

# NVMLab

---

**ver. 0.0.5-20210510**

## 实验介绍

---

本次试验在 [ics.men.ci](https://ics.men.ci) 上进行，编写相关代码片段实现持久化 KV。  
提交需要使用 C++17 nvmlab 语言。

## 实验目的

---

使用 pmdk (libpmem 或 libpmemobj) 实现持久化KV，其中 key value 均由小写字母组成，key 16字节，value 128字节。

需要支持以下操作：

- SET key value
- GET key value
  - 若不存在返回 “-”（不含引号）
- NEXT key
  - 返回 key 的后继
  - 不存在则返回 “-”（不含引号）

需要支持以下特性：

- SET 为原子操作  
完成或者失败，不存在中间状态

## 实验步骤

---

你需要编写 mian.cpp 中的 void mian(std::vector<std::string> args) args 为参数列表，保证有且仅有一个元素，为 pmdk 储存文件路径（目前使用 ~~ramdisk~~ 模拟 现在是 NVM 了）。

在进行相关初始化后，你应当调用 Query nextQuery() 函数获取下一个应当进行的操作。

```
struct Query
{
    enum Operation
    {
        UNKNOWN,
        SET,
        GET,
        NEXT
    };

    Operation type;

    std::string key;
    std::string value;

    std::function<void(std::string)> callback;
};
```

key 和 value 的含义见实验目的。

对于 GET 和 NEXT 操作，你应当调用 callback 传回结果。

若应当调用 callback 却没调用，则下次 nextQuery 时程序出现偏差。

我们假定对于 SET 操作，当你下次调用 nextQuery 时该操作已经 commit。

在 SET 操作时，你的程序可能会退出。

## 离线测试指北

### 安装依赖

在服务器上，你不需要执行此步骤

以 ubuntu 为例，安装 libpmem-dev libpmemobj-dev

```
sudo apt install -y libpmem-dev libpmemobj-dev build-essential
```

## 配置 RamDisk

在服务器上，你不需要执行此步骤

在本地，由于使用 SSD 模拟 pmem 过于缓慢，所以使用 RamDisk 模拟 pmem

以创建在 /ramdisk 为例

```
sudo mkdir /ramdisk
sudo mount -t tmpfs -o size=512m tmpfs /ramdisk
sudo chmod 1777 /ramdisk
```

## 配置 NVMLab

先使用 cp 或 scp 或 rsync 将服务器上 /home/nvmlab 下最新版的 nvmlab-xxx.tar 拷贝至你喜欢的地方

然后使用 tar xvf nvmlab-xxx.tar 解压

之后编辑 Makefile

```
TARGET_DIR = /ramdisk
TARGET_NAME = /pool
```

将 TARGET\_DIR 改为目标目录名称，在服务器上 是 /mnt/pmem0 或 /mnt/pmem1，在本地是 RamDisk 的位置

将 TARGET\_NAME 改为一个你喜欢的名字，以 / 开头，在服务器上注意不要和别人冲突。

## 进行测试

首先先编译

```
make
```

之后进行测试，数据分为六组，每组两个，这里以第一组为例

```
$ make run < data/10000_1_in # 纯写测试
1 0.0759717 ae2fb82f
$ make run < data/10000_1_out # 纯读测试
0 0.0324022 6886687e
```

输出依次为 正确性，时间，校验码。

## 实验细节

---

### 评测流程

你的程序会连续评测6个测试点，在OJ上每个测试点分为两个连续的小测试点，一共12个。

测试点编号	数据量	操作	附加条件
1	~10000	SET	清空储存文件
2	1000	GET NEXT	
3	~10000	SET	清空储存文件
4	1000	GET NEXT	
5	~100000	SET	清空储存文件
6	1000	GET NEXT	
7	~100000	SET	清空储存文件
8	1000	GET NEXT	
9	~1000000	SET	清空储存文件
10	1000	GET NEXT	

11	~1000000	SET	清空储存文件
12	1000	GET NEXT	

在奇数测试点运行到某些阶段（详见数据），你的程序可能会中断并退出。在下一个偶数测试点，会有 GET 和 NEXT 请求检查你的程序是否正确。

## 中断方式

评测 hook 了 `pmem_persist` 和 `pmem_msync`，并可能在合适的时候退出（详见源码）。

PMDK 的函数都会调用这两个函数，一般而言你不需要直接调用它们。

## 判分方式（beta）

在下发的文件里有个假的 `mian.c`，它有正确的行为但并没有使用 PMDK api。它的效率定为 150分，线性增长，和 100 取 min（待定）。

~~因为你们的助教用 libpmem 搓了一个发现和那玩意效率差不多所以基准就不变了，计算公式可能变化~~

**注意：该方法在 OJ 上将不起作用，且不使用 PMDK api 实现将极大影响助教的心情。**