

Linux C Prog

2021 Spring

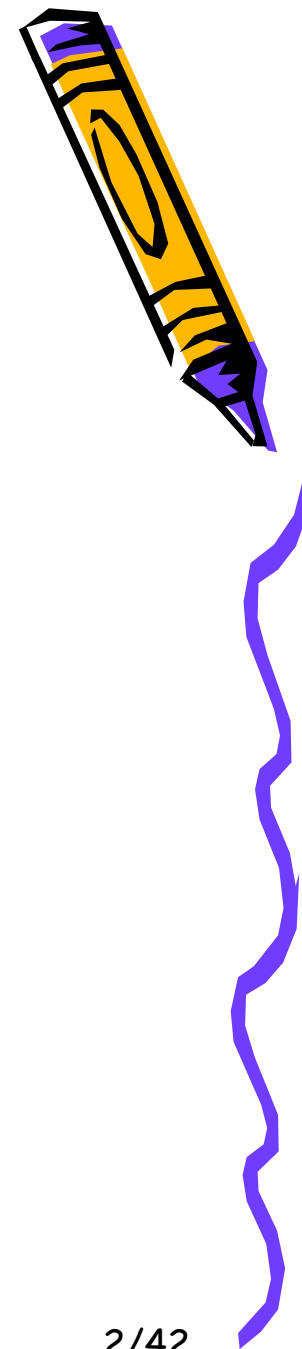
Zhaobin Liu
zhbliu@gmail.com

Outline

- 程序生成
 - 生成vi/vim...
 - 缩排indent
- 程序编译
 - 编译gcc...
 - Makefile
 - 警告
 - 优化
- 程序调试
 - GDB
- 其它编程: java ...

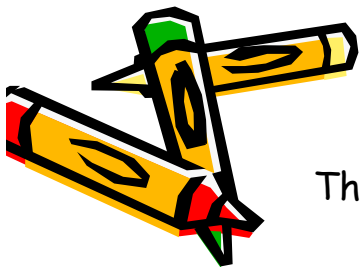
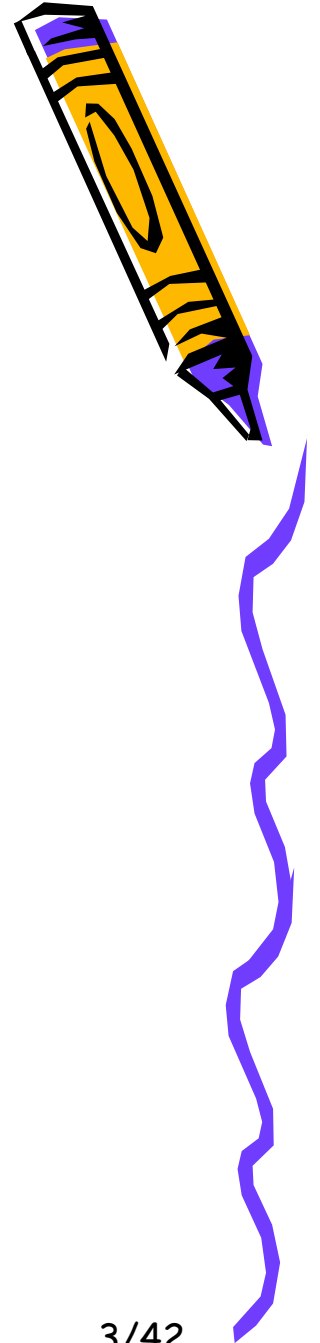


Thursday, May 13, 2021



程序生成工具

- 生成C源文件
 - vi/vim、emacs、nano
- 缩排C源代码
 - cb: C program beautifier
 - bcpp: make C++ beautifier
 - indent



```
buyhorse@ubuntu-server-1804:~/unix/indenttry$ cat first.c
```

```
#include "stdio.h"
void main ()
{printf ("Hello World!\n");
}
```

```
buyhorse@ubuntu-server-1804:~/unix/indenttry$ cat first.c | indent
```

```
#include "stdio.h"
void
main ()
{
    printf ("Hello World!\n");
}
```

```
buyhorse@ubuntu-server-1804:~/unix/indenttry$ █
```

```
buyhorse@ubuntu-server-1804:~/unix/indenttry$ cat second.c
```

```
main ()
{
    int i, j;
    for (i = 0, j = 10; i < j; i++)
    {
        printf ("Hello World!\n");
    }
}
```

```
buyhorse@ubuntu-server-1804:~/unix/indenttry$ cat second.c | indent
```

```
main ()
{
    int i, j;
    for (i = 0, j = 10; i < j; i++)
    {
        printf ("Hello World!\n");
    }
}
```

```
buyhorse@ubuntu-server-1804:~/unix/indenttry$ █
```



```
buyhorse@ubuntu-server-1804:~/unix/indenttry$ cat first-err.c
#include "stdio.h"
void main ()
{printf ("Hello World!\n");
}
}
buyhorse@ubuntu-server-1804:~/unix/indenttry$ cat first-err.c |indent
#include "stdio.h"
void
main ()
{
    printf ("Hello World!\n");
}
indent: Standard input:5: Error:Stmt nesting error.
}
buyhorse@ubuntu-server-1804:~/unix/indenttry$ █
```

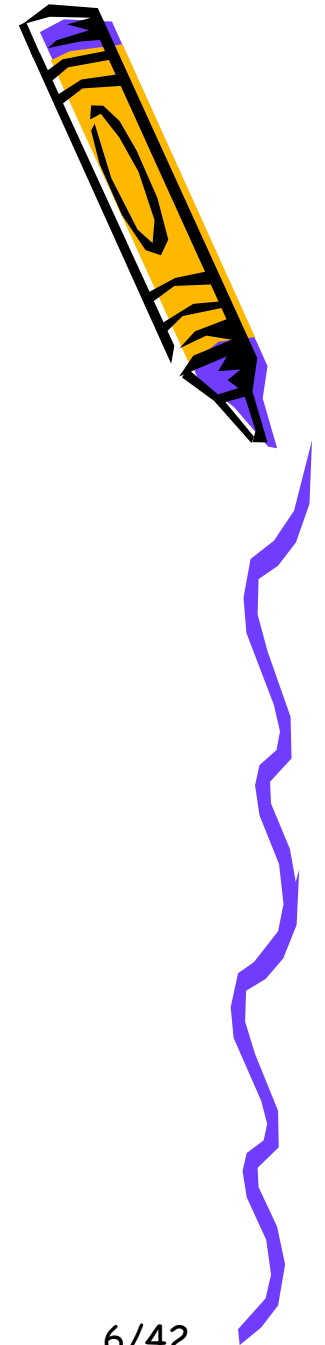


Thursday, May 13, 2021

5/42

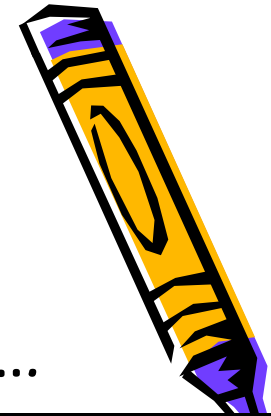
编译C/C++程序

- C编译器
 - cc
 - xlc(AIX); aCC(HP-UX)
 - gcc
- C++编译器
 - cpp: The C Preprocessor
 - CC
 - g++
- C++编译器可以编译C程序
- gcc — -version
- g++ — -version
- 注意：这两个版本最好一致, 否则会报错



gcc/g++ basic

- GNU C Compiler
- GNU Compiler Collection: C, C++, Objective-C, Fortran, Java, Ada, Go ...



```
buyhorse@ubuntu-server-1804:~/unix/indenttry$ gcc first.c
buyhorse@ubuntu-server-1804:~/unix/indenttry$ gcc -o first first.c
buyhorse@ubuntu-server-1804:~/unix/indenttry$ g++ first.c
first.c:2:12: error: '::main' must return 'int'
void main ()
```

```
buyhorse@ubuntu-server-1804:~/unix/indenttry$ gcc hello.cpp
/tmp/ccuggpGd.o: In function `main':
hello.cpp:(.text+0xe): undefined reference to `std::cout'
hello.cpp:(.text+0x13): undefined reference to `std::basic_ostream<char, std::char_traits<char> >& std::operator<< (std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&, char const*)'
hello.cpp:(.text+0x1d): undefined reference to `std::basic_ostream<char, std::char_traits<char> >& std::endl<char, std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&)'
hello.cpp:(.text+0x28): undefined reference to `std::ostream::operator<<(std::ostream& (*) (std::ostream&))'
/tmp/ccuggpGd.o: In function `__static_initialization_and_destruction_0(int, int)':
hello.cpp:(.text+0x58): undefined reference to `std::ios_base::Init::Init()'
hello.cpp:(.text+0x6d): undefined reference to `std::ios_base::Init::~Init()'
collect2: error: ld returned 1 exit status
buyhorse@ubuntu-server-1804:~/unix/indenttry$ g++ hello.cpp
buyhorse@ubuntu-server-1804:~/unix/indenttry$
```

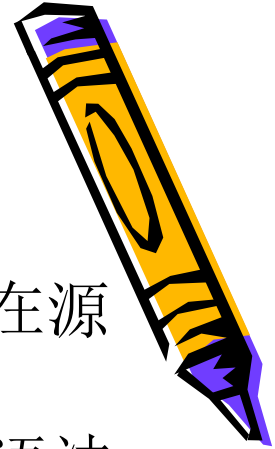


gcc

- **Pre-Processing**: 调用预处理程序 **cpp**，向其中插入“**#include**”语句所包含的内容，并由它负责展开在源文件中定义的宏，生成.i文件 (**gcc -E**)
- **Compilation** : **gcc**首先检查代码的规范性、是否有语法错误等，以确定代码的实际要做的工作，在检查无误后，**cc**把代码翻译成汇编语言“**.s**”文件。该阶段只进行编译而不进行汇编，生成汇编代码。 (**gcc -S**)
- **Assembly**: 调用**as** (**assembler**)把编译阶段生成的“**.s**”文件转成目标“**.o**”文件，不进行链接操作 (**gcc -c**)
- **Linking**: 调用链接程序**ld** (**The GNU linker**)，把生成的目标代码链接成一个可执行程序 (**gcc *.o**)

• **man gcc**

// #565-730



examples(./hello)

- `gcc hello.c`
 - `gcc -o hello hello.c`
 - `gcc -E hello.c -o hello.i`
 - `gcc -S hello.i -o hello.s`
 - `gcc -c hello.s -o hello.o`
 - `gcc hello.o -o hello`
 - `gcc -S hello.c`
 - `cat hello.i` //查看**hello.i**文件
 - `gcc -M hello.c` //生成头文件关联的信息
 - `gcc -v hello.c` //显示编译版本、头文件目录等信息
-
- `readelf -S hello.o` // ELF: Executable and Linking Format
 - `man readelf`
 - `gcc -xc hello.tom`



库文件

- `.a`: 通常表示静态链接库——`.lib`
- `.so`: 通常表示动态(共享)链接库——`.dll`
- `ls /lib/libc*` //64位操作系统之前
- `ls /lib/x86_64-linux-gnu/libc*` (不同OS位置不同)
- `ls /lib64/ld-linux-x86-64.so.2` //动态库加载器/链接器 (一般指向`/lib`)
- `ldd /bin/ls` //ls在启动时, 加载器将所有的动态库加载后然后再将控制权移交给ls程序的入口。
- `man ldd`
- `ldd a.out` 列出a.out运行所需的所有动态依赖关系
- `readelf -l a.out` //gcc包含了一个特殊的 ELF 头: INTERP, 其作用是指定加载器的路径



然后，我们编译并运行ldd命令：

```
xiaomanon@xiaomanon-machine:~/Documents/c_code$ ldd tooltest
linux-gate.so.1 => (0xb775b000)
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7595000)
/lib/ld-linux.so.2 (0xb775c000)
```

我们可以将ldd的输出结果分为3列来看：

- 第一列：程序需要依赖什么库
- 第二列：系统提供的与程序需要的库对应的库名称
- 第三列：依赖库加载的开始地址

Shared Libraries

- Common library routines removed from executable files
- Single copy of common library routines in memory is maintained
- No need to re-link edit every program if a library is updated or changed
- Size is smaller, some run-time overhead



Shared Libraries

- `$ gcc -static dvss.c`
- `$ gcc -o b.out dvss.c`
- `$ ls -l a.out b.out`
- `$ size a.out b.out`

prevent gcc from using
shared libraries

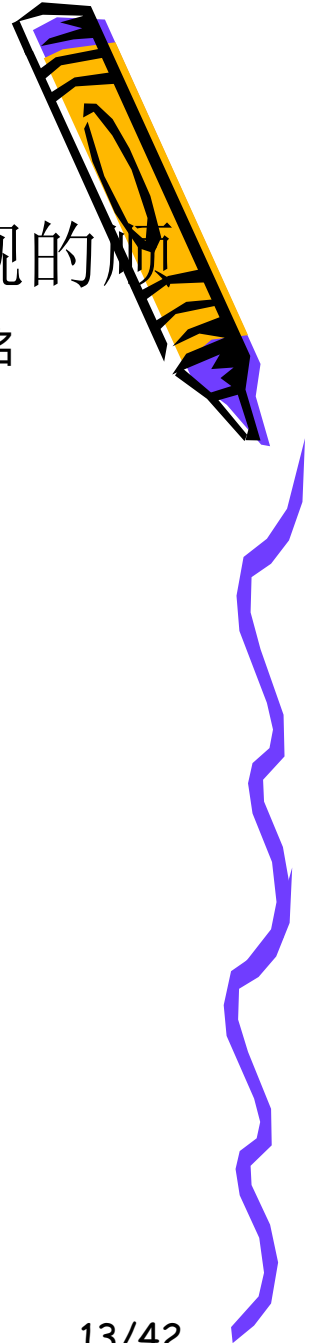
gcc defaults to use shared
libraries

Size is smaller
some run-time overhead



路径问题

- **-I**: 先在所指定的目录查找，然后再按常规的顺序去找 -I参数就是用来指定程序要链接的库，-I参数紧接着就是库名
- **-L**: 指定编译的时候，搜索库的路径
- **<*.h>**: 系统目录
- **“*.h”**: 当前目录




```


buyhorse@ubuntu:~/example$ cat -n power.c
 1  #include <stdio.h>
 2  #include <math.h>
 3
 4  int main ()
 5  {
 6      float x, y;
 7      printf ("The program takes x and y from stdin and displays x^y.\n");
 8      printf ("enter integer x: ");
 9      scanf ("%f", &x);
10      printf ("enter interger y: ");
11      scanf ("%f", &y);
12      printf ("x^y is : %6.3f\n", pow ((double) x, (double) y));
13  }
buyhorse@ubuntu:~/example$ gcc power.c
/tmp/ccGsiyD3.o: In function `main':
power.c:(.text+0x7e): undefined reference to `pow'
collect2: error: ld returned 1 exit status
buyhorse@ubuntu:~/example$ gcc power.c -lm
buyhorse@ubuntu:~/example$ ldd a.out
        linux-vdso.so.1 => (0x00007ffe6c52d000)
        libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007fb6199ac000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fb6195e2000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fb619cb5000)
buyhorse@ubuntu:~/example$ █

```



- **-lm** 编译时需要告诉编译器库函数的路径 **man pow**
- **/usr/lib/x86_64-linux-gnu/libm.a**
- **ldd power** // print shared object dependencies


关于makefile- `makefile_power`



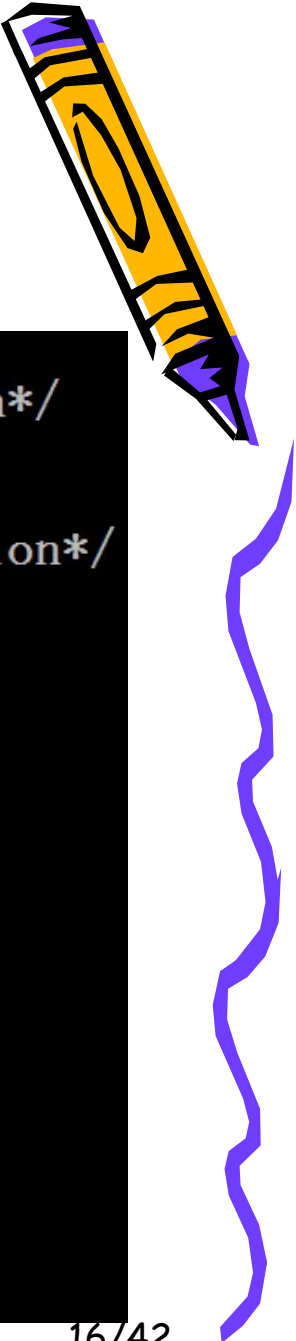
```
buyhorse@ubuntu:~/example$ cat makefile-power
#Sample makefile for the power program
#Remember: each command line starts with a TAB

power: power.c
    cc power.c -o power -lm

buyhorse@ubuntu:~/example$ make -f makefile-power
make: 'power' is up to date.
buyhorse@ubuntu:~/example$ touch power.c
buyhorse@ubuntu:~/example$ make -f makefile-power
cc power.c -o power -lm
buyhorse@ubuntu:~/example$ rm power
buyhorse@ubuntu:~/example$ make -f makefile-power
cc power.c -o power -lm
buyhorse@ubuntu:~/example$
```

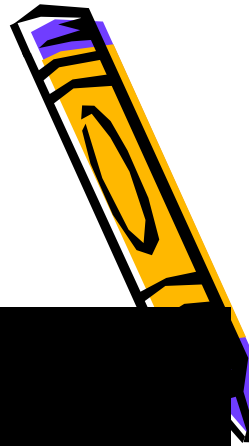


makefile cont'




```
buyhorse@ubuntu:~/example$ cat -n input.h
 1  /*Declaration/Prototype of the "input" function*/
 2  double input(char *);
buyhorse@ubuntu:~/example$ cat -n compute.h
 1  /*Declaration/Prototype of the "compute" function*/
 2  double compute(double,double);
buyhorse@ubuntu:~/example$ cat -n main.h
 1  /*Declaration of prompts to users*/
 2  #define PROMPT1 "Enter the value of x: "
 3  #define PROMPT2 "Enter the value of y: "
buyhorse@ubuntu:~/example$ cat -n input.c
 1  #include "stdio.h"
 2  #include "input.h"
 3  double input (char *s)
 4  {
 5  float x;
 6  printf ("%s", s);
 7  scanf ("%f", &x);
 8  return (x);
 9  }
```




makefile cont'



```
buyhorse@ubuntu:~/example$ cat -n compute.c
 1  #include "math.h"
 2  #include "compute.h"
 3  double compute (double x, double y)
 4  {
 5  return (pow ((double) x, (double) y ));
 6  }
 7
buyhorse@ubuntu:~/example$ cat -n main.c
 1  #include "main.h"
 2  #include "stdio.h"
 3  #include "compute.h"
 4  #include "input.h"
 5
 6  int main()
 7  {
 8  double x,y;
 9  printf("the program takes x and y from stdin and displays x^y.\n")
10  x=input(PROMPT1);
11  y=input(PROMPT2);
12  printf("x^y is: %.3f\n", compute(x,y));
13  }
buyhorse@ubuntu:~/example$
```



makefile cont'

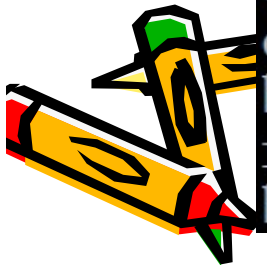


```
buyhorse@ubuntu:~/example$ cat makefile
#Sample makefile for the pow program
#Remember: each command line starts with a TAB
pow: main.o input.o compute.o
    cc main.o input.o compute.o -o pow -lm
main.o: main.c main.h input.h compute.h
    cc -c main.c
input.o: input.c input.h
    cc -c input.c
compute.o: compute.c compute.h
    cc -c compute.c
clean:
    rm *.o
buyhorse@ubuntu:~/example$ make
cc -c main.c
cc -c input.c
cc -c compute.c
cc main.o input.o compute.o -o pow -lm
buyhorse@ubuntu:~/example$
```

Thursday, May 13, 2021

makefile cont'

```
buyhorse@ubuntu:~/example$ cat makefile_new
pow: main.o input.o compute.o
    cc main.o input.o compute.o -o pow -lm
main.o: main.c main.h input.h compute.h
input.o: input.c input.h
compute.o: compute.c compute.h
clean:
    rm *.o
buyhorse@ubuntu:~/example$ make -f makefile_new
cc -c -o main.o main.c
cc -c -o input.o input.c
cc -c -o compute.o compute.c
cc main.o input.o compute.o -o pow -lm
buyhorse@ubuntu:~/example$ touch input.c
buyhorse@ubuntu:~/example$ make
cc -c input.c
cc main.o input.o compute.o -o pow -lm
buyhorse@ubuntu:~/example$ make clean
rm *.o
buyhorse@ubuntu:~/example$
```



makefile cont'

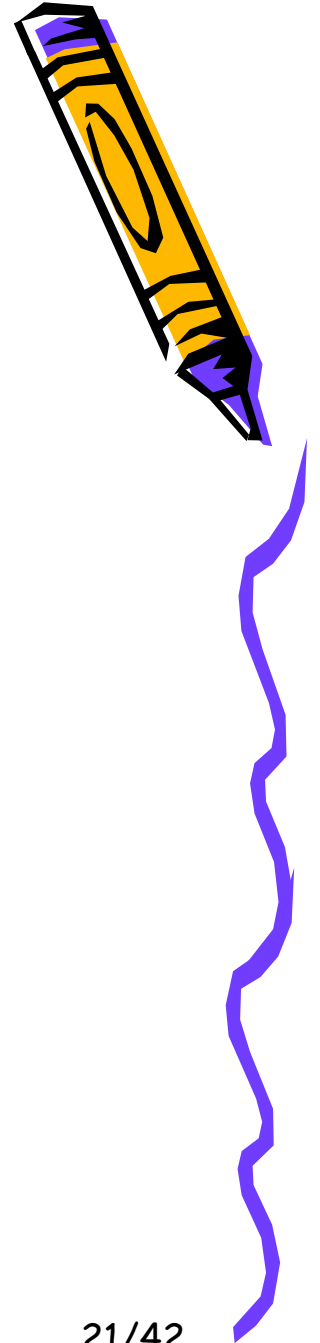
- `$@` //目标文件
- `$^` //所有的依赖文件
- `$<` //第一个依赖文件

```
buyhorse@ubuntu-server-1804:~/example$ cat makefi
pow: main.o input.o compute.o
    gcc -o $@ $^ -lm
.c.o:
    gcc -c $<
clean:
    rm *.o
buyhorse@ubuntu-server-1804:~/example$ make -f ma
gcc -c main.c
gcc -c input.c
gcc -c compute.c
```



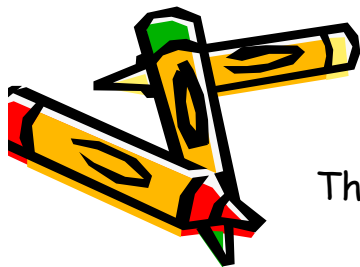
Makefile cont'

- make 执行顺序
 - GNU Makefile
 - makefile
 - Makefile



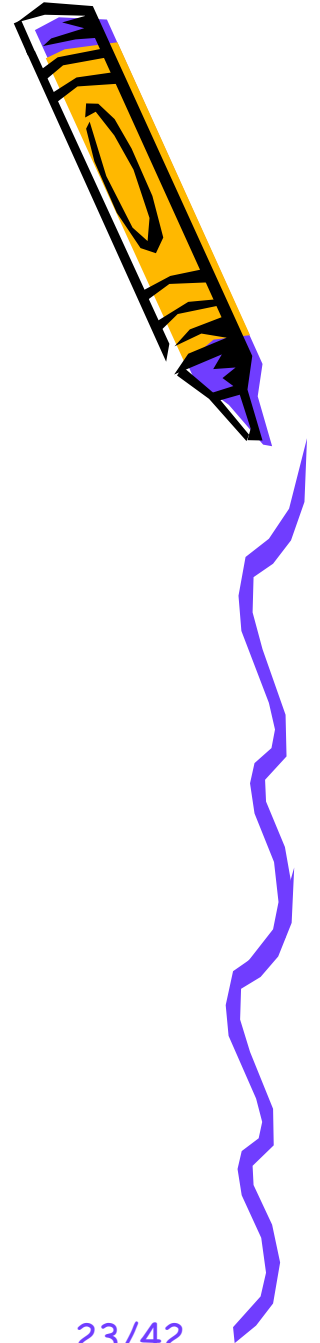
gcc cont'

- `gcc -M hello.c`
- `gcc -MM main.c` //不包含头文件的依赖关系
- `gcc -MM main.c -MF main.makefile` //依赖关系写入main.makefile
- `cat main.makefile`
- `gcc -MM main.c -MT hello.o` //目标对象改名
- `hello.o: main.c main.h`



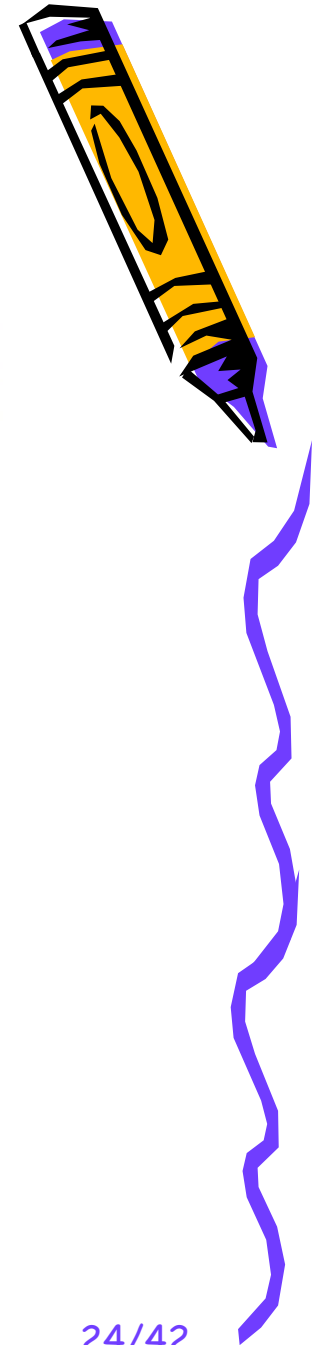
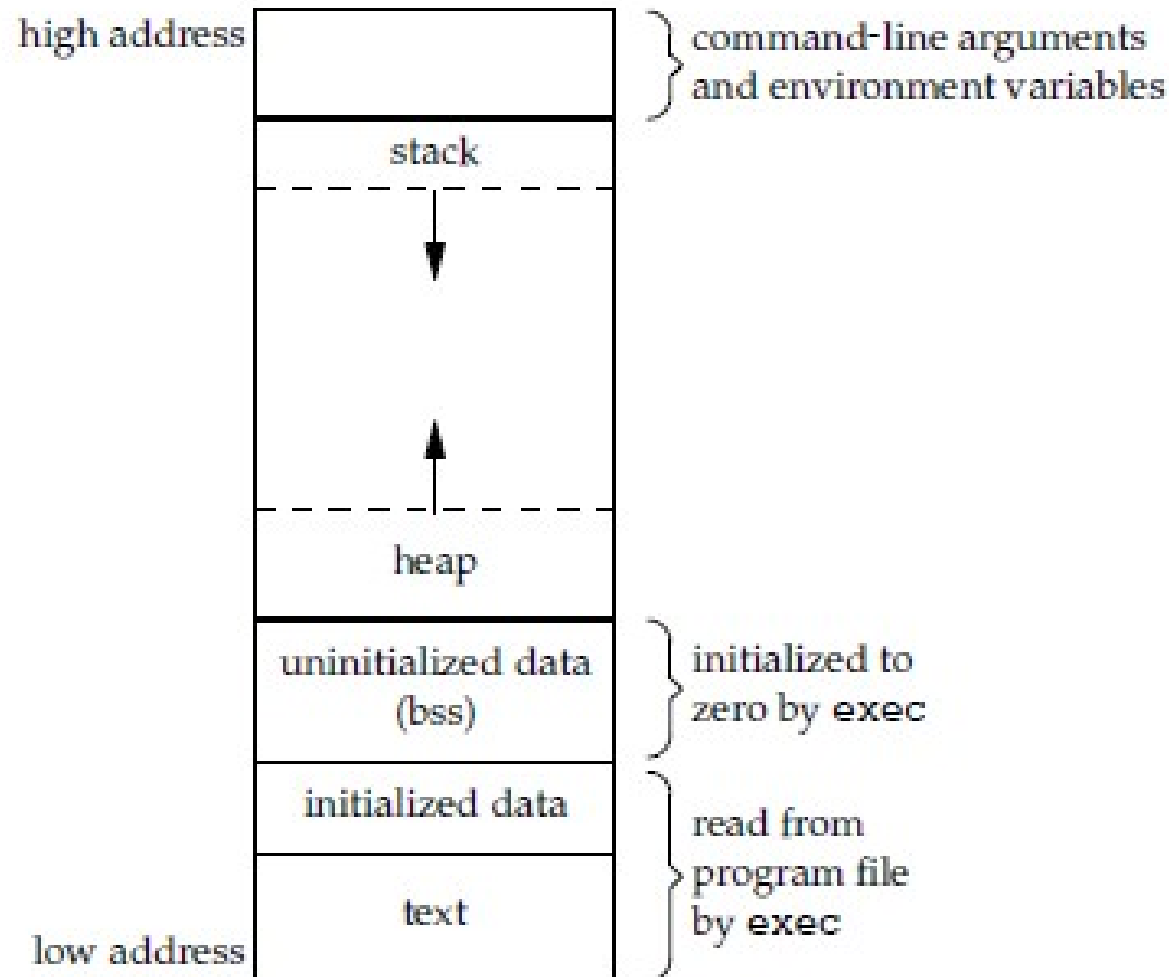
Memory Layout of a C Program

- **Text segment:** Machine instructions
(read-only, sharable)
- **Initialized data segment:**
e.g. `int i = 99;` (initialized!)
- **Uninitialized data segment:**
(bss: block started by symbol)
e.g. `long array[100];`
- **Heap:** dynamic memory allocation
- **Stack:** automatic variables, function calling information, context-switch information,
(recursive functions)



Typical memory arrangement

`man sbrk;` `man malloc`



Size: reports the sizes (in bytes) of the text, data, and bss segments



- `size` //默认查看a.out

```
buyhorse@ubuntu-server-1804:~/apue$ size /bin/ls
  text    data    bss      dec     hex filename
124042   4728   4832  133602  209e2 /bin/ls
buyhorse@ubuntu-server-1804:~/apue$ size /bin/bash
  text    data    bss      dec     hex filename
1061290   47036   39816 1148142  1184ee /bin/bash
buyhorse@ubuntu-server-1804:~/apue$
```



System Calls & Library Functions



System Calls:

Entry points into an OS kernel

- Cannot be changed by user
- A **function** of the same name in the standard C library
- User just calls those C functions whenever system calls are needed

- **man 2 write**

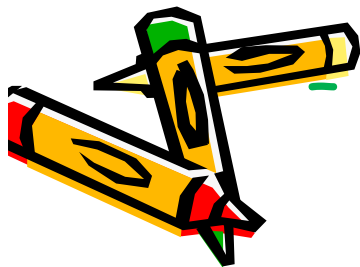
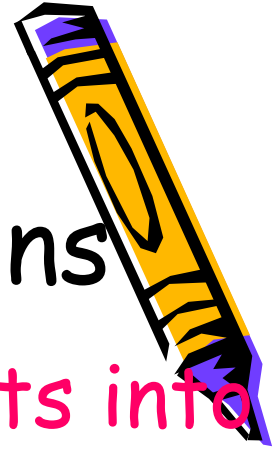


System Calls & Library Functions

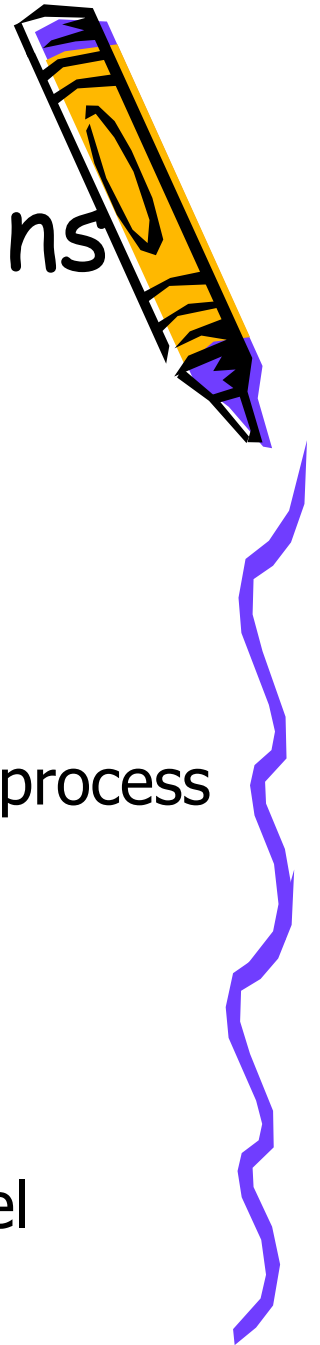
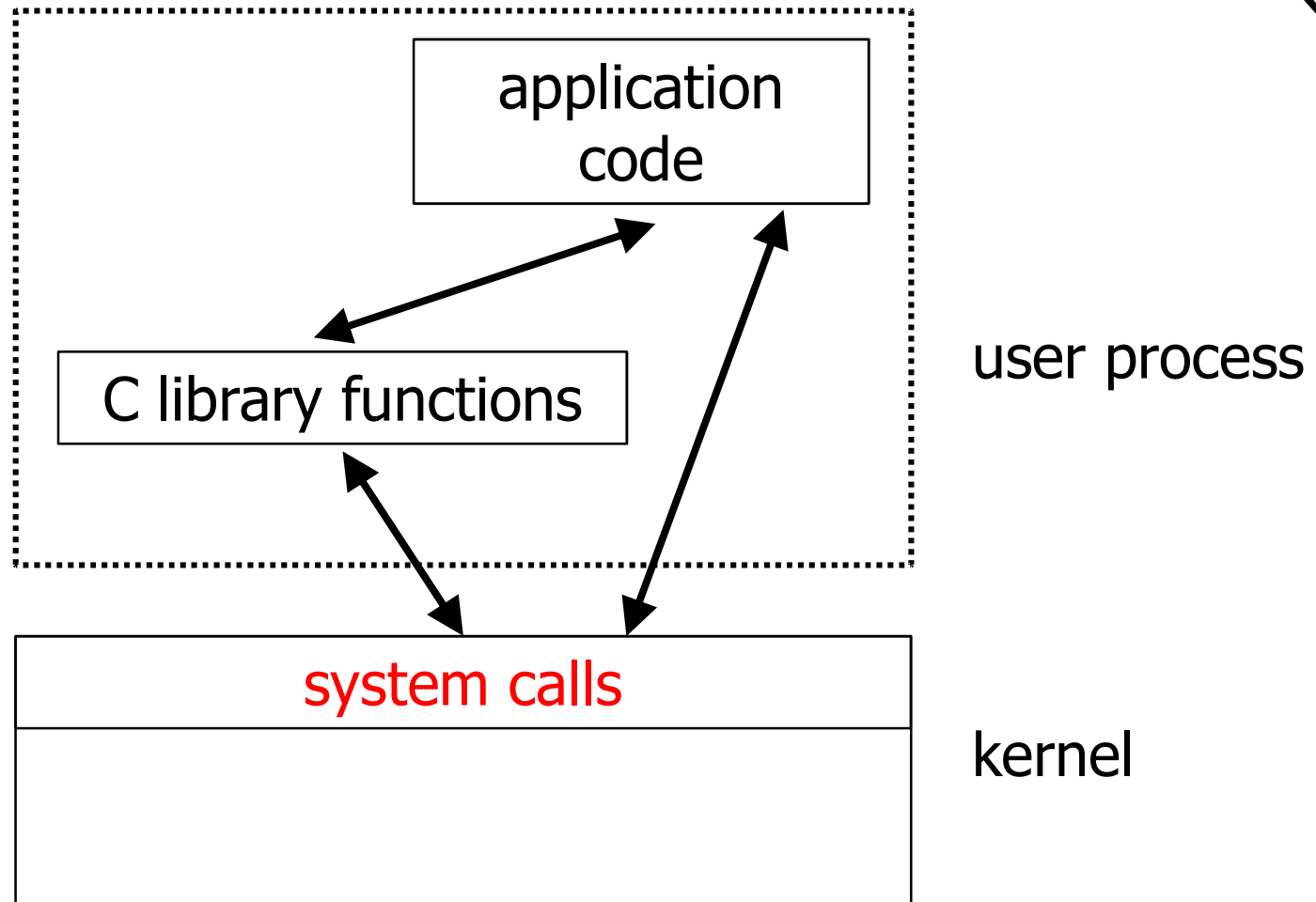
- Library Functions: not entry points into kernel, just functions, but they may invoke one or more system calls

- E.g.: printf() invokes write() system call
- E.g.: strcpy(), atoi(): do not invoke any system call
- Implementor view: fundamental diff
- Programmer view: no critical difference

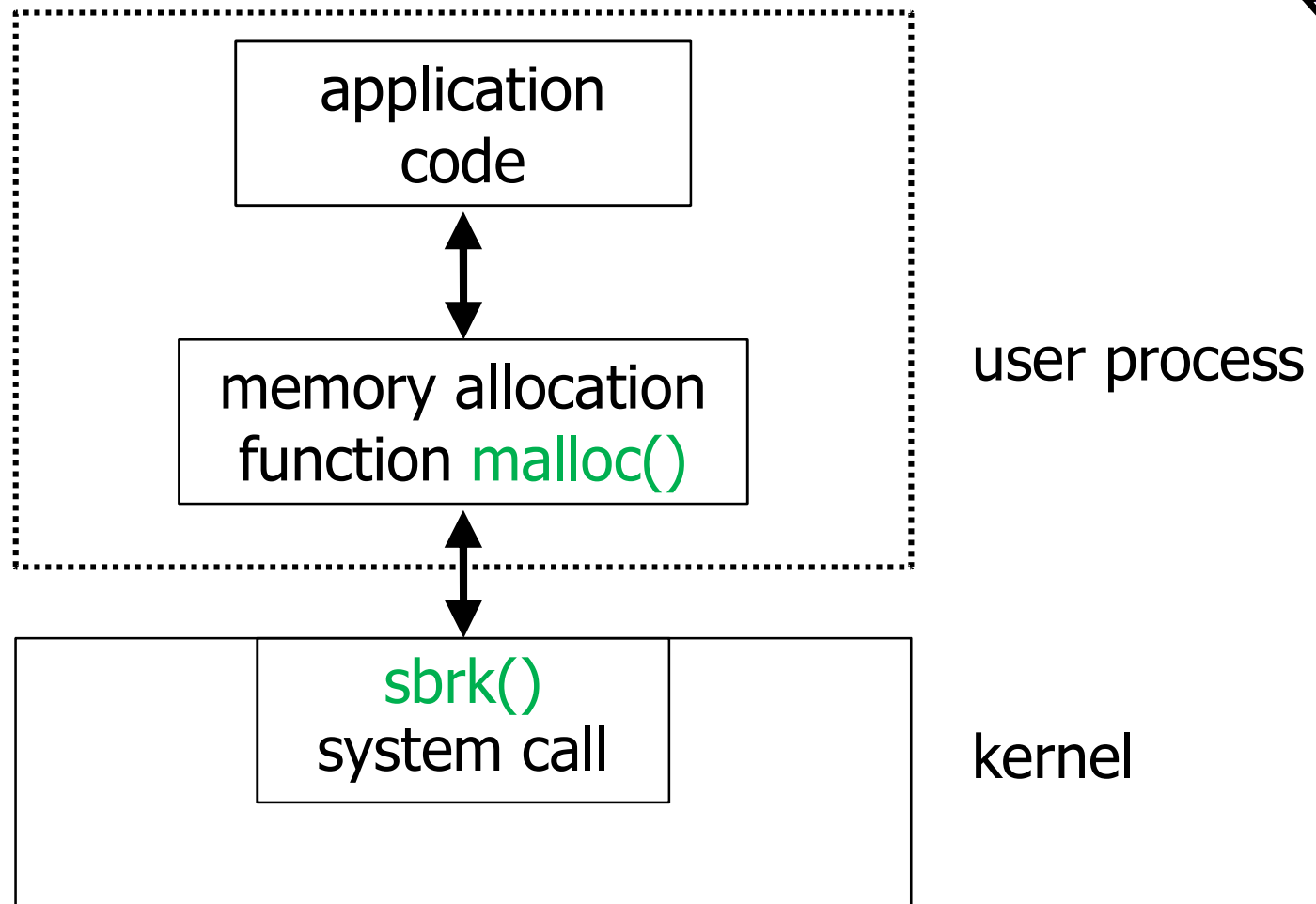
- man 3 printf



System Calls & Library Functions



System Calls & Library Functions



System Calls & Library Functions



- malloc(): Memory allocation
- Many ways to do memory allocation and garbage collection (best fit, first fit)
- sbrk(): UNIX system call, increases or decreases address space of process by a specified number of bytes
- Can implement own mem alloc function

using sbrk() -> `unistd.h`

`man malloc |grep sbrk`



ar 建立库中的目标文件



AR(1)

GNU Development Tools

AR(1)

NAME

ar - create, modify, and extract from archives

SYNOPSIS

```
ar [--plugin name] [-X32_64] [-]p[mod [relpos] [count]] [--target bfdname] archive
[member...]
```

DESCRIPTION

The GNU ar program creates, modifies, and extracts from archives. An archive is a single file holding a collection of other files in a structure that makes it possible to retrieve the original individual files (called members of the archive).

The original files' contents, mode (permissions), timestamp, owner, and group are preserved in the archive, and can be restored on extraction.

GNU ar can maintain archives whose members have names of any length; however, depending on how ar is configured on your system, a limit on member-name length may be imposed for compatibility with archive formats maintained with other tools. If it exists, the limit is often 15 characters (typical of formats related to a.out) or 16 characters (typical of formats related to coff).

ar is considered a binary utility because archives of this sort are most often used as libraries holding commonly needed subroutines.

ar creates an index to the symbols defined in relocatable object modules in the archive when you specify the modifier s. Once created, this index is updated in the archive whenever ar makes a change to its contents (save for the q update operation). An archive with such an index speeds up linking to the library, and allows routines in the library to call each other without regard to their placement in the archive.



ar & nm command



```
[buyhorse@embedded example]$ ls
compute.c  first.c  input.o  main.o      makefile_power  power.c
compute.h  input.c  main.c   makefile    pow              second.c
compute.o  input.h  main.h   makefile_new power            slogan
[buyhorse@embedded example]$ ar r mathlib.a input.o compute.o
ar: creating mathlib.a
[buyhorse@embedded example]$
```

```
[buyhorse@embedded example]$ nm mathlib.a
```

```
input.o:
```

```
00000000 T input
          U printf
          U scanf
```

```
compute.o:
```

```
00000000 T compute
          U pow
```

nm hello.o//显示符号信息

man nm

nm a.out

```
[buyhorse@embedded example]$ nm mathlib.a | grep compute
```



```
compute.o:
```

```
00000000 T compute
```

```
[buyhorse@embedded example]$
```



gcc warning



```
buyhorse@ubuntu:~/gdbexample$ cat cat.c
/*
 * copy stdin to stdout
 */
#include <stdio.h>

main (int argc, char *argv[])
{
    char c;
    int i, j;

    while((c=getchar())!=EOF)
        putchar(c);
}

buyhorse@ubuntu:~/gdbexample$ gcc cat.c
cat.c:6:1: warning: return type defaults to 'int' [-Wimplicit-int]
    main (int argc, char *argv[])
    ^
buyhorse@ubuntu:~/gdbexample$ gcc -Wall cat.c
cat.c:6:1: warning: return type defaults to 'int' [-Wimplicit-int]
    main (int argc, char *argv[])
    ^

cat.c: In function 'main':
cat.c:9:7: warning: unused variable 'j' [-Wunused-variable]
    int i, j;
           ^
cat.c:9:5: warning: unused variable 'i' [-Wunused-variable]
    int i, j;
       ^

buyhorse@ubuntu:~/gdbexample$ █
```

2

gcc warning

- -Wall选项会打开许多有用的外部警告
- -w 选项是关闭警告信息
- gcc -Wall cat.c
- gcc -w cat.c
- gcc -Wall -Wno-unused cat.c
- gcc -Wall -Wno-implicit cat.c
- gcc -Wall -Werror cat.c//警告当做错误处理



优化选项（了解）



- **-O0**: 无优化(默认)
- **-O/-O1**: 主要进行跳转和延迟退栈两种优化；在编译大型程序的时候会显著增加编译时内存的使用。
- **-O2**: 除了完成**-O1**的优化之外，还进行一些额外的调整工作，如指令调整等。
- **-O3**: 优化级别最高，打开所有**-O2**的优化选项并包括循环展开和其他一些与处理特性相关的优化工作。
 - `try-gcc-op.c`
 - `time ./a.out`
 - `time ./b.out`



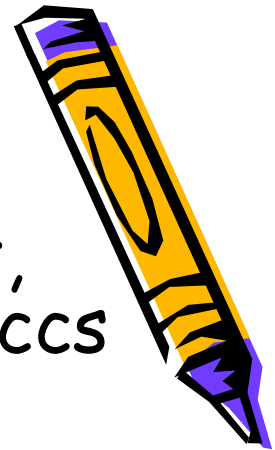
版本控制（了解）

- git
- sccs 源代码控制系统: admin, help, get, sact, unget, delta, prs, rmdel, comb, sccs
- svn
- RCS 版本控制系统: ci, co, rcs, rlog, rcsdiff, rcsmerge

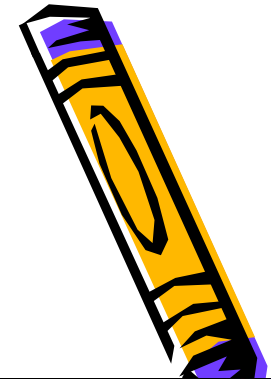
- mkdir RCS

- ci test.c //check in
- co test.c //check out
- co -l test.c //lock
- vi test.c
- co test.c
- rlog test.c
- co -r1.1 test.c

Thursday, May 13, 2021



`gdb: the GNU Debugger`



```
GDB (1)                                GNU Development Tools                                GDB (1)

NAME
  gdb - The GNU Debugger

SYNOPSIS
  gdb [-help] [-nh] [-nx] [-q] [-batch] [-cd=dir] [-f] [-b bps]
      [-tty=dev] [-s symfile] [-e prog] [-se prog] [-c core] [-p procID]
      [-x cmds] [-d dir] [prog|prog procID|prog core]

DESCRIPTION
  The purpose of a debugger such as GDB is to allow you to see what is going on "inside" another
  program while it executes -- or what another program was doing at the moment it crashed.

  GDB can do four main kinds of things (plus other things in support of these) to help you catch
  bugs in the act:

  • Start your program, specifying anything that might affect its behavior.

  • Make your program stop on specified conditions.

  • Examine what has happened, when your program has stopped.

  • Change things in your program, so you can experiment with correcting the effects of one bug
    and go on to learn about another.

  You can use GDB to debug programs written in C, C@t{++}, Fortran and Modula-2.
```



gdb example

- `gcc -g`
- Example: `string-debug.c`

```
#include "stdio.h"
static char buff[256];
static char *string;
int
main ()
{
    printf ("Please input a string:\n");
    gets (string);
    printf ("\nYour string is: %s\n", string);
}
```

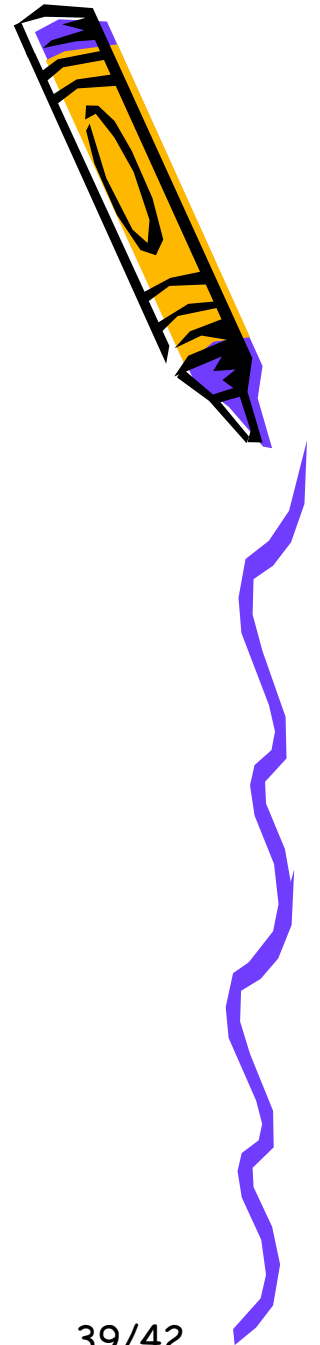


gdb cont'

- `gcc -g string-debug.c`
- `gdb a.out`
- `file a.out` //装入被调试的可执行文件
- `run`
- `where`
- `list` //列出源代码
- `print string`
- `break string-debug.c:7`
- `run`
- `set variable string=(char*)malloc(100)`
- `next` //单步运行且不进入函数内部
- `q`



Thursday, May 13, 2021



ulimit

- `ulimit -c`
- `ulimit -c unlimited`
- `gdb a.out core***`
- `cat /proc/sys/kernel/core_uses_pid`
//若为1则文件名为core.pid格式



simple examples

- pid.c:
- uidgid.c
- stdin-to-out.c-1: stdin → stdout using unbuffered I/O
- stdin-to-out-2.c: stdin → stdout using standard I/O
- myls.c: bare bones implementation of ls command
- callexe.c:
- errno.c
- sum.c
- holefile.c

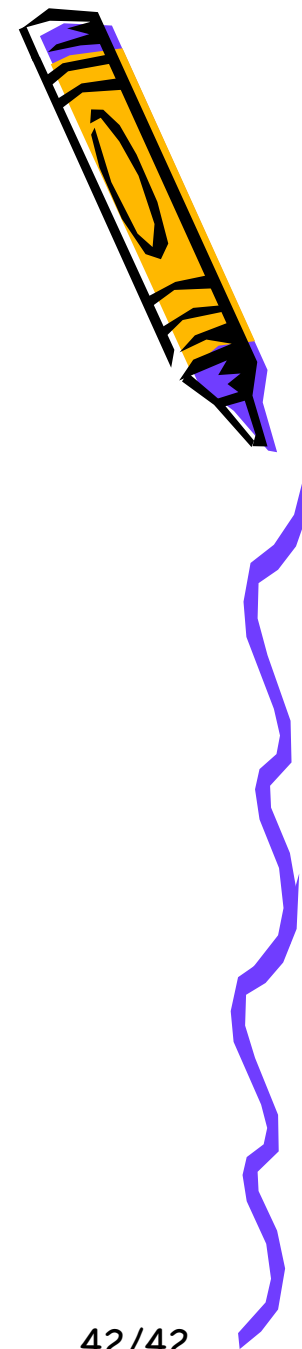


本章小结

- 程序生成
 - 编辑vi/vim...
 - 缩排indent
- 程序编译
 - 编译gcc...
 - **Makefile**
 - 警告
 - 优化
- 程序调试
 - **GDB**



Thursday, May 13, 2021



42/42