并行与分布式作业

“作业名字”

第三次作业

姓名：TRY

班级：18 级计科

学号：

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

1、检测数据竞争

利用 LLVM 检测是否存在数据竞争，可根据老师提供的链接里面的方法实现。

ThreadSanitizer(又名 TSan)是 C/C++的数据竞争检测器。数据争用是并发系统中最常见，最难调试的错误类型之一。当两个线程同时访问同一变量并且至少其中一个访问被写入时，发生数据争用。C ++ 11 标准正式禁止将数据争用作为未定义的行为。

LLVM(Low level machine)不仅仅是一个编译器，同时提供了模块化的功能和库，用于编译器的开发和功能扩展。常规的一个编译器分为前端、优化器和后端，LLVM 编译器也不例外，Clang 就是属于一个编译器的前端部分，LLVM 属于优化器和后端，当然 LLVM 也可以支持其他类型的前端，比如 GCC 前端编译器。LLVM-Clang 就是使用 Clang 前端的 LLVM 编译器。

具体操作步骤如下：

（1）在 linux 操作系统下安装 clang 和 llvm

```
sudo apt-get install clang
sudo apt-get install llvm
```

（2）创建 simple_race 文件

```
shelly@shelly-virtual-machine:~$ gedit simple_race.cc
shelly@shelly-virtual-machine:~$ cat simple_race.cc
#include <pthread.h>
#include <stdio.h>

int Global;

void *Thread1(void *x) {
  Global++;
  return NULL;
}

void *Thread2(void *x) {
  Global--;
  return NULL;
}

int main() {
  pthread_t t[2];
  pthread_create(&t[0], NULL, Thread1, NULL);
  pthread_create(&t[1], NULL, Thread2, NULL);
  pthread_join(t[0], NULL);
  pthread_join(t[1], NULL);
}
```

注意：cat 命令用于连接文件并打印到标准输出设备上。

（3）编译程序，并与-fsanitize = thread 链接，系统会输出数据竞争的 warning

```
shelly@shelly-virtual-machine:~$ clang++ simple_race.cc -fsanitize=thread -fPIE
-pie -g
shelly@shelly-virtual-machine:~$ ./a.out
==================
WARNING: ThreadSanitizer: data race (pid=8209)
  Write of size 4 at 0x56189e5a4d88 by thread T2:
    #0 Thread2(void*) /home/shelly/simple_race.cc:12:9 (a.out+0xbcbda)

  Previous write of size 4 at 0x56189e5a4d88 by thread T1:
    #0 Thread1(void*) /home/shelly/simple_race.cc:7:9 (a.out+0xbcb7a)

  Location is global 'Global' of size 4 at 0x56189e5a4d88 (a.out+0x00000114fd88)

  Thread T2 (tid=8212, running) created by main thread at:
    #0 pthread_create <null> (a.out+0x28446)
    #1 main /home/shelly/simple_race.cc:19:3 (a.out+0xbcc4e)

  Thread T1 (tid=8211, finished) created by main thread at:
    #0 pthread_create <null> (a.out+0x28446)
    #1 main /home/shelly/simple_race.cc:18:3 (a.out+0xbcc33)

SUMMARY: ThreadSanitizer: data race /home/shelly/simple_race.cc:12:9 in Thread2(
void*)
==================
ThreadSanitizer: reported 1 warnings
```

注意：使用-g 在警告消息中获取文件名和行号。

2、检测是否存在不可重入函数

使用 LLVM 将”.cpp”文件 LLVM IR 中间文件（文本形式），然后再编程序来对.11 形式的 IR 中间文件进行自动检测。

具体步骤如下：

（1）LLVM 处理”.cpp”文件为文本形式的 IR 文件”.11”

```
shelly@shelly-virtual-machine:~$ clang -O3 -emit-llvm test.cpp -S -o test.ll
```

（2）编写程序，进行自动检测

打开”test.11”文件后，观察 IR 语法。

```
@Global = common global i32 0, align 4

; Function Attrs: norecurse nounwind uwtable
define noalias i8* @Thread1(i8* nocapture readnone %x) #0 {
  %1 = load i32, i32* @Global, align 4, !tbaa !1
  %2 = add nsw i32 %1, 1
  store i32 %2, i32* @Global, align 4, !tbaa !1
  ret i8* null
}

; Function Attrs: norecurse nounwind uwtable
define noalias i8* @Thread2(i8* nocapture readnone %x) #0 {
  %1 = load i32, i32* @Global, align 4, !tbaa !1
  %2 = add nsw i32 %1, -1
  store i32 %2, i32* @Global, align 4, !tbaa !1
  ret i8* null
}
```

```
; Function Attrs: nounwind uwtable
define i32 @main() #1 {
  %t = alloca [2 x i64], align 16
  %1 = bitcast [2 x i64]* %t to i8*
  call void @llvm.lifetime.start(i64 16, i8* %1) #5
  %2 = getelementptr inbounds [2 x i64], [2 x i64]* %t, i64 0, i64 0
  %3 = call i32 @pthread_create(i64* %2, %union.pthread_attr_t* null, i8* (i8*)* nonnull @Thread1, i8* null) #5
  %4 = getelementptr inbounds [2 x i64], [2 x i64]* %t, i64 0, i64 1
  %5 = call i32 @pthread_create(i64* %4, %union.pthread_attr_t* null, i8* (i8*)* nonnull @Thread2, i8* null) #5
  %6 = load i64, i64* %2, align 16, !tbaa !5
  %7 = call i32 @pthread_join(i64 %6, i8** null) #5
  %8 = load i64, i64* %4, align 8, !tbaa !5
  %9 = call i32 @pthread_join(i64 %8, i8** null) #5
  call void @llvm.lifetime.end(i64 16, i8* %1) #5
  ret i32 0
}

; Function Attrs: norecurse nounwind uwtable
define i32 @main() local_unnamed_addr #3 {
  %1 = tail call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5 x i8], [5 x i8]* @.str, i64 0, i64 0))
  ret i32 0
}
```

经过观察，可以发现.ll 文件的几个特点：（分别对应可重入函数特点）

- 全局变量是用”@”开头的，如上图中的“@Global”变量。
- 函数名是用“@”开头的，如上图的“@printf”，“@pthread_create”。
- 函数定义是用“define”开头的。
- 函数的调用是通过“call”和“@函数名”实现的。

而针对于“可重入函数”的特点，可以通过编写以下程序实现自动检索：

```
if (file)
{
    bool inside = false;//是否进入函数
    while (getline(file, line))
    {
        if (inside == false)//不在函数中
        {
            if (line.find("define") != line.npos)
                inside = true;
        }
        else//已经进入函数中
        {
            if (line == "}")//到达函数尾部
                inside = false;
            else if(line.find("call")!=line.npos)//调用了下面这些函数
            {
                if (line.find("@printf") != line.npos || line.find("@scanf")!=line.npos ||line.find("@puts") != line.npos ||
                    line.find("@gets") != line.npos ||line.find("@malloc") != line.npos || line.find("@new") != line.npos ||
                    line.find("@free") != line.npos)
                {
                    reentrant = false;
                }
            }
            else//没有调入输入输出、开辟空间等函数,则判断有无全局变量
            {
                if (line.find("@")!=line.npos)
                    reentrant = false;
            }
        }
    }
    if (reentrant)
        cout << "程序中不存在不可重入的函数" << endl;
    else
        cout<< "程序中存在不可重入的函数" << endl;
```

注 1：这里可以使用 string.find（）函数来实现查找是否含特定函数的功能。

注 2：此处检索的是文件里面的所有函数（包括 main 函数），因为题目要求只是说“多

线程程序中是否含有不可重入函数”，并没有强调是线程函数，所以此处对文件中所有的函数都进行了检索。

三、 实验结果

1、检测数据竞争

```
WARNING: ThreadSanitizer: data race (pid=8209)
  Write of size 4 at 0x56189e5a4d88 by thread T2:
    #0 Thread2(void*) /home/shelly/simple_race.cc:12:9 (a.out+0xbcbda)

  Previous write of size 4 at 0x56189e5a4d88 by thread T1:
    #0 Thread1(void*) /home/shelly/simple_race.cc:7:9 (a.out+0xbcb7a)

  Location is global 'Global' of size 4 at 0x56189e5a4d88 (a.out+0x00000114fd88)

  Thread T2 (tid=8212, running) created by main thread at:
    #0 pthread_create <null> (a.out+0x28446)
    #1 main /home/shelly/simple_race.cc:19:3 (a.out+0xbcc4e)

  Thread T1 (tid=8211, finished) created by main thread at:
    #0 pthread_create <null> (a.out+0x28446)
    #1 main /home/shelly/simple_race.cc:18:3 (a.out+0xbcc33)

SUMMARY: ThreadSanitizer: data race /home/shelly/simple_race.cc:12:9 in Thread2(
void*)
==================
ThreadSanitizer: reported 1 warnings
```

2、检测是否存在不可重入函数

将“1”中的 test.cc 程序用 LLVM 生成为“test.11”中间文件，并用上图中编写的程序打开文件，运行，输出结果如下图所示：

D:\College\并行与分布式计算\Homework\homework3\数据竞争\Debug\数据竞争.exe

```
程序中存在不可重入的函数
请按任意键继续. . .
```

证明 test.cc 中存在不可重入函数。这也与实际相符，运行结果正确。

```
#include <pthread.h>
#include <stdio.h>

int Global;

void *Thread1(void *x) {
  Global++;
  return NULL;
}

void *Thread2(void *x) {
  Global--;
  return NULL;
}

int main() {
  pthread_t t[2];
  pthread_create(&t[0], NULL, Thread1, NULL);
  pthread_create(&t[1], NULL, Thread2, NULL);
  pthread_join(t[0], NULL);
  pthread_join(t[1], NULL);
}
```

四、 遇到的问题及解决方法

本次实验是并行的第三次作业，整体来说并不难，我遇到了以下的几个问题。

第一，实验环境的配置！！！本来，我已经在 vmware 中配置好了 ubuntu16，并使用良好。但在完成第一个实验的时候，发现在安装完 clang 和 LLVM 之后运行 ThreadSanitizer 时竟然出现了下面这个错误：

```
shelly@shelly-virtual-machine:~$ g++ simple_race.cc -fsanitize=th
read -fPIE -pie -g
shelly@shelly-virtual-machine:~$ ./a.out
FATAL: ThreadSanitizer: unexpected memory mapping 0x55d5667ab000-
0x55d5667ac000
```

后来，经过网上资料的查询，发现好像是由于 linux 内核版本过低导致的。（具体原因我也不太确定，因为按照网上的说法，我所安装的 clang\LLVM\gcc 的版本已经满足要求，所以极有可能是因为 linux 版本过低）所以，我只能再次开始 ubuntu18 的重装过程。

第二，则是对于 LLVM 和 clang 的理解。经过查阅资料，我初步了解了 LLVM 和 clang 的关系，并且学会了阅读.ll 文件，但仍迷惑于如何实现“自动检索”。后来，经过与同学讨论，发现可以使用 string.find()函数进行关键字的搜索。

五、 参考文献

[1] 数据竞争检测器
https://github.com/google/sanitizers/wiki/ThreadSanitizerCppManual
[2] LLVM IR 语法 https://blog.csdn.net/11b202/article/details/10062687
[3] LLVM IR 语法小例子
https://blog.csdn.net/melissa_cjt/article/details/75202004
[4] LLVM 编译原理和使用
https://blog.csdn.net/yayaayaya123/article/details/83993041
[5] LLVM-Clang 编译器安装和使用
https://blog.csdn.net/rikeyone/article/details/100020145?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-2&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-2