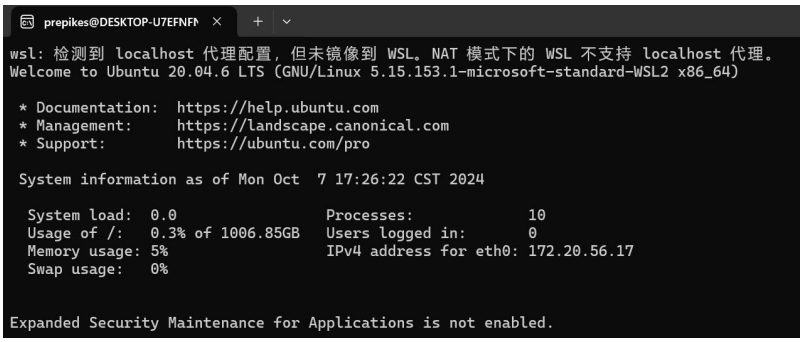
0.运行准备

我这里用的是 Ubuntu On Windows。

不知道为什么，这个自动检测设置必须要关掉。



如果不关的话，

<- Ubuntu 会这样

`lmbench` 程序虽然能运行，但是检测不出任何内容。

以及，直接执行 `make` 编译后将会看到类似有 `Permission denied` 等一系列错误。
所以输入更改文件的权限的指令，然后就可以成功运行了。

1.测试过程

执行 make results，执行后需要设置的有：

- **MULTIPLE COPIES**： 同时运行并行测试，对应生成结果中的 `scal load` 项；由于在并行运行多个副本时，系统资源的竞争会导致性能波动，从而使结果不稳定，所以这里 `default 1`。
- **Job placement selection**: 作业调度控制方法，当然选 1，允许作业调度；
- **Options to control job placement**: 默认选 1；
- **Memory**: 设置为大于 4 倍的 `cache size`，该值越大结果越精确，同时运行时间越长；
- **SUBSET**: 默认选 all；
- **FASTMEM**：选择 no，因为快速内存延迟测试不够精确；
- **SLOWFS**：肯定是不跳过，选择 no；
- **DISKS**: 默认 (但是用 lsblk 命令查到的 sdc 磁盘挂载目录`/mnt/wslg/distro` 会提醒没有 root，就算在 root 模式下也是这样，原因未知)
- **REMOTE**：none

到这里其实就已经设置完了，然后是对于结果输出的一些选项

- **FSDIR**：存储测试文件，用 default 的 /var/tmp
- **Status output file**：处理输出状态信息，用 default 的 /dev/tty
- **Mail results** ：不 Mail，谢谢

然后，这个程序居然跑了两个小时。
利用 `make see` 写入，然后去 `summary.out` 查看运行结果。

2.测试结果

1.系统结果信息

> “

```
Basic system parameters
----------------------------------------------------------------------------------------------
Host                 OS Description              Mhz  tlb      cache   mem    scal
                                                     pages     line    par    load
                                                               bytes
--------- ------------- ----------------------- ---- ----- ----- ------ ----------------------------
DESKTOP-U Linux 5.15.15       x86_64-linux-gnu 4382           64               1
DESKTOP-U Linux 5.15.15       x86_64-linux-gnu 4111    52     64    5.9600     1
```

这里说的是基本系统参数:

- **Host**: 主机名
- **Linux 5.15.15**: 操作系统的名称
- **x86_64-linux-gnu**：版本系统架构
- **Mhz**: 处理器的主频，我的两个处理器频率分别为：4382 MHz，4111 MHz
- **tlb pages**: TLB（Translation Lookaside Buffer，翻译后备页表）中的页数
- **cache line bytes**: 缓存行的大小，以字节为单位
- **Mem par**: 存储器分层并行化，描述了在不同层次的存储器（如缓存和主内存）之间同时处理多个内存请求的能力
- **scal**: 同时运行 lmbench 个数

2.处理器性能

> “

```
Processor, Processes - times in microseconds - smaller is better
----------------------------------------------------------------------------
Host                 OS  Mhz  null  null      open slct  sig  sig  fork exec sh
                             call  I/O stat clos TCP  inst hndl proc proc proc
--------- ------------- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
DESKTOP-U Linux 5.15.15 4382 0.05 0.07 0.23 0.49 1.14 0.10 0.53 100. 255. 743.
DESKTOP-U Linux 5.15.15 4111 0.08 0.10 0.40 1.12 1.62 0.16 0.91 340. 680. 1631
```

主要描述是处理器与进程的性能，单位 us 且越小越好，因为有两个处理器所以输出了两行:

- **null call**: 执行 getppid 需要的时间；
- **null I/O**: 从 `/dev/zero` 读取一个字节的时长 t1，写一个字节到 /dev/null 的时长 t2，t1、t2 取平均值即为该项结果；
- **stat**: stat 一个文件（即得到一个文件的信息）所需时长；
- **open clos**: open 一个文件接着再 close 掉该文件一共所用时间（不包含读目录和节点的时间）；
- **slct TCP**: 通过 TCP 网络连接选择 100 个文件描述符所消耗的时间；
- **sig inst**:`install signal` 所耗时长；
- **sig hndl**: `handler signal` 所耗时长；
- **fork proc**: fork 一个完全相同的 process，并把原来的 process 关掉一共所消耗的时间；
- **exec proc**: 模拟一个 shell 进程的工作过程：fork 一个新进程执行新命令消耗的时间。
- **sh proc**: fork 一个进程，同时询问系统 shell 来找到并运行一个新程序所用时间。

3.数学运算

然后下面是数学运算：

（1）整型运算

“

```
Basic integer operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS  intgr intgr  intgr  intgr  intgr
                         bit   add    mul    div    mod
--------- ------------- ------ ------ ------ ------ ------
DESKTOP-U Linux 5.15.15 0.1600 0.2300 0.7000 2.5400 3.7800
DESKTOP-U Linux 5.15.15 0.2900 0.4200 1.5700 4.9300 5.9200
```

- **intgr bit**: 整数位数，这里通常表示为操作的位宽（如 32 位或 64 位）。
- **add**: 整数加法操作的时间。表示对两个整数进行相加的时间。
- **mul**: 整数乘法操作的时间。表示对两个整数进行相乘的时间。
- **div**: 整数除法操作的时间。表示对两个整数进行相除的时间。
- **mod**: 整数取模操作的时间。表示对两个整数进行取余的时间。

（2）无符号整型运算

“

```
Basic uint64 operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS int64  int64  int64  int64  int64
                        bit    add    mul    div    mod
--------- ------------- ------ ------ ------ ------ ------
DESKTOP-U Linux 5.15.15  0.160         0.7000 3.4500 4.4600
DESKTOP-U Linux 5.15.15  0.250         0.9700 5.0700 7.3800
```

(同上，略)

（3）浮点型运算

“

```
Basic float operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
```

```
Host                 OS  float  float  float  float
                         add    mul    div    bogo
--------- ------------- ------ ------ ------ ------
DESKTOP-U Linux 5.15.15 0.4600 0.9300 2.6300 0.7400
DESKTOP-U Linux 5.15.15 0.7300 1.4900 4.0200 1.1100
```

- **add**: 浮点加法操作的时间。表示对两个浮点数进行相加的时间。
- **mul**: 浮点乘法操作的时间。表示对两个浮点数进行相乘的时间。
- **div**: 浮点除法操作的时间。表示对两个浮点数进行相除的时间。
- **bogo**: 这是一个“虚假”或“无效”计算

（4）双精度浮点型运算

“

```
Basic double operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS  double double double double
                         add    mul    div    bogo
--------- ------------- ------  ------ ------ ------
DESKTOP-U Linux 5.15.15 0.6200 0.9300 3.3900 0.9900
DESKTOP-U Linux 5.15.15 0.6600 1.2800 4.5100 1.4600
```

(同上，略)

4.上下文切换

如下输出结果单位均为 us，数值越小表示性能越好。

“

```
Context switching - times in microseconds - smaller is better
-------------------------------------------------------------------------
Host                 OS  2p/0K 2p/16K 2p/64K 8p/16K 8p/64K 16p/16K 16p/64K
                         ctxsw  ctxsw  ctxsw ctxsw  ctxsw   ctxsw   ctxsw
--------- ------------- ------ ------ ------ ------ ------ ------- -------
DESKTOP-U Linux 5.15.15
DESKTOP-U Linux 5.15.15   19.6   17.7   19.1   22.2   20.4    21.6    21.0
```

- 多个进程用 unix pipe 环连接起来，每个进程从自己的管道中读取 token，执行任务，然后将 token 写给下一个进程。
- context swithing 时间包括: 切换进程的时间, 加上恢复进程所有状态所用的时间 (包含恢复 cache 状态)。
- 2p/0k: 每个进程的 size 为 0（不执行任何任务），进程数为 2 时上下文切换所消耗的时间;
- 2p/16k: 每个进程 size 为 16K（执行任务），进程数为 2 时上下文切换所消耗的时间;

之后的测试项以此类推。

5.本地通讯时延

如下输出结果单位均为 us，数值越小表示性能越好。

“

```
-------------------------------------------------------------------
```

```
Host                 OS 2p/0K  Pipe AF     UDP  RPC/   TCP  RPC/ TCP
                        ctxsw       UNIX         UDP         TCP conn
--------- ------------- ----- ----- ---- ----- ----- ----- ----- ----
DESKTOP-U Linux 5.15.15        28.9 33.2
DESKTOP-U Linux 5.15.15  19.6  76.5 128.  86.0        95.4        91.
```

- **2p/0k**: 每个进程的 size 为 0（不执行任何任务），进程数为 2 时上下文切换所消耗的时间；
- **Pipe**: 即所谓的 hot potato 测试，两个没有具体任务的进程之间使用 pipe 通信，一个 token 在两个进程间来回传递，传递一个来回所消耗时长的平均值；
- **AF UNIX**: 同 Pipe 测试项，但进程间通信使用的是 socket 通信；
- **UDP**: 同 Pipe 测试项，但进程间通信使用的是 UDP/IP 通信；
- **RPC/UDP**: 同 Pipe 测试项，但进程间通信使用的是 sun RPC 通信，默认情况下，RPC 采用 UDP 协议传输；
- **TCP**: 同 Pipe 测试项，但进程间通信使用的是 TCP/IP 通信；
- **RPC/TCP**: 同 Pipe 测试项，但进程间通信使用的是 sun RPC 通信，指定 RPC 采用 TCP 协议传输；
- **TCP conn**: 创建 socket 描述符和建立连接所用时间。

6.远程通讯时延

“

```
*Remote* Communication latencies in microseconds - smaller is better
---------------------------------------------------------------------
Host                 OS   UDP  RPC/  TCP   RPC/ TCP
                          UDP        TCP  conn
--------- ------------- ----- ----- ----- ----- ----
DESKTOP-U Linux 5.15.15
DESKTOP-U Linux 5.15.15
```

（这个没选）

7.文件、内存延迟

如下输出结果单位均为 us，数值越小表示性能越好。

“

```
File & VM system latencies in microseconds - smaller is better
-------------------------------------------------------------------------------
Host                 OS   0K File      10K File     Mmap    Prot   Page   100fd
                        Create Delete Create Delete Latency Fault  Fault  selct
--------- ------------- ------ ------ ------ ------ ------- ----- ------- -----
DESKTOP-U Linux 5.15.15                                    0.330         0.441
DESKTOP-U Linux 5.15.15 7.8080 8.0556   18.5   19.6  7232.0 0.616 0.05010 0.791
```

- **0K File Create**: 0K 文件创建所用时间；
- **0K File Delete**: 0K 文件删除所用时间；
- **10K File Create**: 10K 文件创建所用时间；
- **10K File Delete**: 10K 文件删除所用时间；
- **Mmap Latency**: 将指定文件的开头 n 个字节 mmap 到内存，然后 unmap，并记录每次 mmap 和 unmap 共消耗的时间，去每次消耗时间的最大值；
- **Port Fault**: 保护页延时时间；
- **Page Faule**: 缺页延时时间；

- **100fd selct**: 对 100 个文件描述符配置 select 的时间。

8.本地通信带宽

如下输出结果单位均为 MB/s，数值越大表示性能越好。

“

```
*Local* Communication bandwidths in MB/s - bigger is better
----------------------------------------------------------------------------
Host                OS  Pipe AF    TCP  File   Mmap  Bcopy  Bcopy  Mem   Mem
                             UNIX      reread reread (libc) (hand) read write
--------- ------------- ---- ---- ---- ------ ------ ------ ------ ---- -----
DESKTOP-U Linux 5.15.15
DESKTOP-U Linux 5.15.15 2830 4605 3843 9519.3  17.4K  10.4K 7516.0 15.K 10.9K
```

- **Pipe**: 在两个进程建立 pipe，pipe 的每个 chunk 为 64K，通过该管道移动 50MB 数据所消耗的时间；
- **AF UNIX**: 两个进程之间建立 unix stream socket 连接，每个 chunk 为 64K，通过该 socket 传输 10MB 数据所用的时间；
- **TCP**: 同 Pipe 测试项，但进程间使用 TCP/IP socket 通信，传输数据量为 3MB；
- **File reread**: 读文件并将其汇总一起所用的时间；
- **Mmap reread**: 将文件 mmap 到内存中，从内存中读文件并将其汇总一起所用时间；
- **Bcopy(libc)**: do bw_mem $i bcopy，从指定内存区域拷贝指定数量的字节内容到另一个指定内存区域的速度；
- **Bcopy(hand)**: do bw_mem %i fcp，把数据从磁盘的一个位置拷贝到另一个位置所用的时间；
- **Mem read**: bw_mem $i frd，累加数组中的整数值，测试把数据读入 processor 的带宽；
- **Mem write**: do bw_mem $i fwr，把整数数组的每个成员设置为 1，测试写数据到内存的带宽。

9.内存操作延迟

如下输出结果单位均为 ns，数值越小表示性能越好。

“

```
Memory latencies in nanoseconds - smaller is better
    (WARNING - may not be correct, check graphs)
------------------------------------------------------------------------------
Host                 OS   Mhz   L1 $   L2 $    Main mem    Rand mem    Guesses
--------- -------------   ---   ----   ----    --------    --------    -------
DESKTOP-U Linux 5.15.15  4382     -      -           -    Bad mhz?
DESKTOP-U Linux 5.15.15  4111 1.1750 2.2900    8.497000      170.5
```

本地测试执行 lat_mem_rd，将整数数组中每第 4 个元素的值累加起来；测试的是读数据到 processor 的带宽。

- **L1**：缓存 1
- **L2**：缓存 2
- **Main Mem**：连续内存
- **Rand Mem**：内存随机访问延时
- **Guesses**: 假如 L1 和 L2 近似，会显示 “No L1 cache?”，假如 L2 和 Main Mem 近似，会显示 “No L2 cache?”。

3.代码测试

1. 三个有系统调用的测量项

1.1 Open-Clos 时间测量

测量一个文件打开并关闭的时间，用到了 `open ()` 以及 `close ()` 的系统调用，（Imbench 测量的）一次开启关闭花费了 1.12us。

1.2 OK-File Delete 时间测量

删除一个 0kb 文件用时，用到了 `unlink ()` 系统调用，（Imbench 测量的）一次开启关闭花费了 8.0556us。

1.3 Mmap Reread 时间测量

文件映射到内存中，并读文件并将其汇总的用时（其实是速度），主要用到了 `mmap ()` 系统调用，（Imbench 测量的）速度为 17.4KMB/S。

2. python 代码

1.1 时间测量

“

```
start = time.perf_counter()
result = func(*args, **kwargs)
end = time.perf_counter()
```

可以用这种方式对时间进行测量，只需要把想测量的函数传递给 `func`，`func` 就会把传入的函数运行一遍，然后上下时间相减，就可以得到函数运行的时间。

1.2 函数编写

首先是 `open_close` 函数，先创建文件，再开始计时，用 open 函数把文件用只读模式打开再关闭即可。
然后是测 `delete`，先创建文件，再计时，然后用 `unlink` 函数删除。
最后是 `mmap` 时间测量，先创建一个文件，然后往文件里面写入 1024 个 1，开始计时，然后打开文件，内存映射，内存汇总（计算字节和），最后关闭内存映射。如果说要得知速度，计算一下即可。
（为了防止数据被编码和解码的问题，需要在二进制模式下打开，写入以及关闭）

1.3 结果

由于时间一直在波动，于是使用 for 循环 100 次取平均值。运行结果如下：

“

```
Average open close time: 0.0000694950 seconds
Average 0K File Delete time: 0.0002353170 seconds
Average Mmap Latency time: 0.0001509280 seconds
```

可以看到，这三项和 Lmbench 的差距是 10 的三次方，后一项速度仅有 2.12MB/s，误差稍微有点大了吧。我一开始有点怀疑是 python 的问题，因为 python 是在 PVM 上运行的，也许用 C++测量更加的精确一点。

3. C++代码

1.1 时间测量

使用了适用于 Linux 和 Windows 的<chrono>，不过所需的 C++版本比较高（11 以上）。`<chrono>`库有`high_resolution_clock`，该时钟使用的是最高分辨率的时钟，测量精度应该是够的。

1.2 函数编写

C++里面有`CreatFile`函数可以用来创建和打开一个文件，然后可以用`CloseHandle`函数可以用来关闭一个文件。但是`CreatFile`函数用来创建并打开肯定是会花费多余的时间的，所以要先用`creat all`模式运行一下，然后用只读模式运行，这样可以避免花费创建的时间，不过`creatfile`的只读模式有没有做出什么多余的操作。

`FileDelete`函数也是同理，只是不能把`FileCreate`的时间也计入其中。

`Mmap reread`：将文件`mmap`到内存中，从内存中读文件并将其汇总一起所用时间；

但是我并不知道`lmbench`他的从文件里读了多少内存，所以说我暂且认为是 1kb，到时候直接算速度就行了。为了做到这点，我需要先创建一个文件（在磁盘），然后只写入一个“1”（在内存里操作），在函数结束前会把文件重新放入磁盘。

然后是`measure_mmap_latency()`函数。先记录开始时间，然后打开文件，获取文件大小，然后文件映射到地址，最后读取文件内容并累加，解除映射，关闭计时。

1.3 结果

看起来错怪 python 了，运行时间依旧和 Lmbench 差了很多。

“

```
open/close time: 980us
File delete time: 332us
Sum: 49, mmap read and sum time: 335us
```

这里我也懒得算了，差距和 lmbench 太大了。

4. 结果分析

1.1 误差分析

既然什么语言都会用这么长的时间，那么问题大头肯定不是冗余逻辑或者说是语言问题之类的。之后又在 WSL 上运行了一下 py 代码，发现误差要小非常多。所以 Windows 肯定是没有全力运行我的代码，而且我好像也没有什么办法让他全力干这个事情，所以只能用 Ubuntu 运行我的 py 代码试试。

首先 CPU 利用率会对实验的结果会有一定的影响，然后我再试了一下插拔电池（最大处理器状态 10%）也会对实验有一定影响。

首先是一部分进程未被关闭：

“

```
Average open close time: 0.0000030650 seconds
Average 0K File Delete time: 0.0000066237 seconds
Average Mmap Latency time: 0.0000310329 seconds
```

然后是把能关的进程全部关掉：

“

```
Average open close time: 0.0000023120 seconds
Average 0K File Delete time: 0.0000065934 seconds
Average Mmap Latency time: 0.0000308134 seconds
```

最后是不插电:

“

```
Average open close time: 0.0000098666 seconds
Average 0K File Delete time: 0.0000194906 seconds
Average Mmap Latency time: 0.0000650447 seconds
```

就算是多次运行取平均值，这些时间依旧不是稳定的，甚至波动还很大，大概在±25%左右，上面取的是多次运行后的比较居中的值。

可以看到，如果说取最大处理器 100%以及进程基本全部关闭的状态，第一项与 1us 相比还是大了一点，第二项相比 8us 小了一点，重点是第三项，只有 15MB/s,在检查了一遍后，发现可能是这个问题：

“

```
# 从内存中读取内容并计算字节和
total_sum = sum(mm[i] for i in range(size))
```

我把这段代码去掉后，他直接飙到了 80kMB/s,当然这是有点赖皮的，因为这仅仅是映射而没有汇总。但是在 lmbench 里，汇总的代码应该不会像 python 这段代码的逻辑一样冗余。

然后我用 C++代码在 WSL 跑了一下，

“

```
mmap read and sum time: 5545us
```

第三项是 180MB/s，可能 C++这里的代码逻辑比 py 效率更高，但是还是没好到哪里，去可能 lmench 的汇总

不是读多少数字，sum 就++多少的方式，而是其他更加省时间的方式。

1.2 结果总结

Open-clos：2.3us vs 1.12us
Delete：6.59us vs 8.0556us
Mmap Reread：180MB/s vs 15kMB/s

至于为什么会有这种结果，我认为可能的原因有：

- **不同操作环境**

  在不同的系统运行相同的代码，得到的结果完全不同，就我看来，`Windows` 系统更划水一点，不会全力运行代码，`WSL` 做得更好一点，但是我想 `Linux` 虚拟机或者直接在 `Linux` 操作系统上运行可能和 `lmbench` 的误差更小一点。

- **系统负载和 CPU 利用率**

  不同的处理器利用率以及不同的系统负载会比较明显的影响他运行的速度，这里我上面也试过了。

- **偶然波动**

  不知道为什么，就算代码是多次运行取平均值，他还是会各种波动，这个的原因我也无能为力，但是会影响实验结果是真的。这也可能是 `Delete` 比 `Lmbench` 小的最大原因。

- **代码逻辑**

  `Lmbench` 作为比较基准的测试工具，代码逻辑和计时逻辑应该是要比我高明得多的。就比如汇总那里，他用的应该不是 `sum++`这种？而是另外一些更加高明的方法。`Open-clos` 和 `Mmap Reread` 是稳定比 `Lmbench` 效率低的，特别是 `Mmap Reread`。