Creación del archivo donde se guardarán las aplicaciones UEFI.
Asegure tener actualizado el sistema

```
rodrigo@RodrigoHP:~$ sudo apt update && sudo apt upgrade -y
```

sgdisk / gdisk

```
rodrigo@RodrigoHP:~$ sudo apt-cache search sgdisk
rodrigo@RodrigoHP:~$ sudo apt-cache search gdisk
forensics-extra - Forensics Environment - extra console components (metapackage)
gdisk - GPT fdisk text-mode partitioning tool
rodrigo@RodrigoHP:~$ sudo apt install gdisk
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  groff-base libuchardet0
Suggested packages:
  groff
The following NEW packages will be installed:
  gdisk groff-base libuchardet0
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 1,205 kB of archives.
After this operation, 4,880 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Tras instalar verifique la instalación del archivo con el parámetro *–version*
Utilice el parámetro -l para imprimir a MBR partition table/info o el parametro -P para una GPT partition table

Adicionalmente no olvide verificar que cuenta con alguno de los siguientes compiladores (gcc o clang):

```
rodrigo@RodrigoHP:~$ gcc --version
gcc (Debian 12.2.0-14+deb12u1) 12.2.0
Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

rodrigo@RodrigoHP:~$ clang --version
Debian clang version 14.0.6
Target: x86_64-pc-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
rodrigo@RodrigoHP:~$
```

Descargue el repositorio de trabajo en su computadora local (de preferencia que sea nuevo)

```
rodrigo@RodrigoHP:~$ sudo git clone https://github.com/RodrigoTorresR/UEFI-GPT-image
-creator.git
Cloning into 'UEFI-GPT-image-creator'...
warning: You appear to have cloned an empty repository.
rodrigo@RodrigoHP:~$ ls
coreboot  minipro-fork  UEFI-GPT-image-creator
rodrigo@RodrigoHP:~$ cd UEFI-GPT-image-creator/
rodrigo@RodrigoHP:~/UEFI-GPT-image-creator$ ls
rodrigo@RodrigoHP:~/UEFI-GPT-image-creator$
```

Creamos los archivos *makefile* y *write_gpt.c:*

```
rodrigo@RodrigoHP:~/UEFI-GPT-image-creator$ sudo touch write_gpt.c
rodrigo@RodrigoHP:~/UEFI-GPT-image-creator$ sudo touch makefile
rodrigo@RodrigoHP:~/UEFI-GPT-image-creator$ ls
makefile  write_gpt.c
```

comenzamos a editar el archivo makefile en vim (*sudo vim write_gpt.c):*

```c
#include <stdio.h>

int main(void){
        puts("TESTING");
        return 0;
}
```

Y despues a editar el archivo makefile en vim (*sudo vim makefile).:*

```
.POSIX:
.PHONY: all clean

TARGET = write_gpt.c
CC = gcc
#CC = clang
CFLAGS = -std=c17 -Wall -Wextra -Wpedantic -02

all: $(TARGET)

clean:
        rm -f $(TARGET)
```

Y probamos que nuestro archivo de reglas *makefile* funcione correctamente:

```
rodrigo@RodrigoHP:~/UEFI-GPT-image-creator$ sudo make
gcc -std=c17 -Wall -Wextra -Wpedantic -O2    write_gpt.c   -o write_gpt
rodrigo@RodrigoHP:~/UEFI-GPT-image-creator$ ls
makefile  write_gpt  write_gpt.c
rodrigo@RodrigoHP:~/UEFI-GPT-image-creator$ ./write_gpt
TESTING
rodrigo@RodrigoHP:~/UEFI-GPT-image-creator$
```

Podemos corroborar que el archivo ejecutable *write_gpt* se ha creado correctamente y es funcional.

Ahora modificamos el archivo anterior con el código siguiente, para crear la tabla GPT donde cargaremos nuestras aplicaciones UEFI.

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>

//i-----------------------------------
//Globarl Typedef
//i-----------------------------------
//Master Boot Partition
typedef struct{
        uint8_t boot_indicator;
        uint8_t starting_chs[3];
        uint8_t os_type;
        uint8_t ending_chs[3];
        uint32_t starting_lba;
        uint32_t size_lba;
} __attribute__((packed)) Mbr_Partition;

//Master Boot Record
typedef struct{
        uint8_t boot_code[440];
        uint32_t mbr_signature;
        uint16_t unknown;
        Mbr_Partition partition [4];
        uint16_t boot_signature;
} __attribute__((packed)) Mbr;

//i-----------------------------------
//Global VariablesL
//i-----------------------------------
//Master Boot Record
char *image_name = "test.img";
uint64_t lba_size = 512;
uint64_t esp_size = 1024*1024*33;       // 33 MiB
```

```c
uint64_t data_size = 1024*1024*1;       // 1 MiB
uint64_t image_size = 0;
uint64_t esp_lbas, data_lbas, image_size_lbas;

//===================================
//convert bytes to LBAs
//===================================
uint64_t bytes_to_lbas(const uint64_t bytes){
        return (bytes / lba_size) + (bytes % lba_size > 0 ? 1 :0);
}

//===============================
// Write Protective MBR
// ==============================
bool write_mbr(FILE *image){
        if (image_size_lbas > 0xFFFFFFFF) image_size_lbas = 0x100000000;
        Mbr mbr = {
                .boot_code = { 0 },
                .mbr_signature = 0,
                .unknown = 0,
                .partition[0] = {
                        .boot_indicator = 0,
                        .starting_chs = { 0x00, 0x02, 0x00 },
                        .os_type = 0xEE,        //Protective GPT
                        .ending_chs = {0xFF, 0xFF, 0xFF},
                        .starting_lba = 0x00000001,
                        .size_lba = image_size_lbas - 1,
                },
                .boot_signature = 0xAA55,
        };

        //Write to file
        if (fwrite(&mbr, 1, sizeof mbr, image) != sizeof mbr)
                return false;

        return true;
}
//===================================
//MAIN
//===================================
int main(void){
        FILE *image = fopen(image_name, "wb+");
        if (!image){
                fprintf(stderr, "Error: coud not open file %s\n", image_name);
                return EXIT_FAILURE;
        }
        //Set sizes
        image_size = esp_size + data_size + (1024*1024); // add some extra padding for GPTs /MBR
        image_size_lbas = bytes_to_lbas(image_size);

        if (!write_mbr(image)){
                fprintf(stderr, "Error: coud not protective MBR for file %s\n", image_name);
                return EXIT_FAILURE;
        }

        return EXIT_SUCCESS;
}
```

No podemos usar *gptdisk -l* para verificar *test.img* porque el tamaño del archivo es demasiado pequeño como para contener una verdadera GPT table.

```
rodrigo@RodrigoHP:~/UEFI-GPT-image-creator$ gdisk -l test.img
GPT fdisk (gdisk) version 1.0.9

Warning! Read error 25; strange behavior now likely!
Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: not present

Creating new GPT entries in memory.
Disk is too small to hold GPT data (1 sectors)! Aborting!
rodrigo@RodrigoHP:~/UEFI-GPT-image-creator$
```

En su defecto utilizamos el siguiente comando:
Instalar el comando xxd en nuestro WSL debian (en caso de contar con el):

```
rodrigo@RodrigoHP:~/UEFI-GPT-image-creator$ sudo apt install xxd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  xxd
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 84.1 kB of archives.
After this operation, 138 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian bookworm/main amd64 xxd amd64 2:9.0.1378-2+deb12u2 [84.1 kB]
Fetched 84.1 kB in 1s (141 kB/s)
Selecting previously unselected package xxd.
(Reading database ... 35523 files and directories currently installed.)
Preparing to unpack .../xxd_2%3a9.0.1378-2+deb12u2_amd64.deb ...
Unpacking xxd (2:9.0.1378-2+deb12u2) ...
Setting up xxd (2:9.0.1378-2+deb12u2) ...
```

Verificamos la correcta instalación:

```
rodrigo@RodrigoHP:~/UEFI-GPT-image-creator$ xxd --version
xxd 2022-01-14 by Juergen Weigert et al.
```

Revisamos el contenido de nuestro archivo *test.img* para verificar que nuestra GPT se ha creado correctamente.