并行与分布式作业

LLVM检测

第三次作业

姓名：何泽

班级：18级计科（超算方向）

学号：18340052

一、问题描述

> 利用 `LLVM` （ `C` 、 `C++` ）或者 `Soot` （ `Java` ）等工具检测多线程程序中潜在的数据竞争以及是否存在不可重入函数，给出案例程序并提交分析报告。

二、解决方案

使用老师给出的参考，即 `ThreadSanitizer` ，首先由LLVM将程序转为 `IR code` ，然后使用 `Clang` 编译器结合 `ThreadSanitizer` 自动生成分析，由此便可得到结果。

三、实验结果

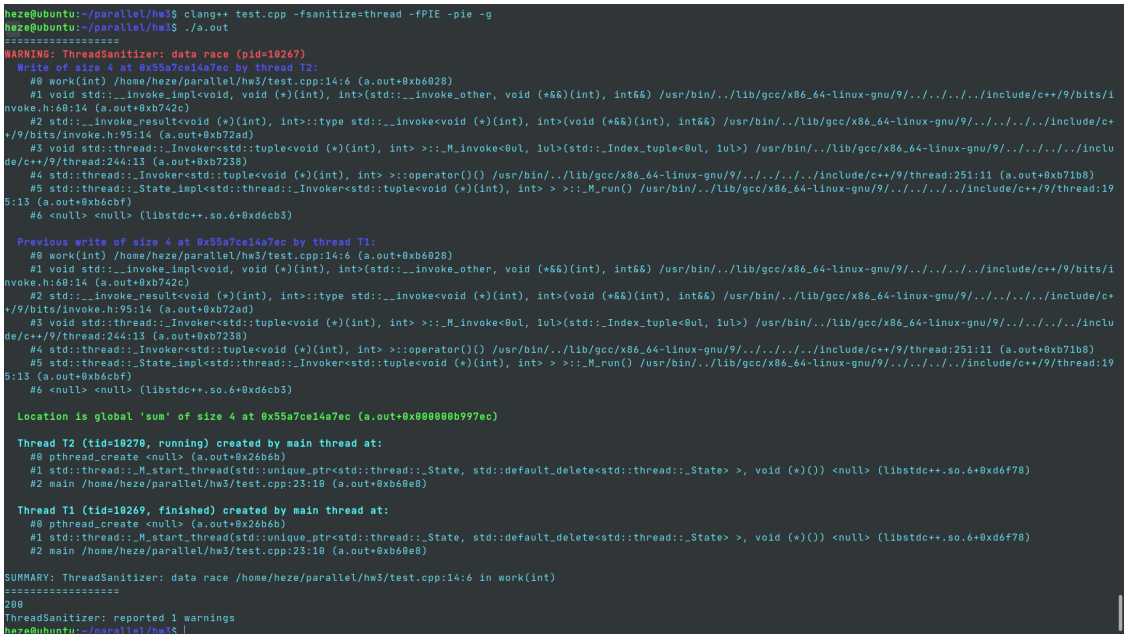
1.对全局变量的访问

```
#include <iostream>
#include <thread>
#include <mutex>
using namespace std;

int sum = 0;

void work(int index)
{
    for (int i = 0; i < 100; i++) {
        sum++;
    }
}

int main()
{
    thread t[tCount];
    for (int n = 0; n < 2; n++){
        t[n] = thread(work, n);
    }
    for (int n = 0; n < 2; n++){
        t[n].join();
    }
    cout << sum << endl;
    return 0;
}

```

- 上面的程序声明了全局变量 `sum` ，并且使用了2个线程分别对全局变量递增，因为对全局变量进行访问，所以 `work` 这个函数是不可重入函数.
- 使用以下命令编译

```
clang++ test.cpp -fsanitize=thread -fPIE -pie -g
```

- 可得分析结果：



- 首先就分析出有 `data race` ，之后指明了全局变量和两个线程的信息，最后指明有不可重入函数。

2.加锁

对于上面的程序，如果在线程函数访问全局变量前后加锁，函数如下：

```
mutex m;

void work(int index)
{
    for (int i = 0; i < 100; i++) {
        m.lock();       // 加锁
        sum++;
        m.unlock();
    }
}
```

那么就不会有数据冲突，运行结果如下：

```
heze@ubuntu:~/parallel/hw3$ clang++ test.cpp -fsanitize=thread -fPIE -pie -g
heze@ubuntu:~/parallel/hw3$ ./a.out
200
```

可见已没有警告，没有了数据冲突。

四、遇到的问题及解决方法

这次作业遇到的主要问题就是LLVM的概念以及不可重入的概念和消除方法。