Разработка программного обеспечения ОС UNIX

API ОС UNIX

Рекомендуемая литература

1. У. Стивенс. UNIX: Взаимодействие процессов . – СПб.:Питер, 2002, - 576 с.
2. У. Стивенс. UNIX: разработка сетевых приложений. – СПб.:Питер, 2004 (2-е изд.), 2007 (3-е изд.). – 1086 с.
3. Т. Чан. Системное программирование на C++ для UNIX. – К.:BHV, 1997, 1999, - 592 с.
4. М. Митчелл, Дж. Оулдем, А.Самьюэл. Программирование для Linux. Профессиональный подход. М.: Изд.дом «Вильямс», 2002. – 288 с.
5. У. Стивенс, С. А. Раго. UNIX. Профессиональное программирование. 3-е изд. — СПб.: Питер, 2018. — 944 с..
6. www.opengroup.org (Single UNIX specification)
   www.unix.org (UNIX 03 v4)
7. docs.oracle.com (man pages)
8. rus-linux.net (статьи о программировании для Linux)

Разработка программ в UNIX. Компилятор

Программирование в UNIX:

скриптовые языки	shell
интерпретаторы	Python, Ruby
компилируемые	C, C++, Fortran

Компилятор языка Си.

cc, c89, c99

Стандарт K&R 1978 г.

Как узнать?

```
int function( a, b, c)
        int a, c; double b;
{ …
}
```

C89, C99, C11, C18

C89 1989 г. – первый формальный стандарт
прототипы функций
битовые поля, объединения и т.п.
стандартная библиотека
функции с переменным числом аргументов, void, volatile …

C99 1999 г.
массивы с размерностью, определяемой переменной
комплексные числа (несовместимо с Си++)

C11 2011 г., 2017 г.(исправления и уточнения)
C18 расширенная поддержка UNICODE
поддержка дополнений проверяется макросами
поддержка многопоточности
проверка выхода за границы массивов
анонимные структуры, вывод типа и др.

Компиляция программы

Расширения файлов

Си	.c			.h		
Си++	.cpp	.C	.c++	.h	.hpp	.h++
Objective-C	.m	.i				
Objective-C++	.mm	.M	.i	.ii		
Объектный файл		*.o				
Ассемблированный файл		*.s				
Выполняемый файл	расширение любое, обычно без расширения					

Получение выполняемого файла:

cc file.c → a.out

Указать имя результирующего файла

cc file.c -o result → result

Получение объектного файла

cc -c file.c → file.o

Стандартные опции компилятора

-o файл	результирующий файл
-c	компилировать
-S	выдать текст на ассемблере
-E	применить только препроцессор
-Iкаталог	включить каталог[и] в путь поиска файлов #include <…> по умолчанию /usr/include
-Lкаталог	включить каталог[и] в путь поиска библиотек по умолчанию /usr/lib
-lбиблиотека	включить библиотеку, формат: libmath.so -lmath; libC.a -lC
-Dмакро[=значение]	определить макро
-Uмакро	отменить определение макро
пример:	cc -Ddebug=YES file1.c –Udebug file2.c -o myprogram1
@файл	взять опции из файла

Компилятор gcc. Опции

gcc.gnu.org mingw/mingw64 tdm-gcc.tdragon.net

Вызов:

gcc //Си, все языки

g++ //C++ с подключением libstdc++

Поколения gcc Новый ABI !

gcc2.7… gcc2.95-2.96

gcc3.4.5

gcc4.1 - 4.7 – 4.8… (Objective-C++, C++11, Objective-C++ 2.0 …)

gcc5.1 – 10.2 …

Узнать версию: gcc –v

-x lang явно задать язык файла: c, c++ , objective-c, obj-c++, fortran…

-Wall включить все предупреждения

-W отключить все предупреждения

-Wпредупр. включить/отключить конкретное предупр.,

например -Wno-write-strings (преобразование const char[] в char*)

-g включить отладочную информацию

-s не включать. Удалить имеющуюся strip файл

-ansi соответствие стандарту ANSI

Опции gcc. Продолжение

-std= c89 c99 c++98 c++11 c++14 c++17 Используемый стандарт языка

-Wl,опция передает опцию компоновщику

-static -shared тип компоновки

Оптимизация

-O0 не оптимизировать

-O1

-O2 рекомендуемая оптимизация

-O3 усиленная оптимизация

-Os оптимизация по размеру

-fomit-frame-pointer (отладка будет невозможна)

-funroll-loops

-m32 -m64 разрядность программы/платформы

-mieee-fp режим обработки вещественных чисел

-mtune=… -march=…

i386 i486 i586 i686 pentium pentium-mmx pentium2
pentium3 pentium4 pentium-m core2 corei7
athlon64 k8-sse3 native

-msse -msse3 -mno-sse3 -msse-4.2 -mavx -mavx2

Опции gcc. Окончание

Архитектура SPARC

- -mcpu=v9
- -mvis
- -mhard-quad-float

Архитектура ARM

- -mcpu=cortex-a15
- -march=armv7

Расположение в Solaris

/usr/local/bin	prefix по умолчанию в GNU
GCC сборки www.sunfreeware.com	
/opt/sfw/bin	утилиты GNU, собранные SUN/Oracle
/opt/sfw/gcc-3/bin	gcc3, собранные SUN/Oracle
/usr/ccs/bin	вспомогательные утилиты Solaris

Примеры использования GCC

1. gcc a.c
2. gcc a.c b.c
3. gcc a.c b.c -o a
4. gcc a.c b.c -o a
5. gcc a.c b.c -o a –lrt
6. gcc -c a.c
7. gcc -c b.c
8. gcc a.o b.o -o a.app -lrt
9. gcc a.c b.o -o a.app -lrt
10. strip a.app
11. gcc –m64 –s –O2 a.c b.c -o a.app –lrt
12. gcc –m64 –s –O2 –mieee-fp –march=corei7 –I. –L. a.c b.c -o a.app –lrt

Библиотеки

Библиотека – совокупность объектных файлов, объединённых в один файл.

Прототипы функций из библиотеки размещаются во включаемых файлах.

Названия файлов, как правило, не совпадают с именам библиотеки, одной библиотеке соответствует, обычно, несколько include-файлов.

Библиотеки статической компоновки lib*.a.*

Динамически связываемые библиотеки lib*.so.*

Подключение прототипов: #include <файл>

Подключение библиотеки: -lимя

где файл библиотеки libимя.a.* или libимя.so.*

Создание статических библиотек

ar [опции] файл_архива [файл]

Опции:

-q добавить файл в конец архива
-d удалить файл из архива
-t напечатать таблицу содержимого архива
-p напечатать содержимого указанных файлов или всего архива
-r заменить или добавить файлы
-x извлечь из архива файл, если не указать – все.
-m переместить файл, используется вместе с –a, –b, –i.
-s обновить таблицу символов

Модификаторы

-a файл после этого файла
-b файл перед этим файлом
-i файл перед этим файлом
-u обновить файлы (вместе с –r).

В Linux после внесения изменений: ranlib файл_архива.a
(для обновления __SYMDEF)

Создание разделяемых библиотек

В исполняемом файле – только ссылка на библиотеку

Компилировать, указывая –fPIC position independent code

Объединить в библиотеку –shared –fPIC

Пример:

gcc –c –fPIC d1.c

gcc –c –fPIC d2.c

gcc –shared –fPIC –o libnew.so d1.o d2.o

gcc a.c –o awlnew –L. –lnew

export LD_LIBRARY_PATH=.

./awlnew

или

gcc a.c –o awlnew –L. –lnew –Wl,-rpath,/export/home/user/project1

Зависимость библиотек друг от друга:	статические	динамические
порядок указания важен?	ДА	НЕТ

Динамическая загрузка и выгрузка (явная компоновка)

Использовать: <dlfcn.h> libdl.so

```
int     dlclose(void *);
char  *dlerror(void);
void  *dlopen(const char *, int);
    флаги    RTLD_LAZY,  RTLD_NOW,  RTLD_GLOBAL,  RTLD_LOCAL
void  *dlsym(void *restrict, const char *restrict);
```

Пример использования (без обработки ошибок!):

```
void *handle = dlopen(“libnew.so”, RTLD_LAZY );       // if ( handle ==0 …
void (*test)() = dlsym( handle, “newfunction” );          // if ( test ==0 …
(*test)();
dlclose( handle );
```

```
gcc –rdynamic –o foo foo.c -ldl
```

Конструкторы и деструкторы библиотеки

```
void _init() {
…
}
void _fini() {
…
}
// заработают, если указать –nostartfiles
```

Обычно в gcc:

```
void __attribute__((constructor)) init() {
…
}
void __attribute__((destructor)) fini() {
    …
}
```

Управление версиями программ

SCCS: source code control system
хранит историю изменений файла: s.файл

Программы/команды

admin	создает файлы, изменяет параметры
delta	вносит изменения в файл истории
get	запрашивает файл для редактирования или компиляции
prs	печатает файл с комментариями
rmdel	удаляет внесенные изменения
sact	паказывает, кто работает с файлом
unget	отказывается от намерения внести изменения
val	проверяет историю изменений
what	ищет подстановку по %Z%

Дополнительные команды-аргументы

Псевдоутилиты

check	эквивалент info (в основном)
clean	числит каталог от файлов, которые могут быть восстановлены
create	создает, использует опции admin
delget	выполняет delta + get
deledit	выполняет delta + get -e
diffs	выдает различия файлов
edit	эквивалент get –e
fix	для исправления ошибок, удаляет именованное изменение
info	список редактируемых файлов
print	эквивалентно sccs prs
tell	список редактируемых файлов, разделенных \n.
unedit	отказ от get -e

Версии файлов

file	g-файл - сам исходный файл
d.file	файл различий
p.file	блокировка
q.file	временный
s.file	файл истории (хранилище)
x.file	временная копия s-файла
z.file	временная блокировка

Нумерация версий

X.X.Y.Y, где X, Y – 1…9999, Y – номер ветви, определяется последовательностью порождения.

1.1→1.2→1.3→1.4→2.1→2.2

|→1.4.1.1 →1.4.1.2→1.4.1.3

|→1.4.2.1→1.4.2.2

Взять из хранилища

get или get –e ?

Подстановка вместо ключевых сочетаний символов

Ключевые последовательности

%M%	Module name.
%I%	SCCS identification (SID) (%R%.%L% or %R%.%L%.%B%.%S%) .
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the t flag in the SCCS file.
%F%	SCCS filename.
%P%	SCCS absolute pathname.
%Q%	The value of the q flag in the file.

Ключевые последовательности - 2

%C% Current line number.
%Z% The four-character string "@(#)" recognizable by what.
%W% A shorthand notation for constructing what strings:
 %W%=%Z%%M%<tab>%I%
%A% Another shorthand notation for constructing what strings:
 %A%=%Z%%Y%%M%%I%%Z%

Запросы get
-k запрет замены
-p печать текста
-c YY[MM[DD[HH[MM[SS]]]]] не новее чем
-b создать ветвь

Спецификации печати команды prs

:Dt:	Delta information			
:DL:	Delta line statistics			
:Li:	Lines inserted by Delta			
:Ld:	Lines deleted by Delta			
:Lu:	Lines unchanged by Delta			
:R:	Release number			
:L:	Level number			
:B:	Branch number			
:S:	Sequence number			
:D:	Date delta created	:DY:	:Dm:	:Dd:
:T:	Time delta created	:Th:	:Tm:	:Ts:
:P:	Programmer who created Delta			
:M:	Module name			
:FB:	Floor boundary			
:CB:	Ceiling boundary			

и др.

Внесение изменений: delta

-n не удалять g-файл
-y задает комментарий
-p перед внесением изменений печатается информация о различиях

Порядок работы:

EDITOR → file → admin → s.file → get –e → EDITOR → delta →s.file →
→ get → cc → a.out
→ get –e → EDITOR → delta →s.file → get → cc → a.out

Управление сборкой проектов: make

Назначение – выполнение минимально необходимых дествий по пересборке проекта.

Описание проекта и зависимостей файлов – makefile.

Файлы по умолчанию:

./makefile ./Makefile.

./s.makefile, SCCS/s.makefile, ./s.Makefile, SCCS/s.Makefile

Опции:

-f makefilename задает имя файла с описанием зависимостей

-i игнорирует коды завершения операций

-k при ошибке продолжает сборку других целей

-n печатает команды, но их не выполняет

-p печатает макросы на терминал

-q код возврата не 0, если что-то надо делать

-r очищает список суффиксов и не использует встроенные правила

-t обновляет время целей, но сборку не производит

-s не выводит выполняемые команды

MAKEFLAGS= опции по умолчанию.

Синтаксис makefile

#комментарий
цель [цель]: [зависимость] [зависимость] [; действие]
\t действие
\t действие

Префиксы команд:

-	игнорируется код завершения	.IGNORE
@	команда не печатается	.SILENT
+	команда выполняется, даже если опции –n, –q, -t.	

.DEFAULT	выполняется, если другие цели отсутствуют
.PRECIOUS	указанные файлы не будут удалены при ошибке

Пример:

```
applic: applic.o
        gcc applic.o –s –O2 –lrt –o applic
        @strip applic
applic.o: applic.c
        gcc –c applic.c
```

Makefile: макросы

Макро=значение

Использование: $(Макро) $Макро (1 символ в имени)

Пример:

```
CC=gcc
TARGET=applic
applic: applic.o
        $(CC) $(TARGET).o –s –O2 –lrt –o $(TARGET)
        strip $(TARGET)

$(TARGET).o: $(TARGET).c
        $(CC) –c $(TARGET).c
```

Встроенные макросы

MAKE=make
AR=ar
ARFLAGS=-rv
YACC=yacc
YFLAGS=
LEX=lex
LFLAGS=
LDFLAGS=
CC=c99
CFLAGS=-O
FC=fort77
FFLAGS=-O 1
GET=get
GFLAGS=
SCCSFLAGS=
SCCSGETFLAGS=-s

Символические макроподстановки и суффиксные правила

$@	Имя цели
$%	элемент библиотеки
$?	Устаревшая зависимость
$<	Устаревшая зависимость в суффиксных правилах
$*	имя цели без суффикса и префикса

```
.c:
    $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $<

.f:
    $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $<

.sh:
    cp $< $@
    chmod a+x $@

.c~:
    $(GET) $(GFLAGS) -p $< > $*.c
    $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $*.c
```

Суффиксные правила

```
.c.o:
    $(CC) $(CFLAGS) -c $<

.f.o:
    $(FC) $(FFLAGS) -c $<

c~.o:
    $(GET) $(GFLAGS) -p $< > $*.c
    $(CC) $(CFLAGS) -c $*.c

.c.a:
    $(CC) -c $(CFLAGS) $<
    $(AR) $(ARFLAGS) $@ $*.o
    rm -f $*.o
```

Задание пользовательского суффикса.

```
.SUFFIXES: .o .c .y .l .a .sh .f .c~ .y~ .l~ .sh~ .f~
```

Пример использования макроподстановок

```
CC=gcc
CFLAGS= –fomit-frame-pointer –O2 –s –m64 –mavx –mtune=corei7
PROGLIST= program1 program2

all: $(PROGLIST)
    echo “Выполнено”

clean:
    rm –f *.o
    echo “Очищено”

$(PROGLIST): $$@.c
    $(CC) $(CFLAGS) $? –o $@
```

_____ Динамические зависимости _____

```
    AT&T:       $(targets): $$@.o libm1.a
    GNU:        $(targets): %: %.o libm1.a
```

Дополнительные возможности

Включение другого файла include файл

Условный выбор gcc / nmake (VC)

```
ifeq (…, … )                    !if “…”=“…”
…..                             …..
endif                           !endif
```

ifdef переменная

SHELL= макро определяет вызываемый интерпретатор команд
Каждая строка – новая копия.
Выполнить программы в одном shell-е:

```
aa: a.c
    cd build; gcc –O2 ../a.c; cp a.out ../aa
```

Условное выполнение

```
mks: mks.c
        if …..; \
        then \
        $(CC) …..\
        else \
        …
        fi
```

Специальные цели

.INIT	цель выполняется раньше других
.DONE	цель выполняется последней, если успешно
.FAILED	цель выполняется при ошибке
.PHONY	описание абстрактных целей

Пример.

```
.PHONY: clean

clean:
	rm *.o *.d
```

Работа с файлами внутри библиотек

```
foolib(hack.o) : hack.o
	ar cr foolib hack.o
```