C, Memory

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Announcements

- Homework 8 due tonight on Gradescope
- Homework 9 available soon
- Lab 10 tomorrow: Memory Errors

Common Memory Bugs (reading)

More on man pages

① man man : ask for the manual on the manual

② synopsis: very short overview.

③ something underlined, it's linking it back to something at the top.

④ It's divided in sections and there are 9 sections of the manual.

⑤ Mostly going to be in section 3. (Library calls).

⑥ Examples of how to use the man.

⑦ "/": slash for search. "/example". it will find example on the page. "n" will be the next, and "shift" is the previous one.

⑧ try "man printf", "man 3 printf" (check the printf in section 3). then seach for examples, you can find some examples for printf.

⑨ If you're not quite sure what you're looking for, use "-k" to do larger searches.

More on man pages

(10). man -k "square root" (find the instruction using keyword) - very fast.
it tells you some information of square root like sqrt function.

(11). man sqrt : This is the function of square root.

(12). man -f is equivalent to "whatis" (I know what I want, for example, I want to check printf in section 3. (man 3 printf)

(13). man -K : search all things in man, find where this word appears. (very slow).

$ man strsep.

Also maybe one char or an array of char

```
char *strsep(char **s, const char *d);
```

→ a pointer to one character. or an array of pointers

It's changable!

→ It could be:

ptr → ptr → char ⇒ a waste of space.

ptr → array of chars ✓

array → pointers to char

array of arrays of chars

} maybe not an array of sth. because there is no length provided. How many pointers do I have?

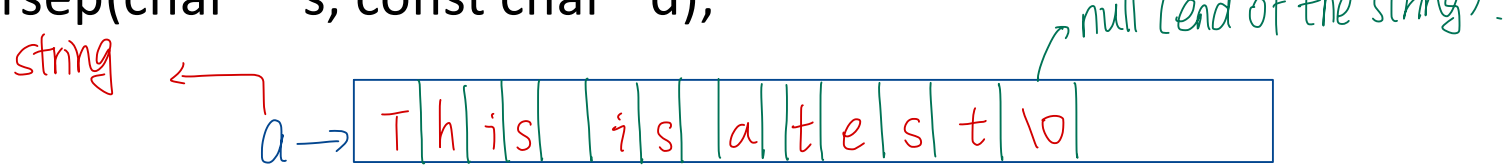
✅ When a function returns a pointer, it usually means TWO things:

① The function allocated memory for you (via malloc).
It created new memory on the heap.
Now you own it → you must free it later.
Example idea: makeList() returns a newly allocated list.

② The function is returning something you already own.
You gave it memory (a buffer, a struct, etc.).
It used or modified the memory.
It returns a pointer into that same memory.
You still free it only once, when you're done.

❌ What it NEVER means:
It NEVER returns a pointer to a local variable on the stack.
Stack memory disappears after the function ends → pointer becomes invalid.

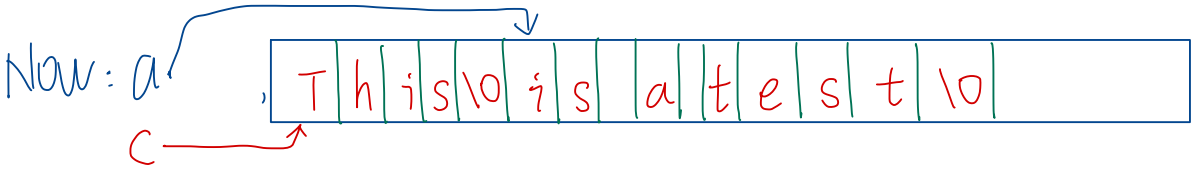
char *strsep(char **s, const char *d);



I should tell it where the variable "a" lives so that they can change it.

c = strsep(&a, b)

run through the string "a" and look for anything in "b" so it will find the first space in "a" (which is what "b" has) and then replace it with "\0"



char *strsep(char **s, const char *d);

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

int main() {
    char *s = "This is a test";
    char *d = " ";
    char *token = strsep(&s, d);
    puts(token);
    return 0;
}
```

segmentation fault! Why?

"char *s" is a string → read only memory.

so we need to put this string on heap.

char *strsep(char **s, const char *d);

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

int main() {
    char *s = strdup("This is a test");
    char *d = " ";
    char *token = strsep(&s, d);
    puts(token);
    return 0;
}
```

malloc some space for me, put the string there, and gave me a ponter to that string.

Everything good now?

No. I forget to free it.

char *strsep(char **s, const char *d);

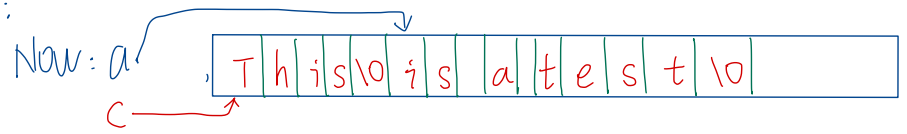
```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main() {
    char *s = strdup("This is a test");
    char *d = " ";
    char *token = strsep(&s, d);
    puts(token);
    free(s);
    return 0;
}
```

How about this version?

```
This
free(): invalid pointer
Aborted (core dumped)
```

recall:



What we malloc at very begining (a) now points to other place. No longer the same pointer we got from malloc() So I can not free it because malloc didn't give me that pointer.

char *strsep(char **s, const char *d);

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

int main() {
    char *s = strdup("This is a test");
    char *deleteme = s;
    char *d = " ";
    char *token = strsep(&s, d);
    puts(token);
    puts(s);
    free(deleteme);
    return 0;
}
```

We use a new pointer to remember the address of original "a". Now it works.

char *strsep(char **s, const char *d);

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

int main() {
    char *s = strdup("This is a test");
    char *deleteme = s;
    char *d = " ";
    while (s) {
        char *token = strsep(&s, d);
        puts(token);
    }

    free(deleteme);
    return 0;
}
```

It will give you:

```
This
is
a
test
```

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

int main() {
    char *s = strdup("This is a  test");
    char *deleteme = s;
    char *d = " ";
    while (s) {
        char *token = strsep(&s, d);
        puts(token);
    }

    free(deleteme);
    return 0;
}
```

It will give you:

```
This
is
a

test
```

All the processes running on the machine : ps -A | less

How many things are running ? ps -A | wc -l (All of them have a view of memory as if they were the only program running in memory.

Processes

Process - approximately what we think of as a "running program"

- Operating System effectively has a giant array of processes started since computer turned on
- Try `ps -A`
- Has access to all memory (but only its own!)
- Operating System maintains data structure about each process
  - What program is running, who ran it, when it started, …
  - Array of "file like objects"

1. [18 points] Consider the following C code. Write the function `examtwo()` in x86-64 assembly. Your function should store x in register `rax` and y in register `rbx`. When using 32-bit values, use the correct 32-bit instruction and register.

```
1  long mathfun(long x, int y);
2
3  long examtwo() {
4      long x = 0x42;          // store in rax
5      int y = 1;              // store in rbx
6      while (y <= 5) {
7          x = mathfun(x, y);
8          y++;
9      }
10     return x;
11 }
```

```
examtwo:
        pushq   %rbx
        movq    $0x42, %rax
        movl    $0x1, %ebx
before: cmpl    $5, %ebx
        jg      after
        movq    %rax, %rdi
        movl    %ebx, %esi
        callq   mathfun
        addl    $1, %ebx
        jmp     before
after:  popq    %rbx
        retq
```

For the next two questions, assume the first eight registers and the given segment of memory have the following values before the next few instructions.

Register	Value (hex)
rax	0x2130
rcx	0xc501e1f0
rdx	0x1000000088
rbx	0x4
rsp	0xc501e20b
rbp	0xc501e20f
rsi	0x67
rdi	0xf000000ff

Mem Addr.	Value (hex)
0xc501e201	0x01
0xc501e202	0xe2
0xc501e203	0x01
0xc501e204	0x50
0xc501e205	0xff
0xc501e206	0xa0
0xc501e207	0x04
0xc501e208	0x34
0xc501e209	0x12

Mem Addr.	Value (hex)
0xc501e20a	0xb2
0xc501e20b	0x59
0xc501e20c	0x01
0xc501e20d	0x46
0xc501e20e	0xcf
0xc501e20f	0xea
0xc501e210	0x65
0xc501e211	0x7d
0xc501e212	0x4e

2. [16 points] Which program registers are modified, and to what values, by the following instructions? Leave spaces blank if fewer registers change than there are lines. If no registers are changed, write "none" in the first register box with no new value. *Each instruction below is independent; do not use the result of one as input for the next.* (4 points each)

`popl %eax`

Register	New Value
%rax	0xcf46b159
%rsp	0xc501e20f

(popl is 4 bytes b+4=f)

`andl %edi, %edx` (0xff and 0088)

Register	New Value
%edx	0x88

0xc501e20b q: 8 bytes

`leaq 0x6(%rcx, %rbx, 4), %rdi`

Register	New Value
%rdi	0xc501e20b

`movq 0x6(%rcx, %rbx, 4), %rdi`

Register	New Value
%rdi	0x46b159b21234d4a0

0x6+(%rcx+%rbx*4)

0x4 *4 = 0x10

↓

0xc501e1f0

For the next two questions, assume the first eight registers and the given segment of memory have the following values before the next few instructions.

Register	Value (hex)
rax	0x2130
rcx	0xc501e1f0
rdx	0x100000088
rbx	0x4
rsp	0xc501e20b
rbp	0xc501e20f
rsi	0x67
rdi	0xf000000ff

Mem Addr.	Value (hex)
0xc501e201	0x01
0xc501e202	0xe2
0xc501e203	0x01
0xc501e204	0x50
0xc501e205	0xff
0xc501e206	0xa0
0xc501e207	0x04
0xc501e208	0x34
0xc501e209	0x12

Mem Addr.	Value (hex)
0xc501e20a	0xb2
0xc501e20b	0x59
0xc501e20c	0x01
0xc501e20d	0x46
0xc501e20e	0xcf
0xc501e20f	0xea
0xc501e210	0x65
0xc501e211	0x7d
0xc501e212	0x4e

16 bits in binary → 4 bits in hex

3. [6 points] Assume we stored an array of 16-bit unsigned numbers that started at memory address `0xc501e20a` (i.e., the far right column). Write the hexadecimal values of the first two numbers in the array, if the array was stored in memory using:

A. little-endian storage.

0x5962	0x4601

B. big-endian storage.

0x6259	0x0146

4. [12 points] True/False questions on our discussion of a backdoor in our Toy ISA processor. For each of the statements below, fill in the **T** circle *completely* if you think the statement is True. If you think it's False, fill in the **F** circle *completely*.

   (F) A backdoor can allow others to take control of your computer without your knowledge.

   (T) Our exploit only provided a way to execute code that was programmed into the hardware.

   (F) Our backdoor's malicious code is executed when the user's program read the bytes of the passcode in order.

   (T) Our backdoor required additional gates for calculation, which would noticeably slow the normal operation of the processor.

5. [6 points] We discussed copyrights and patents in class. Assume that Jane has written a new program—using a novel approach—to solve matrix multiplication faster than anyone else. Which of the following would allow Jane to protect her intellectual property? *Fill in the circle complete for all that apply.*

   ○ The implementation (source code) is immediately copyrighted, preventing others from using it.

   ● Jane patents her algorithm, requiring anyone who wants to use it to pay royalties.

   ● Jane does not release the source code, but requires users to accept a license agreement restricting re-distribution.

   ○ Jane publishes her code on her blog, including a copyright notice and text describing limits on re-using the code.

6. [4 points] Suppose you have a file named `hello.c`, which includes a `main()` function. What command on the CS portal would you use to compile this C code to an executable, with a moderate level of optimization (i.e., 2 levels or level 2), so that it can be run later as `./a.out`?

Answer
clang -O2 hello.c

7. [6 points] True/False questions. For each of the statements below, fill in the **T** circle *completely* if you think the statement is True. If you think it's False, fill in the **F** circle *completely*.

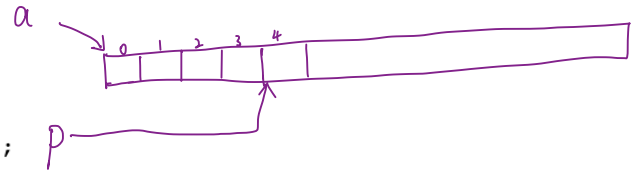
(T) (●) An error will occur in the **type-checking** stage of compilation when compiling our code if we call the function `csoprintf()` which does not exist. linking time

(●) (F) If we accidentally wrote Java instead of C when initializing an array, an error would occur in the **parsing** stage of compilation: `int[100] arr = new arr[100];`

All grammar related things → parsing

For the next two questions, consider the following C code snippet.

```
1  void addressfun() {
2      int a[10];
3      int *p = a + 4;
4      __________ = 42;
5      size_t asize = sizeof(a);
6      return;
7  }
```



8. [6 points] Which of the expressions below could be used in the blank on line 4 to set the 5th element in `a` (i.e., `a[4]`) to 42? *Fill in the circle completely for all that apply.*

- (●) `*(a + 4)`
- (●) `p[0]`
- (●) `*p`
- ( ) `*(a + 16)`
- ( ) `*a + 16`
- ( ) `p[3]`

9. [4 points] What value is stored in `asize` on line 5?

Answer
40

4x10

For the next question, consider the following struct and function header.

```
1  typedef struct {
2      char *name;
3      unsigned long age;
4      double latitude;
5      double longitude;
6  } person;
7
8  int locate(person waldo);
```

10. [4 points] When the code is compiled to x86-64 assembly, where is the first parameter to the function `locate` placed when calling the function?

- ○ In register `%rdi`
- ○ In register `%rax`
- ● On the stack
- ○ This is not valid C

It XORs together:
① all the numbers that should appear, from 0 to n.
② all the numbers that actually apper in the array.
Since every number except the missing one appears once in both groups, those matching values cancel out under XOR. The only value left at the end is the missing number.

11. [18 points] Write a C function named `missingNumber` that takes two parameters (in order): an array of integers named `nums` given as a pointer, and an integer `n`; it should return an integer.

`nums` points to a valid array of length `n`, which contains `n` distinct integers selected from the range `0` through `n`, inclusive, in random order. Exactly one integer in that range is missing. Your function should find and return the missing integer. If `n = 0`, return `-1`.

Do not allocate another array or modify the contents of `nums`.

For example: given `nums = {3,0,1}` and `n = 3`, return 2; given `nums = {1,4,3,2}` and `n = 4`, return 0; given `nums = {}` and `n = 0`, return `-1`.

method 1:

```
int missingNumber(int *nums, int n){
    for(int x=0; x<=n; x++){
        int found=0;
        for(int i=0; i<n; i++){
            if(nums[i]==x){
                found=1;
            }
        if(!found){ return x; }
        }
    }
    return -1;
}
```

method 2:

```
int missingNumber(int *nums, int n){
    int expected = n*(n+1)/2;
    int actual=0;
    for(int i=0; i<n; i++){
        actual+=nums[i];
    }
    return expected-actual;
}
```

method 3:

```
int missingNumber(int *nums, int n){
    int x=n;
    for(int i=0; i<n; i++){
        x^=i;
        x^=nums[i];
    }
    return x;
}
```

why? $a \wedge a = 0$, $a \wedge 0 = 1$