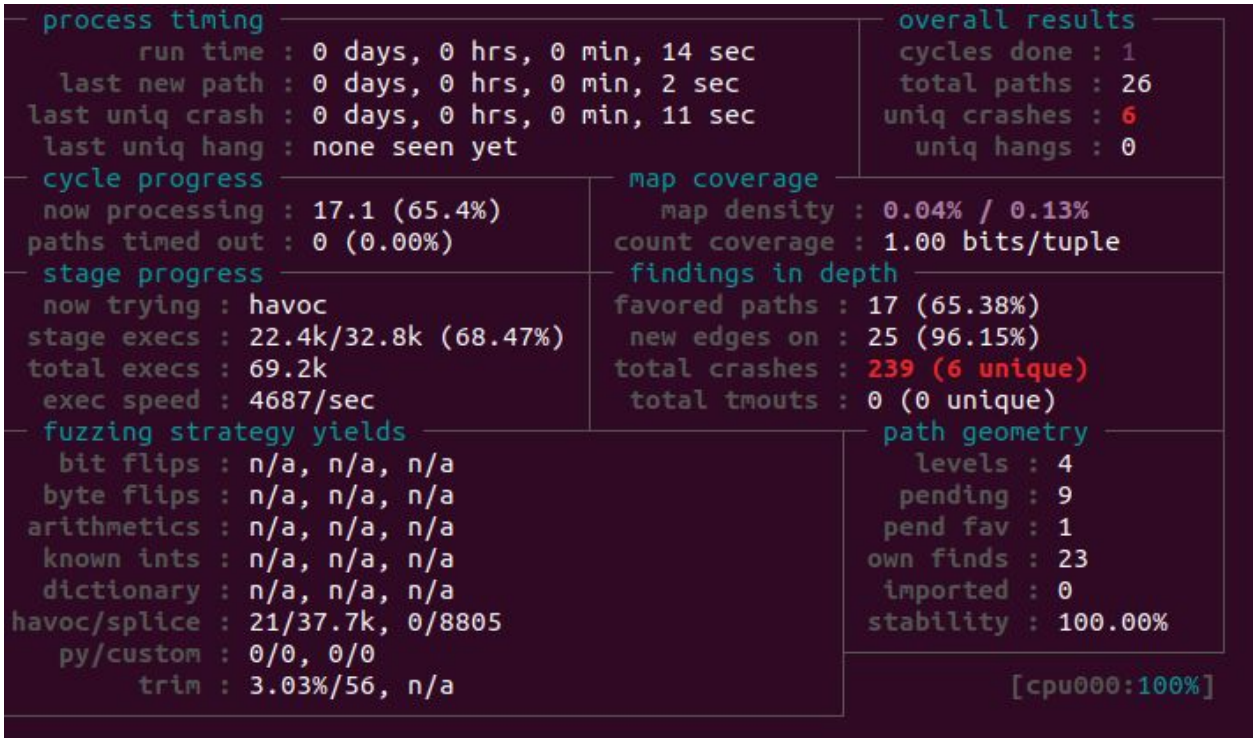
In a new build folder, do the below for 5 rounds
CC=/usr/local/bin/afl-gcc CXX=/usr/local/bin/afl-g++ cmake ..
make
afl-fuzz -i ../testcases -o ../findings ./afldemo
rm -rf ./

Round 1:

For part 1:**Fuzz testing on a vulnerable program**



(1) how long was the fuzzing process
14 sec

(2) how many inputs AFL found that could make the program crash or hang.
6
wenhui@wenhui:~/Downloads/afl-demo/findings/default/crashes$ ls
id:000000,sig:06,src:000002,time:80,op:havoc,rep:8
id:000001,sig:11,src:000002,time:147,op:havoc,rep:2
id:000002,sig:06,src:000002,time:1302,op:havoc,rep:4
id:000003,sig:11,src:000002,time:1317,op:havoc,rep:16
id:000004,sig:06,src:000002,time:1585,op:havoc,rep:16
id:000005,sig:06,src:000002,time:3449,op:havoc,rep:4

For part 2: **Isolating and fixing the bugs**

(1) describe your process of identifying bugs from AFL-generated inputs and what bugs you found in the program;

The AFL inputs are in folder /findings/default/crashes, execute the program with this input , and use printf to identify the issue,

a. u -1 strin, It seems like N should not be a negative number, it causes malloc errors
b. u -18 aBCd 18 aBC11, it seems like negative N causes malloc issues
c. u 11 {BCdddd@ddddMddddddd0aB9zde, it seems like characters, such as {, other than small letters causes transiting issues
d. u 10 aBCded 10 aBCde0 aBC aBCd 10 aBCde0 aBCd, it seems like spaces and capital letters causes translating issues
e. u 0aBCu 8 4 aBCde, it seems like spaces and capital letters causes translating issues
f. u 0aBCu 8 4 aBCde aBCdeed����, it seems like spaces and capital letters causes translating issues

(2) describe how you fixed the bugs and why your fixes worked.

For INPUT in case a and b: Bound the input N to positive integers, bound the N so that later malloc won't have issues

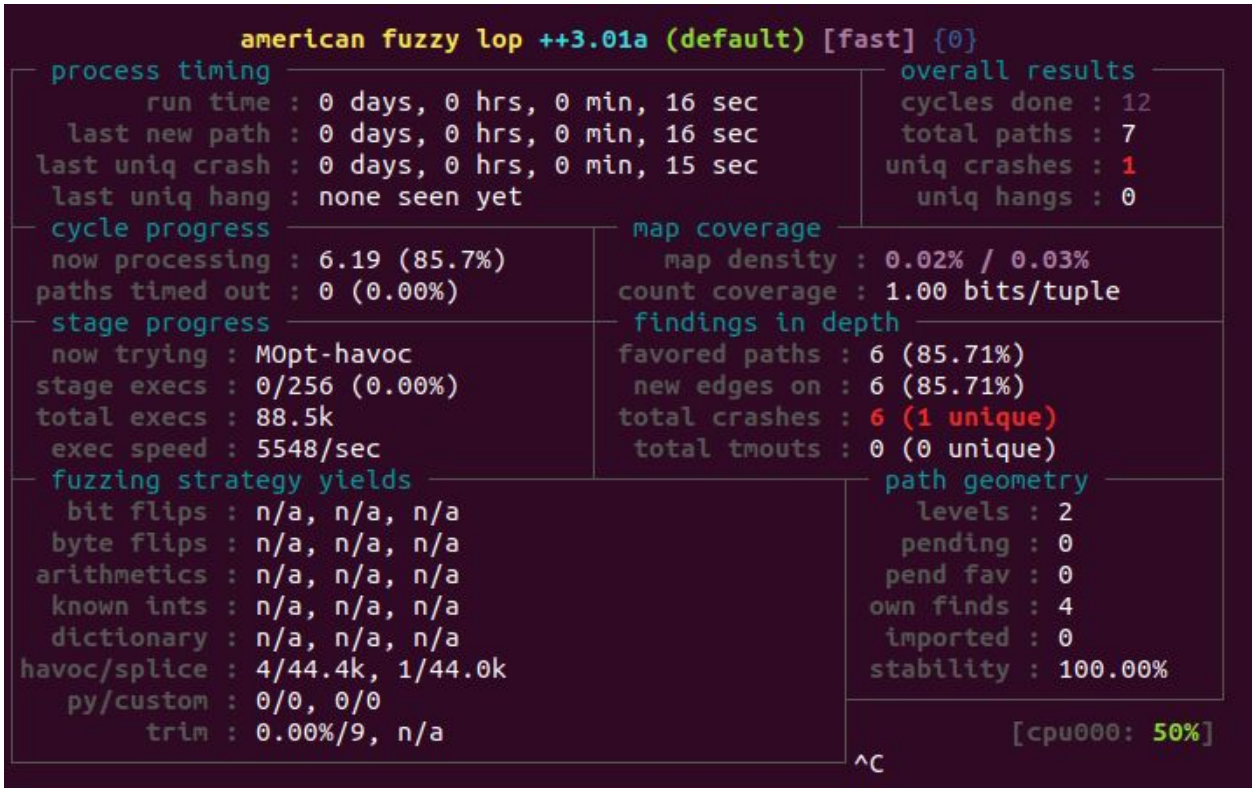
```
len = strtol(input + 2, &rest, 10); // how many characters of the string to upper-case
if(len < 0){
        printf("Specified length %d should be positive\n", len);
        return 1;
}
```

For INPUT in case c, d, e, f, Bound the input string to only small letters, so that ASCII translation works

```
}
for (i = 0; i != len; i++)
{
        if(rest[i] >= 'a' || rest[i] <= 'z'){
                printf("Letters should be small letters, this program only handles ASCII\n");
                return 1;
        }
        out[i] = rest[i] - 32; // only handles ASCII
}
```

Round 2:

For part 1:**Fuzz testing on a vulnerable program**



(1) how long was the fuzzing process;

16 sec

(2) how many inputs AFL found that could make the program crash or hang.

```
wenhui@wenhui:~/Downloads/afl-demo/findings/default/crashes$ ls
id:000000,sig:06,src:000000+000002,time:783,op:splice,rep:16  README.txt
```

1 input for crash:

u u �� �� 3 �� >B

For part 2: **Isolating and fixing the bugs**

(1) describe your process of identifying bugs from AFL-generated inputs and what bugs you found in the program;

Input “u u �� �� 3 �� >B” , execute the program with this input , and use printf to identify the issue, N should be an integer instead of a character

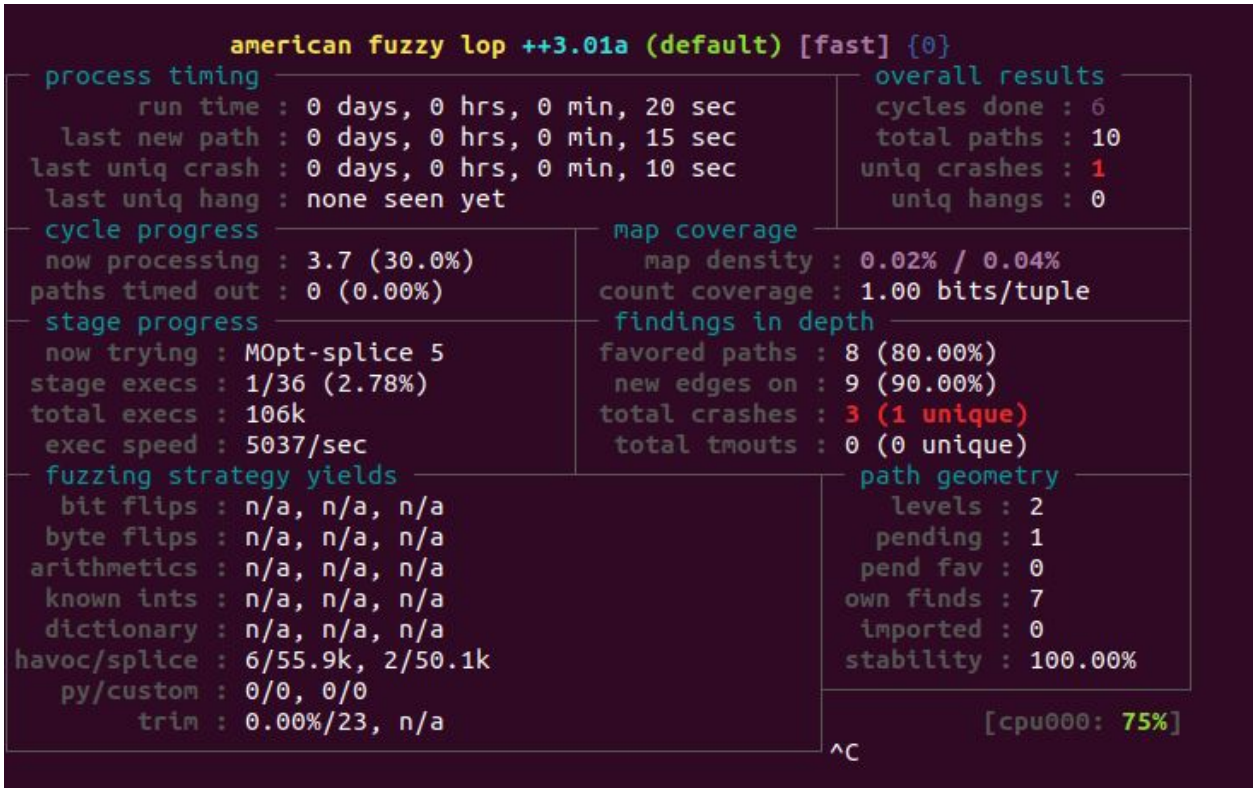
(2) describe how you fixed the bugs and why your fixes worked.

Bound N to a number, atoi returns 0 if it is a number 0 or not a number. Get the first char to see if the N is 0, if N is not 0 and atoi returns 0, then the input is not valid.

```
upper case command
    char *rest;
    len = strtol(input + 2, &rest, 10); // how many characters of the string to upper-case
    newlen = atoi(input+2);
    int firstN = input[2] - '0';
    if(newlen == 0 && firstN != 0){
            printf("N should be a number\n");
            return 1;
    }
```

Round 3:

For part 1:**Fuzz testing on a vulnerable program**



(1) how long was the fuzzing process;

20 sec

(2) how many inputs AFL found that could make the program crash or hang.

1 input

```
wenhui@wenhui:~/Downloads/afl-demo/findings/default/crashes$ ls
id:000000,sig:06,src:000007+000002,time:10698,op:splice,rep:16  README.txt
wenhui@wenhui:~/Downloads/afl-demo/findings/default/crashes$ cat id:000000,sig:06,src:000007+000002,time:10698,op:splice,rep:16
J 0•••u 10 aB••••••••••'••u aB• 0 ••••••••p•••Â•'••u 10 aBB••••••••••aBwenhui@wenhui:~/Downloads/afl-demo/findings/default/crashes$
```

For part 2: **Isolating and fixing the bugs**

(1) describe your process of identifying bugs from AFL-generated inputs and what bugs you found in the program;

Input “u 0���u 10 aB������□��u aB□ 0 ������p���Ã□��u 10 aBB□□□□□□□□□aB” , execute the program with this input , and use printf to identify the issue, input string should be small letters

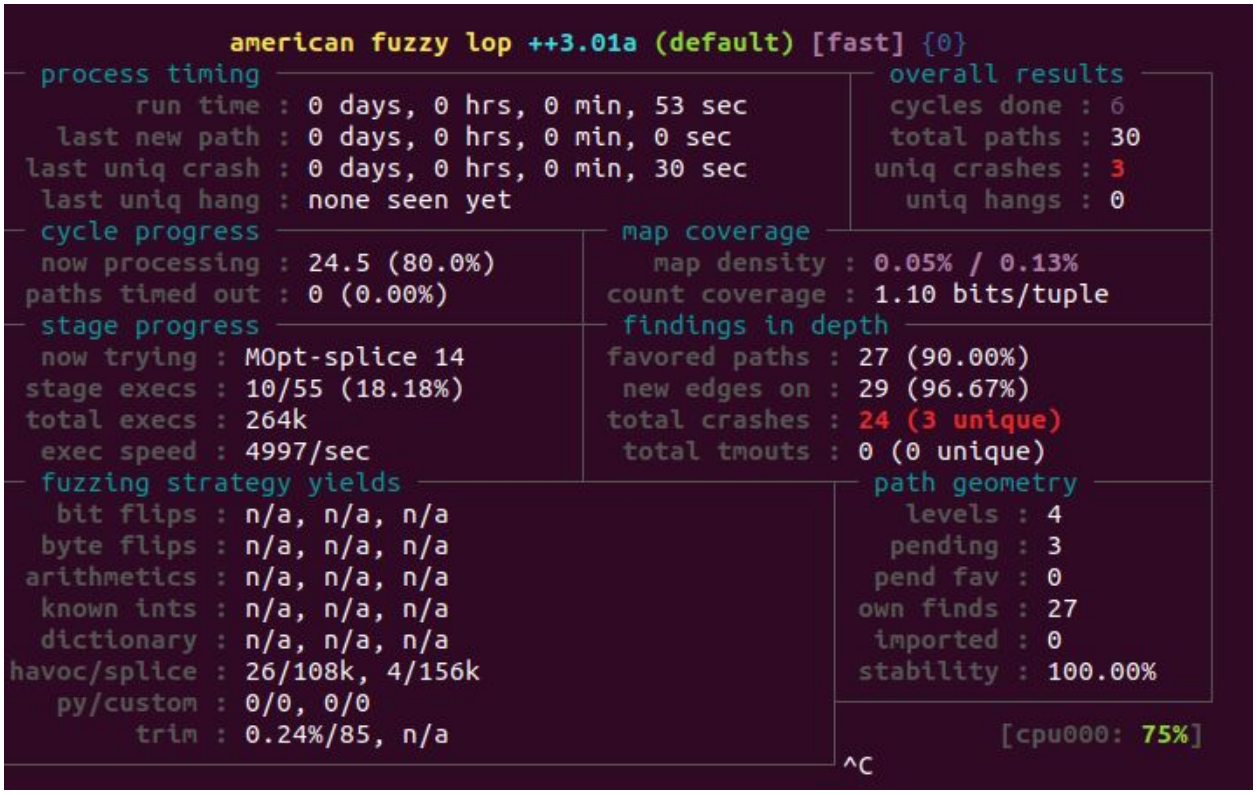
(2) describe how you fixed the bugs and why your fixes worked.

All characters in string must be between ‘a’ and ‘z’.

```
    }
    for (i = 0; i != len; i++)
    {
            if(!((rest[i] - 'a' >= 0) &&  (rest[i] - 'z' <= 0))){
                    printf("Letters should be small letters, this program only handles ASCII\n");
                    return 1;
            }
            out[i] = rest[i] - 32; // only handles ASCII
    }
```

Round 4:

For part 1:**Fuzz testing on a vulnerable program**



(1) how long was the fuzzing process;

53 sec

(2) how many inputs AFL found that could make the program crash or hang.

3

```
wenhui@wenhui:~/Downloads/afl-demo/findings/default/crashes$ ls
id:000000,sig:06,src:000024,time:7236,op:havoc,rep:16
id:000001,sig:06,src:000018+000020,time:14737,op:splice,rep:8
id:000002,sig:06,src:000021+000028,time:22355,op:splice,rep:8
README.txt
wenhui@wenhui:~/Downloads/afl-demo/findings/default/crashes$ cat id:000000,sig:06,
src:000024,time:7236,op:havoc,rep:16
u 1d@@@@@@@@@@@@@@@@@@@@@@@@ddX 1dddc◆wenhui@wenhui:~/Downloads/afl-demo/findings/def
ault/crashes$ cat id:000001,sig:06,src:000018+000020,time:14737,op:splice,rep:8
u 2yuuuuuuuuuuuuuuuuuuuuuuuuuuwenhui@wenhui:~/Downloads/afl-demo/findings/default/cr
ashes$ cat id:000002,sig:06,src:000021+000028,time:22355,op:splice,rep:8
u 0u 1sssssssssssssssswsss ddddddXa@sssssWssss◆sscwenhui@wenhui:~/Downloads/afl-de
mo/findings/default/crashes$
```

For part 2: **Isolating and fixing the bugs**

(1) describe your process of identifying bugs from AFL-generated inputs and what bugs you found in the program;

execute the program with this input , and use printf to identify the issue, there should be a space after the N, also the input should terminated with \n, there should be a space after N

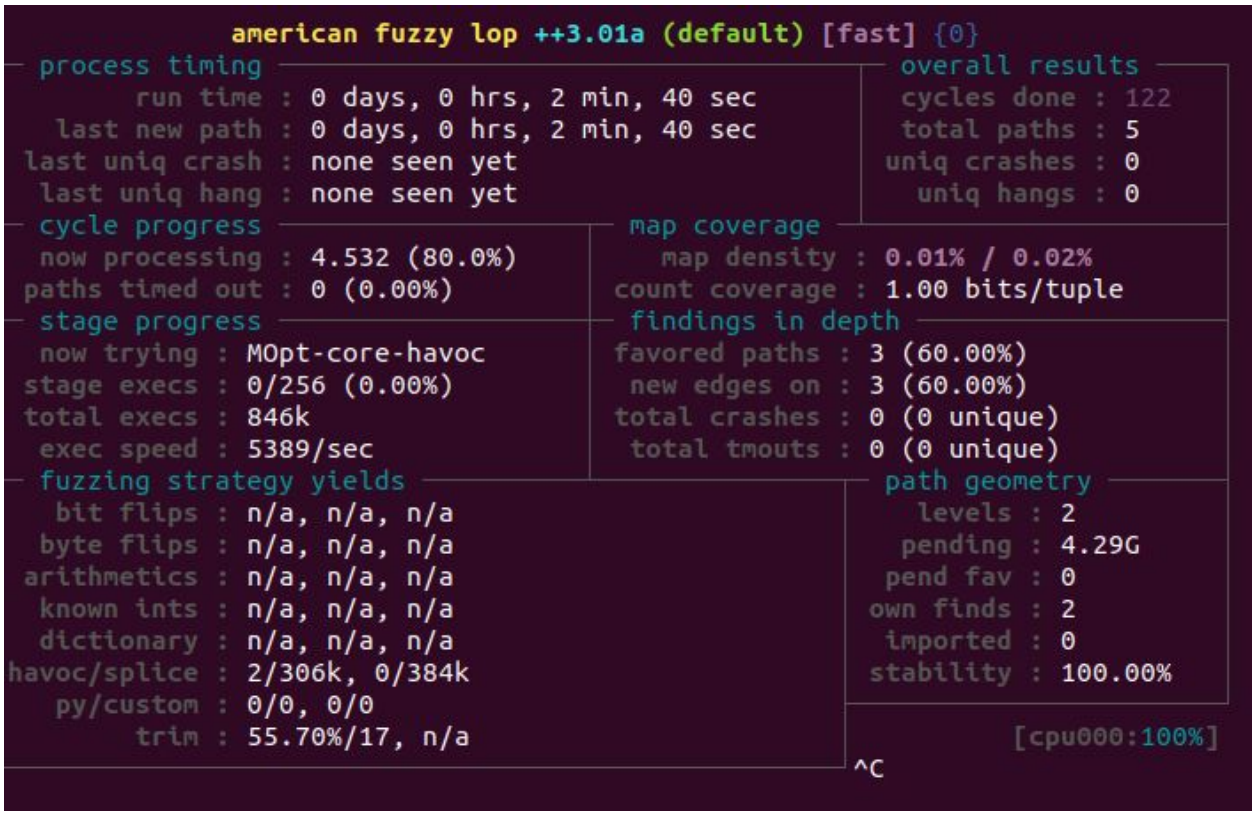
(2) describe how you fixed the bugs and why your fixes worked.

Compare the length of string N with digit N, if they are the same length, then it is an integer, otherwise it is not an integer

```
    int i;
    input[INPUTSIZE-1] = '\n';
    if (strncmp(input, "u ", 2) == 0)
    { // upper case command
            char *rest;
            char delim[] = " ";
            char *ptr0 = strtok(input, delim);
            char *ptr1 = strtok(input, delim);
            int n = atoi(ptr1);
            int count = 0;
            while (n != 0) {
                    n /= 10;      // n = n/10
                    ++count;
            }
            if ( strlen(ptr1 ) != count) {
                    printf("its not an integer\n");
                    return 1;
```

Round 5:

For part 1:**Fuzz testing on a vulnerable program**



(1) how long was the fuzzing process;
2 mins 40 sec

(2) how many inputs AFL found that could make the program crash or hang.
none

For part 2: **Isolating and fixing the bugs**

(1) describe your process of identifying bugs from AFL-generated inputs and what bugs you found in the program;
N/A

(2) describe how you fixed the bugs and why your fixes worked.
N/A