1. 了解汇编

尝试理解下面的命令

$gcc -S -m32 lab0\_ex1.c

接着我们将得到lab0\_ex1.s文件，请写出汇编代码与c代码之间的关系。

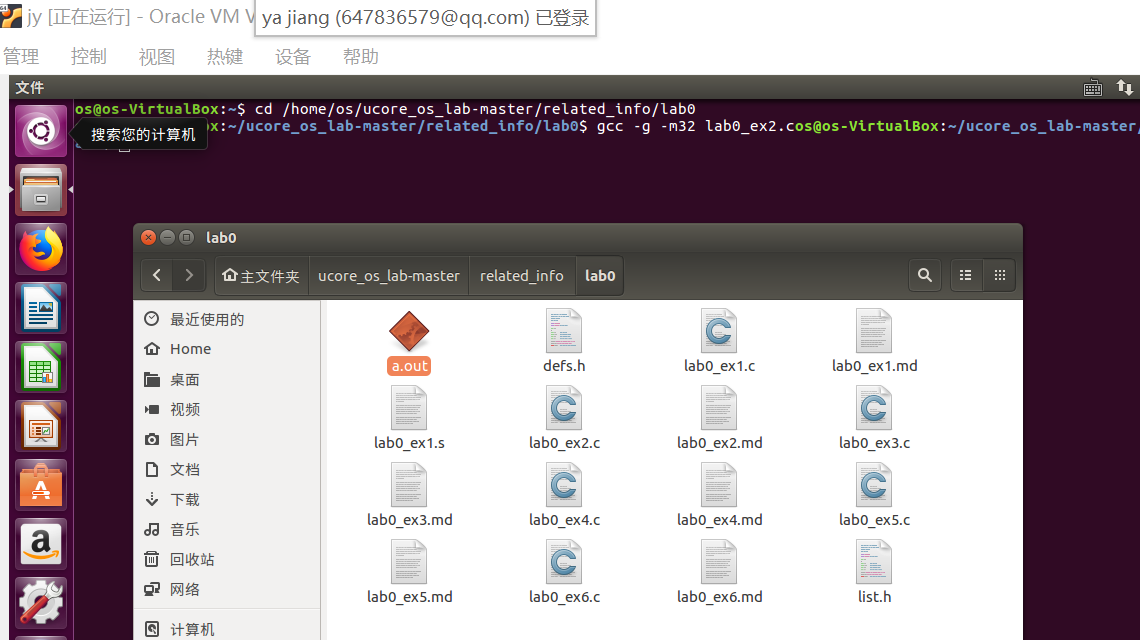
汇编语言与机器代码是一一对应的，编译器直接将汇编语言转换成机器代码，对于c语言编译器先将c转化成汇编语言然后再转换成机器代码。

1. 用gdb调试

尝试下面的命令，

$gcc -g -m32 lab0\_ex2.c

接着我们会得到a.out文件，请用gdb调试，并写出设置断点、单步执行及查看变量的过程。



[1] 运行 “gdb a.out” ，加载 a.out 可执行文件；

$gdb a.out

[2] 执行装入的 a.out 命令；

(gdb) run

[3] 使用 where 命令查看程序出错的地方；

(gdb) where

[4] 利用 list 命令查看调用 gets 函数附近的代码；

(gdb) list

[5] 在 gdb 中，我们在第 4 行处设置断点，看看是否是在第11行出错；

(gdb) break 4

[6] 程序重新运行到第 4 行处停止，这时程序正常，然后执行单步命令next；

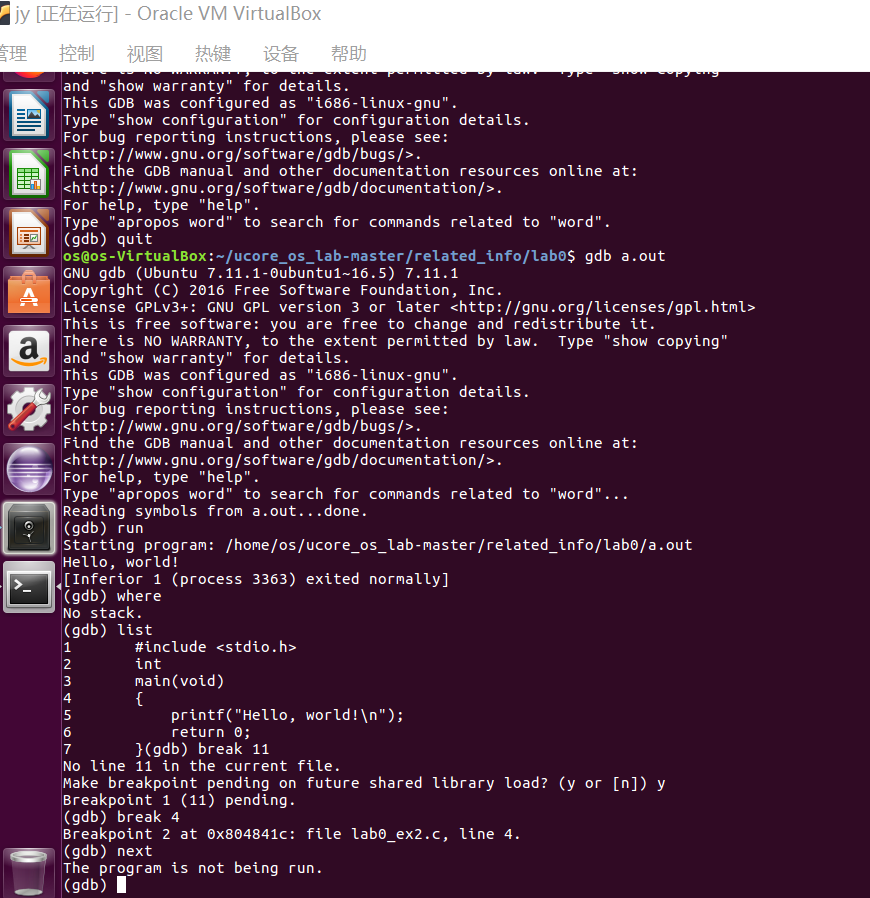
(gdb) next

[7] 程序确实出错，能够导致 gets 函数出错的因素就是变量 string。重新执行测试程，用 print 命令查看 string 的值；

(gdb) run (gdb) print string (gdb) $1=0x0

[8] 问题在于string指向的是一个无效指针，修改程序，在10行和11行之间增加一条语句 “string=buff; ”，重新编译程序，然后继续运行，将看到正确的程序运行结果。

用gdb查看源代码可以用list命令，但是这个不够灵活。可以使用"layout src"命令，或者按CtrlX再按A，就会出现一个窗口可以查看源代码。也可以用使用-tui参数，这样进入gdb里面后就 能直接打开代码查看窗口。其他代码窗口相关命令：



1. 掌握指针和类型转换相关的Ｃ编程

分析如下代码段

写出gintr和intr的结果，试着编译这段代码，如果遇到错误进行改正，并分析错误原因。

intr is 0x10002

gintr is 0xee0000010002

4. 掌握通用链表结构相关的Ｃ编程

查看list.h和lab0\_ex4.c，编写一个程序，利用list.h中的链表结构，将26个英文字母存入链表中，并逆序打印出来。

#ifndef \_\_LIBS\_LIST\_H\_\_

#define \_\_LIBS\_LIST\_H\_\_

#ifndef \_\_ASSEMBLER\_\_

#include <defs.h>

/\* \*

\* Simple doubly linked list implementation.

\*

\* Some of the internal functions ("\_\_xxx") are useful when manipulating

\* whole lists rather than single entries, as sometimes we already know

\* the next/prev entries and we can generate better code by using them

\* directly rather than using the generic single-entry routines.

\* \*/

struct list\_entry {

struct list\_entry \*prev, \*next;

};

typedef struct list\_entry list\_entry\_t;

static inline void list\_init(list\_entry\_t \*elm) \_\_attribute\_\_((always\_inline));

static inline void list\_add(list\_entry\_t \*listelm, list\_entry\_t \*elm) \_\_attribute\_\_((always\_inline));

static inline void list\_add\_before(list\_entry\_t \*listelm, list\_entry\_t \*elm) \_\_attribute\_\_((always\_inline));

static inline void list\_add\_after(list\_entry\_t \*listelm, list\_entry\_t \*elm) \_\_attribute\_\_((always\_inline));

static inline void list\_del(list\_entry\_t \*listelm) \_\_attribute\_\_((always\_inline));

static inline void list\_del\_init(list\_entry\_t \*listelm) \_\_attribute\_\_((always\_inline));

static inline bool list\_empty(list\_entry\_t \*list) \_\_attribute\_\_((always\_inline));

static inline list\_entry\_t \*list\_next(list\_entry\_t \*listelm) \_\_attribute\_\_((always\_inline));

static inline list\_entry\_t \*list\_prev(list\_entry\_t \*listelm) \_\_attribute\_\_((always\_inline));

static inline void \_\_list\_add(list\_entry\_t \*elm, list\_entry\_t \*prev, list\_entry\_t \*next) \_\_attribute\_\_((always\_inline));

static inline void \_\_list\_del(list\_entry\_t \*prev, list\_entry\_t \*next) \_\_attribute\_\_((always\_inline));

/\* \*

\* list\_init - initialize a new entry

\* @elm: new entry to be initialized

\* \*/

static inline void

list\_init(list\_entry\_t \*elm) {

elm->prev = elm->next = elm;

}

/\* \*

\* list\_add - add a new entry

\* @listelm: list head to add after

\* @elm: new entry to be added

\*

\* Insert the new element @elm \*after\* the element @listelm which

\* is already in the list.

\* \*/

static inline void

list\_add(list\_entry\_t \*listelm, list\_entry\_t \*elm) {

list\_add\_after(listelm, elm);

}

/\* \*

\* list\_add\_before - add a new entry

\* @listelm: list head to add before

\* @elm: new entry to be added

\*

\* Insert the new element @elm \*before\* the element @listelm which

\* is already in the list.

\* \*/

static inline void

list\_add\_before(list\_entry\_t \*listelm, list\_entry\_t \*elm) {

\_\_list\_add(elm, listelm->prev, listelm);

}

/\* \*

\* list\_add\_after - add a new entry

\* @listelm: list head to add after

\* @elm: new entry to be added

\*

\* Insert the new element @elm \*after\* the element @listelm which

\* is already in the list.

\* \*/

static inline void

list\_add\_after(list\_entry\_t \*listelm, list\_entry\_t \*elm) {

\_\_list\_add(elm, listelm, listelm->next);

}

/\* \*

\* list\_del - deletes entry from list

\* @listelm: the element to delete from the list

\*

\* Note: list\_empty() on @listelm does not return true after this, the entry is

\* in an undefined state.

\* \*/

static inline void

list\_del(list\_entry\_t \*listelm) {

\_\_list\_del(listelm->prev, listelm->next);

}

/\* \*

\* list\_del\_init - deletes entry from list and reinitialize it.

\* @listelm: the element to delete from the list.

\*

\* Note: list\_empty() on @listelm returns true after this.

\* \*/

static inline void

list\_del\_init(list\_entry\_t \*listelm) {

list\_del(listelm);

list\_init(listelm);

}

/\* \*

\* list\_empty - tests whether a list is empty

\* @list: the list to test.

\* \*/

static inline bool

list\_empty(list\_entry\_t \*list) {

return list->next == list;

}

/\* \*

\* list\_next - get the next entry

\* @listelm: the list head

\*\*/

static inline list\_entry\_t \*

list\_next(list\_entry\_t \*listelm) {

return listelm->next;

}

/\* \*

\* list\_prev - get the previous entry

\* @listelm: the list head

\*\*/

static inline list\_entry\_t \*

list\_prev(list\_entry\_t \*listelm) {

return listelm->prev;

}

/\* \*

\* Insert a new entry between two known consecutive entries.

\*

\* This is only for internal list manipulation where we know

\* the prev/next entries already!

\* \*/

static inline void

\_\_list\_add(list\_entry\_t \*elm, list\_entry\_t \*prev, list\_entry\_t \*next) {

prev->next = next->prev = elm;

elm->next = next;

elm->prev = prev;

}

/\* \*

\* Delete a list entry by making the prev/next entries point to each other.

\*

\* This is only for internal list manipulation where we know

\* the prev/next entries already!

\* \*/

static inline void

\_\_list\_del(list\_entry\_t \*prev, list\_entry\_t \*next) {

prev->next = next;

next->prev = prev;

}

#endif /\* !\_\_ASSEMBLER\_\_ \*/

#endif /\* !\_\_LIBS\_LIST\_H\_\_ \*/