28.06.22

**Callback Functions:** Callback functions will constantly monitor a certain entry point, although this doesnt mean they are polling. Inherently, they are polling, but async monitoring is a little bit different in essence. Instead of starting the executable one time and then running the function one time, you can create a function which constantly looks for a change in a certain event and this function will initiate a callback after the change has been registered. Passing a callback to a regular function is done via a function pointer.

If a callback is handled inside a separate thread which has spawned from the main thread, the values that the callback sees are based on the info that the current thread can see at the moment of invocation.

To ensure thread safety, all access to the global variables must be locked at the time of callback execution. This can be done by creating a mutex lock when inside the callback while access to global variables is at play.

Callback functions are misnomers. They should be named "call after" because they are called after the function they are passed into. The function automatically invokes the callback, when it itself finishes the function body, this means that it happens at the very end of the caller function.

In order to make callback functions traceable, name them similarly when it goes into the monitor and when it gets processed inside it.

Callback functions ideally return nothing to the main thread, ensuring that they stay non-blocking.

The callback function passed to the monitor function is via function pointer.

29.06.22

If you declare a pointer with null, you cant assign fuck all to it unless you define it. There has to be at least one time variable definition to the pointer, so that you dont get a null point deref error at the end.

If you null declare a type that contains multiple types inside it, defining it to avoid null pointer deref error may be tricky. You may need to individually instantiate the internals of the type.

If a function doesnt return but inherently changes the passed argument, the argument is passed as a reference or a pointer, which may be then used outside of the scope of the function because it is irrelevant to the function.

Multiple type variables must be defined on their outer shell first (either dynamically or reggie value assignment) so that inner fields obtained with **.** or **->** may be defined.

If you have a list of elements that you dynamically created, you can push\_back() them into a list, iterate over them and delete them one by one in a for loop for easier readability.

Imagine the main thread just going itss thing and you have defined a class in another cpp which wants to access the variables handled inside the main thread. If the class has member variables, when the main thread calls upon the class, the internal variables can be assigned to the member variables, filling them up with information, which may be queried at a later point.

Compiling an executable successfully doesnt mean that it wont crash. Some problems are only found in run-time like sigsegvs and pointer access violations. These can be prevented by compiler warnings written by people who foresee such errors.

30.06.22

Pay attention to how the threads wait for each other. Youre not thinking intuitively because you are not thinking in parallel. After a thread is finished with itss work, it will either store the results in a map type structure or will not do anything critical with the data, otherwise how can you be sure if you actually finished or not?

4.7.22

STL: a part of the standard library, just under a more specified name to confuse everybody cuz the people writing this language were retarded. It includes things like vector, deque, arrays.

If a function has been decorated with "constant", it won't modify objects or call any non-const member functions. It specifies that this function is "read-only". Const written after.

If a member is declared static, it can be created one time amongst many instantiations of the same class. All instantiations share the same static variable.

While writing a complex list of class functions, categorize them into high level meanings. There may be many categories depending on the algos ,but the followings must be standard in all:

* Constructor/destructor
* Setters/getters
* Thread
* Entry/exit points to the algorithms
* A callback function for various works
* Status update

Accessing functions through classes is architecture. This keeps everything intact and allows you to see things in compartments. Before starting any project, you need to create the scaffolding with the classes, housing the functions.

Encapsulation: binds together the data and the function that manipules it. It abstracts away the details. Getters and setters are the best example for encapsulation.

Standard library headers are extremely low level, dealing with the individual ones and zeroes. They are filled with preprocessor macroes.

11.7.22

**ALL CAPS** written variables are #define pre-processor macros.

**\_t** at the end of a variable is a typedeffed, and usually retardedly convoluted way of saying int,float,double,char and other basic types.

**Lwip**: lightweight ip stack, used on embedded systems for socket communications. It is an implementation of the TCP/IP suite. Aims to use fewer resources than normal.

**Port** is a logical construct, meaning that it doesn’t get reserved for any particular process, although a limited number of ports are reserved for other purposes and thus cant be accessed.

**Multiplexing** : different analog and digital streams of transmission are processed over a shared link (the same medium like going through the same box) the medium is shared, thus becomes low capacity, but we gain flexibility and multi-sending instead. Timing, synchronicity and bit manipulation are key points for multiplexing activities.

**Bit shifting:** the byte which is processed has no meaning as a whole but rather holds multiple meaningful bit structures, that have to be individually processed. The byte is treated as individual bits. When we want to send multi-faceted information over one byte chain, bit shifting will be done to decipher the individual information present inside. Also used in microcontrollers to describe flags which are represented by individual bits.

**pull-up/pull-down resistors:** ensures a known state for a signal by physically interrupting the connection of subsequent components to ground or to Vcc.

12.7.22

**Build order:** there are multiple ways to build c++ programs (apps, applications, processes, executables, binaries). Altogether, the steps fall into 3 categories: building static libs, dynamic libs and executables. You can build with the following tools:

* From the command line with specific user inputs
* Boost build system.
* IDE like MSVC, which lets you build the project/solution just with the click of a button, because all the steps are hidden from the user and/or presented to the user in the GUI, so you can change them easily.
* GNU make.

The compiler takes .cpp and .h files as input and produces object files, obj files contain a mixture of machine code and symbolic references to functions and data.

The archiver takes object files as input and produces a static library (or an archive) which is the collection of obj files grouped for convenient use.

The linker takes obj files and libraries and resolves their symbolic references to produce an executable or a dynamic library. It matches each use of a symbol to its definition.

A dynamic library is a body of machine code which gets loaded into memory and can be shared by multiple applications.

Obj and stat libs are needed only when the executable is built. Dyn libs must be present previously on the OS before the executable is run.^

You don’t wanna remember every step of the build process, so you use toolchains that do the job for you when you click the build button. This is virtually impossible if you have a complex project with multiple src, obj, a, dll files. You just cant keep everything in your head.

IDEs organize src into collections called projects. One project is associated with one binary file (variants of it are debug and release). Project collections go into a solution. A solution also describes the dependencies between projects.

A build system provides a text file, formatted to describe the collection of src and bins to be generated, a build tool that reads this formatted text file and generates the bins by invoking the command-line tools step by step.

Ides organize into projects, build systems organize into targets. Targets are usually the final bin files or they can also be further steps to install a program.

GNU make is the defacto standard build tool. It reads the formatted text file “makefile”. Steep learning curve especially for complex cxx programs.

The command line tools, which get invoked either by the IDE (by clicking build on Visual Studio) or by

calling make on the terminal, can be found in /usr/bin on UNIX, and in VC/bin on MS.

GCC and MSVC are top dogs in the game. They both implement the c++ standard library in their own ways, but are relatively similar.

In order for a symbol defined in a dyn lib to be available to code which uses this said dyn lib, the symbol must be exported when the dyn lib is built and imported when an executable or another dyn lib is built, which wants to use this said symbol.

**Import libraries:** static libs containing information needed to invoke functions in a DLL.

**Module definition:** with the extension of .def, this text file describes the funcs and data exported by the DLL.

You export functions from the DLL by manually writing the \_declspec keyword and “extern” while writing the dll.cpp, the tutorials are present on the web.

**Environment Variable**: pairs of strings maintained by the OS. Command-line tools refer to env var to learn details about the OS and to obtain configuration information that otherwise would have to be entered manually to the command line.

On MS, PATH is the upper most environment variable, where executables and binaries are looked for. the executable can be written with its short name, and the path variable will append the full path to it.

**RunTime Library:** while the application is running, it makes calls to the OS which in turn utilizes hardware. The context in which these calls must happen is done within the runtime library. UNIX has appropriate runtime libraries pre installed, not as much the case for MS applications.

**IDE building:** do the following steps to create an executable from scratch.

* Create proj, specify what you want to build (st, dyn, exe)
* Name of library and directory of creation/storage
* Write source files
* Include directories where compiler should search for non-defaultly

**Make:** all make variants are tailored for their particular toolset. GNU make is the defacto benchmark. Make can execute complex shell scripts, which accomplish a part of the build process. So running make one time corresponds to running these scripts sequentially, thereby creating a chain reaction.

13.7.22

**File Descriptor on Sockets:** you can connect to an IP/Port number combination with many socket file descriptors. The established connection is unique and them closing doesn’t mean that the socket closes. You can have a listener file descriptor that calls “socket” , “bind” and “listen” while another file descriptor can be created to call “accept”. You can shutdown the listener file descriptor, while still having an active connection which was established through accept.

**Chunk:** any large amount of data which is a part of any size of a whole.

**Block:** fixed amount of data, which is a part of a whole.

**Offset:** positional distance, between beginning of something and position of interest. Offset is exactly the byte distance, not one less.

**Buffer:** a piece of memory in which things are collected in order to process them as a whole when the buffer is full. Buffers can have fixed sizes or dynamically changing sizes.

**Sector:** like a block, it is a fixed size part of a whole, but related to a technically significant origin. Sectors contain blocks.

**Returning:** return from main is equivalent to exit. The program terminates immediately when exit is encountered. Return from a non-main functions will terminate immediately at the point where it has been called. Exit from anywhere in the program will terminate the executable immediately. If exit status is zero, the program terminated successfully, anything else is an anomaly.

14.7.22

You create private functions by using variables that are declared static inside the translation unit where you implement the function.

15.7.22

**Event Handling:** When a low priority component like the wifi-module, the hall sensor etc generates an event, this event is sent to a high priority event task via a queue. The user defined event handler function is called in the context of this task. Event task stack size and event queue size can be adjusted.

**Embedded RAM:** embedded hardware/chips usually have limited memory due to place constraints, hence the need to optimize the apps in order to squeeze as much RAM as possible. Such hardware are deliberately designed to utilize external RAM, so that you can execute certain computationally expensive actions. When the external RAM is initialized at startup (bootloading), you can utilize the memory as if it already is a part of the system. Applications can manually place data in external memory by creating pointers to this region. these pointers will have been mapped by the virtual memory satep that has taken place previously.

**Flash Memory:** when you flash your embedded hardware which doesn’t have any OS running, you create a sequence of subroutines that gets called one by one. You use C standard libraries, that get compiled into object files (assembly code) where the whole library doesn’t get compiled, just the function that you need and the dependence hierarchy that this function creates internally. On the next step when the linker gets in on the action, it will optimize code even further. Thus, you can have a really skinny image installed on the flash at the end.

19.7.22

**Unsigned char pointer:** pointer to the binary data buffer, as unsigned char guarantees the copying of the exact bitwise image. Unsigned char is used for raw memory actions, especially with memcpy. Ptr is used to access the data byte by byte, as you can do pointer arithmetics on it and iterate over the data.

**Bit Timing:** each BIT is logically divided into 4 quanta. CAN controller hardware can distinguish bits if it can discern the timing of these 4 quanta, thes quanta are separated into the following:

* Synchronization segment
* Propagation segment
* Phase segment 1
* Phase segment 2

These segments compensate for delays and tolerances. The firmware must time these quanta and send them to the can responder. CAN controllers sample multiple times during a bit. The sampling occurs on the borders of two quanta.

**Synchronization Jump Width:** To adjust the on-chip bus clock, the CAN controller may shorten or prolong the length of a bit by an integral number of quanta. The maximum value of these bit time adjustments are termed the “synchronization jump width”.

**Hard Synchronization:** occurs on the recessive-to-dominant transition of the start bit.

**Resynchronization:** occurs when a bit edge doesn’t occur within the Synchronization segment in a message. One of the phase segments are shortened or lengthened with an amount that depends on the phase error in the signal. The maximum amount is determined by SJW.

**Bit Timer Register Calculation:** CAN controller allow the programmer to set the bit timing using the following parameters:

* Clock prescaler value
* Number of quanta before sampling point
* Number of quanta after sampling point
* Number of quanta in the SJW

**Cyclic Redundancy Check**: detects accidental changes to digital data. Blocks of data get a short “check value” attached, based on the remainder of a polynomial division of their contents. If the check value doesn’t match, correction can be used against data corruption. Its called redundancy because it expands the message without adding any information to it. They are simple to implement in binary hardware and good at detecting common errors, like noise. The data is treated as a binary number, and divided by another binary number. the checksum is appended to the transmitted message. If the division result of the checksum isnt zero, an error occurred.

20.7.22

**Service Discovery:** automatic detection of devices and services on a computer network. An SDP, a service discovery protocol is a network protocol that accomplishes this. For example, DHCP and SLP are examples of these protocols.

**Service Location Protocol:** finds other devices and services in a local area network without prior configuration. Devices announce services on a local network. Each service has a URL that is used to locate the service. Devices are in scopes and they cant see outside of their scope.

**Case Fallthrough:** your switch clause deliberately has unhandled cases, in which the code sequentially goes from one case to the next, checking for all scenarios, but only executes the content on the handled case. This method ensures that more than one case can be handled under one roof.

**Hardware Abstraction:** Each type of CPU has a specific ISA. ISA represents the primitive operations of the machine that are available for use by assembly programmers and compiler writers. One of the main functions of a compiler is to allow a programmer to write an algorithm that is not CPU-specific. Compiler must generate a CPU-specific executable. The same type of abstraction applies to operating systems, the OS APIs represent the primitive operations of the machine, rather than the ISA. This means that you don’t have to know how each hardware system communicates with each other. The OS generates a hardware dependent instruction that gets sent to the device.

The hardware abstraction layer is implemented in software, between the physical hardware of the computer and the OS kernel. HAL sees to it so that the OS kernel-mode code doesn’t need to be changed to run on different hardware. HAL is the driver for the motherboard and allows hi-lev instrs to comms with low-lev devices, but no direct access to the low lev devices is present.

In Windows, the HAL is inside the kernel space, between hardware and the executive services that are contained in the file NTOSKRNL.EXE. this allows portability to processors, with different MMUs and IO bus archs.

21.7.22

**Build Process in Embedded Systems:** Unlike ready to go systems like MSVC or GCC, which produce the binary for the platform that you are writing the source code on, building on embedded systems requires multiple steps. Firstly, since you don’t have a lot of memory available, don’t expect an elaborate development environment in which you can debug everything with precision. Embedded systems almost always have unix-variants, meaning that you need to employ “make” to build and flash. Executing the make file on directory will build the binary. Flashing this binary on the hardware requires make flash, or something that will call upon a flashing shell script.

Embedded systems have partitions, once you are running inside the main loop, you need to flash the device and reboot it, so that the new partition can be used primarily. When a new firmware is installed, the old partition will be active at that time, as long as the system doesn’t reboot.

**OSI in Vehicles:** Every developer of every technology has a different way of understanding complex systems. Such is the case with vehicle communication, since there have been many protocols with many implementations overall.

22.7.22

**UART:** computer hardware device used for async serial communication. Communication parameters such as the data format and the transmission speeds are configurable. It sends data bits one by one, from the least significant to the most significant, framed by start and stop bits so that precise timing is handled by the communication channel. One of the grandpas of comms between multiple computer systems. The hardware connects to a “serial port” of a computer or an embedded device, in which it partakes in “serial communication”. Simplex, full duplex or half duplex comms types are present. The transmitting and the receiving side must both configure the following parametrs:

* Baud rate
* Parity bit
* Data bits size
* Stop bits size
* Flow control

An UART frame is shaped as the following:

* Idle
* Start bit
* Data bits
* Parity bit
* Stop bit

Receivers and transmitters both have timing constratints, in which they fulfill certain bit timings.

**Forward decl a struct:** a struct without a typedef needs to be with the keyword struct when used. If you typedef your struct you can leave out the struct keyword. You typedef your struct with the same name at the beginning of the translation unit, in case you want to use the struct containers in convoluted manners.

25.7.22

**Queues:** in embedded systems, tasks (processes) can communicate with each other via queues. Message between tasks and between tasks and interrupts are processed on queues. Especially in callback and asynchronous applications, queues play a critical role. Data is copied onto the queue, meaning that on the queue there are literally the raw bytes that you need, instead of a pointer to a different memory location. If you want to send an unsigned char, short, int or other basic variables, it is convenient to access the queue and obtain exactly what you need. There is no need to allocate a buffer for the message and then copy the variable into the allocated buffer. If one task sends raw bytes to the queue, containing some numbers, another task can read from the queue directly without any interpretation, meaning that tasks have directly communicated with each other. Copying the variable means that the task that sent it can further use its local copy for other purposes. This makes flexible use of the variable.

If the variable is too large, you can still pass it by reference. Queues allow both methods.

Variable sized messages are passed as C-structs. C-structs are seen often in such cases, because they hold the relevant data together.

Queues can receive different message types, from different locations.

You can implement blocking on the queues as well.

**Error Catching:** it may seem flat out retarded to encapsulate every function call with an if clause and return the result of the function within the body. Its because you think that the function will most likely succeed (which it will) but in the case that it doesn’t, catching run time errors is hell. By encapsulating each of these invocations with error code returns, you give your future self a way to cope with these run time errors, in case one pops up (very unlikely if you have written the code correctly, if not, get fucked retard).

1.8.22

Use regular int 0s and 1s as booleans when ur writing low level bare metal C because C doesnt have built-in boolean unless you want to include std\_bool.h

 The members of the dialout group have full and direct access to serial ports.

Add your user (you can also add root to this as well with sudo)

$ sudo usermod -a -G dialout <*username*>

 Restart the terminal to let changes take effect.

This solves the "permission denied" errors on external hardware.

 Write groups on the terminal to show to which groups the currently logged in user belongs to.

 Dmesg:

 Through trial and error, the sequence of actions that needs to be carried out has been optimized. These actions are present in many individual norms, hence the user should follow them in hierarchy. For Kline init, the sequence (bare metal to abstraction) follows:

- J1850 PWM

- J1850 VPW

- ISO 9141-2

- ISO 14230-4

- ISO 15765-4

If you send the signal too fast before the previous init steps have been correctly executed, you will encounter errors. The ECU + Tester comms time intervals should be kept intact. Thus, after each successful timing event, you must return error codes that give you status updates.

 If it seems like communication is stuck, it may be a deadlock on threads or a timing interval is giving a false positive.

 Initialization is completely different from the regular data exchange. Any subsequent comms after the init process will be standard data requests and responses.

Requests start with Service 1 PID 00

Some vehicle manufacturers have anomalies of course, thus the init steps may require some auxilliaries.

3.8.22

**Error:** “storage class specified for parameter” usually pops up in a file which you havent touched due to a ; being missing in a header file which has the function declaration. To solve it, look at new additions, or just try to mock commit and see the changes there.

If you feed enum values to a switch-case, you have to handle all of them, even if it has to be with a mock break.

4.8.22

**Null Pointer Function Callback Guards:** null guard dereferenced pointers. If your algorithm must have the pointer, you don’t need it, because it will get set regardless of other factors. If it MAY have it and is not detrimental to your app logic, you must null guard it.

**UML State Machine:** general FSM approach, expressed in the UML. It organizes the way a device, a computer program or a technical process works such that an entity or each of its subentities is always in exactly one of a number of possible states. There must be also well defined conditional transitions between these states. UML state machines have the characteristics of Moore/Mealy machines, they support actions that depend on both the state of the system and the triggering event, as well as entry and exit actions

**TTY in UNIX:** historical accuracy for teletypewriter. Represents an abstract entity for a device, which gets connected to the OS. A USB stick, a keyboard are physical inputs. Sometimes it is a virtual device, on which the user may interact with the kernel.

5.8.22

**Memory Paging:** a memory management scheme, it allows a process to be stored in a memory in a non-contiguous manner. It solves external fragmentation. The OS retrieves the data from secondary storage in same-size blocks called “pages”. Programs can exceed the size of available physical memory. The first storage is RAM, the second storage is the disk. Sometimes literally, sometimes in abstract. The secondary storage may also be called as “logical memory”. Phys and log mem are divided into fixed sized blocks. Phys is called frames, log is called page. When a process is executed, pages are loaded into frames. The address generated by CPU for accessing the frame is “page number” and “page offset”.

**Memory Segmentation:** the process is divided into variable size segments and loaded to the logical memory address space. The logical address space is a collection of variable size segments, with names and lengths. During execution, segments are loaded into phys mem space.

**Memory Fragmentation:** computer programs (processes) request memory from the OS during their execution. The blocks of memory are allocated in chunks. When the program is finished with the chunk, it will give it back to the OS. Over time, the blocks of memory that the OS distributes to the processes may be so short/interrupted, that the OS cant allocate the required memory and the process that needs this memory can function properly.

* Internal Frag: page frames are given in fixed sizes, even though the program may need less than this. Memory is allocced in multiples of 4 bytes, so when a process wants 29 bytes, 32 bytes are given. 3 bytes are wasted.
* External frag: the memory, although plents, is divided into subparts inefficiently, where none of the subparts satisfy the demands of the processes.
* Data frag: collection of data is broken up into many places that are not close together.

**Embedded System Tasks:** Tasks are semantically identical to threads happening on commercial Oses.

**Reactive (event driven) systems:** The response of the program depends on the nature of the event (the unique algorithm that defines the phenomenon) and the history of past events. This means that the system must have a memory that holds the past inputs, where these are accesible through many doors. Most reactive programs start very simple, but as features get added on top, they have complex “if-else” and “switch-case” scenarios. This means more flags and variables are introduced to the system, making “spaghetti code”. The behavior of the system can be divided into a number of non-overlapping states, where the responses depend only on the current level and not on the sequence of past events.

**Interrupt Service Routine (ISR):** aka interrupt handler, it is a process invoked by an interrupt request from a hardware device. It handles the request from the device, sends the results to the CPU, an in turn the CPU will halt the process, because the hardware device has sent the request to do just that. The most powerful ISR on commercial computers is keyboard strokes.

Without an interrupt request, the computer will preserve its current state. Device drivers allow communication between multiple hardware and the CPU.

Instead of the CPU polling on every hardware device and checking whether any changes have occurred or not, the hardware device will take control and inform the CPU about the changes.

CPU will stop tending to the running tasks and handle the high-priority interrupt first.

ISRs are the portion of the program code, where the content is executed, when an interrupt is obtained.

When the interrupt comes, the CPU breaks away from what it is doing, puts the “instruction pointer” to the beginning of the ISR and executes the content.

Embedded systems are mainly ISRs because most of the app logic occurs with interrupts .

**Static Keyword on functions:** you cant make function static if they are going to be used through different translation units, meaning that if the implemented function in a .c file is used via its header on a different .c file, this function must not have static.

**Stack Depth:** when a task is created on FreeRTOS, the stack size must be manually allocated for it. The task must have enough stack memory to handle its variables.

8.8.22

**Low Level Mutexing:** One task checks if a mutex exists before taking it. Mutexes are global variables, which you produce similarly to int a = 5; where the scope of the mutex must align with your shared resources. You can define local scope mutexes as well, but they will live only in that scope. Mutex operations are atomic, meaning no other tasks can interrupt the checking and taking of the said mutex. Some CPU archs have specific test and set assembly instructions, that guarantee this atomicity and in other scenarios, this must be guaranteed on the OS level. If the mutex exists, its value drops to zero and the task takes it, meaning that theres no mutex available for other operations that access shared resources. Suppose that one task has taken the mutex and gone into a function, which takes a fixed amount of time. during this time, another task wants to access the same shared resource, which is currently being processed inside the other function. The other task finds out that the mutex is zero, so it doesn’t have the key to access the shared resource. However, it doesn’t return from the “access mutex and do your thing” function. It waits, however humanly inconcievable, till the mutex is available, and does its thing from then on.

FreeRTOS implements semaphores and mutexes similarly, hence they are under the same roof.

“SemaphoreHandle\_t mutex;” or “SemaphoreHandle\_t semaphore;”

Mutex = xSemaphoreCreateMutex(); //returns the handle.#

xSemaphoreTake(mutex,0)

**Fatal Errors on Embedded Systems:** the kernel will catch fatal errors, and go into its panic handler, where it tries to trace to the address of where the error occurred. The execution of the program, which is the whole int main() since embedded systems bootload into the entry point of the int main, cant continue in a well defined way. There are three categories: 1) cpu exceptions, 2) system level checks and safeguards, 3) failed assertions

* Illegal instruction
* Load/store alignment error
* Load/store prohibited error
* Double exception
* -------------------------------
* Interrupt watchdog timeout
* Task watchdog timeout
* Cache access error
* Brownout detection event
* Stack overflow
* Stck smashing protection check
* Heap integrity check
* Undefined behavior check
* ---------------------------------------
* Assert and configassert spazzing out8/29/2022

You can set the CONFIG\_ESP\_SYSTEM\_PANIC configuration choice, to adjust the panic handler behavior.

Panic handlers are default programmed to reset the chip and create a crash dump.

The backtrace line contains PC:SP = program counter : stack pointer.

Kernels are historically programmed to have a panic() subroutine which handles such cases. These output an error message to the console, dump an image of kernel memory

9.8.22

**Memory Corruption:** also termed as violation of memory safety. This is caused when memory is modified outside of the intention of the programmer. It usually originates from bad programming, aka software bugs. Heap corruption is why usually apps on user space crash. The source of memory corruption and its manifestation are usually far apart due to abstraction/compertmentalization, therefore making it hard to debug. Corruption is mainly categorized into four.

* Uninitialized memory: these values are garbage, will lead to UB.
* Non-owned memory: will cause OS exception and a program crash.
* Beyond memory: will cause a buffer overflow. Usually happens with arrays because C/C++ doesn’t check arrays explicitly.
* Incorrect heap management: memory leaks.

**Illegal Opcode:** this opcode is not mentioned in the official documentation released by the CPUs designer. Illegal opcodes are a side-effect, they exist due to emergent properties of transistor wirings. They combine functions, which are not intended to be combined.

**Bad Sector:** a part of disk storage, which is perm damaged. Bad sectors are unrecoverable, the OS will mark it as such and skip it forever.

**Microcode:** a technique that interposes a layer of computer organization between the CPU hardware and the ISA of the computer. They are hardware-level insturctions that implement higher-level machine code. Microcode can also be used as a synonym for “firmware”. Microcode is written during the design of a processor, so it has insider knowledge. Microcode controls the CPU at hardware circuitry level. It is the closest thing to literal transistor switching a written code can come to.

**NX Bit:** used in CPUs to segregate areas of memory for use. It can be either storage for processor instructions (code) or storage of data. Used vastly in harvard architecture, although von neumann is also present. It marks memory regions as non executable, even thought code may reside inside. If you attempt to execute, an exception will be thrown. “executable space protection”

**ASLR:** address space layout randomization, the OS randomizes the location where system executables are loaded into memory. This protects the computer against buffer-overflow attacks.

**Backdoor in computing:**

**Halt and Catch Fire:** HCF assembly mnemonic, the machine code instruction tells the CPU to cease all ops and restart the computer. X86 intel procs have HLT (halt), only way out is to restart the CPU.

**Return oriented programming:** the attacker gains control of the “call stack” to hijack program control flow and then executes machine instruction sequences, which are already present in machine’s memory. These instr end in a return statement, which returns from the subroutine immediately to the point of return address. The OSes have somewhat of a protection against buffer overflow attacks, the overflown part is marked as nonexecutable. ROP doesn’t overflow and thus inject malicious code but uses alread present instrs by manipulating return addresses.

18.8.22

**Static C funcs:** only defined inside .c files without header declaration, because it defeats the purpose. You only need the function inside that file, why would you declare it inside headers? This wont give you any errors, not even runtime, but it is good practice to keep it inside one file for semantic purposes.

**Python/C comms on socket:** you can transfer messages with different names, corresponding to the same semantic concept. Create a class on python, your names correspond to hex values. Create a typedef enum on c, your names correspond to hex values. Names, altough not identical, will correspond to each other exactly, if their hex values match. You can use this to deal with if else cases

19.8.22

**Printing binary numbers:** you cant print out binary numbers like 010101 with %b, this isnt a built-in function. However, a mini function can be found inside the code bank,

22.8.22

**C-struct init is obj:** there is no “class” in c. but you can initialize the values of variables inside a struct explicitly through a function. Semantically, it is similar to creating the same process with classes (albeit without the elegant features of OOP), this way your specific type will have a user given value set at start-up.

**Bitwise Shifting:** using >> will shift your byte sequence to the right by the amount that you want. (yourVariable >> 1) will shift every block of byte to the right by one. the most left block will be filled with zero. This happens for every block that arises by the shift procedure.

Similarly, by using << you will shift the amount to the left. This will fill the most right block with zero as well.

If continuously shifted, all values will revert back to 0x00.

**Writing at the address location of a byte sequence :**imagine having a sequence of memory, you have a pointer that points to the beginning of this sequence. You can get the address of the pointer (meaningless, a random memory value) or you can get the real value of the pointee, to which the pointer points.

23.8.22

**Using pointers to go inside a function:** unreachable scope, return as ptr, done in kernel progs and just general dick swinging behavior. This is the prime way to cause dangling pointers, hence read below to counter that.

**Undefined reference to:** means that you forgot to implement the function in corresponding .c src file of a header from which you extract the functions.

**C Preproc on address assignment:** you cant get the address of a literal by doing &(value + 2). Create a new variable; variable = value +2 and then “&variable” will work. Otherwise, C preproc error will be thrown.

**Dangling pointer:** you created a pointer and passed it to a function. Inside the function, you created a locally scoped variable, and did things with it and assigned the address to the pointer. You went out of the function, this variable doesn’t exist anymore and you are now referencing to non-existent memory. This is undefined behavior. Declare the locally scoped variable as static to avoid. Has the downside of this variable being accessed from everywhere, security gets fucked.

24.8.22

**API Key:** this identifies a user, a developer or a calling program to an API. An entity having this key may interact with the API and even modify it, if a modification is offered. API keys are not secure.

**Compiler Extension:** makes your code run on different architecture / different language version or other incompatible shit. This is on by default on your IDE, which may let you think that some features are inherently there(you get filtered if youre a brainlet!). Compiler extensions make your program non-compliant to C++ standards, they must be turned off iff the norm is to be conformed.

**CI/CD:** imagine starting up your project / solution everytime, build it, package it, nitpick every small detail and then burn it to a drive and then literally bring it to the customer’s doorstep. Sounds retarded right? Well you don’t have to! Employ a CI/CD system to automate these processes for you. When you are working with your local copy, make some changes, commit these to your repo, Jenkins is aware of these changes and will prompt your build automation to build the project again with the latest changes. In complete projects, a lot of logging files are produced. Here you can trace the build process as well. Jenkins will also give you some tracing information regarding the progress of the build.

**Design Steps of a program:**

* Define a goal
* Define requirements
* Define tools, targets and a backup plan
* Hierarchical structuring of bigger problems
* Sequence of events / prog flow
* Outline the int main
* Entry points for each function

**Casting:** compilers have hardcoded checks, where they look for type conformity. A smaller type can usually fit into a bigger type without explicitly informing the programmer. This is implicit conversion. If you want to fit a bigger type into a smaller one, you will lose the bytes. You can also explicitly convert types by writing the name manually. C-style conversion has the type directly engulf the var that you want to convert, C++ conversion will use static\_cast<yourNewType> and the likes to convert the var depending on the use case.

**Bit Mask a state:** bit masking can be used to set/reset/toggle the state of a variable depending on the mask. Make the mask equal to isState and compare it with the variable you want. The result should give you the isState if it matches, and nothing it if doesn’t. More under c++ patterns.

25.8.22

**Linking against a library:** a library contains already compiled code. This may be produced inside your solution at the location where you said the build tool to store the data. In this case, you would have your dll, pdb, lib files here, and can directly run the .exe at this path. By #include, you tell the compiler that the symbols you want to use exists somewhere at the location you provided, so it trusts you already. However, it must be completely linked nonetheless. You provide the external library through the configuration selection with MSVC, or through Makefiles on UNIX. If the symbols cant be found in the library that the compiler thought it would be, you will have “unresolved references”. Linking against a library is usually problematic with external third party libs, reggie shit wont throw this error, because the compilers have thought about virtually all cases.

29.8.22

**Core Dump / Crash Dump:**

**Kernel Panic:**

30.8.22

31.8.22

1.9.22

2.9.22

5.9.22

6.9.22

7.9.22