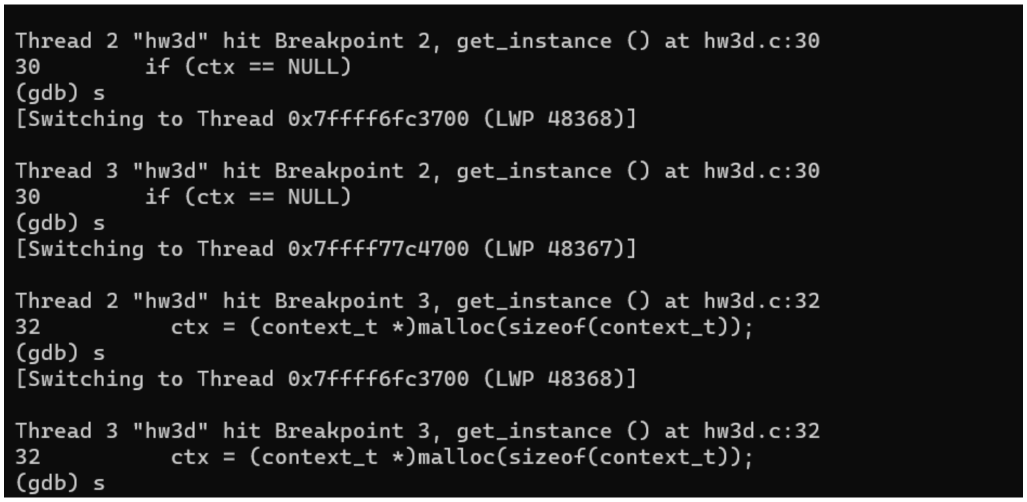
Benedictus Alvian Sofjan

3035603095

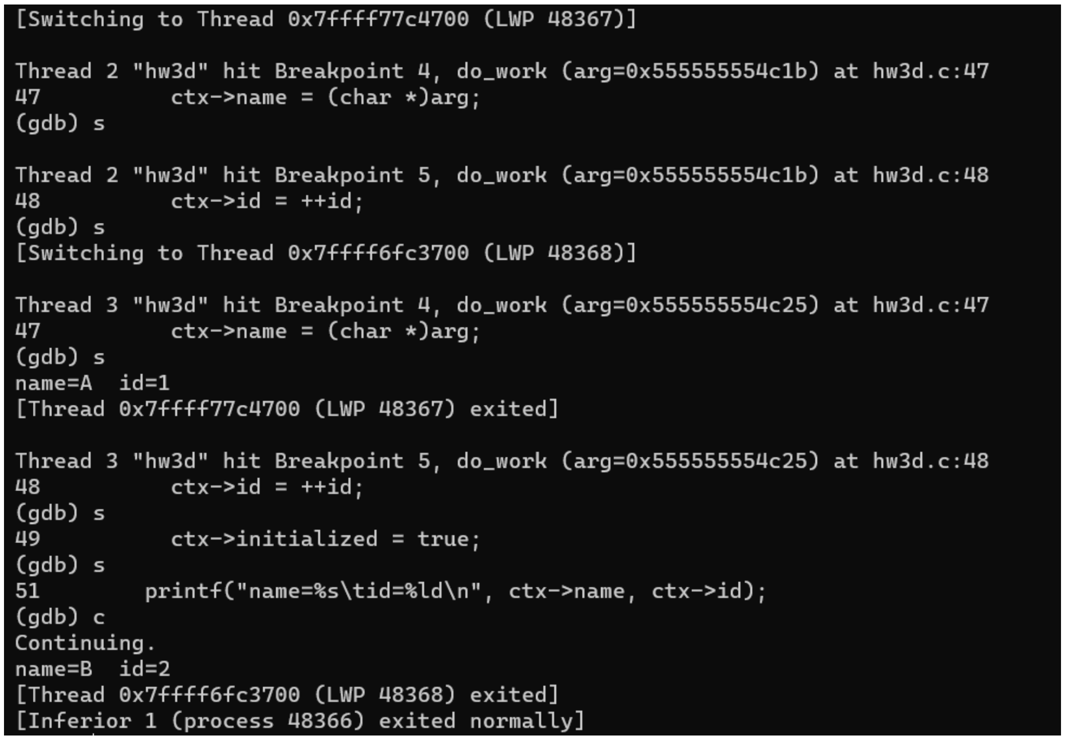
COMP3230 Tutorial 3 Exercise 3

Debugging Evaluation



```
Thread 2 "hw3d" hit Breakpoint 2, get_instance () at hw3d.c:30
30        if (ctx == NULL)
(gdb) s
[Switching to Thread 0x7ffff6fc3700 (LWP 48368)]

Thread 3 "hw3d" hit Breakpoint 2, get_instance () at hw3d.c:30
30        if (ctx == NULL)
(gdb) s
[Switching to Thread 0x7ffff77c4700 (LWP 48367)]

Thread 2 "hw3d" hit Breakpoint 3, get_instance () at hw3d.c:32
32            ctx = (context_t *)malloc(sizeof(context_t));
(gdb) s
[Switching to Thread 0x7ffff6fc3700 (LWP 48368)]

Thread 3 "hw3d" hit Breakpoint 3, get_instance () at hw3d.c:32
32            ctx = (context_t *)malloc(sizeof(context_t));
(gdb) s
```

Thread “A” and thread “B” both evaluate if (ctx == NULL) at the same time, and thus two instances of *context_t* was created. This is a bug.

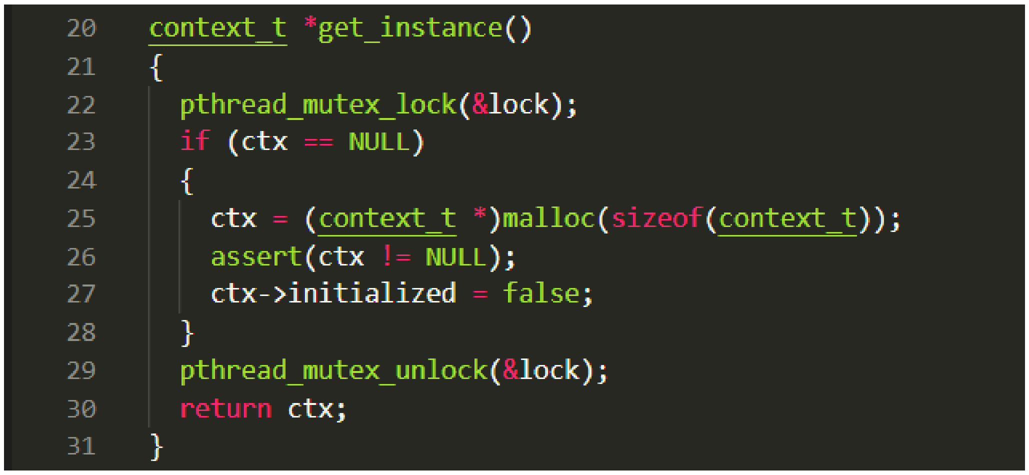


```
[Switching to Thread 0x7ffff77c4700 (LWP 48367)]

Thread 2 "hw3d" hit Breakpoint 4, do_work (arg=0x555555554c1b) at hw3d.c:47
47            ctx->name = (char *)arg;
(gdb) s

Thread 2 "hw3d" hit Breakpoint 5, do_work (arg=0x555555554c1b) at hw3d.c:48
48            ctx->id = ++id;
(gdb) s
[Switching to Thread 0x7ffff6fc3700 (LWP 48368)]

Thread 3 "hw3d" hit Breakpoint 4, do_work (arg=0x555555554c25) at hw3d.c:47
47            ctx->name = (char *)arg;
(gdb) s
name=A  id=1
[Thread 0x7ffff77c4700 (LWP 48367) exited]

Thread 3 "hw3d" hit Breakpoint 5, do_work (arg=0x555555554c25) at hw3d.c:48
48            ctx->id = ++id;
(gdb) s
49            ctx->initialized = true;
(gdb) s
51        printf("name=%s\tid=%ld\n", ctx->name, ctx->id);
(gdb) c
Continuing.
name=B  id=2
[Thread 0x7ffff6fc3700 (LWP 48368) exited]
[Inferior 1 (process 48366) exited normally]
```

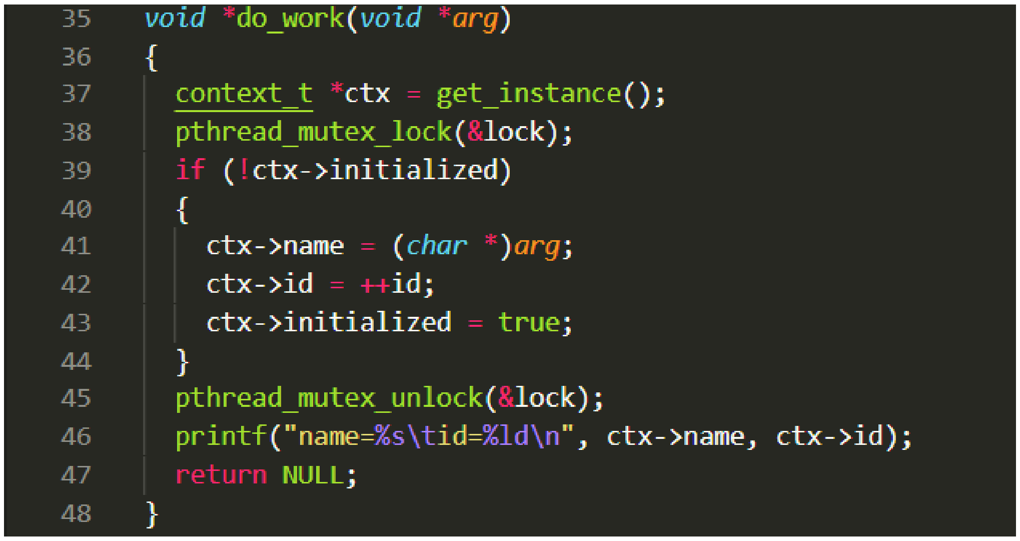
Thread “A” and thread “B” both access ctx->name and ctx->id at the same time, initialization is done twice, and race condition occurs. Thus, two different contexts are printed by thread “A” and “B”.

 I fixed the bug by adding a mutex in get_instance() so only one thread may evaluate the if (ctx == NULL) expression and thus only a single instance of *context_t* will be created.

However, this is not enough to entirely eliminate the bug, as there is a very slim chance thread “A” and “B” will evaluate the if (!ctx->initialized) expression at the same time and thus initialize ctx twice. Thus, another mutex is necessary to be added before the if (!ctx->initialized) expression.



```c
context_t *get_instance()
{
    pthread_mutex_lock(&lock);
    if (ctx == NULL)
    {
        ctx = (context_t *)malloc(sizeof(context_t));
        assert(ctx != NULL);
        ctx->initialized = false;
    }
    pthread_mutex_unlock(&lock);
    return ctx;
}
```



```c
void *do_work(void *arg)
{
    context_t *ctx = get_instance();
    pthread_mutex_lock(&lock);
    if (!ctx->initialized)
    {
        ctx->name = (char *)arg;
        ctx->id = ++id;
        ctx->initialized = true;
    }
    pthread_mutex_unlock(&lock);
    printf("name=%s\tid=%ld\n", ctx->name, ctx->id);
    return NULL;
}
```