

并行与分布式计算

数据竞争与可重入函数程序分析

第三次作业

姓名：罗仁良

班级：计科五班

学号：18340126

1 数据竞争

1.1 实验内容

利用 ThreadSanitizer 分析程序中是否有潜在数据竞争出现。

1.2 实验结果

实验中设计了几个简单的存在数据竞争的函数，利用 ThreadSanitizer 分析结果，大多数结果都与符合实际，但遇到一种情况比较特殊，与实际相反。

1.2.1 程序一

```
int global;
void * f1() {
    global = 1;
    return NULL;
}

int main(void) {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, f1, NULL);
    pthread_create(&t2, NULL, f1, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    return 0;
}
```

线程 t1 和线程 t2 同时访问全局变量 global，且存在写入，有数据竞争。结果如下图：

```

hw3 git:master > ./DataRace_sample
=====
WARNING: ThreadSanitizer: data race (pid=97880)
  Write of size 4 at 0x00010a51e05c by thread T2:
    #0 f1 <null>:1600848 (DataRace_sample:x86_64+0x10000cfb)

  Previous write of size 4 at 0x00010a51e05c by thread T1:
    #0 f1 <null>:1600848 (DataRace_sample:x86_64+0x10000cfb)

  Location is global 'global' at 0x00010a51e05c (DataRace_sample+0x00010000205c)

  Thread T2 (tid=2324054, running) created by main thread at:
    #0 pthread_create <null>:1600928 (libclang_rt.tsan_osx_dynamic.dylib:x86_64h+0x2aacd)
    #1 main <null>:1600928 (DataRace_sample:x86_64+0x10000e8a)

  Thread T1 (tid=2324053, finished) created by main thread at:
    #0 pthread_create <null>:1600928 (libclang_rt.tsan_osx_dynamic.dylib:x86_64h+0x2aacd)
    #1 main <null>:1600928 (DataRace_sample:x86_64+0x10000e6e)

SUMMARY: ThreadSanitizer: data race (DataRace_sample:x86_64+0x10000cfb) in f1
=====
ThreadSanitizer: reported 1 warnings
[1] 97880 abort ./DataRace_sample

```

1.2.2 程序二

```

static int s;

void *f2() {
    s = 2;
    printf("f2:%d\n", s);
    return NULL;
}

void *f3() {
    s = 3;
    printf("f3:%d\n", s);
    return NULL;
}

int main(void) {
    pthread_t t3, t4;
    pthread_create(&t3, NULL, f2, NULL);
    pthread_create(&t4, NULL, f3, NULL);
    pthread_join(t3, NULL);
    pthread_join(t4, NULL);
    return 0;
}

```

```
hw3 git:master > clang -fsanitize=thread -O1 -o DataRace_sample DataRace_sample.c
hw3 git:master > ./DataRace_sample
f2:2
f3:3
```

使用 threadSanitizer 检测时程序并没有报错，结果如上图：但是分析程序可以知道，printf 是非原子操作，若此时另外的线程对 s 进行写，就可能出现数据竞争。为了检测数据竞争，将源程序进行了改进。循环打印 s，每次打印 sleep(1)，得到如下结果：从检测结果可以知道，存在数据竞争，此

```
f2:2
=====
WARNING: ThreadSanitizer: data race (pid=650)
  Write of size 4 at 0x00010a918058 by thread T2:
    #0 f3 <null>:1600816 (DataRace_sample:x86_64+0x10000dc1)

  Previous write of size 4 at 0x00010a918058 by thread T1:
    #0 f2 <null>:1600816 (DataRace_sample:x86_64+0x10000d41)

  Location is global 's' at 0x00010a918058 (DataRace_sample+0x000100002058)

  Thread T2 (tid=2340500, running) created by main thread at:
    #0 pthread_create <null>:1600896 (libclang_rt.tsan_osx_dynamic.dylib:x86_64h+0x2aacd)
    #1 main <null>:1600896 (DataRace_sample:x86_64+0x100000e61)

  Thread T1 (tid=2340499, running) created by main thread at:
    #0 pthread_create <null>:1600896 (libclang_rt.tsan_osx_dynamic.dylib:x86_64h+0x2aacd)
    #1 main <null>:1600896 (DataRace_sample:x86_64+0x100000e4a)

SUMMARY: ThreadSanitizer: data race (DataRace_sample:x86_64+0x10000dc1) in f3
=====
f3:3
f2:3
f3:3
f2:3
f3:3
f2:3
f3:3
f2:3
f3:3
f2:3
f3:3
f2:3
f3:3
f2:3
f3:3
f2:3
f3:3
f2:3
f3:3
ThreadSanitizer: reported 1 warnings
[1] 650 abort ./DataRace_sample
```

时 threadSanitizer 也分析出有数据竞争。

2 IR 自动分析程序设计

2.1 设计目标

设计一个程序自动分析 llvm 的中间代码，判断一个函数是否为可重入函数。

2.2 设计思路

llvm 编译产生的中间代码具有一定的易读性和特点。自动分析程序主要利用了 IR 的两大特点：

- (1) 全局变量由 @ 和 global 标识
- (2) 函数声明有 define 和 @ 标识

所以设计的主要思路就是分析中间代码中由 @ 标识的全局变量和用户声明的函数中是否存在使用全局变量和使用系统调用的函数。

2.3 代码分析

2.3.1 数据定义

```
// 记录函数名和标识是否为可重入函数
typedef struct Function {
    string name;    // 函数名
    int Reentrant;  // 不可重入：0；可重入：1
    Function(string s, int Reentrant):name(s),
        Reentrant(Reentrant) {}
} Function;

vector<Function> func_lib; // 库函数记录，主要是系统调用
vector<Function> user_func; // 用户函数记录
```

每分析出一个函数，在记录它的可重入性的同时，将它添加到库函数和用户函数记录中。若存在函数调用，就在库函数中查询调用函数的可重入性。初始的库函数记录中已经记录了系统调用相关的函数信息，如，printf, malloc 等函数。

2.3.2 算法分析

使用每次读一行，对 IR code 进行分析，首先寻找是否存在 @ 标识符，再确定是否为全局变量。

```
it = find(line.begin(), line.end(), '@');
```

如果是全局变量，分析出变量名，添加到库函数记录中，标记为非可重入“函数”。

```
if( line.find("global") != string::npos) {  
    string name;  
    it++;  
    while(*it != ' ') {  
        name += *it;  
        it++;  
    }  
    func_lib.push_back(Function(name, 0));  
}
```

搜索完全局变量，再通过 **define** 寻找用户函数声明

```
if( line.find("define") != string::npos )  
// 找到函数声明分析函数名  
string func_name;  
int Reentrant = 1;  
it++;  
while(*it != '(') {  
    func_name += *it;  
    it++;  
}
```

找到函数声明，分析函数中是否使用全局变量、调用非可重入函数、系统调用等。寻找方式同上面描述的方式一样，搜索全局变量标识符 @。找到之后在库函数记录中对比即可。

```
vector<Function>::iterator i = func_lib.begin();
while( i != func_lib.end() ) {
    if(i->name == name) {
        if(i->Reentrant == 0) {
            Reentrant = 0;
            break;
        }
    }
    i++;
}
```


2.4 运行结果

2.4.1 样例程序

```
//===== 使用全局变量 =====
int value = 1;
void *f1() {
    value = 4;
    return NULL;
}
//===== 使用堆 =====
void *f2() {
    int *p = malloc(8*sizeof(int));
    for(int i=0; i < 8; i++) {
        p[i] = i;
    }
    return NULL;
}
```

```
//===== 调用不可重入函数 =====  
void *f3() {  
    f1();  
    return NULL;  
}  
//===== 系统调用 I/O =====  
void *f4() {  
    printf("stdio\n");  
    return NULL;  
}  
//===== 返回全局变量 =====  
  
void *f5() {  
    return &value;  
}  
//===== 可重入函数 =====  
void *f(){  
    return NULL;  
}  
int fff(int a) {  
    return a+1;  
}
```

2.4.2 结果截图



```
f1:no  
f2:no  
f3:no  
f4:no  
f5:no  
f:Reentrant  
fff:Reentrant
```


3 实验总结

通过数据竞争的实验熟悉了 ThreadSanitizer 在多线程程序中检测数据竞争的方法，加深了数据竞争产生的原因以及在多线程编程中尽量使得数据本地化避免数据竞争。自动分析程序设计其实难度是挺大的，只是我在设计的过程中简化了判断方式。还是有很多缺陷的，比如函数同名等问题没有解决，分析程序也只是给出了一个函数是否为可重入函数，没有给出更加具体的分析。这些都是值得改进的地方。总而言之，本次实验基本了解 llvm 中的部分知识和使用其中的部分工具帮助程序分析，还有很多地方需要进一步的学习、探索。