COMPUTER SYSTEMS AND ORGANIZATION

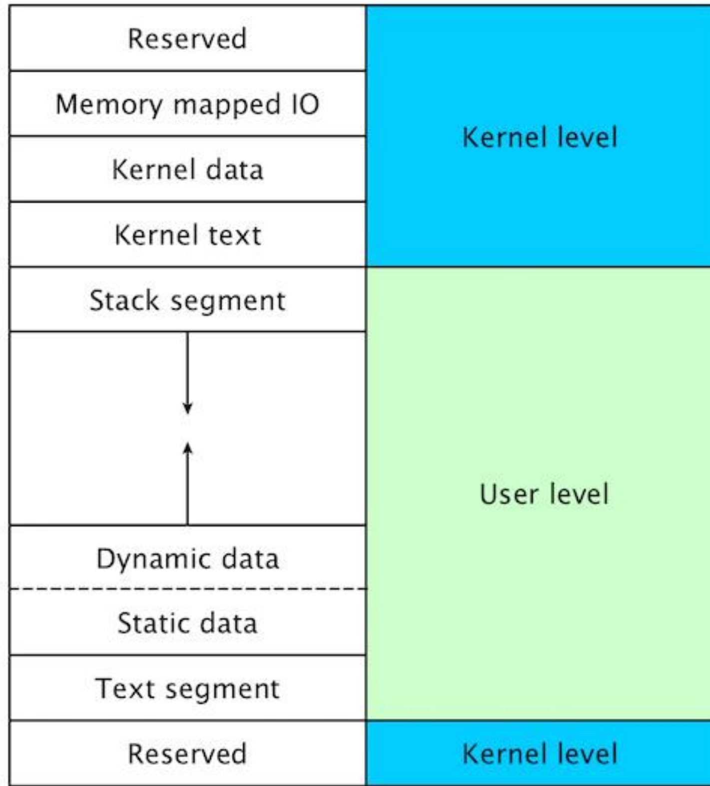
Generic Swap, Memcmp

Daniel G. Graham Ph.D

ENGINEERING

1. Syscall Continued
2. Being careful with C functions fscanf, example
3. There are some function you should never use.
4. Generic Swap
5. Memcp, and other mem operations
6. Memory Error Puzzle strsep

USER SPACE VS KERNEL SPACE LINUX



Kernel layout for MIPS chips

https://www.it.uu.se/education/course/homepage/os/vt18/module-0/mips-and-mars/mips-memory-layout/

The layout of the arm chips can be found here.
https://www.kernel.org/doc/html/v5.7/arm/memory.html

UNIVERSITY of VIRGINIA | ENGINEERING

BUT REMEMBER OUR GOAL IS TO BE ABLE REASON FROM FIRST PRINCIPLES ABOUT THIS:

```
#include <stdio.h>

int main(){
    printf("Hello World !\n");
}
```

Not quite there yet. Let's think about a simpler example.

UNIVERSITY of VIRGINIA | ENGINEERING

WHAT ABOUT THIS

```
#include <stdio.h>

int main(){
    putchar(65); //putchar takes int
                 //A is 65 in decimal
}
```

- How is this implemented?
- Yes it calls the putchar function implemented in stdio.c.
- But what assembly instructions eventually get run?
- Notice we didn’t print the new line char (Important!!)

University of Virginia ENGINEERING

WHAT ABOUT THIS

```
#include <stdio.h>

int main(){
    putchar(65); //putchar takes int
                 //A is 65 in decimal
     putchar(10);//New line \n
}
```

- How is this implemented?
- Yes it calls the putchar function implemented in stdio.c.
- But what assembly instructions eventually get run?
- Now we've added new line char print

University of Virginia | ENGINEERING

```
GNU nano 6.3
.global main



putchar:
        movq %rdi,-0x8(%rsp)    #place param1 on the stack
        movq $1, %rdx           # message length
        leaq -0x8(%rsp),%rsi    # address of message to write
        movq $1,%rdi            # file descriptor (stdout)
        movq $1,%rax            # system call number (sys_write)
        syscall                 # call kernel aka int 0x80
        ret

main:
        movq $65, %rdi          #mov decimal for A into 1st param
        call putchar
        movq $10, %rdi          #mov /n into 1st parm
        call putchar
        xorq %rax, %rax
        ret

^G Help       ^O Write Out   ^W Where Is   ^K Cut
^X Exit       ^R Read File   ^\ Replace    ^U Paste
```

```
dgg6b@portal05:~/CSO-Code-Examples/SystemCalls$ clang write.s
dgg6b@portal05:~/CSO-Code-Examples/SystemCalls$ ./a.out
A
dgg6b@portal05:~/CSO-Code-Examples/SystemCalls$
```

HERE IS SOME ASSEMBLY THAT PRINTS A

UNIVERSITY of VIRGINIA | ENGINEERING

```
 GNU nano 6.3
.global main


putchar:
        movq %rdi,-0x8(%rsp)    #place param1 on the stack
        movq $1, %rdx           # message length
        leaq -0x8(%rsp),%rsi    # address of message to write
        movq $1,%rdi            # file descriptor (stdout)
        movq $1,%rax            # system call number (sys_write)
        syscall                 # call kernel aka int 0x80
        ret

main:
        movq $65, %rdi          #mov decimal for A into 1st param
        call putchar
        movq $10, %rdi          #mov /n into 1st parm
        call putchar
        xorq %rax, %rax
        ret

^G Help      ^O Write Out   ^W Where Is   ^K Cut
^X Exit      ^R Read File   ^\ Replace    ^U Paste
```

```
dgg6b@portal05:~/CSO-Code-Examples/SystemCalls$ clang write.s
dgg6b@portal05:~/CSO-Code-Examples/SystemCalls$ ./a.out
A
dgg6b@portal05:~/CSO-Code-Examples/SystemCalls$
```

So many questions.

What is a file descriptor?

Why are we moving 1 into rax?

What is syscall?

What is a system call number?

Let’s start with: What is a file descriptor?

SYSTEM CALL CALLING CONVENTION

1. **Register Usage for Arguments**:
   1. %rax: System call number. Each system call has a unique number that you place in this register to tell the kernel which system call you're making.
   2. %rdi, %rsi, %rdx, %r10, %r8, %r9: Used for passing up to six arguments to system calls. %rdi is for the first argument, %rsi for the second, and so on. If a system call needs more than six arguments, a pointer to a block containing the arguments is passed as one of these registers.
2. **Making the System Call**:
   1. The syscall instruction is used to switch to kernel mode and invoke the system call. The kernel examines the value in %rax and understands which system call is being requested.
3. **Return Value**:
   1. After the system call, the return value is placed in %rax. This value typically indicates success or an error code.

UNIVERSITY of VIRGINIA | ENGINEERING

THING ABOUT HOW YOU IMPLEMENT THE WRITE SYSTEM CALL TO STDOUT

```
write(1, message, message_length);
```

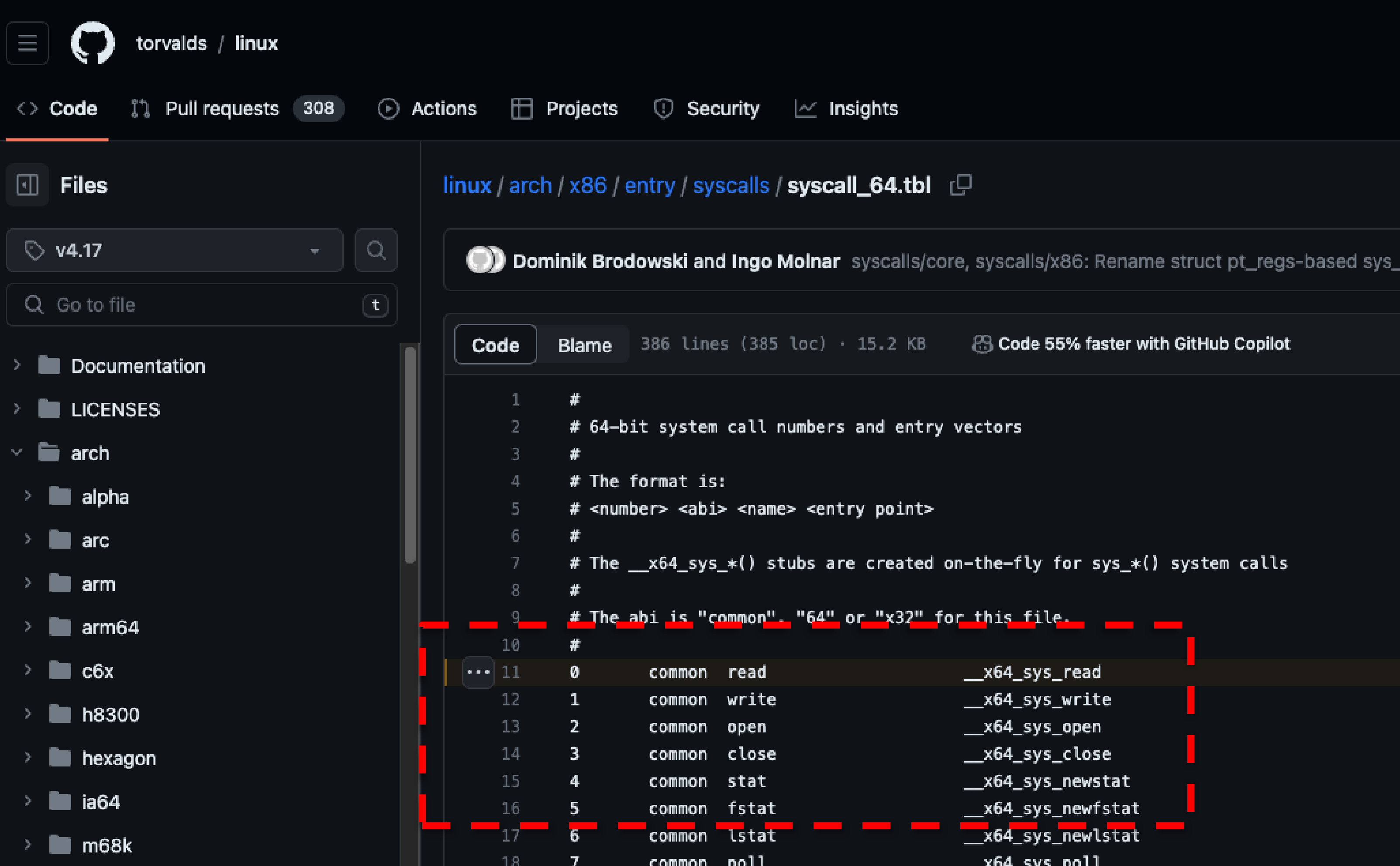
1. **Register Usage for Arguments**:
    1. %rax: System call number. Each system call has a unique number that you place in this register to tell the kernel which system call you're making.
    2. %rdi, %rsi, %rdx, %r10, %r8, %r9: Used for passing up to six arguments to system calls. %rdi is for the first argument, %rsi for the second, and so on. If a system call needs more than six arguments, a pointer to a block containing the arguments is passed as one of these registers.
2. **Making the System Call**:
    1. The syscall instruction is used to switch to kernel mode and invoke the system call. The kernel examines the value in %rax and understands which system call is being requested.
3. **Return Value**:
    1. After the system call, the return value is placed in %rax. This value typically indicates success or an error code.

University of Virginia ENGINEERING

SYSTEM CALL CALLING CONVENTION

```
.global _start
.text
_start:
    # write(1, message, 18)
    mov     $1, %rax               ; syscall number for write (1)
    mov     $1, %rdi               ; file descriptor 1 (stdout)
    lea     message(%rip), %rsi    ; load the address of the message
    mov     $18, %rdx              ; message length (18 bytes)
    syscall                        ; perform the system call


.section .rodata                   ; Read-only data section
message:                           ; Label for the message
    .ascii  "Computer Systems 1" ;
```

WHERE CAN I FIND THE SYSTEM CALL NUMBERS



Linux github repo.

https://github.com/torvalds/linux/blob/v4.17/arch/x86/entry/syscalls/syscall_64.tbl

ENGINEERING

SYSTEM CALLS

```
int main() {
    int fd;
    char *text = "CS01";

    // Open a file for writing (create it if it doesn't exist)
    fd = open("output.txt", O_WRONLY | O_CREAT, 0644);
```

User space

Returns file descriptor

System Call Interface

Kernel space

Call #	Function pointer
0	read
1	write
2	open
3	close

open()
implementation of open
file descriptor setup etc.

return

WHAT DOES THE FOLLOWING ASSEMBLY DO?

```
.global _start
.text
_start:
    # What does this snippet of assembley do?
    mov     $3, %rax            ;
    mov     $1, %rdi            ;
    syscall                     ;
```

Call #	Function pointer
0	read
1	write
2	open
3	close

A. Write Perror
B. Write stdout
C. Open stdout
D. Open Perror
E. Read from Perror
F. Close Perror
G. Read stdout
H. Close stdout
I. Read stdin
J. Close std in

UNIVERSITY of VIRGINIA ENGINEERING

A SIMPLE PRINT EXAMPLE

int printf(const char *format, ...);

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

int main()
{
      char cs_array[] = “C.S.O is fun”;
      printf(“%s”, cs_array);
      printf(“\n”);

}
```

What gets printed?

C.S.O is fun

UNIVERSITY of VIRGINIA | ENGINEERING

SOMETIMES C FUNCTIONS AREN'T INTUITIVE

Let's start by looking at fscanf

```
int fscanf(FILE *stream, const char *format, ...);
```


UNIVERSITY of VIRGINIA | ENGINEERING

WHAT DO WE THINK THE FOLLOWING SNIPPET OF CODE DOES

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

int main()
{
	char line[255];
	FILE *fptr = fopen("haiku.txt", "r");
	fscanf(fptr, "%s", line); // What does line contain?
	printf("%s", line);     // What does this line print?
	print("\n");

}
```

haiku.txt

C.S.O. is fun
Knowledge blooms as the end looms
It is almost done

University of Virginia | ENGINEERING

~ — ssh portal.cs.virginia.edu

```
GNU nano 6.3                    fprintfExample.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        char line[255];
        FILE *fptr = fopen("haiku.txt", "r");
        fscanf(fptr, "%s", line); // What does line contain?
        printf("%s", line);// What this line print
        printf("\n");

}
```

~ — ??M? — -zsh

```
dgg6b@portal08:~/Lecture-Code/lecture-34$ clang fprintfExample.
dgg6b@portal08:~/Lecture-Code/lecture-34$ ./a.out
C.S.O.
dgg6b@portal08:~/Lecture-Code/lecture-34$
```

Wait what it just prints C.S.O

Why….?

haiku.txt

C.S.O. is fun
Knowledge blooms as the end looms
It is almost done

LET'S CHECK THE DOCUMENTATION

```
X      Equivalent to x.

f      Matches an optionally signed floating-point number; the next pointer
       must be a pointer to float.

e      Equivalent to f.

g      Equivalent to f.

E      Equivalent to f.

a      (C99) Equivalent to f.

s      Matches a sequence of non-white-space characters; the next pointer
       must be a pointer to the initial element of a character array that
       is long enough to hold the input sequence and the terminating null
       byte ('\0'), which is added automatically.  The input string stops
       at white space or at the maximum field width, whichever occurs
       first.
```

Different behavior
for fscanf
And printf

UNIVERSITY of VIRGINIA | ENGINEERING

GENERAL GUIDANCE ON CHOOSING FUNCTIONS

Read the documentation closely ☺

That’s it. C was first so let’s learn to love all its quirks and features and then program in RUST ☺

UNIVERSITY of VIRGINIA | ENGINEERING

SWAP

```c
#include <stdio.h>

void swapShorts(short *x, short *y) {
    short temp = *x;
    *x = *y;
    *y = temp;
}

int main() {
    short a = 10, b = 20;
    printf("Before swapping: a = %d, b = %d\n", a, b);

    swapShorts(&a, &b);

    printf("After swapping: a = %d, b = %d\n", a, b);

    return 0;
}
```

SWAP TWO STRINGS

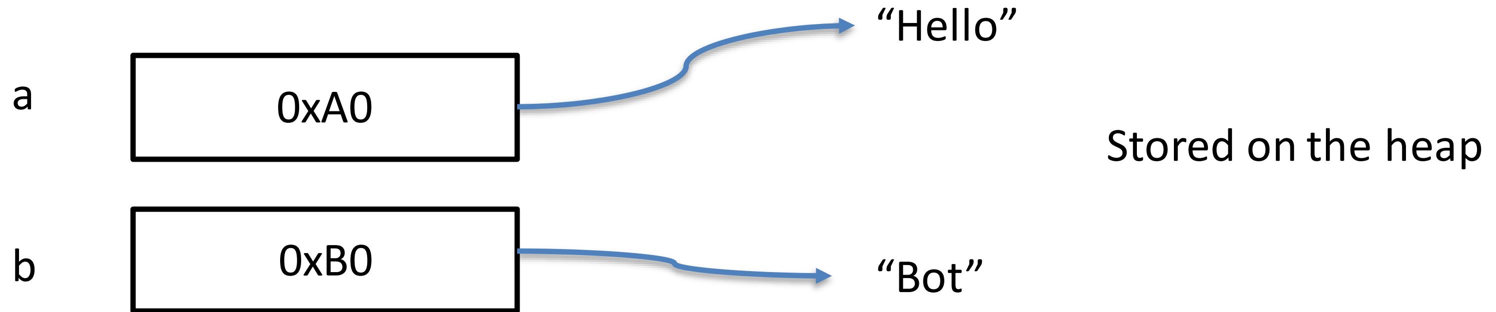
How could you do this?

UNIVERSITY of VIRGINIA | ENGINEERING

SWAP TWO STRINGS

How could you do this?
Just swap the value of the pointers ;)

```
char* a = strdup("Hello");
char* b = strdup("Bot");
```

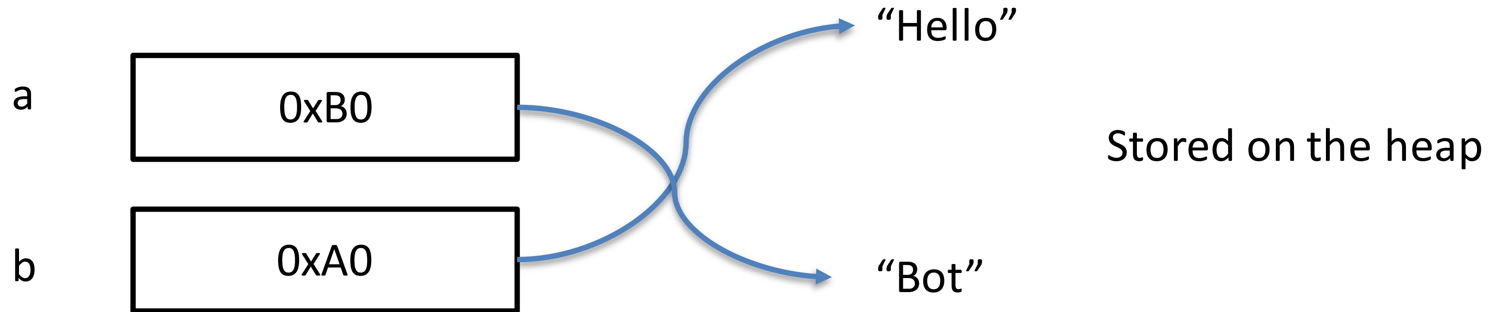


UNIVERSITY of VIRGINIA | ENGINEERING

SWAP TWO STRINGS

How could you do this?
Just swap the value of the pointers ;)

```
char* a = strdup("Hello");
char* b = strdup("Bot");
```



UNIVERSITY of VIRGINIA ENGINEERING

SWAP STRINGS

```
#include <stdio.h>

void swapStrings(char **str1, char **str2) {
    char *temp = *str1;
    *str1 = *str2;
    *str2 = temp;
}

int main() {
    char *a = "Hello";
    char *b = "Bot";

     printf("Before swapping: a = %s, b = %s\n", a, b);
     swapStrings(&a, &b);
     printf("After swapping: a = %s, b = %s\n", a, b);

    return 0;
}
```


University of Virginia | ENGINEERING

GENERIC SWAP

Could we write a generic that could pass any two pointers and then have the memory pointer swap

UNIVERSITY of VIRGINIA | ENGINEERING

MEMCPY

memcpy is a function in the C standard library, defined in the header file <string.h>. It is used to copy a specified number of bytes from one memory location to another. However, we should avoid using it in overlapping regions as it will result in undefined behavior that could possibly break our code.

```
void *memcpy(void *dest, const void *src, size_t n);
```

- dest: Pointer to the destination array where the content is to be copied.
- src: Pointer to the source of data to be copied.
- n: Number of bytes to copy.
- Returns a pointer to dest.

UNIVERSITY of VIRGINIA | ENGINEERING

MEMCPY EXAMPLE

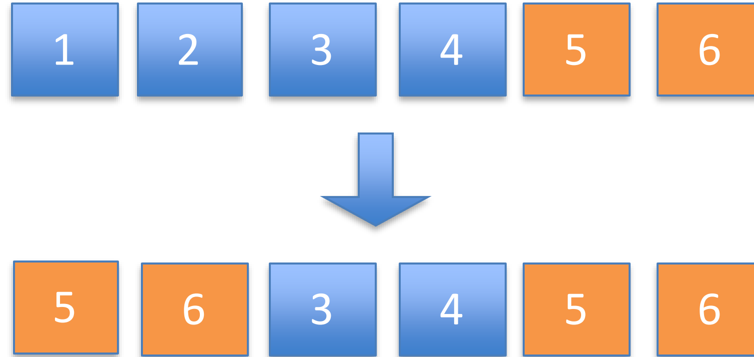
```
#include <stdio.h>
#include <string.h>

int main() {
    char src[50] = "This is the source string.";
    char dest[50];

    // Copy src to dest
     // Added to make we get the null terminator
    memcpy(dest, src, strlen(src) + 1);
    printf("dest = \"%s\"\n", dest);

    return 0;
}
```

MEMMOVE

It is similar to memcpy, but the key difference is that memmove is safe to use when the source and destination memory regions overlap. This is because memmove takes care of the possibility of overlapping regions by ensuring that the copying of bytes does not interfere with the original content in the case of overlap.

```
void *memmove(void *dest, const void *src, size_t n);
```

MEMMOVE



University of Virginia | ENGINEERING

TALK TO YOUR NEIGHBOR

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

int main() {
    char str[50] = "Hello, World!";

    printf("Original string: %s\n", str);
    memmove(str + 7, str + 0, 5);

    printf("After memmove: %s\n", str);

    return 0;
}
```

What get's printed after memmove?

University of Virginia | ENGINEERING

TALK TO YOUR NEIGHBOR

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

int main() {
    char str[50] = "Hello, World!";

    printf("Original string: %s\n", str);
    memmove(str + 7, str + 0, 5);

    printf("After memmove: %s\n", str);

    return 0;
}
```

What get’s printed after memmove?

Answer: Hello, Hello!

UNIVERSITY of VIRGINIA | ENGINEERING

BACK TO GENERIC SWAP

```
void swap(pointer to data1, pointer to data2) {
        store a copy of data1 in temporary storage
        copy data2 to the location of data1
        copy data1 in temporary storage to the location of data2
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr) {
        store a copy of data1 in temporary storage
        copy data2 to the location of data1
        copy data1 in temporary storage to the location of data2
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr) {
        store a copy of data1 in temporary storage
        copy data2 to the location of data1
        copy data1 in temporary storage to the location of data2
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr) {
        store a copy of data1 in temporary storage
        copy data2 to the location of data1
        copy data1 in temporary storage to the location of data2
}
```

If we don't know the data type, we don't know how many bytes it is.
Let's take that as another parameter.

UNIVERSITY of VIRGINIA | ENGINEERING

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {
        store a copy of data1 in temporary storage
        copy data2 to the location of data1
        copy data1 in temporary storage to the location of data2
}
```

If we don't know the data type, we don't know how many bytes it is. Let's take that as another parameter.

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {
	//store a copy of data1 in temporary storage
	void *temp = malloc(nbytes);
	memcpy(temp, data1ptr, nbytes);
	copy data2 to the location of data1
	copy data1 in temporary storage to the location of data2
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {
        void *temp = malloc(nbytes);
        memcpy(temp, data1ptr, nbytes);
        copy data2 to the location of data1
        copy data1 in temporary storage to the location of data2
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {
	void *temp = malloc(nbytes);
	memcpy(temp, data1ptr, nbytes);
	memcpy(data1ptr, data2ptr, nbytes);
	copy data1 in temporary storage to the location of data2
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {
	void *temp = malloc(nbytes);
	memcpy(temp, data1ptr, nbytes);
	memcpy(data1ptr, data2ptr, nbytes);
	copy data1 in temporary storage to the location of data2
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {
        void *temp = malloc(nbytes);
        memcpy(temp, data1ptr, nbytes);
        memcpy(data1ptr, data2ptr, nbytes);
        memcpy(data2ptr, temp, nbytes);
}
```

Just one more thing.

UNIVERSITY of VIRGINIA | ENGINEERING

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {
        void *temp = malloc(nbytes);
        memcpy(temp, data1ptr, nbytes);
        memcpy(data1ptr, data2ptr, nbytes);
        memcpy(data2ptr, temp, nbytes);
        free(temp);
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

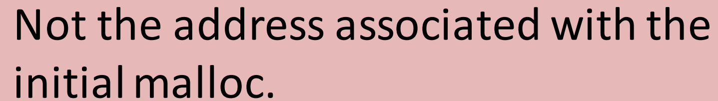
PUZZLE 1

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


int main(){
    int *x = (int *) malloc(40*sizeof(int));
    x+= 5;
    free(x);
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

PUZZLE 1

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


int main(){
    int *x = (int *) malloc(40*sizeof(int));
    x+= 5;
    free(x);
}
```



University of Virginia | ENGINEERING

ANOTHER FUN EXAMPLE

char *strsep(char **stringp, const char *delim);

stringp is a pointer to a pointer to the string that we want to parse

delim is a string that contains multiple delimiters

UNIVERSITY of VIRGINIA | ENGINEERING

STRSEP

char *strsep(char **stringp, const char *delim);

```
DESCRIPTION
       If  *stringp  is  NULL, the strsep() function returns NULL and does nothing
       else.  Otherwise, this  function  finds  the  first  token  in  the  string
       *stringp,  that is delimited by one of the bytes in the string delim.  This
       token is terminated by overwriting the delimiter with a null  byte  ('\0'),
       and  *stringp is updated to point past the token.  In case no delimiter was
       found, the token is taken to be the entire string *stringp, and *stringp is
       made NULL.
```


University of Virginia ENGINEERING

STRSEP

Draw visual.

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

int main() {
    char string[] = "a,b,c,d";  // The string to be tokenized
    char *token;
    char *rest = string;

    while ((token = strsep(&rest, ",")) != NULL) {
        printf("%s\n", token);
    }

    return 0;
}
```


University of Virginia | ENGINEERING

PUZZLE 2 DOES THIS CRASH OR NOT?

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    char *string = strdup("a,b,c,d”);  // stored on the heap
    char *token;
    token = strsep(&string, ",");
    printf("%s\n", token);
    free(string);
    return 0;
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

PUZZLE 2 DOES THIS CRASH OR NOT?

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    char *string = strdup("a,b,c,d”);  // stored on the heap
    char *token;
    token = strsep(&string, ",");
    printf("%s\n", token);
    free(string);
    return 0;
}
```

PUZZLE 2 DOES THIS CRASH OR NOT?

~ — ssh portal.cs.virginia.edu

```
 GNU nano 6.3                              strsepfree.c
include <stdio.h>
include <string.h>
include <stdlib.h>
nt main() {
   char *string = strdup("a,b,c,d");   // stored on the heap
   char *token;
   token = strsep(&string, ",");
   printf("%s\n", token);
   free(string);
   return 0;
```

~ — ??M? — -zsh

```
dgg6b@portal08:~/Lecture-Code/lecture-34$ clang -O3 strsepfree.c
dgg6b@portal08:~/Lecture-Code/lecture-34$ ./a.out
a
free(): invalid pointer
Aborted (core dumped)
dgg6b@portal08:~/Lecture-Code/lecture-34$
```


UNIVERSITY of VIRGINIA | ENGINEERING

PUZZLE 2 DOES THIS CRASH OR NOT?

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    char *string = strdup("a,b,c,d”);  // stored on the heap
    char *token;
    token = strsep(&string, ",");
    printf("%s\n", token);
    free(string);
    return 0;
}
```

Crashed because the string pointer is updated to a new address. It is not the original malloced address.

UNIVERSITY of VIRGINIA | ENGINEERING

PUZZLE 2 (HERE'S A FIX)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    char *string = strdup("a,b,c,d”);  // stored on the heap
    char *token;
    char *rest = string;
    token = strsep(&rest, ",");
    printf("%s\n", token);
    free(string);
    return 0;
}
```

Think about "rest" as if it was "temp". By using a "copy" of string and using it as the temporary value to operate on, we can keep the address of string the same as it was before as it is left unmodified. We would be able to successfully free it in this way.

University of Virginia | ENGINEERING

REFERENCES AND CREDIT

https://web.stanford.edu/class/archive/cs/cs107/cs107.1242/lectures/12/Lecture12.pdf

UNIVERSITY of VIRGINIA | ENGINEERING

University of Virginia | ENGINEERING