ADVANCED UNIX PROGRAMMING MIDTERM REPORT

TEAM 9 — 林禾堃、馬毓昇、陳曦

Dec 2023

1. Question 1

The code use pthread_once function to ensure that the pthread key will only create once. After the creation of key, envbuf should be NULL, and therefore malloc a space, associated with the key we've just created.
Then every time we run the getenv function, we only write the result on thread-specific data envbuf. Then we can ensure that even if two threads call this function at the same time, since the result is written in thread-specific data, there won't be inconsistencies.

2. Question 2

No, the reason why this version of getenv is still not async-signal safety is because malloc function is not async-signal safe. For example, when two threads run malloc at the same time, there might be race condition, and therefore not async-signal safe. Therefore simply block signals won't solve this issue.

3. Question 3

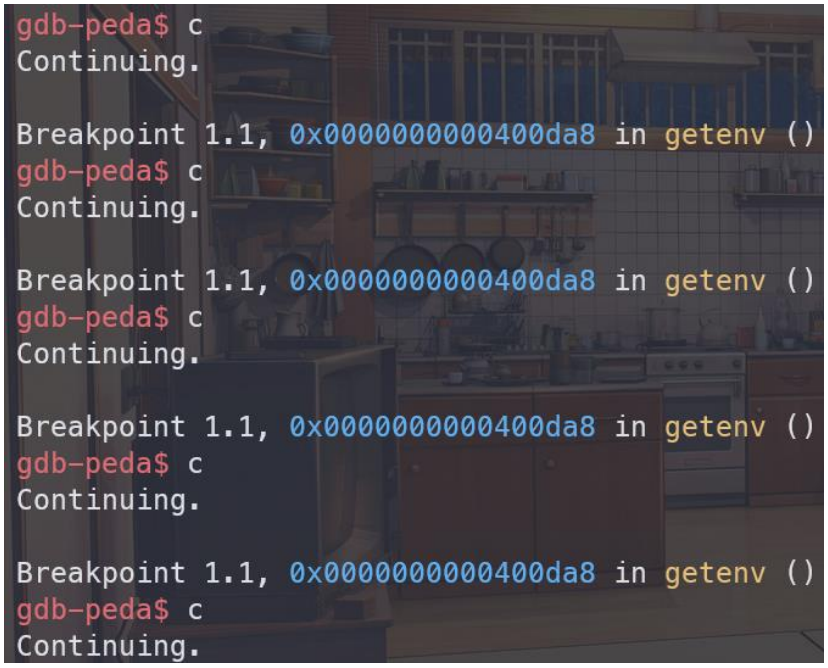
The program cannot run successfully on FreeBSD, to understand why, we tried to use gdb-peda to trace and see what happened while running the program.

We starts with checking function symbols. Below is the result, we found that there are multiple definitions of getenv function.

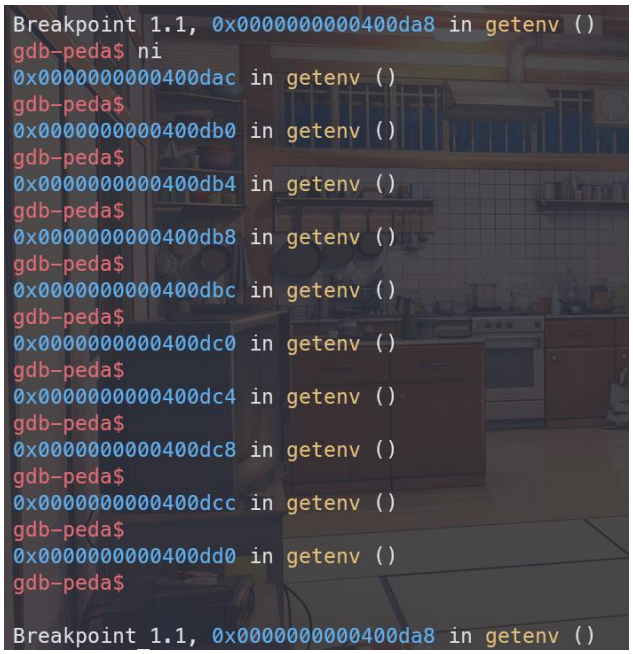
For example, in both stdlib and rtld-elf we found the definition, and also our implementation.

Then we figure out that even we don’t send any arguments, which suppose to print out help message then return directly, but it stuck rather than terminating.

So we set a break point at getenv function, then run debugger. And we found that the program keep stuck in getenv function as below.



To see more details, use ni to see next instruction. Then we found that the program will keep running between 0x400da8 to 0x400dd0.



Below are the instructions between these two addresses.

```
gdb-peda$ x/67i 0x400da4
   0x400da4 <getenv>:    stp     x29, x30, [sp, #-64]!
=> 0x400da8 <getenv+4>:  adrp    x1, 0x400000
   0x400dac <getenv+8>:  add     x1, x1, #0xd90
   0x400db0 <getenv+12>:         mov     x29, sp
   0x400db4 <getenv+16>:         stp     x19, x20, [sp, #16]
   0x400db8 <getenv+20>:         adrp    x19, 0x411000
   0x400dbc <getenv+24>:         stp     x23, x24, [sp, #48]
   0x400dc0 <getenv+28>:         add     x23, x19, #0x340
   0x400dc4 <getenv+32>:         mov     x24, x0
   0x400dc8 <getenv+36>:         add     x0, x23, #0x8
   0x400dcc <getenv+40>:         stp     x21, x22, [sp, #32]
   0x400dd0 <getenv+44>:         bl      0x400940 <pthread_once@plt>
   0x400dd4 <getenv+48>:         add     x0, x23, #0x18
```

Also, when back tracing, we see the same situation.

```
gdb-peda$ bt
#0  __thr_interpose_libc () at /usr/src/lib/libthr/thread/thr_syscalls.c:643
#1  0x0000000004049ec40 in _libpthread_init (curthread=0x0) at /usr/src/lib/libthr/thread/thr_init.c:329
#2  0x00000000404a4204 in _thr_check_init () at /usr/src/lib/libthr/thread/thr_private.h:930
#3  _thr_once (once_control=0x411348 <init_done>, init_routine=0x400d90 <thread_init>) at /usr/src/lib/libthr/thread/thr_once.c:72
#4  0x0000000000400dd4 in getenv ()
#5  0x000000004049ed24 in init_private () at /usr/src/lib/libthr/thread/thr_init.c:501
#6  _libpthread_init (curthread=0x0) at /usr/src/lib/libthr/thread/thr_init.c:332
#7  0x00000000404a4204 in _thr_check_init () at /usr/src/lib/libthr/thread/thr_private.h:930
#8  _thr_once (once_control=0x411348 <init_done>, init_routine=0x400d90 <thread_init>) at /usr/src/lib/libthr/thread/thr_once.c:72
#9  0x0000000000400dd4 in getenv ()
#10 0x000000004049ed24 in init_private () at /usr/src/lib/libthr/thread/thr_init.c:501
#11 _libpthread_init (curthread=0x0) at /usr/src/lib/libthr/thread/thr_init.c:332
#12 0x00000000404a4204 in _thr_check_init () at /usr/src/lib/libthr/thread/thr_private.h:930
#13 _thr_once (once_control=0x411348 <init_done>, init_routine=0x400d90 <thread_init>) at /usr/src/lib/libthr/thread/thr_once.c:72
#14 0x0000000000400dd4 in getenv ()
```

The reason why this issue occur is that some initialize function tried to get environment variable by calling getenv function. But in our implementation, we also use some functions that need initialize by calling getenv function (for example, pthread functions and malloc.), and therefore a dead loop occurs.

We can verify that by output the name in getenv function.

```
root@genet0:~/Advanced-UNIX-Programming/HW10 # ./assignment10
MALLOC_CONF
LIBPTHREAD_BIGSTACK_MAIN
LIBPTHREAD_SPLITSTACK_MAIN
LIBPTHREAD_SPINLOOPS
LIBPTHREAD_YIELDLOOPS
LIBPTHREAD_QUEUE_FIFO
```

So in order to solve this issue, we should make sure that when the initialize functions calling getenv, we won't run thread-safe codes, while after that, just do thread-safe version.

First, declare a flag function in global.

```
// for preventing initialize process dead
static int init_flag = 1;
```

Then since we cannot run malloc, so instead we allocate a static char array, so that even the function return, the value still holds.

```
// buffer for initialize process
static char     buf[MAXSTRINGSZ];
```

For initialize functions, just assign the buf to envbuf, otherwise we run the thread-safe code.

```
if(init_flag == 0){
        pthread_once(&init_done, thread_init);
        pthread_mutex_lock(&env_mutex);
        envbuf = (char *)pthread_getspecific(key);

        if (envbuf == NULL) {
                envbuf = malloc(MAXSTRINGSZ);
                if (envbuf == NULL) {
                        pthread_mutex_unlock(&env_mutex);
                        return(NULL);
                }
                pthread_setspecific(key, envbuf);
        }
}
else{
        envbuf = buf;
}
```

Whenever we are going to run functions like pthread, check whether running initialization or not.

```
len = strlen(name);
for (i = 0; environ[i] != NULL; i++) {
        if ((strncmp(name, environ[i], len) == 0) &&
                        (environ[i][len] == '=')) {
                strncpy(envbuf, &environ[i][len + 1], MAXSTRINGSZ -1);
                if(init_flag == 0){
                        pthread_mutex_unlock(&env_mutex);
                }
                return(envbuf);
        }
}

if(init_flag == 0){
        pthread_mutex_unlock(&env_mutex);
}

return(NULL);
}
```

When running to main function, we shall assume that all initialization are done, so set the initialization flag to 0.

```
// set init flag
init_flag = 0;
```

Below is the output with debug message.