Advanced SUSE Linux Enterprise Server Administration (Course 3038)

Chapter 7

Compile Software from Source

Objectives

- Understand the Basics of C Programming
- Understand the GNU Build Tool Chain
- Understand the Concept of Shared Libraries
- Perform a Standard Build Process

Understand the Basics of C Programming

- Objectives
  - The Difference Between Source Code and an Executable
  - The Structure of a Simple C Program
  - How to Compile a Simple C Program

The Difference Between Source Code and an Executable

- Types of programming languages
  - Script languages
    - Developed with just a text editor
    - Executed with the help of an interpreter program
    - Examples: Perl, PHP, Python, Shell scripting language
  - Compiler languages
    - Created with a text editor
    - Normally there is no interpreter software available
    - Code needs to be converted into a binary format
      - Done by a special program called the *compiler*
    - Examples: C, C++, and Fortran

The Difference Between Source Code and an Executable (continued)

Table 7-1

Language Type	Advantages	DIsadvantages
Script language	• Most script languages are relatively easy to learn. • The development process in a script language is rather fast. • Script programs can basically run on all platforms where the interpreter is available, without changing the program code.	• The execution of scripts application is rather slow. • It's not possible to do system programing with scripting languages (such as operating systems or device drivers).
Compiler language	• The execution is very fast compared with scripting languages. • It is possible to do system programming (such as operating systems or device drivers).	• Compiler languages can be difficult to learn. • The development process usually takes longer. • The source code needs to be recompiled before it can be executed on a different platform.

The Structure of a Simple C Program

- Source code of a simple C program

```
#include <stdio.h>

int main(void)
{
  char name[80];
  printf("Please enter your name: ");
  scanf("%s", name);
  printf("Your name is: %s\n", name);
  return(0);
}
```

How to Compile a Simple C Program

- C program needs to be compiled into a binary file
  - For this you need a C compiler
- gcc (Gnu C Compiler)
  - The standard C compiler in Linux
- Syntax
  - gcc my_name.c -o my_name

Exercise 7-1 Compile a Simple C Program

- In this exercise, you will compile a simple C program

Understand the GNU Build Tool Chain

- Objectives
  - Use configure to Prepare the Build Process
  - Use make to Compile the Source Code
  - Use make install to Install the Compiled Program
  - Install the Required Packages for a Build Environment

Use configure to Prepare the Build Process

- Prepare the source code with a configure script
- Reasons
  - Many applications can be compiled on different UNIX systems and hardware platforms
  - Build process is controlled by a program called make
  - Use configure to enable or disable certain features
- Run configuration script
  - ./configure
- List all available configure options
  - ./configure --help

Use make to Compile the Source Code

- Use tool make
  - To compile multiple source files in the correct order
- Make is controlled by Makefiles
  - Usually generated by the configure script
- Use make to install and uninstall the program
- Makefile can perform the following tasks
  - Compile the program from source
  - Install the program
  - Uninstall the program
  - Clean up directory where compilation is performed

Use make to Compile the Source Code (continued)

- Makefile example

```
# Makefile for my_name
all: my_name
my_name: my_name.c
gcc my_name.c -o my_name
install: my_name
install -m 755 my_name /usr/local/bin/my_name
uninstall: /usr/local/bin/my_name
rm -f /usr/local/bin/my_name
clean:
rm -f my_name
```

Use make install to Install the Compiled Program

- Last step when installing a program from source
  - Install the binary file and additional files
  - Usually done with make and an install target
    - In the corresponding Makefile
- Syntax
  - make install

Install the Required Packages for a Build Environment

- Easiest way to install all required packages
  - Select the selection C/C++ Compiler and Tools
    - In the YaST package manager

Install the Required Packages for a Build Environment (continued)

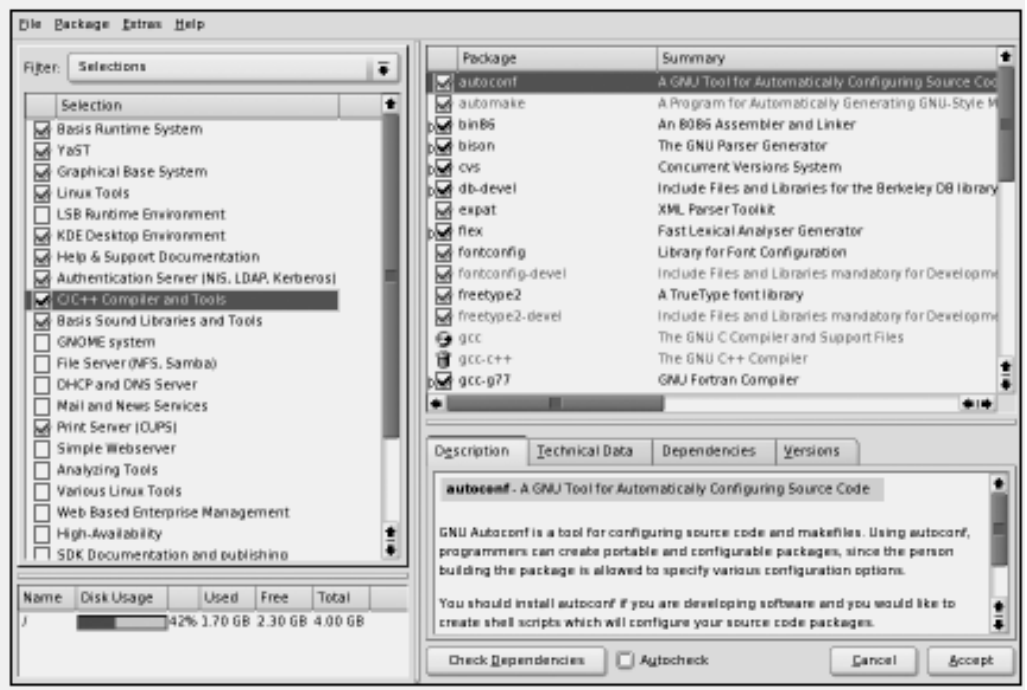

Figure 7-1

Understand the Concept of Shared Libraries

- Tasks on a system
  - Performed by more than one application
- Shared library
  - File on the hard disk
  - Loaded into the main memory
    - When an application requires its functionality
- Task of finding and loading required libraries
  - Performed by the program ld

Understand the Concept of Shared Libraries (continued)

Figure 7-2 illustrates how shared libraries work.

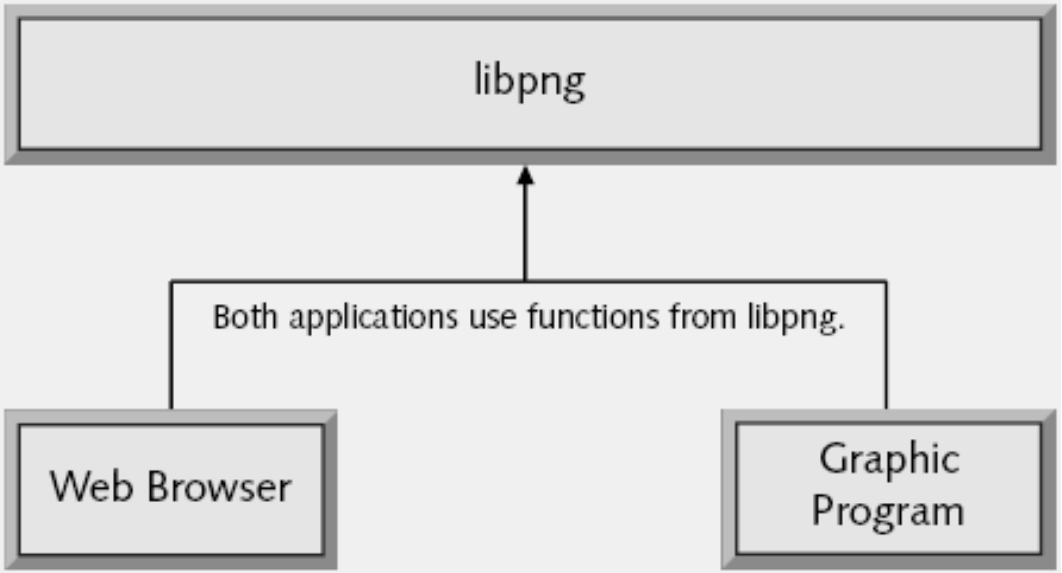

Figure 7-2

Understand the Concept of Shared Libraries (continued)

- Shared library parts
  - The shared library file itself
  - The header files of the library
- C programming language requires
  - That every function used has to be declared
- Header files of a library
  - Need to be installed on a system to compile software

Perform a Standard Build Process

- Objective
  - Compile and install program xpenguins from a source archive
- Extract the tar archive
  - tar xzf xpenguins-2.2.tar.gz
- Change to the source directory
  - cd xpenguins-2.2/
- Run the configure script
  - ./configure

Perform a Standard Build Process (continued)

- Start the compilation process
  - make
- Install the software
  - make install

Exercise 7-2 Compile Software from a Source Package

- In this exercise, you will do the following:
  - Part I: Compile a Source Package
  - Part II: Run the Application

Summary

- Use the GNU Build Tool Chain
  - To compile program source code into software
- Script languages and compiler languages
  - Used to create text files that contain program instructions
- The C programming language
  - Common compiler language
  - Uses the GNU C Compiler (gcc)
- Most C source code files are available on the Internet
  - In compressed tar format

Summary (continued)

- Steps to compile C source code files
  - Run configure script
    - Creates Makefiles
  - Run make command
    - Compiles the program using gcc
  - Run make install
    - To install compiled program
- Shared libraries
  - Functions used by several different programs
- C source code files use header files
  - To locate shared library files