***Лабораторна робота №17***

***«Розробка програм на мові C/C ++ в ОС Linux»***

***Етапи створення програми на мові С/С++***

На рис. 1 представлена послідовність розробки програми на мові С. В ОС Linux один з популярних наборів компіляторів - **GNU Compiler Collection**, або **GCC**. Він включає компілятори з мов **C**, **C++**, **Java**, **Fortran** і ін. Ім'я компілятора мови **С** - ***gcc***, мови **С++** - ***g++***. Для мови **С** також можна використовувати компілятор **cc**, наявний у багатьох ОС сімейства UNIX. Для мови **С** розширення вихідних файлів - ***.c*** і ***.h***, для мови ***С++*** - ***.cpp***, ***.hpp***, ***.cxx***, ***.hxx***, ***.C*** і ***.H***. Розширення об'єктних файлів - ***.o***, бібліотек об'єктних файлів - ***.a***. Виконувані файли в Linux зазвичай не мають розширення.



***Послідовність команд для створення програми***

1. Програма складається з одного вихідного файлу

|  |  |
| --- | --- |
| **Дія** | **Приклад** |
| 1) У текстовому редакторі (**nano**, **vi**, **gedit**, **emacs** і т.п.) створити і зберегти вихідний текстовий файл (наприклад, **myprog.c**) | ***$ nano***  ………………  робота з nano |
| 2) Відкомпілювати і скомпонувати програму. Результат - виконуваний файл (наприклад, **myprog**) | ***$ gcc -o myprog myprog.c*** |
| 3) Виконати програму | ***$ myprog*** або  ***$ myprog param1 param2*** |

1. Програма складається з декількох вихідних файлів

|  |  |
| --- | --- |
| **Дія** | **Приклад** |
| 1) У текстовому редакторі створити і зберегти кожен з вихідних файлів | ***$ nano***  ………………  робота з nano |
| 2) Відкомпілювати окремо кожен текстовий файл. Результат - об'єктні файли (**module1.o**, **module2.o**, **module3.o**) | ***$ gcc -c module1.c***  ***$ gcc -c module2.c***  ***$ gcc -c module3.c*** |
| 3) Скомпонувати об'єктні файли. Результат - здійсненний файл. | ***$ gcc -o myprog module1.o module2.o module3.o*** |
| 4) Виконати програму | ***$myprog*** або  ***$myprog param1 param2*** |

Компілятор за замовчуванням шукає заголовні файли в поточному каталозі і в каталогах, в яких встановлені заголовки для стандартних бібліотек. Якщо заголовки знаходяться в будь-якому іншому місці, то слід використовувати команду для виклику компілятора ***gcc***/***g++*** з опцією ***-I***. Нехай, наприклад, потрібно включити в програму файл ***obr.h*** з каталогу ***/home/ivanov/incl*** і нехай каталог ***/home/ivanov/texts*** є поточним. Тоді слід дати команду:

***$ gcc -c -I ../incl obr.c***

Або можна вказати абсолютну ім'я для каталогу ***incl***:

***$ gcc -c -I /home/ivanov/incl obr.c***

При цьому в програмі директива ***include*** повинна мати вигляд:

***#include "obr.h"***

Стандартна бібліотека **С** (що містить, зокрема, функцію ***printf*** і ін.) компонується в виконуваний файл автоматично. Для включення нестандартної бібліотеки (наприклад, бібліотеки ***libpthreads.a***, яка містить функції стандарту POSIX для роботи з потоками) слід скомпонувати програму з опцією ***-l***, наприклад:

***$ gcc -o obr main.o obr.o –l pthreads***

За цією командою в програму буде включена бібліотека ***libpthreads.a***, при цьому автоматично до імені бібліотеки додасться префікс ***lib*** і суфікс ***a***. Компонувальник буде шукати бібліотеки в ряді стандартних каталогів, включаючи каталоги ***/lib*** і ***/usr/lib***. Якщо ж бібліотека, яку необхідно включити, знаходиться в якому-небудь іншому каталозі, то в командному рядку слід використовувати опцію ***-L*** спільно з опцією ***-l***, наприклад:

***$ gcc -o obr main.o obr.o –L /usr/local/libs –l pthreads***

За цією командою бібліотека ***libpthreads.a*** буде включена з каталогу ***/usr/local/libs***. Якщо даний каталог є поточним, то команду можна задати в такій формі:

***$ gcc -o obr main.o obr.o –L . –l pthreads***

***Використання утиліти Make***

Утиліта ***make*** використовується для автоматизації розробки програм. Для цього в текстовому файлі з ім'ям ***makefile*** вказується наступна інформація:

* цільові файли (цілі), які необхідно побудувати;
* правила для їх побудови;
* залежності, що визначають коли цю мету необхідно перебудувати заново.

Так **makefile** має наступний вигляд:

***CFLAGS=-c***

***CC=gcc***

***obr: main.o obr.o***

***<Tab> $(CC) -o obr main.o obr.o***

***main.o: main.c***

***<Tab> $(CC) $(CFLAGS) main.c***

***obr.o: obr.c***

***<Tab> $(CC) $(CFLAGS) obr.c***

Тут зліва від '***:***' вказана мета, справа - її залежності. Правило для побудови цілі вказано на наступному рядку, яка повинна починатися з символу табуляції. ***CC*** і ***CFLAGS*** - змінні утиліти ***make*** (в даному прикладі ***CC*** задає ім'я компілятора, ***CFLAGS*** - опцію компілятора). Значення для змінної може бути задано у файлі ***makefile*** (як в прикладі) і/або в командному рядку (наприклад, ***$ make CC=g++***).

Після підготовки файлу ***makefile*** для створення програми, що виконується достатньо ввести команду:

***$ make***

або

***$ make імя\_make\_файлу***

(якщо ***імя\_make\_файлу*** відмінне від ***makefile***)

За цією командою утиліта ***make*** виконає файл ***makefile*** з поточного каталогу та автоматично перетранслює тільки ті файли, які необхідно.

***Доступ до параметрів командного рядка***

При запуску програми в командному рядку після імені програми можуть зазначатися параметри, що називаються параметрами (аргументами) командного рядка. Вони поділяються проміжками або, якщо параметри самі містять проміжки, поміщуються в апострофи. Для доступу до цих параметрів у програмі на мові **С** заголовок функції ***main*** повинен містити два параметра: кількість параметрів (***argc***) і масив вказівників на рядки (***argv***), що закінчується покажчиком NULL. i-ий рядок є значенням i-го параметра, при цьому рядок з номером 0 містить ім'я програми.

**Приклад 1.** Друк параметрів командного рядка

***#include <stdio.h>***

***int main (int argc, char \*argv[])***

***{***

***if ( argc < 2 )***

***{***

***printf( “Usage : %s parameter\n”, argv[0] ) ;***

***return 1;***

***}***

***printf("Starting program %s \n", argv[0]) ;***

***printf("with %d parameter(s)\n", argc-1) ;***

***printf("First parameter is %s\n", argv[1]) ;***

***return 0;***

***}***

***Змінні оточення***

Будь-яка програма, що виконується, має своє оточення, що представляє собою набір строкових пар виду ***змінна=значення***. Відповідно до угоди імена змінних пишуться великими літерами.

У програмі можна отримати значення змінної оточення за допомогою функції ***getenv*** з ***<stdlib.h>***. Ця функція приймає ім'я змінної і повертає її значення у вигляді рядка символів або ***NULL***, якщо змінна не визначена в оточенні. Для установки і очищення змінних оточення використовуються функції ***setenv*** і ***unsetenv*** відповідно. Для доступу до всіх змінних оточення використовується спеціальна глобальна змінна ***environ***. Ця змінна має тип ***char\*\**** і є масивом покажчиків на символьні рядки, що закінчується вказівником ***NULL***.

**Приклад 2.** Друк змінних середовища оточення

***#include <stdio.h>***

***extern char\*\* environ;***

***int main ()***

***{***

***char\*\* var;***

***for (var = environ; \*var != NULL; ++var)***

***printf ("%s\n", \*var);***

***return 0;***

***}***

***Обробка помилок в системних викликах***

Більшість системних викликів повертає нуль, якщо операція виконана успішно, і ненульове значення в іншому випадку. У разі помилок в глобальну змінну ***errno*** записується додаткова інформація - ціле число, яке ідентифікує причину помилки. Якщо включити в програму ***<errno.h>***, можна посилатися на помилки за їх символічними іменами, наприклад, ***EACCES*** або ***EINVAL***.

Отримати в програмі текстове повідомлення про помилку можна двома способами:

1) За допомогою функції ***strerror*** із ***<string.h>***; функція приймає ***errno*** і повертає рядок з описом помилки;

2) За допомогою функції ***perror*** з ***<stdio.h>***; функція приймає рядок, яку вона виводить в якості префікса перед повідомленням про помилку (наприклад, це може бути ім'я файлу, де сталася помилка) і виводить опис помилки в потік ***stderr***.

**Приклад 3.** У програмі робиться спроба відкрити файл. Системний виклик ***open*** повертає дескриптор файлу або ***-1*** у випадку помилки.

**Варіант 1**

***fd = open ("inputfile.txt", O\_RDONLY);***

***if (fd == -1)***

***{***

***fprintf (stderr, "error opening file: %s\n", strerror (errno));***

***exit (1);***

***}***

**Варіант 2**

***fd = open ("inputfile.txt", O\_RDONLY);***

***if (fd == -1)***

***{***

***perror("inputfile.txt");***

***exit (1);***

***}***

Зверніть увагу, що успішне виконання системного виклику не приводить до скидання змінної ***errno***.

***Завдання***

1. Знайдіть у довіднику ***man*** призначення основних опцій компілятора ***gcc***: ***-c***, ***-S***, ***-E*** та ***-o***.
2. Створіть у вашому каталозі каталог ***lab2*** для створення у ньому програм на мові **C** і зробіть його поточним для даної лабораторної роботи.
3. Напишіть на мові **С**/**С++** і виконайте програму ***Hello***, яка виводить рядок ***"Hello, world"***.
4. Напишіть на мові **С/С++** і виконайте програму ***obr***, що складається з двох модулів:

* модуль ***obr.c*** містить функцію ***double obr(int i)***, яка повертає число, зворотне числу ***i***;
* модуль ***main.c*** містить функцію ***main()***, яка запитує у користувача ціле число i та виводить значення ***obr(i)***.

1. Створіть файл ***makefile*** для програми із завдання 4, попередньо видаливши файли ***\*.o*** і ***\*.obr*** з поточного каталогу. Потім виконайте команду ***make***.
2. Виконайте **Приклад 1** з даної лабораторної роботи та змініть його таким чином, щоб у якості параметрів командного рядка були цілі числа. Для перетворення рядка в ціле число використовуйте функцію ***atoi***.
3. Виведіть всі змінні середовища оточення програмних шляхом на мові **C/C++**.

***Контрольні питання***

1. Як отримати проміжний (після препроцесора) вихідний текст програми?

2. Як створити програму з декількох модулів?

3. Як додати в програму бібліотеку об'єктних модулів?

4. Як використовувати утиліту ***make*** для створення програм?

5. Як в програмі отримати доступ до параметрів командного рядка і змінним оточення?

6. За допомогою яких функцій можна вивести повідомлення про помилку в системному виклику?