzielnego rozwiązania. Kody źródłowe czytelnik znajdzie w archiwum na serwerze FTP wydawnictwa Helion.

Język C

Przykładowe programy

Użyte programy prezentują wybrane zagadnienia związane z programowaniem systemowym oraz sieciowym. Głównym celem rozdziału jest jednak zaprezentowanie (na użytecznych przykładach programów

przygotowanych w języku C), jakie opcje i ustawienia zastosować w *Code::Blocks IDE*, a także *GCC*, aby skompilować i uruchomić konkretny program.

Ć W I C Z E N I E

3.1 Mnożenie macierzy

W nowych wersjach *gcc* i *icc* został zaimplementowany standard *OpenMP*.¹ Należy napisać program, który będzie mnożył dwie macierze dwuwymiarowe. Mają zostać przygotowane dwie wersje programu — jedna bez użycia *OpenMP*, a druga z użyciem *OpenMP*. Macierze będą zawierać wartości losowo generowane przez sam program. Operator — z linii poleceń — ma podawać trzy parametry. Pierwszy określający wysokość pierwszej macierzy, drugi szerokość pierwszej macierzy i wysokość drugiej, a ostatni szerokość drugiej macierzy. Interesującym dla operatora wynikiem jest czas wykonywania obliczeń.

Listing 3.1.1. Mnożenie macierzy

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #define DEFAULT_SIZE 100
5 #define true 1
6 #define false 0
```

Na początku trzeba dołączyć podstawowe biblioteki do obsługi wejścia i wyjścia, a także czasu. Należy zdefiniować stałe określające rozmiar (*DEFAULT_SIZE*) macierzy i wartości logiczne (*true*, *false*).

```
7 //funkcja generująca macierze
8 int **tabCreate(int y, int x, int losuj) {
9     int **a, i, i2;
10    a = (int **)malloc(sizeof(int *)*y);
11                                     //przydzielenie pamięci dla jednego wymiaru
12    for(i=0; i<y; i++) {
13        a[i] = (int *)malloc(sizeof(int)*x);
14                                     //przydzielenie pamięci dla drugiego wymiaru
15        if(losuj) {
16            for(i2=0; i2<x; i2++) a[i][i2] = rand()%10; //generowanie elementów
17        } else for(i2=0; i2<x; i2++) a[i][i2] = 0;
```

¹ <http://gcc.gnu.org/gcc-4.2/changes.html>

```

16 }
17 return a;
18 }

```

Ustawienie zmiennej `losuj` na `true` powoduje automatyczne generowanie tablicy, w przeciwnym wypadku jest ona wypełniana zerami.

```

19 //funkcja wyliczająca różnicę między dwoma czasami
20 long long int toddiff(struct timeval *tod1, struct timeval *tod2)
21 {
22     long long t1, t2;
23     t1 = tod1->tv_sec * 1000000 + tod1->tv_usec;
24     t2 = tod2->tv_sec * 1000000 + tod2->tv_usec;
25     return t1 - t2;
26 }
27 int main( int argc, char *argv[])
28 {
29     int **a, **b, **c; //deklaracja 3 tablic
30     int i,j,k;
31     struct timeval tod1, tod2;
32     int temp;
33     int w[3];
34     srand(time(NULL));
35     for(i=0;i<3;i++) {
36         if(argc>=2+i) {
37             temp = 0;
38             sscanf(argv[1+i], "%d", &temp);
39             if(temp>0) w[i] = temp;
40             else {
41                 w[i] = DEFAULT_SIZE;
42             }
43         }
44         else {
45             w[i] = DEFAULT_SIZE;
46         }
47     }
48     printf("arg[%d] = %d\n", i, w[i]);
49 }

```

Powyższy kod od liniiki 35 służy do pobrania wymiarów macierzy. Należy podać trzy parametry. Są to kolejno: „wysokość pierwszej macierzy”, „szerokość pierwszej i jednocześnie wysokość drugiej macierzy”, „szerokość drugiej macierzy”. Gdy nie zostaną podane parametry, rozmiary zostaną ustawione automatycznie (`DEFAULT_SIZE`).

```

49 //utworzenie 3 macierzy
50 a = tabCreate(w[0].w[1], true);
51 b = tabCreate(w[1].w[2], true);
52 c = tabCreate(w[0].w[2], false);

```

```

53 printf("Start\n");
54 gettimeofday(&tod1, NULL);
55 //-----
56 for(i=0;i<w[0];i++){
57   for(j=0;j<w[2];j++) {
58     for(k=0;k<w[1];k++) {
59       c[i][j] += a[i][k]*b[k][j];
60     }
61   }
62 }
63 //-----

```

Kod znajdujący się między linijkami 55 i 63 odpowiada za mnożenie macierzy. Przy późniejszej modyfikacji właśnie tu będą wprowadzane zmiany.

```

64 gettimeofday(&tod2, NULL);
65 //zwalnianie pamięci
66 for(i=0;i<w[0];i++) {
67   free(c[i]);
68   free(a[i]);
69 }
70 for(i=0;i<w[1];i++) {
71   free(b[i]);
72 }
73 free(a);
74 free(b);
75 free(c);
76 printf("Czas: %ld milisekund\n", (long int)(toddiff(&tod2, &tod1) /
1000.0));
77 return 0;
78 }

```

Mnożenie macierzy z użyciem OpenMP

Kolejny program w dużej części ma taki sam kod jak poprzedni, dlatego zostaną przedstawione tylko zmienione lub dodane fragmenty kodu. Pierwszą nową czynnością jest dodanie biblioteki umożliwiającej korzystanie z *OpenMP* — czyli dla gcc <omp.h>. Następnie należy zmodyfikować kod znajdujący się między linijkami 55 i 63.

Listing 3.1.2. Mnożenie macierzy z użyciem OpenMP

```

55 //-----
55a #pragma omp parallel shared(a,b,c,w) private(i, j, k)

```

Mamy zamiar wykorzystać algorytm, który „zrównolegli” mnożenia macierzy, więc trzeba o tym poinformować kompilator. Odpowiednią dyrektywę wpisano w linii 55a. *Omp* jest słowem kluczowym dla *OpenMP*, *parallel* służy do wskazania kompilatorowi, który obszar będzie zrównoleglony, zaś *shared* i *private* określają, które zmienne będą wspólne (wszystkie wątki mają dostęp do tej samej zmiennej), a które prywatne (każdy wątek ma własną kopię zmiennej). Jak widać, wszystkie liczniki pętli zostały zmiennymi prywatnymi, dzięki czemu każdy wątek będzie miał swój własny licznik. Warto wspomnieć, że w praktyce za wspólne zmienne zwykło się przyjmować te, które określają liczbę iteracji pętli, a także zmienne używane w działaniach, których wartości w trakcie obliczeń nie zmieniają się.

```
55b {
55c #pragma omp for schedule(dynamic)
```

Użyta tutaj dyrektywa *schedule* jest dopuszczalna tylko dla pętli *for*. Dzięki niej można w pewnym stopniu kontrolować podział iteracji pomiędzy wątki. *Schedule* przyjmuje dwa parametry — sposób podziału iteracji i rozmiar podzbioru (jest to opcjonalny parametr). Zastosowany przez nas zapis oznacza, że każdy wątek będzie miał przydzielony jednoelementowy podzbiór iteracji.

```
56     for(i=0;i<w[0];i++){
57         for(j=0;j<w[2];j++) {
58             for(k=0;k<w[1];k++) {
59                 c[i][j] += a[i][k]*b[k][j];
60             }
61         }
62     }
62a }
63 //------
```

Skoro oba programy są już gotowe, to teraz należy je skompilować i przetestować. Użyjemy do tego celu najpierw kompilatora *GNU GCC*. Pierwszą wersję programu należy skompilować przy standardowych ustawieniach kompilatora, a w drugim przypadku należy użyć specjalnej flagi *-fopenmp*. Dodatkowo należy wykonać kompilację za pomocą kompilatora *ICC* (opcja do kompilacji z *OpenMP* to *-openmp*).

Analizując wpisy w tabeli 3.1, nietrudno zauważyć, że użycie *OpenMP* zasadniczo zwiększa szybkość mnożenia macierzy, ale również rodzaj użytego kompilatora ma duże znaczenie.

Tabela 3.1. Wpływ użytego kompilatora i bibliotek na szybkość pracy programu mnożącego macierze dwuwymiarowe

Wymiar pierwszej macierzy	Wymiar drugiej macierzy	Narzędzie użyte do kompilacji			
		GCC	GCC w wersji z OpenMP	ICC	ICC w wersji z OpenMP
		Czas potrzebny na mnożenie macierzy [ms]			
500×500	500×500	1345	886	109	72
100×1000	1000×1000	3448	1957	237	138
1000×1000	1000×2000	34448	19521	2383	1364



Programy były testowane na komputerze z procesorem Phenom II N620 i 4 GB pamięci RAM.

Ć W I C Z E N I E

3.2 Utworzenie obrazka

Napisz program generujący i zapisujący kolorowy obraz o wymiarach 400 na 400 pikseli w formacie *PPM* (ang. *Portable PixMap*). Tło obrazka powinno przechodzić — w sposób ciągły — od koloru niebieskiego (w lewym górnym rogu) do zielonego (w prawym dolnym rogu). Istotnymi elementami obrazka powinny być zestawy czerwonych koncentrycznych okręgów tworzące regularną siatkę.

Na początku należy poszukać informacji o formacie *PPM*. Można je znaleźć w Internecie lub też uruchomić terminal w Ubuntu i wpisać polecenie `man ppm`. Na stronach elektronicznego podręcznika użytkownika (`man`) dostępna jest dokładna specyfikacja nagłówka, a także sposobu reprezentacji danych dla interesującego nas formatu. *PPM* to prosty format przeznaczony do przechowywania kolorowych grafik w trybie map bitowych.

Format pliku *PPM* zawiera (wg podręcznika `man`²):

1. „magiczny numer” określający rodzaj pliku — P3 (tekstowy) lub P6 (binarny);

² Specyfikacja PPM z podręcznika `man`.

2. biały znak (spacja, tabulator, CR, LF);
3. szerokość i wysokość zapisane jako dziesiętne znaki ASCII rozdzielone białym znakiem;
4. biały znak;
5. maksymalną wartość komponentu kolorowego, znów jako dziesiętne ASCII;
6. wartości kolejnych kolorów zapisane tekstowo (dla P3) lub binarnie (dla P6).

Listing 3.2.1. Generowanie obrazka

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     const int szer = 400; //szerokość
6     const int wys = 400; //wysokość
7     const float wsp = 3.2; //3.2 ~= (szer+wys) / 255
8     int i,j; //liczniki pętli
9     static unsigned char kolor[3]; //tablica kolorów
10    FILE * file; //deskryptor pliku
11    file = fopen("obraz.ppm", "wb"); //otwarcie pliku
```

Funkcja `fopen()` otwiera plik o nazwie i ścieżce określonej przez pierwszy parametr oraz wiąże z nim strumień. Drugi parametr określa sposób, cel, a także tryb otwarcia strumienia. W naszym przypadku otwierany będzie plik `obraz.ppm` (znajdujący się w tym samym katalogu co kod źródłowy) tylko do zapisu ('w') w trybie binarnym ('b'). Inne możliwe parametry to 'r' (tylko do odczytu), 'a' (dodaj na koniec istniejącego pliku lub utwórz nowy), 'r+' (do odczytu i zapisu jednocześnie — modyfikacja pliku), 'w+' (do zapisu i odczytu — jeśli plik istnieje, zostanie nadpisany). Można wybrać także tryb tekstowy ('t'). Wartość zwracana przez funkcję `fopen()` to adres utworzonego strumienia lub `NULL` w przypadku niepowodzenia.

```
12    fprintf(file, "P6\n%d %d\n 255\n", szer, wys); //nagłówek
```

Funkcja `fprintf()` umieszcza sformatowane dane w strumieniu określonym przez pierwszy parametr. W podanym przypadku dane są zapisywane do pliku w formie określonej przez łańcuch formatujący (drugi parametr). Za symbolem `%d` będzie wstawiany ciąg znaków określony przez parametr nieustalony. Wartością tej funkcji staje się liczba wysyłanych znaków lub EOF w przypadku wystąpienia błędu.

```

13   for(i=0;i<wys;i++){
14       for(j=0;j<szer;j++){
15           kolor[0] = abs(j*j+i*i) % 255; //czerwone kółeczka
16   /* zielony gradient — i+j — coraz mocniejszy kolor, 3.2 — żeby w obrazku mieścił się
      ↪ jeden gradient */
17           kolor[1] = round(abs(i+j)/wsp);
18   /* niebieski gradient od całości odejmujemy i i j */
19           kolor[2] = (int)(abs(255-(i+j)/wsp));
20           fwrite(kolor,1,3,file);

```

Funkcja `fwrite()` wysyła do pliku trzy elementy (o rozmiarze jednego bajta każdy) z tablicy `kolor`. Jeżeli operacja powiedzie się, wartością zwracaną jest liczba wszystkich wysłanych elementów. W przypadku błędu może być mniejsza od wartości określanej przez trzeci parametr (może być także równa zero).

```

21       }
22   }
23   fclose(file); //zamknięcie pliku

```

Funkcja `fclose(file)` zamyka strumień podany przez parametr *file*. Wszystkie bufora związane ze strumieniem są czyszczone. Zwracana wartość to 0 w przypadku powodzenia lub EOF, jeżeli wystąpił jakikolwiek błąd.

```

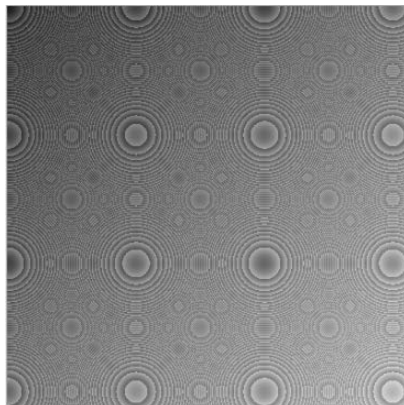
24   return 0;
25 }

```

Po skompilowaniu i uruchomieniu programu zostanie utworzony obrazek (rysunek 3.1), którego należy szukać w katalogu roboczym pliku źródłowego (w przypadku gdy został użyty *Code::Blocks*).

Rysunek 3.1.

Utworzony obraz
w formacie PPM



Na koniec należy zauważyć, że w trakcie pisania tego programu zastosowano sekwencyjny dostęp do danych (od początku do końca). Możliwy jest także swobodny dostęp do danych (np. przez użycie funkcji `fseek()`, `fgetpos()`).



Inspiracją tego zadania była strona³ Wikibooks. Autorzy książki mają świadomość, że opis tego, co ma się znaleźć na obrazku (delikatnie mówiąc), nie jest zbyt precyzyjny i pokazuje, że jeden obraz to więcej niż setki słów.

Ć W I C Z E N I E

3.3

Uprawnienia dla pliku

— zadanie do samodzielnego wykonania

W Linuksie z każdym plikiem powiązane są prawa określające, kto co może z nim zrobić (odczytać, zapisać, wykonać). W praktyce mamy trzy typy użytkowników: właściciel pliku, każdy, kto należy do grupy powiązanej z właścicielem, i inni. Uprawnienia są — z zasady — zapisywane w postaci czterocyfrowych liczb ósemkowych, gdzie 0400 oznacza pozwolenie odczytu, 0200 to pozwolenie zapisu i 0100 umożliwia uruchomienie pliku. Jest tak dla każdej grupy. W celu określenia praw dostępu sumuje się poszczególne wartości.⁴ Najbardziej prawdopodobne uprawnienie to 0755. Zadanie polega na napisaniu programu zmieniającego uprawnienia pliku. Uprawnienia powinny być podane w postaci ósemkowej.



Aby to zrobić, należy użyć `chmod(nazwa pliku, uprawnienia);`.

Ć W I C Z E N I E

3.4

Swobodny dostęp do danych

Zakłada się, że istnieje plik tekstowy, w którym wszystkie linie mają taką samą długość (w naszym przypadku jest 11 znaków, łącznie ze znakiem końca linii). Używając funkcji `fseek()`, należy napisać program wyświetlający zawartość linii o podanym numerze. Na początek

³ http://pl.wikibooks.org/wiki/C/Czytanie_i_pisanie_do_plik%C3%B3w

⁴ Haviland K.: *Unix — Programowanie systemowe*, Wydawnictwo RM, Warszawa 1999.

trzeba utworzyć plik tekstowy o nazwie *lista*. Można posłużyć się np. poleceniem `cat >> lista` lub też dowolnym edytorem tekstowym. Należy pamiętać o stałej liczbie znaków w linii.

Listing 3.4.1. Co jest w linii?

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define DLUGOSC 11
```

Definicja stałej określającej długość linii.

```
4 char linia[DLUGOSC];
```

Definicja zmiennej globalnej określającej bufor, do którego będą wpisywane dane.

```
5 int main()
6 {
7     FILE * plik; //deskryptor pliku
8     int pozycja; //pozycja wskaźnika
9     int linie = 5; //linia, której zawartość mamy wyświetlić
10    long offset;
11    plik = fopen("./lista", "rt"); //otwieramy plik
12    if(plik == NULL)
13    {
14        printf "Nie ma pliku\n";
15    }
16    offset = (linie - 1) * DLUGOSC;
```

Linia 16 określa pozycję, w której należy ustawić „kursor”.

```
17    pozycja = fseek(plik, offset, SEEK_SET);
```

Funkcja `fseek()` zmienia wartość pozycji wskaźnika w pliku związanym ze strumieniem (określonym przez pierwszy parametr) o liczbę bajtów (określoną przez drugi parametr) od wskazanej pozycji (ostatni parametr). Udostępnia trzy wartości sterujące poruszaniem się po pliku: `SEEK_SET` (początek pliku), `SEEK_CUR` (pozycja aktualna) i `SEEK_END` (koniec pliku). Wartość zwracana to 0 w przypadku powodzenia lub -1, gdy wystąpi błąd.

```
18    if(pozycja==-1)
19    {
20        printf "Nie moge ustawic pozycji";
21    }
22    else
23    {
24        fgets(linia, DLUGOSC, plik);
```

Za pomocą funkcji `fgets()` można odczytać linię o długości wskazanej przez drugi parametr z pliku *lista* (trzeci parametr). Wynik jest zapisywany do bufora (pierwszy parametr) i następnie wartość jest zwracana. Funkcja czyta do momentu odczytania `DLUGOSC - 1` znaków lub do napotkania znaku końca linii. Do odczytanego łańcucha dodaje znak końca wiersza i `'\0'` (koniec łańcucha). W przypadku niepowodzenia zwraca `NULL`.

```
25     printf "%s ", linia);  
26 }  
27 fclose(plik);  
28 return 0;  
29 }
```

Można teraz przetestować działanie programu, a następnie utworzyć plik tekstowy z inną liczbą linii i zobaczyć, jak program będzie się zachowywał.

Ć W I C Z E N I E

3.5 Modyfikacja ćwiczenia 3.4

— zadanie do samodzielnego wykonania

Program z ćwiczenia 3.4 ma kilka istotnych wad (np. na sztywno ma wpisany numer linii do wyświetlenia, liczba wierszy musi być znana). Wady te należy usunąć. W szczególności należy napisać funkcję typu `void`, wyświetlającą zawartość linii o podanym numerze, a także — w sekcji `main` — dopisać kod, który będzie wyświetlał zadany przez operatora numer linii z pliku.

Ć W I C Z E N I E

3.6 Wyszukiwanie wzoru

— zadanie do samodzielnego wykonania

Należy napisać program wyszukiujący określoną sekwencję znaków oraz zliczający liczbę jej wystąpień. Wynik (szukaną sekwencję znaków oraz liczbę jej wystąpień) należy zapisać w pliku wyjściowym. Zakłada się, że plik z tekstem został wcześniej przygotowany, a szukana sekwencja znaków będzie zadawana z klawiatury terminalu.



Należy wykorzystać algorytm K-M-P.⁵

Ć W I C Z E N I E

3.7 Program klient-serwer

Należy napisać oprogramowanie, które będzie wyliczało sumę dwóch liczb. Cały proces ma być realizowany przez dwa programy pracujące w trybie klient-serwer. Zadaniem serwera będzie odbieranie od klienta pary liczb, wyliczanie ich sumy i odsyłanie wyniku. Do komunikacji klienta z serwerem należy użyć mechanizmu gniazd oraz protokołu TCP. W drugim kroku program serwera należy zmodyfikować (wykorzystując funkcję `fork()`), tak aby mógł obsłużyć kilka klientów.

Praca serwera to wykonywanie prostej sekwencji działań:

1. start,
2. utworzenie gniazda,
3. bindowanie gniazda i portu,
4. ustawienie gniazda w tryb nasłuchiwania,
5. zaakceptowanie połączenia,
6. wymiana komunikatów z klientem (pętla, w której pobierane są dane od klienta, generowana suma oraz odsyłany wynik sumowania do klienta),
7. zamknięcie połączenia i gniazda.

Działania klienta sprowadzać się będą do:

1. utworzenia gniazda,
2. nawiązania połączenia z serwerem,
3. odebrania wiadomości od serwera,
4. wysłania danych,
5. odebrania wyniku,
6. zakończenia połączenia.

Zgodnie z poleceniem podanym w ćwiczeniu należy użyć gniazd oraz protokołu TCP. Należy wspomnieć, że gniazda sieciowe są dość

⁵ Algorytm K-M-P (ang. *Knutha-Morrisa-Pratta*) wyszukiwania wzorca wykorzystujący tablicę.

uniwersalnym mechanizmem komunikacji między procesami. Ich implementacja w systemie Linux jest wzorowana na kodzie pochodzącym z systemu *BSD-Unix* (ang. *Berkeley System Distribution*). Komunikacja może się odbyć tylko wtedy, gdy oba procesy utworzą gniazda (każdy po swojej stronie). W programie wykorzystano funkcje `socket()`, `bind()`, `listen()`, `accept()`, `send()`, `recv()` i `close()`. Funkcje te wymagają bibliotek `<sys/types.h>` i `<sys/socket.h>`.

Teraz gdy został przedstawiony zarys programu oraz zdefiniowano, jakich narzędzi należy użyć, wystarczy uruchomić *Code::Blocks IDE* i jako pierwszy przygotować nowy projekt (kod dla serwera).

Listing 3.7.1. Kod serwera

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <netinet/in.h>
5 #include <sys/socket.h>
6 #include <sys/types.h>
```

Powyższe linijki zawierają definicje struktur, a także podstawowych funkcji systemowych dla gniazd. `<netinet/in.h>` zawiera definicje struktur `in_addr` i `sockaddr_in` dla rodziny protokołów internetowych. W `<sys/socket.h>` znajduje się struktura `sockaddr`. Zaś `<sys/types.h>` zawiera makrodefinicje oraz definicje typów danych (np. `u_short` — unsigned short). Z funkcji do obsługi gniazd można korzystać po dodaniu `<sys/types.h>` i `<sys/socket.h>`.

```
7 #define MSG_LEN 512
8 #define PORTNUM 60000
```

Powyższe linie prezentują definicje maksymalnej długości wiadomości i numeru portu. Teoretycznie numer portu to liczba z zakresu od 0 do 65 535, ale w praktyce do naszych celów nie możemy użyć tzw. portów dobrze znanych (przedział od 0 do 1023) oraz lepiej nie używać „portów zarejestrowanych” (przedział od 1024 do 49 151). Po prostu te porty są (odpowiednio) zastrzeżone lub zarezerwowane przez IANA (ang. *Internet Assigned Numbers Authority*) dla używanych powszechnie usług.⁶

```
9 int main(void) {
10 int s, result, cs = 0;
```

⁶ Numery portów: <http://www.iana.org/assignments/port-numbers>.

```

11 int a,b, wynik;
12 char msg[MSG_LEN];
13 struct sockaddr_in laddr;
14 socklen_t laddr_len;
15 s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); //tworzymy gniazdo
16 if (s < 0) {//obsługa błędu
17     perror("socket");
18     return 1;
19 }
```

Za pomocą `socket()` tworzone jest gniazdo. Funkcja ta ma trzy parametry. Pierwszy z nich określa rodzinę — w tym przypadku jest to stała `AF_INET` oznaczająca protokoły Internetu (można spotkać także nazwę `PF_INET`). Przedrostek `AF_` oznacza rodzinę adresów, a `PF_` rodzinę protokołów. Obie notacje można stosować zamiennie (są równoważne). Drugi argument funkcji `socket()` określa rodzaj gniazda (`SOCK_STREAM` — gniazdo strumieniowe). Ostatni parametr wskazuje protokół. Zgodnie z treścią zadania należy zastosować protokół TCP, więc naturalnym będzie podanie stałej `IPPROTO_TCP`. Funkcja `socket()` zwraca liczbę całkowitą, którą nazywa się deskryptorem gniazda.

```

20 memset(&laddr, 0, sizeof(struct sockaddr)); //zerowanie struktury laddr
21 laddr.sin_family = AF_INET; //protokoły Internetu
22 laddr.sin_port = htons(PORTNUM); //serwer będzie nasłuchiwać na porcie
23 laddr.sin_addr.s_addr = htonl(INADDR_ANY); //każdy klient może się połączyć
```

Linijki 20 – 23 powodują dowiązanie adresu lokalnego. Dzięki temu klient będzie mógł wysyłać dane do serwera.

```

24 result = bind(s, (struct sockaddr*)&laddr, sizeof(struct
    ↪sockaddr)); //bindujemy port z gniazdem
25 if (result < 0) {//obsługa błędu
26     close(s);
27     perror("bind");
28     return 1;
29 }
```

Po utworzeniu fizycznego gniazda należy powiązać je z nazwą, tak aby inne procesy mogły się do niego łatwo odwołać. Służy do tego funkcja `bind()`. W jej wywołaniu trzeba podać trzy parametry. Pierwszy określa deskryptor gniazda, drugi to wskaźnik do struktury zawierającej adres, a ostatni podaje rozmiar struktury.

```

30 result = listen(s, 1); //ustawienie gniazda w tryb nasłuchiwania
31 if (result < 0) {//obsługa błędu
32     close(s);
33     perror("listen");
34     return 1;
35 }
```

Kolejnym krokiem jest wskazanie, że serwer „chce przyjmować połączenia — określoną liczbę połączeń”. Służy do tego funkcja `listen()` wymagająca podania dwu parametrów. Pierwszym jest deskryptor gniazda, a drugi określa maksymalną liczbę połączeń (w naszym przypadku jest to jedno połączenie), które system umieszcza w kolejce w oczekiwaniu na wykonanie funkcji `accept()`.

```
36 laddr_len = sizeof(struct sockaddr);
37 cs = accept(s, (struct sockaddr*)&laddr, &laddr_len);
                                     //akceptowanie połączenia
38 if (cs < 0){//obsługa błędu
39     perror("accept");
40     return 1;
41 }
```

Funkcja systemowa `accept()` wywoływana jest po `listen()`. Pobiera ona pierwsze żądanie połączenia z kolejki i tworzy nowe gniazdo o takich samych właściwościach, jakie ma jej pierwszy argument (deskryptor gniazda). Dwa pozostałe adresy określają klienta, którego dotyczy połączenie. Po wywołaniu `accept()` wątek serwera jest blokowany i czeka na połączenie.

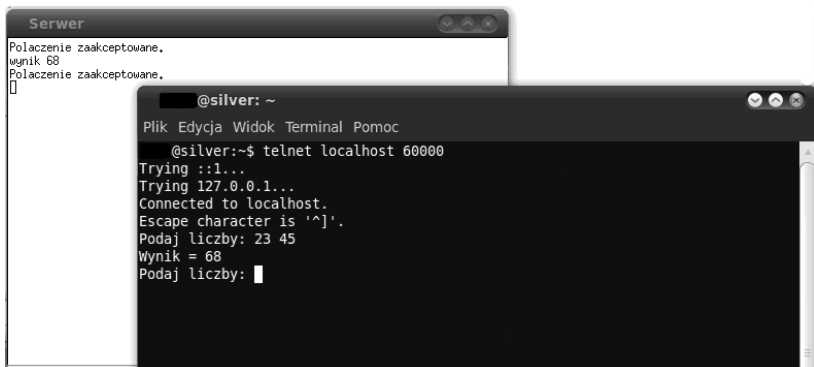
```
42 while(cs){
43     memset(msg, 0, MSG_LEN); //wyzeroowanie bufora wiadomości
44     printf("Połączenie zaakceptowane.\n");
45     strncpy(msg, "Podaj liczby: ", MSG_LEN);
46     result = send(cs, msg, strlen(msg), 0); //wyslij tekst do klienta
47     if (result <= 0) { //obsługa błędu
48         close(cs); //zamknięcie gniazda
49         perror("send");
50         return 1;
51 }
```

Kolejną funkcją umożliwiającą komunikację jest funkcja `send()` służąca do wysyłania danych. Wymaga podania czterech parametrów: deskryptora gniazda, wiadomości w postaci ciągu znaków, długości wiadomości w bajtach oraz flagi (u nas nie korzysta się z żadnej, więc należy podać 0). W takiej postaci funkcja `send()` jest funkcją blokującą. Zostanie odblokowana wtedy, gdy wszystkie dane zostaną wysłane. Taką samą listę parametrów przyjmuje funkcja `recv()`, która odbiera dane.

```
52 memset(msg, 0, MSG_LEN); //wyzeroowanie bufora wiadomości
53 result = recv(cs, msg, MSG_LEN, 0); //oczekiwanie na odpowiedź klienta
54 if (result <= 0) { //obsługa błędu
55     close(cs);
56     perror("recv");
```

```
57     return 1;
58 }
59 msg[result-2]='\0';
60 sscanf(msg,"%d %d",&a,&b); //odczytanie liczb
61 wynik=a+b; //wykonanie działania
62 printf("wynik %d\n", wynik); //wydrukowanie wyniku na serwerze
63 sprintf(msg,"Wynik = %d\n",wynik); //wpisanie wyniku do zmiennej msg
64 result = send(cs, msg, strlen(msg)+1, 0); //wyslij tekst do klienta
65 if (result < 0) { //obsługa błędu
66     close(cs);
67     perror("send");
68     return 1;
69 }
70 } //koniec pętli
71 close(cs); //zamknięcie połączenia
72 close(s); //zamknięcie gniazda
73 return 0;
74 }
```

Po napisaniu kodu programu i wykreowaniu projektu należy uruchomić serwer. W efekcie powinno ukazać się puste okienko. Kolejnym krokiem będzie przetestowanie serwera. Najprościej wykorzystać do tego celu *telnet*, jako parametry podając nazwę hosta i port, z którym ma zostać nawiązane połączenie. Po uruchomieniu terminalu i pojawieniu się znaku zachęty należy wpisać: `telnet localhost 60000`. Gdy wszystko zadziała prawidłowo, w oknie terminalu powinna ukazać się wiadomość od serwera. Rysunek 3.2 przedstawia wynik działania omawianego testu.



Rysunek 3.2. Wynik testowania serwera

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA



Helion SA

Programowanie w **Linuksie**. ĆWICZENIA

Odkryj moc drzemiącą w Linuksie!
Naucz się w nim programować!



Systemy należące do rodziny Linux zdobyły ogromną popularność jako stabilne, wydajne, bezpieczne i darmowe środowiska operacyjne zarówno wśród użytkowników prywatnych, jak i wielkich korporacji. Dziś środowiska te można spotkać już niemal wszędzie, a różne dystrybucje walczą o palmę pierwszeństwa i bez kompleksów konkurują z zamkniętymi komercyjnymi systemami, takimi jak MS Windows czy Mac OS X. Wzrostowi popularności Linuksa wśród „zwykłych” użytkowników towarzyszy oczywiście wzrost zapotrzebowania na oprogramowanie użytkowe, ponieważ nikt nie będzie chciał korzystać z systemu operacyjnego, jeśli nie będzie w stanie znaleźć odpowiednich dla siebie narzędzi pracy.

Wielu programistów niechętnie odnosi się do tworzenia aplikacji działających pod Linuksem, ponieważ wydaje im się, że brak w nim odpowiednich narzędzi, system stawia wyższe wymagania, a sam kod pisze się trudniej. Tymczasem Linux obsługuje się podobnie jak inne systemy, a możliwości, jakie proponuje, są naprawdę spore. *Programowanie w Linuksie. Ćwiczenia* to książka, która w praktyczny sposób prezentuje najbardziej popularne języki programowania w tym systemie oraz ich zastosowania. Pomaga też wyposażać warsztat programisty w bezpłatne narzędzia i właściwie je skonfigurować. Informuje, jak rozpocząć tworzenie aplikacji i rozwinąć swoje umiejętności.

- Przegląd najpopularniejszych języków programowania w Linuksie
- Wybór, instalacja oraz konfiguracja narzędzi i środowisk programistycznych
- Konfiguracja i korzystanie z edytorów kodu oraz kompilatorów
- Wykrywanie i poprawianie błędów w programach
- Automatyzacja pracy programisty dzięki użyciu skryptów
- Tworzenie aplikacji graficznych, systemowych i sieciowych
- Programowanie wielowątkowe
- Generowanie i przetwarzanie plików w różnych formatach

helion.pl
księgarnia
internetowa

Nr katalogowy: 8011



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:
0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:

➤ <http://helion.pl/promocje>

Książki najchętniej czytane:

➤ <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

➤ <http://helion.pl/nowosci>

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: helion@helion.pl

<http://helion.pl>

sięgnij po **WIECEJ**



KOD KORZYŚCI

ISBN 978-83-246-4345-5



Cena 39,00 zł

Informatyka w najlepszym wydaniu

9 788324 164345