The bug is the uncontrolled scheduling of the two threads. Each thread creates one instance and there are two instances created in total. The global variable id is increased by 1 from the first thread. Then, the global variable id is increased by 1 (id = 2) from the second thread. It violates the single instance requirement.

```
cylee@workbench:~/Desktop$ ./test
name=A  id=1
name=B  id=2
Starting program: /student/19/cs/cylee/Desktop/test
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffff77c4700 (LWP 32829)]
[New Thread 0x7ffff6fc3700 (LWP 32830)]
[Switching to Thread 0x7ffff77c4700 (LWP 32829)]

Thread 2 "test" hit Breakpoint 4, do_work (arg=0x555555554c1b) at test.c:46
46        printf("name=%s\tid=%ld\n", ctx->name, ctx->id);
(gdb) p ctx->id
$53 = 1
(gdb) p ctx->name
$54 = 0x555555554c1b "A"
(gdb) c
Continuing.
[Switching to Thread 0x7ffff6fc3700 (LWP 32830)]

Thread 3 "test" hit Breakpoint 4, do_work (arg=0x555555554c25) at test.c:46
46        printf("name=%s\tid=%ld\n", ctx->name, ctx->id);
(gdb) p ctx->id
$55 = 2
(gdb) p ctx->name
$56 = 0x555555554c25 "B"
```

Moreover, the output sometimes looks correct but may violate thread-safe requirement because the two threads do multiple initialization. The first and second thread do initialization and the first thread run much faster than the second thread so that initialization becomes true and thus the second thread prints the same name and id as the first thread.

```
cylee@workbench:~/Desktop$ ./test
name=A  id=1
name=A  id=1
```

```
Starting program: /student/19/cs/cylee/Desktop/test
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffff77c4700 (LWP 67160)]
[New Thread 0x7ffff6fc3700 (LWP 67161)]
[Switching to Thread 0x7ffff77c4700 (LWP 67160)]

Thread 2 "test" hit Breakpoint 1, do_work (arg=0x555555554b4b) at test.c:36
36        printf("name=%s\tid=%ld\n", ctx->name, ctx->id);
(gdb) p ctx->id
$1 = 1
(gdb) p ctx->name
$2 = 0x555555554b4b "A"
(gdb) c
Continuing.
[Switching to Thread 0x7ffff6fc3700 (LWP 67161)]

Thread 3 "test" hit Breakpoint 1, do_work (arg=0x555555554b55) at test.c:36
36        printf("name=%s\tid=%ld\n", ctx->name, ctx->id);
(gdb) p ctx->id
$3 = 1
(gdb) p ctx->name
$4 = 0x555555554b4b "A"
```

In order to fix the bug, I add a mutex lock from line 32 to line 39 inside do_work. After the first thread get an instance, it will change the bool initialized is true so that the second thread cannot run the if clause.

For the bonus question, I use pthread_once in order to ensure the thread only run once which fulfill the single instance and thread-safe requirement.

```
pthread_once_t once = PTHREAD_ONCE_INIT;

// singleton
void get_instance() {

  ctx = (context_t *)malloc(sizeof(context_t));
  assert(ctx != NULL);
  ctx->initialized = false;

}

int id = 0;

void *do_work(void *arg) {
  pthread_mutex_lock(&lock);
  int rc = pthread_once(&once,get_instance);
  assert(rc == 0);
  if (!ctx->initialized) {
    ctx->name = (char *)arg;
    ctx->id = ++id;
    ctx->initialized = true;
  }
  pthread_mutex_unlock(&lock);
```