实验环境与工具

计算机系统概论实验导引(0)

张宇轩
yuxuanzh23@mails.tsinghua.edu.cn

本地实验环境搭建

- 系统课组均使用Linux环境（推荐Ubuntu 20.04 或 22.04）开展实验
- Windows：WSL 2
  - 前置要求笔记本BIOS中启用了虚拟化技术：https://zhuanlan.zhihu.com/p/586751199
    （**不要盲目修改BIOS**，必要时可以寻求【科技服务队】的帮助）
  - 默认安装：https://learn.microsoft.com/zh-cn/windows/wsl/install
  - 非系统盘安装：https://damsteen.nl/blog/2018/08/29/installing-wsl-manually-on-non-system-drive

    下载发行版 https://learn.microsoft.com/zh-cn/windows/wsl/install-manual#downloading-distributions，解压时与教程会有不同
- MacOS：VMWare、Parallel Desktops
  - VMWare Fusion： https://www.vmware.com/cn/products/fusion.html
  - Parallel Desktops：https://www.parallels.com/

    安装 Ubuntu Desktop镜像 https://cn.ubuntu.com/download/desktop（直接使用，体积较大）
    安装Ubuntu Server镜像 https://cn.ubuntu.com/download/server/step1 （使用SSH远程接入虚拟机，轻量）

VS Code推荐插件

VS Code上手教学：https://code.visualstudio.com/docs/introvideos/basics

Remote Development v0.24.0 预览版

Microsoft microsoft.com | 4,465,237 | ★★★★½ (108)

An extension pack that lets you open any folder in a container, on ...

禁用 | 卸载

此扩展已全局启用。

GitLens — Git supercharged

GitKraken gitkraken.com | 25,931,208 | ★★

Supercharge Git within VS Code — Visualize code aut...

禁用 | 卸载 | 切换到预发布版本

已在"WSL: Ubuntu"上启用扩展

C/C++ v1.17.5

Microsoft microsoft.com

C/C++ IntelliSense, debugging, and code browsing.

禁用 | 卸载

已在"WSL: Ubuntu"上启用扩展

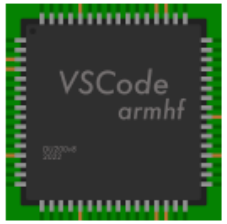
x86 and x86_64 Assembly v3.1

13xforever | 638,887 | ★★★★☆ (15)

Cutting edge x86 and x86_64 assembly syntax highlig...

禁用 | 卸载

此扩展已全局启用。



Arm Assembly v1.7.4

dan-c-underwood | 499,608 | ★★★★★ (6)

Arm assembly syntax support for Visual Studio Code

禁用 | 卸载

此扩展已全局启用。

RISC-V Support v0.0.8

zhwu95 | 53,071 | ★★★★★ (2)

Provides syntax highlighting and snippets for RISC-V assembly lan...

禁用 | 卸载

此扩展已全局启用。

提示：

可以将汇编代码保存为 `.x86asm`, `.armasm`, `.rvasm`, 并指定文件类型关联至相应的插件上

实验环境配置

- 安装build-essential、gcc-multilib、g++-multilib、gdb；可选git

```
yuxuan-z@DESKTOP-VBL6Q41:~$ sudo apt install -y build-essential gcc-multilib g++-multilib gdb
```

- 测试gcc是否安装成功
  - which 命令：在【环境变量】中寻找【可执行文件】

```
yuxuan-z@DESKTOP-VBL6Q41:~$ which gcc
/usr/bin/gcc
```

  - whereis命令：在【环境变量 & 系统目录】中寻找【含关键词的所有文件】

```
yuxuan-z@DESKTOP-VBL6Q41:~$ whereis gcc
gcc: /usr/bin/gcc /usr/lib/gcc /usr/share/gcc /usr/share/man/man1/gcc.1.gz
```

- 在课程服务器上已安装上述工具
- 建议使用git管理作业项目的版本 https://www.bilibili.com/video/BV1Wh4y1s7Lj

C/C++代码编译流程

- 从 .c/.cpp文件 → 可执行文件 的过程：
  1. 预处理：处理 #include 和 #define

```
yuxuan-z@DESKTOP-VBL6Q41:~$ gcc -E test.c -o test.i
```

  2. 编译：C代码→汇编代码

```
yuxuan-z@DESKTOP-VBL6Q41:~$ gcc -S test.i -o test.s
```

  3. 汇编：汇编代码→对象文件

```
yuxuan-z@DESKTOP-VBL6Q41:~$ gcc -c test.s -o test.o
```

  4. 链接：对象文件+标准库→可执行文件

```
yuxuan-z@DESKTOP-VBL6Q41:~$ gcc test.o -o test
```

每个步骤的具体命令可以使用 -v 选项来查看

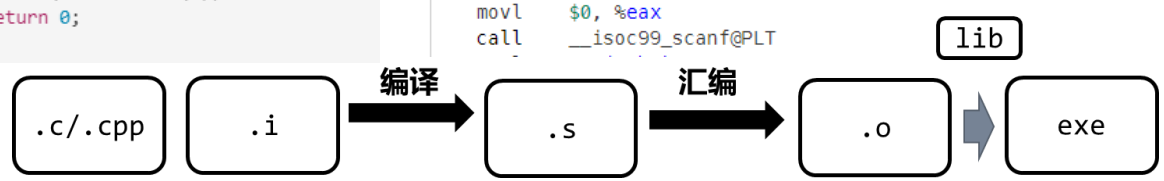
```
yuxuan-z@DESKTOP-VBL6Q41:~$ gcc -v test.c -o test
```

```c
// test.c
#include <stdio.h>

int fib(int n) {
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}

int main(int argc, char const *argv[]) {
    int n;
    scanf("%d", &n);
    printf("%d\n", fib(n));
    return 0;
}
```

```
main:
.LFB1:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $32, %rsp
    movl    %edi, -20(%rbp)
    movq    %rsi, -32(%rbp)
    movq    %fs:40, %rax
    movq    %rax, -8(%rbp)
    xorl    %eax, %eax
    leaq    -12(%rbp), %rax
    movq    %rax, %rsi
    leaq    .LC0(%rip), %rdi
    movl    $0, %eax
    call    __isoc99_scanf@PLT
```




```c
// test.i
...
# 858 "/usr/include/stdio.h" 3 4
extern int __uflow (FILE *);
extern int __overflow (FILE *, int);
# 873 "/usr/include/stdio.h" 3 4

# 2 "fib.c" 2


# 3 "fib.c"
int fib(int n) {
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}

int main(int argc, char const *argv[]) {
    int n;
    scanf("%d", &n);
    printf("%d\n", fib(n));
    return 0;
}
```

```
00000460  00 2E 73 79 6D 74 61 62 00 2E 73 74 72 74 61 62  . . s y m t a b . . s t r t a b
00000470  00 2E 73 68 73 74 72 74 61 62 00 2E 72 65 6C 61  . . s h s t r t a b . . r e l a
00000480  2E 74 65 78 74 00 2E 64 61 74 61 00 2E 62 73 73  . t e x t . . d a t a . . b s s
00000490  00 2E 72 6F 64 61 74 61 00 2E 63 6F 6D 6D 65 6E  . . r o d a t a . . c o m m e n
000004A0  74 00 2E 6E 6F 74 65 2E 47 4E 55 2D 73 74 61 63  t . . n o t e . G N U - s t a c
000004B0  6B 00 2E 6E 6F 74 65 2E 67 6E 75 2E 70 72 6F 70  k . . n o t e . g n u . p r o p
000004C0  65 72 74 79 00 2E 72 65 6C 61 2E 65 68 5F 66 72  e r t y . . r e l a . e h _ f r
000004D0  61 6D 65 00 00 00 00 00 00 00 00 00 00 00 00 00  a m e . . . . . .
000004E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . .
```

C/C++代码编译与汇编

- C代码→ 汇编代码

```
yuxuan-z@DESKTOP-VBL6Q41:~$ gcc -S test.c -o test.s
```

```c
// test.c
#include <stdio.h>

int fib(int n) {
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}

int main(int argc, char const *argv[]) {
    int n;
    scanf("%d", &n);
    printf("%d\n", fib(n));
    return 0;
}
```

```
main:
.LFB1:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $32, %rsp
    movl    %edi, -20(%rbp)
    movq    %rsi, -32(%rbp)
    movq    %fs:40, %rax
    movq    %rax, -8(%rbp)
    xorl    %eax, %eax
    leaq    -12(%rbp), %rax
    movq    %rax, %rsi
    leaq    .LC0(%rip), %rdi
    movl    $0, %eax
    call    __isoc99_scanf@PLT
```

- C代码→ 对象文件

```
yuxuan-z@DESKTOP-VBL6Q41:~$ gcc -c test.c -o test.o
```

  - 通过file命令来查看

```
yuxuan-z@DESKTOP-VBL6Q41:~$ file test.o
test.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
```

  - 通过readelf命令来查看

```
yuxuan-z@DESKTOP-VBL6Q41:~$ readelf -h test.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          1240 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         14
  Section header string table index: 13
```

对象文件链接

- 对象文件→ 可执行文件

```
yuxuan-z@DESKTOP-VBL6Q41:~$ gcc test.o -o test
```

  - 通过file命令来查看

```
yuxuan-z@DESKTOP-VBL6Q41:~$ file test.o
test.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
yuxuan-z@DESKTOP-VBL6Q41:~$ file test
test: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=b59fa953a3e67bd68355edb6d76
74ef6c68b0155, for GNU/Linux 3.2.0, not stripped
```

  - 通过readelf命令来查看

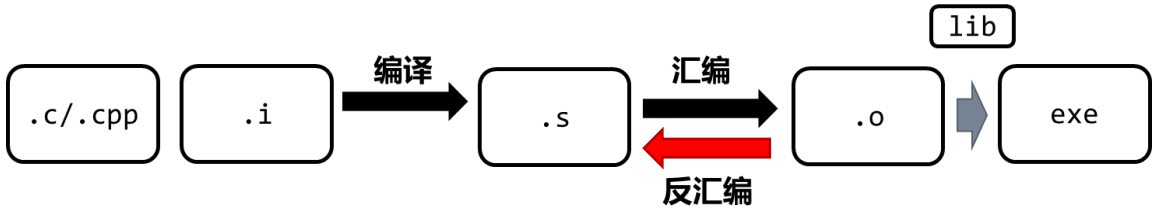
```
yuxuan-z@DESKTOP-VBL6Q41:~$ readelf -h test.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          1240 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         14
  Section header string table index: 13
```

```
yuxuan-z@DESKTOP-VBL6Q41:~$ readelf -h test
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x10a0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          14840 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

反汇编

- 对象文件→ 汇编文件

```
yuxuan-z@DESKTOP-VBL6Q41:~$ objdump -S test.o > test.S
```



- 显示的信息包括：
  1. 偏移
  2. 指令十六进制编码
  3. 汇编指令

```
0000000000000000 <fib>:
   0: f3 0f 1e fa           endbr64
   4: 55                    push   %rbp
   5: 48 89 e5              mov    %rsp,%rbp
   8: 53                    push   %rbx
   9: 48 83 ec 18           sub    $0x18,%rsp
   d: 89 7d ec              mov    %edi,-0x14(%rbp)
  10: 83 7d ec 01           cmpl   $0x1,-0x14(%rbp)
  14: 7f 05                 jg     1b <fib+0x1b>
  16: 8b 45 ec              mov    -0x14(%rbp),%eax
  19: eb 1e                 jmp    39 <fib+0x39>
  1b: 8b 45 ec              mov    -0x14(%rbp),%eax
  1e: 83 e8 01              sub    $0x1,%eax
  21: 89 c7                 mov    %eax,%edi
  23: e8 00 00 00 00        callq  28 <fib+0x28>
  28: 89 c3                 mov    %eax,%ebx
  2a: 8b 45 ec              mov    -0x14(%rbp),%eax
  2d: 83 e8 02              sub    $0x2,%eax
  30: 89 c7                 mov    %eax,%edi
  32: e8 00 00 00 00        callq  37 <fib+0x37>
  37: 01 d8                 add    %ebx,%eax
  39: 48 83 c4 18           add    $0x18,%rsp
  3d: 5b                    pop    %rbx
  3e: 5d                    pop    %rbp
  3f: c3                    retq
```

编译与反汇编工具小结

- gcc常用选项
  - `-o [filename]`　指定输出文件名
  - `-E`　预处理
  - `-S`　编译
  - `-c`　汇编
  - `-v`　显示完整过程
  - `-g`　开启调试信息
  - `-O{1,2,3,g,fast,s}` 优化选项，如-O2
- objdump常用选项
  - `-d`　反汇编可执行的段(如.text)
  - `-D`　反汇编所有的段，一率当做指令反汇编
  - `-S`　整合反汇编代码与调试信息一起显示
  - `-t`　显示函数的符号列表
  - `--adjust-vma OFFSET` 把反汇编出来的地址都加上一个偏移，常用于嵌入式开发
  - `-M {intel,att}` 用 Intel 或者 AT&T 风格显示 X86 汇编；默认 AT&T 风格

gdb调试#0

- gdb是一个可以监控程序运行时行为的调试器
  - 设置断点：在设定点停止程序
  - 观察变量值、寄存器值变化
  - 路径跟踪：[单/多]步 执行 指令或代码语句
  - 错误查找：自动停在abort处
- 为了方便演示，我们重写test.c，
  将fib()由递归改成递推实现
  并通过argv[1]来传递n

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

int fib(int n) {
    if (n < 2) return n;
    int dp[3] = {0, 1, -1};
    for (int i = 2; i <= n; i++) {
        dp[2] = dp[0] + dp[1];
        dp[0] = dp[1];
        dp[1] = dp[2];
    }
    return dp[2];
}

int main(int argc, char const *argv[]) {
    if (argc < 2) exit(-1);
    int n = atoi(argv[1]);
    int ans = fib(n);
    printf("fib(%d)=%d\n", n, ans);
    return 0;
}
```

gdb调试#0

- 启动、运行与退出
  - 编译程序并加载可执行文件进入gdb

```
yuxuan-z@DESKTOP-VBL6Q41:~$ gcc -g test.c -o test
yuxuan-z@DESKTOP-VBL6Q41:~$ gdb -q ./test
Reading symbols from ./test...
(gdb)
```

  - 运行并传入argv[1]

```
(gdb) r 25
Starting program: /home/yuxuan-z/test 25
fib(25)=75025
[Inferior 1 (process 5305) exited normally]
```

  - 清屏（通过!来调用Shell）

```
(gdb) !clear
```

  - 退出

```
(gdb) q
```

- 断点设置
  - 打断点：b [函数名 / 文件:行数 / *addr]
  - 继续执行：c
  - 查看断点：info b
  - 删除断点：d [断点id]

TRY 在第10行打断点，执行两次后，删除断点

```
(gdb) b test.c:10
Breakpoint 1 at 0x1201: file test.c, line 10.
(gdb) r 25
Starting program: /home/yuxuan-z/test 25

Breakpoint 1, fib (n=25) at test.c:10
10              dp[1] = dp[2];
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000555555555201 in fib at test.c:10
        breakpoint already hit 1 time
(gdb) c
Continuing.

Breakpoint 1, fib (n=25) at test.c:10
10              dp[1] = dp[2];
(gdb) d 1
(gdb) info b
No breakpoints or watchpoints.
```

gdb调试#0

- 查看变量值：p [变量名 / $寄存器]

TRY 在第10行打断点，查看1st和2nd迭代中dp[]的值，删除断点

```
(gdb) b test.c:10
Breakpoint 1 at 0x1201: file test.c, line 10.
(gdb) r 25
Starting program: /home/yuxuan-z/test 25

Breakpoint 1, fib (n=25) at test.c:10
10              dp[1] = dp[2];
(gdb) n
7           for (int i = 2; i <= n; i++) {
(gdb) p dp
$1 = {1, 1, 1}
(gdb) c
Continuing.

Breakpoint 1, fib (n=25) at test.c:10
10              dp[1] = dp[2];
(gdb) n
7           for (int i = 2; i <= n; i++) {
(gdb) p dp
$2 = {1, 2, 2}
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000555555555201 in fib at test.c:10
        breakpoint already hit 2 times
(gdb) d 1
(gdb) info b
No breakpoints or watchpoints.
```

- 单/多步执行
  - n：停在下一行源代码（无论是否为函数调用）
  - s：若为函数则进入函数体，否则同n
  - [n/s] 空格 步数：多步执行

TRY 进入fib()并查看参数n的值

```
(gdb) b test.c:18
Breakpoint 1 at 0x1265: file test.c, line 18.
(gdb) r 25
Starting program: /home/yuxuan-z/test 25

Breakpoint 1, main (argc=2, argv=0x7fffffffddc8) at test.c:18
18          int ans = fib(n);
(gdb) s
fib (n=21845) at test.c:4
4       int fib(int n) {
(gdb) p $rdi
$1 = 25
(gdb) p n
$2 = 21845
(gdb) n
5           if (n < 2) return n;
(gdb) p n
$3 = 25
```

为什么n的值不对呢？
可以用指令粒度的调试！

gdb调试#0

- 以指令为粒度的单/多步执行
  - ni：停在下一行源代码（无论是否为函数调用）
  - si：若为函数则进入函数体，否则同ni
  - [ni/si] 空格 步数：多步执行

- 显示当前的执行状态
  - layout src ：仅显示源代码状态
  - layout asm ：仅显示汇编代码状态
  - layout split ：同时显示源代码和汇编代码状态

TRY 进入fib()并查看参数n的值，查看n的值何时改变

HINT 查看参数n的地址及写入该地址时的时刻

```
test.c
 1          #include <stdio.h>
 2          #include <stdlib.h>
 3
>4          int fib(int n) {
 5              if (n < 2) return n;
 6              int dp[3] = {0, 1, -1};
 7              for (int i = 2; i <= n; i++) {
 8                  dp[2] = dp[0] + dp[1];
 9                  dp[0] = dp[1];
10                  dp[1] = dp[2];
11              }
12              return dp[2];
13          }
14
15          int main(int argc, char const *argv[]) {
16              if (argc < 2) exit(-1);
17              int n = atoi(argv[1]);
B+18            int ans = fib(n);
19              printf("fib(%d)=%d\n", n, ans);
20              return 0;
21          }
```

```
0x5555555551a9 <fib>    endbr64
0x5555555551ad <fib+4>  push   %rbp
0x5555555551ae <fib+5>  mov    %rsp,%rbp
0x5555555551b1 <fib+8>  sub    $0x30,%rsp
>0x5555555551b5 <fib+12> mov    %edi,-0x24(%rbp)
0x5555555551b8 <fib+15> mov    %fs:0x28,%rax
0x5555555551c1 <fib+24> mov    %rax,-0x8(%rbp)
0x5555555551c5 <fib+28> xor    %eax,%eax
0x5555555551c7 <fib+30> cmpl   $0x1,-0x24(%rbp)
0x5555555551cb <fib+34> jg     0x5555555551d2 <fib+41>
0x5555555551cd <fib+36> mov    -0x24(%rbp),%eax
0x5555555551d0 <fib+39> jmp    0x555555555216 <fib+109>
0x5555555551d2 <fib+41> movl   $0x0,-0x14(%rbp)
0x5555555551d9 <fib+48> movl   $0x1,-0x10(%rbp)
0x5555555551e0 <fib+55> movl   $0xffffffff,-0xc(%rbp)
0x5555555551e7 <fib+62> movl   $0x2,-0x18(%rbp)
0x5555555551ee <fib+69> jmp    0x55555555520b <fib+98>
0x5555555551f0 <fib+71> mov    -0x14(%rbp),%edx
0x5555555551f3 <fib+74> mov    -0x10(%rbp),%eax
0x5555555551f6 <fib+77> add    %edx,%eax
0x5555555551f8 <fib+79> mov    %eax,-0xc(%rbp)
0x5555555551fb <fib+82> mov    -0x10(%rbp),%eax
```

gdb调试#0

- 其它基础命令
  - `bt` ： 查看函数调用栈
  - `finish` ： 执行完当前函数
  - `until 空格 文件:行数` ： 一直执行至指定行数
  - `info reg`：查看当前时刻所有寄存器的值

```
(gdb) bt
#0  fib (n=21845) at test.c:4
#1  0x000055555555526f in main (argc=2, argv=0x7fffffffddc8) at test.c:18
(gdb) finish
Run till exit from #0  fib (n=21845) at test.c:4
0x000055555555526f in main (argc=2, argv=0x7fffffffddc8) at test.c:18
18          int ans = fib(n);
Value returned is $1 = 75025
(gdb) until test.c:20
fib(25)=75025
main (argc=2, argv=0x7fffffffddc8) at test.c:20
20          return 0;
```

这就完了吗？不！gdb之旅才刚刚开始 o(￣T￣*)ゞ

- 我们会在Attack Lab开始前再深入学习gdb
- 课外·gdb进阶练习：https://pwn.college/fundamentals/debugging-refresher
- 课外·《100个gdb调试小技巧》：https://github.com/hellogcc/100-gdb-tips