11.22

关于cache

- 一道小思考题
- 假设有四个线程，现在要做一个长度为4N的向量加法的程序，有两种方法（假设两个向量分别叫**X**和**Y**）
  - A：第一个线程计算区域1~N，第二个线程计算N+1~2N … 以此类推
  - B：第一个线程计算1,5,9,13…4k+1…; 第二个线程计算2,6,10…4k+2…; 以此类推
- 那种方法会快一些呢？

下半学期要学习的内容

- 链接
  - 程序怎么样正确的找到函数和变量的定义（“definition”）
  - 到底如何生成一个二进制可执行文件
  - 大型项目中，代码如何组织，如何编译
- ECF
  - 进程之间的交流与协作，系统是怎么管理它们的
  - 在shell里键入 ./main 执行程序的时候到底发生了什么？
  - 用Ctrl+C的时候又发生了什么？

下半学期要学习的内容

- I/O
  - 比fopen, fprintf, std::fstream更底层的IO操作
    - 系统怎么知道我的进程打开了哪些文件?
- 虚拟内存
  - 系统怎么给一个进程营造出一种美好的假象
    - “你可以用4T的内存，而且他们全是属于你的”
- 网络和并行
  - 我的机器如何与远端的另一台机器交流?
    - 或者说，我的进程如何与远端的另一个进程交流?
  - 如何利用多线程技术，有什么需要注意的地方
    - Race?

关于链接

- 大家编译一些程序的时候很常见的报错信息
  - Undefined reference to “foo”
  - Multiple definition of “foo”
- 这就是链接错误
- 链接是什么？
  - 这个你们上课说过了

为什么要链接？

- 我写一个hello world，还要自己重新编译一次libc的代码？
- 我希望能把我的程序变成很多功能独立的模块
  - 如果一个很大的项目，修改了一个文件，就要全部重新编译一遍
  - 这是很恐怖的
  - 编译有时很耗时间——如手动编译linux kernel或者Qt 5.4

一些书本上的重要内容

- 各种symbol的分类，以及ELF文件的layout
- 计算relocation entry
  - 见书上对应部分的练习题 —— *考试会考*
- 动态链接和静态链接的区别

如何组织一些大型工程

- 对于C工程
  - .h文件存放变量和函数的声明，.c文件存放变量和函数的定义
  - .c文件中不想被别人访问到的全局变量和helpers加上static修饰符
  - “module”的设计思想，一对.h和.c文件组成了一个模块，提供了一些数据结构和访问修改这些数据结构的函数 —— 封装不必要的细节
- 对于c++工程
  - .hpp文件里存放类的声明，模板类/函数的定义
  - .cpp文件里写非模板类的实现，类中static变量的初始化等
  - 把raw的全局变量封装到类中
  - namespace的使用：防止重名 —— 尽量少用using namespace std
  - stl库很强大，大家有兴趣可以多学学
    - 不仅仅是程设实习里面的那些

如何组织一些大型工程

- 编译脚本的使用
- Makefile
  - 很基本的技能。对于一些简单的工程，手写几行Makefile就可以搭建起框架
  - 需要程序员描述“目标”与“依赖”
- CMake
  - 通过另一种语法描述“目标”与“依赖”
  - 可以自动找一些library或module的位置
  - 自动生成Makefile

如何组织一些大型工程

举个例子:
这是我写数算/数算实习作业的时候用的Makefile
效果大概是每次新写一个程序就可以直接make来编译

```
CXX = g++
FLAG = -std=c++11 -g

SRC  = ${wildcard *.cpp}
EXEC = ${patsubst %.cpp, %, ${SRC}}

.PHONY: clean

all: ${EXEC}

%: %.cpp
    ${CXX} $^ -o $@ ${FLAG}

clean:
    rm -f ${EXEC}
```

如何组织一些大型工程

- 在从零开始写程序之前，就要想好自己的代码框架大概长什么样
- 搭建好了很简单、什么功能都没有的代码框架后，把编译脚本写出来
  - 让自己的工程时刻处于可以编译的状态
  - 这样，新加入一个功能，就可以立即测试
- 某些测试工具的使用，比如gTest等（一般用不到）