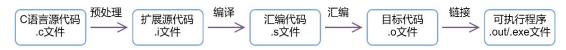
##写在前面

本人之前没有过任何 C 语言的编程经验,所以这次任务从自学开始一路摸爬滚打走来,再加上看到 Dian 招新任务已经是 9 月 16 号左右了,时间比较紧,如果有做得不够好的地方还请各位前辈见谅~

##关于 gcc 编译过程

实验流程

首先,通过简单的 google 我们知道, C语言的编译过程大致如下:



我们先对最简单的 Hello World 文件进行编译

进入命令行用 gcc -save-temps hello.c -o compilation, 把所有的编译产生的文件保存下来 gcc 的具体编译指令学习到 gcc Hello.c -o Hello.i, 将.c 转换为.i 看 hello.i

```
# 0 "clult-ins"
# 0 "ccommand-lines"
# 1 "hello.c"
# 1 "rello.c"
# 1 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/stdio.h" 1
# 9 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/corecrt_stdio_config.h" 1
# 1 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/corecrt_stdio_config.h" 3
# 1 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/corecrt.h" 1
# 1 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/corecrt.h" 3
# 1 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/_mingw.h" 1
# 10 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/_mingw.h" 3
# 10 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/_mingw.mac.h" 3
# 10 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/_mingw_mac.h" 3
# 384 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/_mingw_mac.h" 3
# 11 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/_mingw.mac.h" 3
# 12 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/_mingw.h" 2 3
# 1 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/_mingw.h" 3
# 12 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/_mingw.h" 3
# 1 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/_mingw.h" 3
# 10 "c:/Program Files/mingw64/x86_64-w64-mingw32/include/_mingw.h" 3
#
```

观察到这一步首先把头文件内容引入了, 貌似还替换了宏常量好奇去 c 盘找到对应的文件

stdio.h 打开

```
#ifndef _INC_STDIO
#define _INC_STDIO
#include <corecrt_stdio_config.h>
#pragma pack(push,_CRT_PACKING)

#pragma push_macro("snprintf")
#undef snprintf
#pragma push_macro("vsnprintf")
#undef vsnprintf
#pragma push_macro("snwprintf")
#undef snwprintf
#pragma push_macro("vsnwprintf")
#undef snwprintf
#ifdef __cplusplus
extern "C" {
#endif

#define BUFSIZ 512
#define _NFILE _NSTREAM_
#define _NSTREAM_ 512
#define _NSTREAM_ 512
#define _IOB_ENTRIES 20
#define _EOF (-1)

#ifndef _FILE_DEFINED
    struct_iobuf {
#ifdef _UCRT
        void *_Placeholder;
#else
        char *_ptr;
        int _cnt;
        char *_base;
        int _flag;
        int _file;
        int _charbuf;
        int _bufsiz;
        char *_tmpfname;
#endif
    };
    typedef struct_iobuf FILE;
#define _FILE_DEFINED
#endif
```

通过搜索了解到#ifndef #define #endif 实现了宏定义并且防止了重复定义, 貌似可以防止重复编译, 好像还 include 了另一个头文件

```
# 3 "hello.c"
int main()
{
    printf("hello world!\n");
    return 0;
}
```

Hello.i 最下面依然出现了原来代码内容,加入注释后编译发现注释被删。 再看 hello.s

```
#include <stdio.h>

int main()

{

printf("hello world!\n")

return 0;

}
```

对代码处理使其语法错误,然后预处理成.i,使用 gcc -E hello.c 让文件内容直接输出

```
# 3 "hello.c"
int main()
{
    printf("hello world!\n")
    return 0;
}
```

错误依然被输出, 然后进行编译, gcc -S hello.c

报错,说明在编译时才会检查语法,弹出报错消息

```
.file "hello.c"
   .text
          __main; .scl 2; .type 32; .endef
   .def
   .section .rdata,"dr"
.LC0:
   .ascii "hello world!\0"
   .text
   .globl main
   .def main; .scl 2; .type 32; .endef
   .seh_proc main
main:
   pushq %rbp
   .seh_pushreg
                 %rbp
   movq %rsp, %rbp
                 %rbp, 0
   .seh_setframe
   subq $32, %rsp
   .seh stackalloc 32
   .seh_endprologu
         __main
   call
          .LCO(%rip), %rax
   leaq
   movq %rax, %rcx
   call
         puts
         $0, %eax
   movl
   addq $32, %rsp
   popq
          %rbp
   ret
   .seh_endproc
   .ident "GCC: (x86_64-posix-seh-rev1, Built by MinGW-Builds project) 13.1.0"
          puts; .scl 2; .type 32; .endef
   .def
```

当然, 汇编代码我是一点不懂/头秃

打开是一堆乱码,不过用 <u>not</u>epad++和 hexeditor 可以查看机器码,hello.o 就是将汇编转化为更低级的机器码,最后还有链接一步,学习得知这步是将库文件对应的定义添加到这个对象文件中,才能实现最终的执行。

##关于 level 1-1 和 level 1-2

一些最基本的 C 语言程序,大概从零开始看了两三个晚上教程就上手去做了,在报错和 bug 中收获了很多要注意的小规范

##关于 level 1-3

这是开始让我比较头大的一关,首先我先浏览了一下各个算法,决定使用最符合直觉也相对高效的 LOOK 算法作为要实现的目标。于是乎我就 STFW, 但是网上能找到的资料和代码实现方案全是基于磁盘读写的,逻辑上跟电梯不符合,而且也没有考虑到超载的情况,于是转战自己独自写代码。

基本思路就是将乘客输入数组后分上下行分组,然后再对上下行分别处理, 电梯跑到最远距离后会返回, 然后反向运行。

期间面对各种 debug,发现了各种各样的小问题,再经过亿点点时间的修 bug 后终于正常了。

复盘发现我写的这套算法有一个问题,就是但同一层有 5 个人及以上请求某一个方向的电梯时,去往距离最远的那个人就上不了电梯只能等下一趟,电梯会空跑到最远距离,不过时间也来不及 fix 了,目前做到的最好就只能这样。

可能的解决方案或许是把上行的乘客输入到电梯乘客的数组里,然后再进行排序取最大楼层,实现"进入电梯按下目的楼层"的效果,(但是其实在这种情况下是不知道哪个人能挤上电梯的)但是时间不足,这种操作在 level2-1 里做了,没时间移植到 level 1-3 里,有点可惜。

##关于 level 2-1

简单参考了一下一篇论文,关于单部电梯的实时调度的。参考论文中给出的方法,通过考虑等待电梯时间和乘坐电梯时间简单定义了一下代价函数,每走一个有需求的楼层计算一次,然后参考其中对中间层和整数层分开处理的方法,试图实现了一下调度算法,但是目前实现的版本有几率出现在两个楼层之间反复横跳的问题,还有在获取代价函数时貌似出现了故障,但是受限于时间无法 fix 了 wwww

(可能跟定义函数时数组的大小没有规定有关)

而且该算法的局限性就是不像 FCFS 和 LOOK 那样直观,容易上下折返,但是总体数据应该更好看。

##关于 level 2-2

没时间写,麻了