



Technical report

Operating Systems ST0257



This document by Juan Manuel Young Hoyos is for educational purposes only.
This document should not be printed or shared.

March 01 of 2022

Contents

1	General Research	2
1.1	Assignment	2
1.2	Conceptual Map	2
1.3	Timeline	2
2	Operating Systems Interruptions	4
2.1	What is an O.S. Interruption?	4
2.2	Buffer overflow	4
2.2.1	How can we avoid it?	6
2.3	Heap Memory Leak	6

1 General Research

The first activity is simple, the idea is just research about the history of the operating systems.

1.1 Assignment

- Make a conceptual map about operating systems.
- Make a timeline that reflects the history and generation of operating systems.

1.2 Conceptual Map

This is the conceptual map using draw.io:

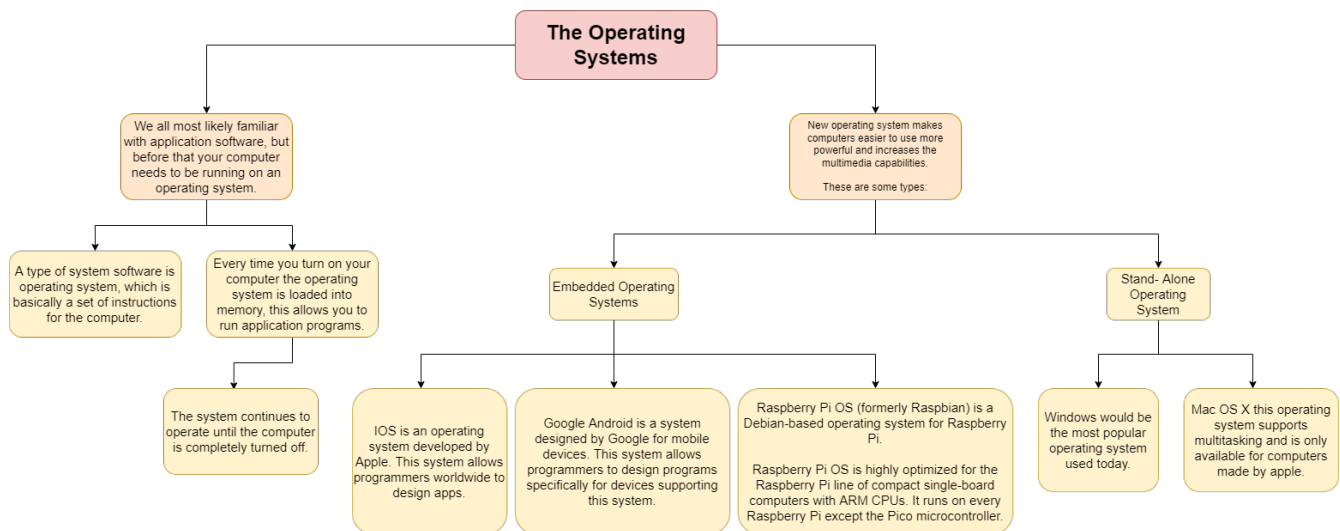


Figure 1: Operating Systems Conceptual Map

URL

[ConceptualMap.png](#) [GitHub file](#)

1.3 Timeline

This is the Operating System Timeline using draw.io:

URL

[OperatingSystemsTimeLine.png](#) [GitHub file](#)

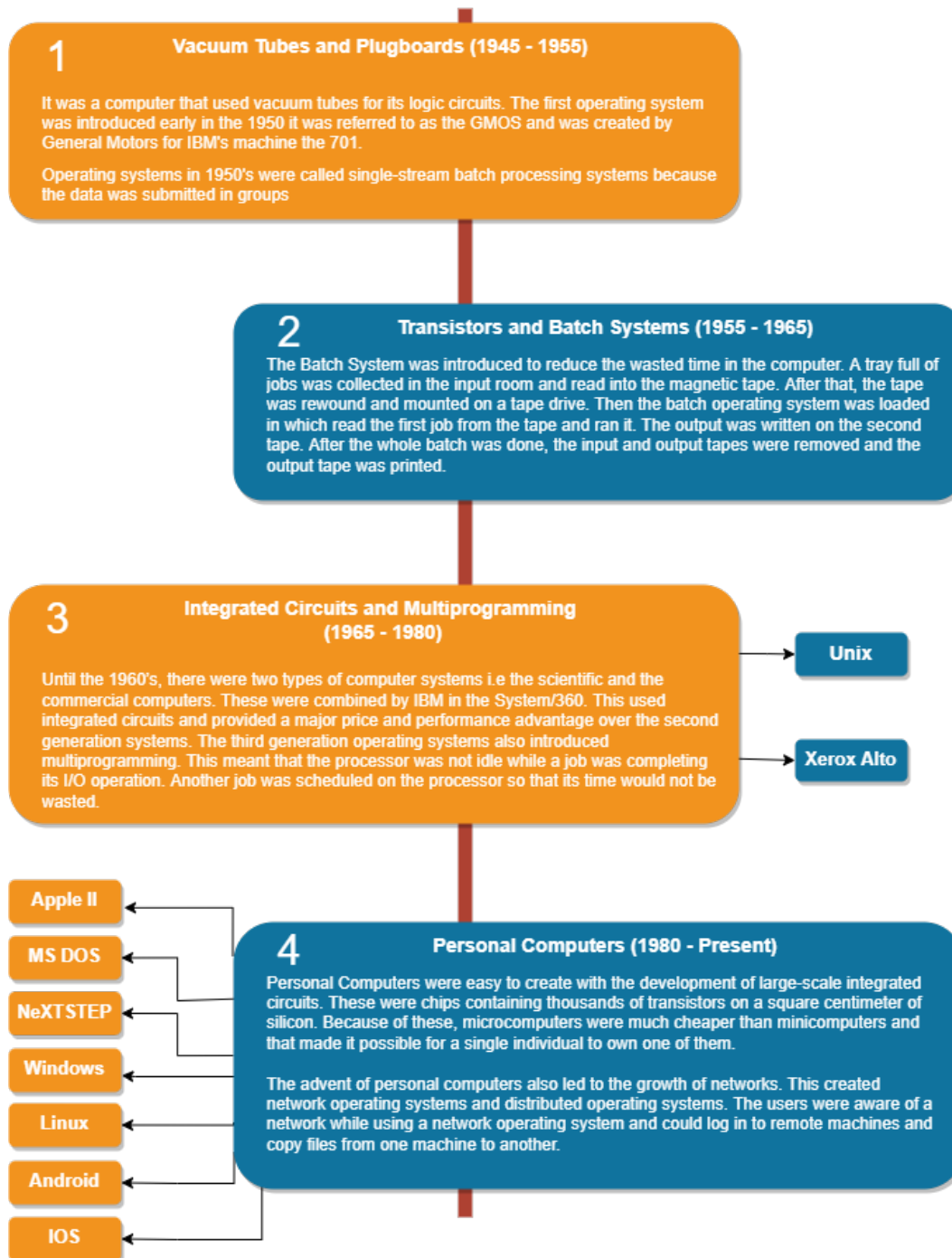


Figure 2: Operating Systems Timeline

2 Operating Systems Interruptions

The idea is just research and implementation of some of the operating system interruptions.

2.1 What is an O.S. Interruption?

An interrupt is a signal emitted by hardware or software when a process or an event needs immediate attention. It alerts the processor to a high-priority process requiring interruption of the current working process.

2.2 Buffer overflow

A buffer overflow occurs when a program or process attempts to write more data to a fixed-length block of memory, or buffer, than the buffer is allocated to hold. Buffers contain a defined amount of data; any extra data will overwrite data values in memory addresses adjacent to the destination buffer.

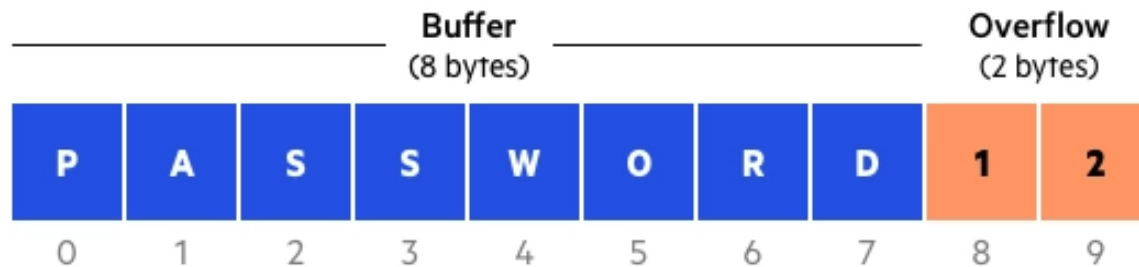


Figure 3: Buffer overflow diagram, image taken from imperva.com.

Now let's test it out with the language C and Rust. So we will have the following files for this test:

- ***Makefile***, this file allow us to build or clean the project.
- ***avoid memory leaks.rs***, this file shows an implementation of how Rust handles the memory leaks.
- ***buffer overflow.c***, this file shows an implementation of how to do a buffer overflow with C.
- ***get memory leaks.sh***, this script shows if Valgrind can help us with type of problems.

Now let's run the following *Makefile* with the command *make*, and we should get an executable.

```
1 # https://github.com/Youngermaster/ST0257-Operating-Systems/blob/main/Challenges/Challenge_1/
  BufferOverflow/Makefile
2
3 CC=gcc
4 CFLAGS=-g -Wall
5 EXE=buffer_overflow
6 REXE=avoid_memory_leaks
7
8 all: $(EXE)
9
10 %: %.c
11     $(CC) $(CFLAGS) $< -o $@
12
13 clean:
14     rm -rf *.o $(EXE)
15
16 rust:
17     rustc $(REXE).rs
18
19 cleanRust:
20     rm -rf *.o $(REXE)
```

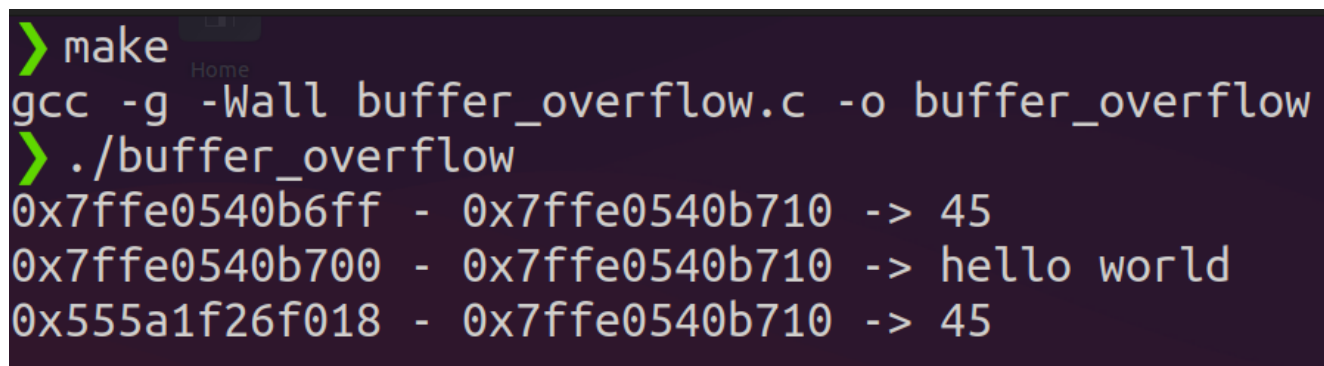
Code 1: This file allow us to build or clean the project.

The executable is made from the following code, where we are accessing to an outbound position.

```
1 // https://github.com/Youngermaster/ST0257-Operating-Systems/blob/main/Challenges/Challenge_1/
  BufferOverflow/buffer_overflow.c
2
3 #include <stdio.h>
4
5 int main(int argc, char const *argv[]) {
6     char *s = "hello world";
7     char c = s[20];
8     printf("%p - %p -> %d\n", &c, __builtin_frame_address(0), c);
9     printf("%p - %p -> %s\n", &s, __builtin_frame_address(0), s);
10    printf("%p - %p -> %d\n", &s[20], __builtin_frame_address(0), s[20]);
11    return 0;
12 }
```

Code 2: This file shows an implementation of how to do a buffer overflow with C.

Now, after the build we run it, and we get the following output:



```
> make
gcc -g -Wall buffer_overflow.c -o buffer_overflow
> ./buffer_overflow
0x7ffe0540b6ff - 0x7ffe0540b710 -> 45
0x7ffe0540b700 - 0x7ffe0540b710 -> hello world
0x555a1f26f018 - 0x7ffe0540b710 -> 45
```

Figure 4: Screenshot of the commands made in the terminal and the results.

We get the word "hello world", however, as we can see we got a random number, in this case the number 45. And what is the problem with that? The problem is that we are able to access to a random memory value, due to security issues and/or unexpected behaviours, imagine an airplane crash due to a little calculus modified by a Buffer Overflow.

2.2.1 How can we avoid it?

We have some options, but right now the most common are those two:

- We should pay attention to our C programs memory management.
- We can use another languages that solves that problem, using *Garbage Collectors*, *Borrowing and checkers*, etc.

Maybe a *Garbage Collector* approach is not a bad idea, however if we want maximum performance just maybe that is not the right path. Now let's check the *Borrowing and checkers* option with the language *Rust*. Now let's run the following code with the command `rustc ourRustFile.rs`

```
1 // https://github.com/Youngermaster/ST0257-Operating-Systems/blob/main/Challenges/Challenge_1/
  // BufferOverflow/avoid_memory_leaks.rs
2
3 fn main() {
4     // Fixed-size array (type signature is superfluous)
5     let xs: [i32; 5] = [1, 2, 3, 4, 5];
6
7     // Indexing starts at 0
8     println!("first element of the array: {}", xs[0]);
9
10    // ! Rust compilation will break this code immediately!
11    println!("first element of the array: {}", xs[20]);
12 }
```

Code 3: This file shows an implementation of how Rust handles the memory leaks.

And there is and interesting output, even if the *C* language allowed us to get a memory value far from our scope, although *Rust* before compilation does not allow us to compile "*bad code*".

```
>rustc avoid_memory_leaks.rs
error: this operation will panic at runtime
--> avoid_memory_leaks.rs:9:48
9 |     println!("first element of the array: {}", xs[20]);
  |                                         ^^^^^^ index out of bounds: the length is 5 but the index is 20
= note: `[deny(unconditional_panic)]` on by default
error: aborting due to previous error
```

Figure 5: Screenshot of the commands made in the terminal and the results of the Rust code.

2.3 Heap Memory Leak

Memory leak occurs when programmers create a memory in heap and forget to delete it. The consequences of memory leak is that it reduces the performance of the computer by reducing the amount of available memory. Eventually, in the worst case, too much of the available memory may become allocated and all or part of the system or device stops working correctly, the application fails, or the system slows down vastly.

Now let's test it out with the language C. So we will have the following files for this test:

- *Makefile*, this file allow us to build or clean the project.
- *buffer overflow.c*, this file shows an implementation of how to do a buffer overflow with C.
- *get memory leaks.sh*, this script shows if Valgrind can help us with type of problems.