Linux系统下软件编译与安装

目录

01 常用编译器介绍

02 Linux系统下软件编译流程

03 静态库和静态链接

04 动态库和动态链接

05 Linux系统下软件安装示例

目录

01 常用编译器介绍

02 Linux系统下软件编译流程

03 静态库和静态链接

04 动态库和动态链接

05 Linux系统下软件安装示例

常用编译器介绍

✓ GNU编译器

GCC（GNU Compiler Collection，GNU编译器套件）

- GNU开发的编程语言编译器。它是一套以GPL及LGPL许可证所发行的自由软件，也是GNU计划的关键部分，亦是自由的类Unix及苹果电脑Mac OS X 操作系统的标准编译器。GCC（特别是其中的C语言编译器）也常被认为是跨平台编译器的事实标准
- GCC可处理C、C++、Fortran、Pascal、Objective-C、Java以及Ada等语言

编程语言	编译器调用名称
C	gcc
C++	g++
Fortran77	gfortran
Fortran90/95	gfortran

常用编译器介绍

✓ PGI编译器

PGI Accelerator

- PGI编译器是The Portland Group推出的一款编译器产品，支持C、C++和Fortran
- PGI编译器还支持HPF（High Performance Fortran，Fortran90的并行扩展）编程语言，支持CUDA Fortran
- 已经被NVIDIA收购

编程语言	编译器调用名称
C	pgcc
C++	pgCC
Fortran77	pgf77
Fortran90/95	pgf90/pgf95
HPF	pghpf

常用编译器介绍

✓ Intel编译器

Intel Composer XE

- Intel编译器是Intel公司发布的x86平台（IA32/INTEL64/IA64/MIC）编译器产品，支持C/C++/Fortran编程语言
- Intel编译器针对Intel处理器进行了专门优化，性能优异，在其它x86处理器平台上表现同样出色

编程语言	编译器调用名称
C	icc
C++	icpc
Fortran77	ifort
Fortran90/95	ifort

常用编译器介绍

- ✓ 其它x86编译器

Open64（C、C++、Fortran77/90/95）

PathScale （C、C++、Fortran77/90/95）

EKOPath（C、C++、Fortran77/90/95）

Absoft Fortran Compiler

g95 Fortran Compiler

Lahey Fortran Compiler

…

目录

01 常用编译器介绍

02 Linux系统下软件编译流程

03 静态库和静态链接

04 动态库和动态链接

05 Linux系统下软件安装示例

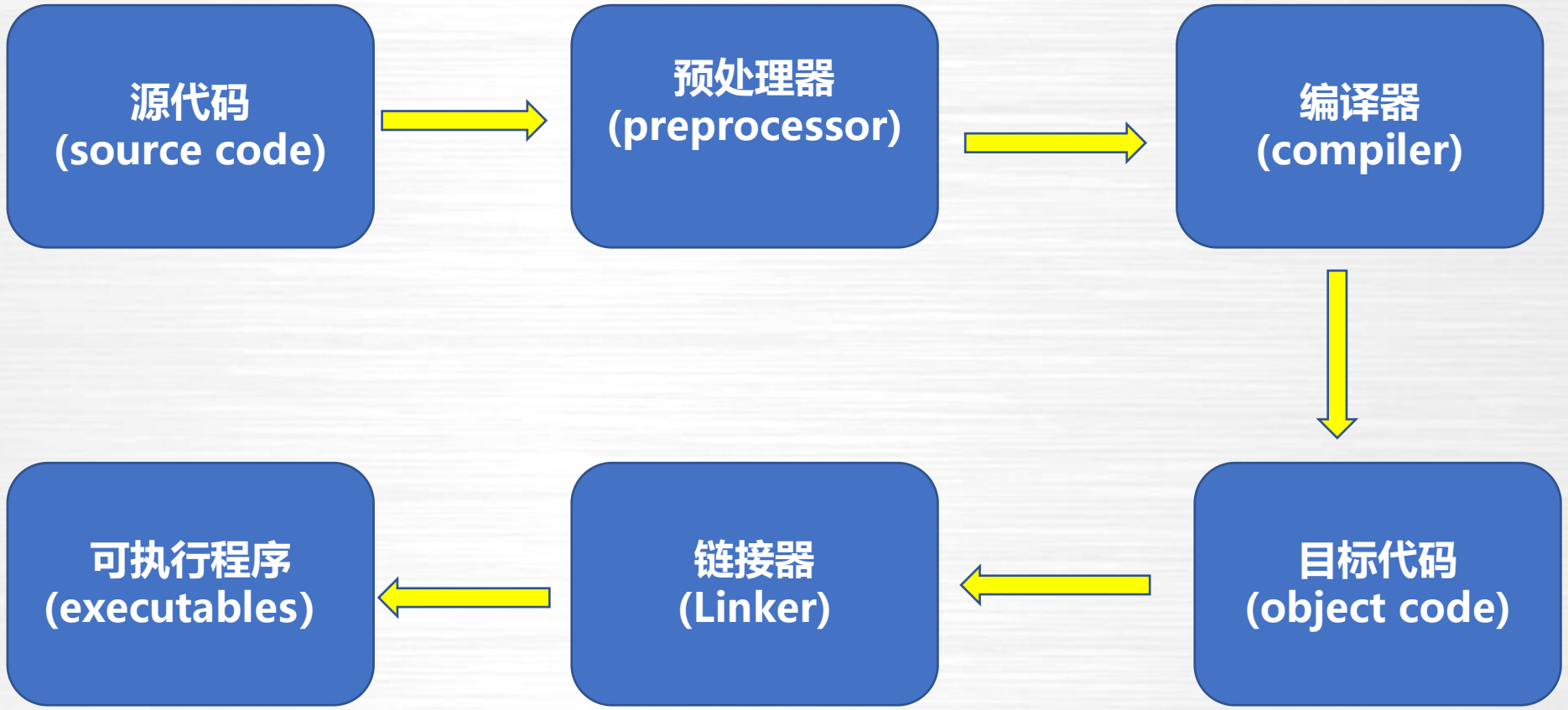
Linux环境下软件编译流程

- 示例代码如下：

```c
#include <stdio.h>
#include <time.h>

#define Maxtime 100

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

        int StartTime = 0;
        int EndTime = 0;
        int Sum = 0;
        int i = 0;

        StartTime = clock();

        for (i = 0; i < Maxtime; i ++){
                printf("%d\n",i);
        }

        EndTime = clock();

        printf("\n\nRunning time: %dms\n", EndTime-StartTime);
}
```

Linux环境下的软件编译流程

- 软件编译整体流程如下所示：



Linux环境下的软件编译

■程序编译生成可执行文件过程分为四个步骤：

- .c文件到.i文件（预处理）
- .i文件到.s文件（编译）
- .s文件到.o文件（汇编）
- .o文件到可执行文件（链接）

目录

01 常用编译器介绍

02 Linux系统下软件编译流程

03 静态库和静态链接

04 动态库和动态链接

05 Linux系统下软件安装示例

Linux环境下的软件编译流程

■在Linux中，链接主要分为静态链接和动态链接。

- 静态链接是指将目标文件中所需的函数和库文件代码全部复制到可执行文件中的链接方式。
- 当程序链接时，静态库的目标文件被直接链接到可执行文件中

■ 静态库和静态链接的关系：

- 静态库是静态链接的一种实现方式，命名规范为libXXX.a，程序在编译时将这些目标文件和库文件全部链接到程序中。

Linux环境下的软件安装示例

静态库文件有如下特点：

静态库

- 命名规范为libXXX.a
- 库函数会被连接进可执行程序，可执行文件体积较大
- 可执行文件运行时，不需要从磁盘载入库函数，执行效率较高
- 库函数更新后，需要重新编译可执行程序

目录

01 常用编译器介绍

02 Linux系统下软件编译流程

03 静态库和静态链接

04 动态库和动态链接

05 Linux系统下软件安装示例

Linux环境下的软件编译流程

■动态库和动态链接：

- 动态链接是指将目标文件中所需的函数和库文件代码在程序运行时复制到可执行文件中的链接方式。
- 当程序运行时，动态库的目标文件被直接链接到可执行文件中。

■ 动态库和动态链接的关系：

- 动态库是动态链接的一种实现方式，命名规范为libXXX.so。程序在运行时将这些目标文件链接到程序中，而不需要将库文件的目标代码复制到可执行文件中。

Linux环境下的软件安装示例

动态库有如下特点：

动态库

- 命名规范为libXXX.so
- 库函数不被连接进可执行程序，可执行文件体积较小
- 可执行文件运行时，库函数动态载入
- 使用灵活，库函数更新后，不需要重新编译可执行程序

目录

01 常用编译器介绍

02 Linux系统下软件编译流程

03 静态库和静态链接

04 动态库和动态链接

05 Linux系统下软件安装示例

Linux环境下的软件安装示例

- Linux环境下开源C/C++项目源码编译安装流程：
  - configure
  - make
  - make install

- configure主要有以下作用：
  - 定义需要的功能选项
  - 检测系统环境是否符合要求
  - 把定义好的功能选项和检测系统环境的信息写入makefile文件

Linux环境下的软件安装示例

- 常见configure参数选项

--prefix=	指定软件安装路径
--enable-shared	允许软件编译生成动态库
--enable-static	允许软件编译生成静态库
CC	指定C编译器
CXX	指定C++编译器
LDFLAGS	引用非默认路径的动态函数库
LIBS	引用非默认路径的静态函数库
INCLUDE	引用非默认路径的头文件

Linux环境下的软件安装示例

- make：make命令执行时，需要一个makefile文件，以告诉make命令需要怎样去编译和链接程序。

- Makefile编写规则如下：
  - target：可以是一个object file（目标文件），也可以是一个执行文件，还可以是一个标签。
  - prerequisites：生成该target所依赖的文件和/或target
  - command：该target要执行的命令（任意的shell命令）target这一个或多个的目标文件依赖于prerequisites中的文件，其生成的规则定义在command中。

Linux环境下的软件安装示例

■示例

```
edit : main.o kbd.o command.o display.o \
        insert.o search.o files.o utils.o
    cc -o edit main.o kbd.o command.o display.o \
        insert.o search.o files.o utils.o

main.o : main.c defs.h
    cc -c main.c
kbd.o : kbd.c defs.h command.h
    cc -c kbd.c
command.o : command.c defs.h command.h
    cc -c command.c
display.o : display.c defs.h buffer.h
    cc -c display.c
insert.o : insert.c defs.h buffer.h
    cc -c insert.c
search.o : search.c defs.h buffer.h
    cc -c search.c
files.o : files.c defs.h buffer.h command.h
    cc -c files.c
utils.o : utils.c defs.h
    cc -c utils.c
clean :
    rm edit main.o kbd.o command.o display.o \
        insert.o search.o files.o utils.o
```

Linux环境下的软件安装示例

- install：就是将make生成的执行文件按照一定规则拷贝或者链接到指定的目录或者文件。

```
%install
rm -rf $RPM_BUILD_ROOT
mkdir -p -m755 $RPM_BUILD_ROOT%{_sysconfdir}/ssh
mkdir -p -m755 $RPM_BUILD_ROOT%{_libexecdir}/openssh
mkdir -p -m755 $RPM_BUILD_ROOT%{_var}/empty/sshd
make install DESTDIR=$RPM_BUILD_ROOT
rm -f $RPM_BUILD_ROOT%{_sysconfdir}/ssh/ldap.conf

install -d $RPM_BUILD_ROOT/etc/pam.d/
install -d $RPM_BUILD_ROOT/etc/sysconfig/
install -d $RPM_BUILD_ROOT/etc/rc.d/init.d
install -d $RPM_BUILD_ROOT%{_libexecdir}/openssh
install -d $RPM_BUILD_ROOT%{_libdir}/fipscheck
install -m644 %{SOURCE2} $RPM_BUILD_ROOT/etc/pam.d/sshd
install -m644 %{SOURCE6} $RPM_BUILD_ROOT/etc/pam.d/ssh-keycat
install -m755 %{SOURCE3} $RPM_BUILD_ROOT/etc/rc.d/init.d/sshd
install -m644 %{SOURCE7} $RPM_BUILD_ROOT/etc/sysconfig/sshd
install -m755 %{SOURCE13} $RPM_BUILD_ROOT/%{_sbindir}/sshd-keygen
install -d -m755 $RPM_BUILD_ROOT/%{_unitdir}
install -m644 %{SOURCE9} $RPM_BUILD_ROOT/%{_unitdir}/sshd@.service
install -m644 %{SOURCE10} $RPM_BUILD_ROOT/%{_unitdir}/sshd.socket
install -m644 %{SOURCE11} $RPM_BUILD_ROOT/%{_unitdir}/sshd.service
install -m644 %{SOURCE12} $RPM_BUILD_ROOT/%{_unitdir}/sshd-keygen.service
install -m755 contrib/ssh-copy-id $RPM_BUILD_ROOT%{_bindir}/
install contrib/ssh-copy-id.1 $RPM_BUILD_ROOT%{_mandir}/man1/
```

Linux环境下的软件安装示例

- 以编译python3为例
  - 在官网下载源码放在家目录下

```
[test1@admin1 sourcecode]$ wget https://www.python.org/ftp/python/3.8.5/Python-3.8.5.tgz
--2020-08-11 11:16:23--  https://www.python.org/ftp/python/3.8.5/Python-3.8.5.tgz
Resolving www.python.org (www.python.org)... 151.101.108.223, 2a04:4e42:36::223
Connecting to www.python.org (www.python.org)|151.101.108.223|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 24149103 (23M) [application/octet-stream]
Saving to: 'Python-3.8.5.tgz'

 0% [                                                                                 ] 131,548     26.5KB/s  eta 14m 45s
```

  - 下载完成后解压源代码

```
Python-3.8.5/Objects/stringlib/stringdefs.h
Python-3.8.5/Objects/stringlib/localeutil.h
Python-3.8.5/Objects/stringlib/asciilib.h
Python-3.8.5/Objects/typeslots.py
Python-3.8.5/Objects/clinic/
Python-3.8.5/Objects/clinic/typeobject.c.h
Python-3.8.5/Objects/clinic/odictobject.c.h
Python-3.8.5/Objects/clinic/codeobject.c.h
Python-3.8.5/Objects/clinic/enumobject.c.h
```

Linux环境下的软件安装示例

- 执行 make && make install
- 开始编译并安装编译好的文件

```
[test1@admin1 Python-3.8.5]$ make && make install
gcc -pthread -c -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -O3 -Wall    -std=c99 -Wextra -Wno-unused-result -Wno-unused-parameter -Wno-m
ing-field-initializers -Werror=implicit-function-declaration  -I./Include/internal  -I. -I./Include    -DPy_BUILD_CORE -o Programs/python.o ./Progr
/python.c
gcc -pthread -c -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -O3 -Wall    -std=c99 -Wextra -Wno-unused-result -Wno-unused-parameter -Wno-m
ing-field-initializers -Werror=implicit-function-declaration  -I./Include/internal  -I. -I./Include    -DPy_BUILD_CORE -o Parser/acceler.o Parser/a
ler.c
gcc -pthread -c -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -O3 -Wall    -std=c99 -Wextra -Wno-unused-result -Wno-unused-parameter -Wno-m
ing-field-initializers -Werror=implicit-function-declaration  -I./Include/internal  -I. -I./Include    -DPy_BUILD_CORE -o Parser/grammar1.o Parser/
mmar1.c
gcc -pthread -c -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -O3 -Wall    -std=c99 -Wextra -Wno-unused-result -Wno-unused-parameter -Wno-m
ing-field-initializers -Werror=implicit-function-declaration  -I./Include/internal  -I. -I./Include    -DPy_BUILD_CORE -o Parser/listnode.o Parser/
tnode.c
gcc -pthread -c -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -O3 -Wall    -std=c99 -Wextra -Wno-unused-result -Wno-unused-parameter -Wno-m
ing-field-initializers -Werror=implicit-function-declaration  -I./Include/internal  -I. -I./Include    -DPy_BUILD_CORE -o Parser/node.o Parser/node
gcc -pthread -c -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -O3 -Wall    -std=c99 -Wextra -Wno-unused-result -Wno-unused-parameter -Wno-m
ing-field-initializers -Werror=implicit-function-declaration  -I./Include/internal  -I. -I./Include    -DPy_BUILD_CORE -o Parser/parser.o Parser/pa
```

注意：某些软件支持并行编译可以在make 命令后面加 –j参数指定编译的进程数。比如make -j 8

谢 谢