

Centro Federal de Educação Tecnológica de Minas Gerais  
Departamento de Engenharia de Computação  
Sistemas Operacionais

# Adição de um módulo ao kernel

Aluno: Gabriel Siqueira Silva  
Professor: Bruno Santos

Agosto  
2022

Centro Federal de Educação Tecnológica de Minas Gerais  
Departamento de Engenharia de Computação  
Sistemas Operacionais

# Relatório

Primeiro Relatório de Sistema Operacional com o  
objetivo de entender o processo de alteração do kernel de uma distribuição Linux

Aluno: Gabriel Siqueira

Professor: Bruno Santos

Agosto  
2022

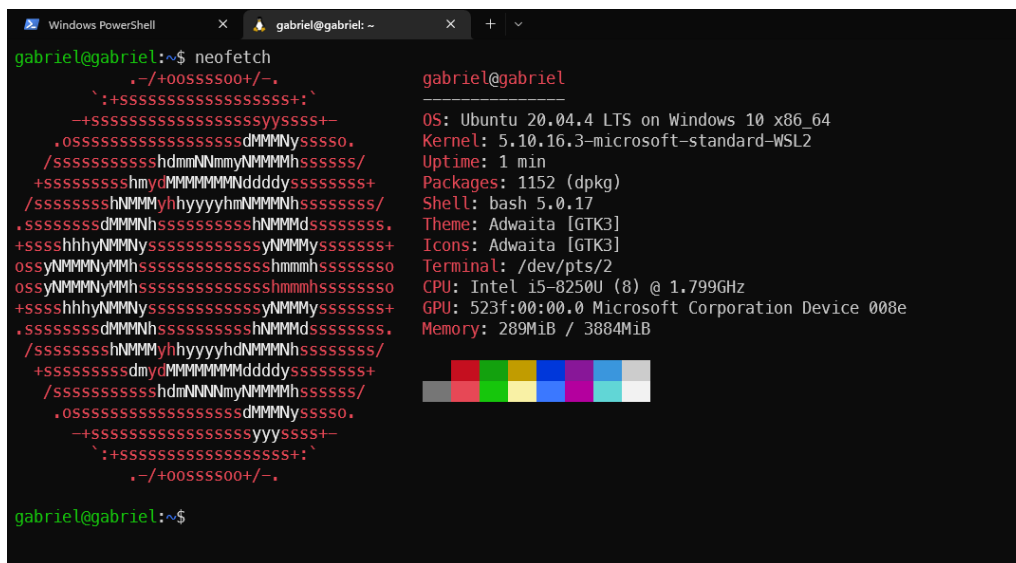
# Conteúdo

<b>1</b>	<b>Adição do módulo Hello World ao kernel</b>	<b>1</b>
1.1	Procedimentos iniciais . . . . .	1
1.2	Hello World . . . . .	1
1.3	Compilação no WSL2 . . . . .	5
1.4	Configuração . . . . .	6
1.5	Teste . . . . .	7
<b>2</b>	<b>Questões</b>	<b>8</b>
	<b>Bibliografia</b>	<b>10</b>

# 1 Adição do módulo Hello World ao kernel

## 1.1 Procedimentos iniciais

Para realizar esse trabalho prático utilizou-se o WSL2 na distribuição Ubuntu 20.04 que tem o kernel base 5.10.16.3-microsoft-standard-WSL2. O kernel modificado será a versão 5.15.y disponível no github da microsoft e para tal utilizou-se o git clone para utilizar o repositório.



```
Windows PowerShell
gabriel@gabriel: ~
gabriel@gabriel:~$ neofetch
      .-/+00SSSS00+/-+.
      `:+SSSSSSSSSSSSSSSS+:`
      -+SSSSSSSSSSSSSSSSyySSSS+-
      .oSSSSSSSSSSSSSSSSdMMNNySSSSo.
      /SSSSSSSSSSShdmmNNmyNNMMhSSSSS/
      +SSSSSSSSShmydMMMMMMNdddySSSSSSS+
      /SSSSSSSShNNMyhyyyyhmNNNNhSSSSSSS/
      .SSSSSSSSdMMNNhSSSSSSSSShNNMdSSSSSSS.
      +SSShhhyNNMySSSSSSSSSSyNNMySSSSSSS+
      ossyNNMMNyMMhSSSSSSSSSSShmmhSSSSSSSo
      ossyNNMMNyMMhSSSSSSSSSSShmmhSSSSSSSo
      +SSShhhyNNMySSSSSSSSSSyNNMySSSSSSS+
      .SSSSSSSSdMMNNhSSSSSSSSShNNMdSSSSSSS.
      /SSSSSSSShNNMyhyyyyhdNNNNhSSSSSSS/
      +SSSSSSSSdmydMMMMMMNdddySSSSSSS+
      /SSSSSSSSShdmmNNNNmyNNMMhSSSSS/
      .oSSSSSSSSSSSSSSSSdMMNNySSSSo.
      -+SSSSSSSSSSSSSSSSyySSSS+-
      `:+SSSSSSSSSSSSSSSS+:`
      .-/+00SSSS00+/-+.

OS: Ubuntu 20.04.4 LTS on Windows 10 x86_64
Kernel: 5.10.16.3-microsoft-standard-WSL2
Uptime: 1 min
Packages: 1152 (dpkg)
Shell: bash 5.0.17
Theme: Adwaita [GTK3]
Icons: Adwaita [GTK3]
Terminal: /dev/pts/2
CPU: Intel i5-8250U (8) @ 1.799GHz
GPU: 523f:00:00.0 Microsoft Corporation Device 008e
Memory: 289MiB / 3884MiB
```

Figura 1: Kernel antigo

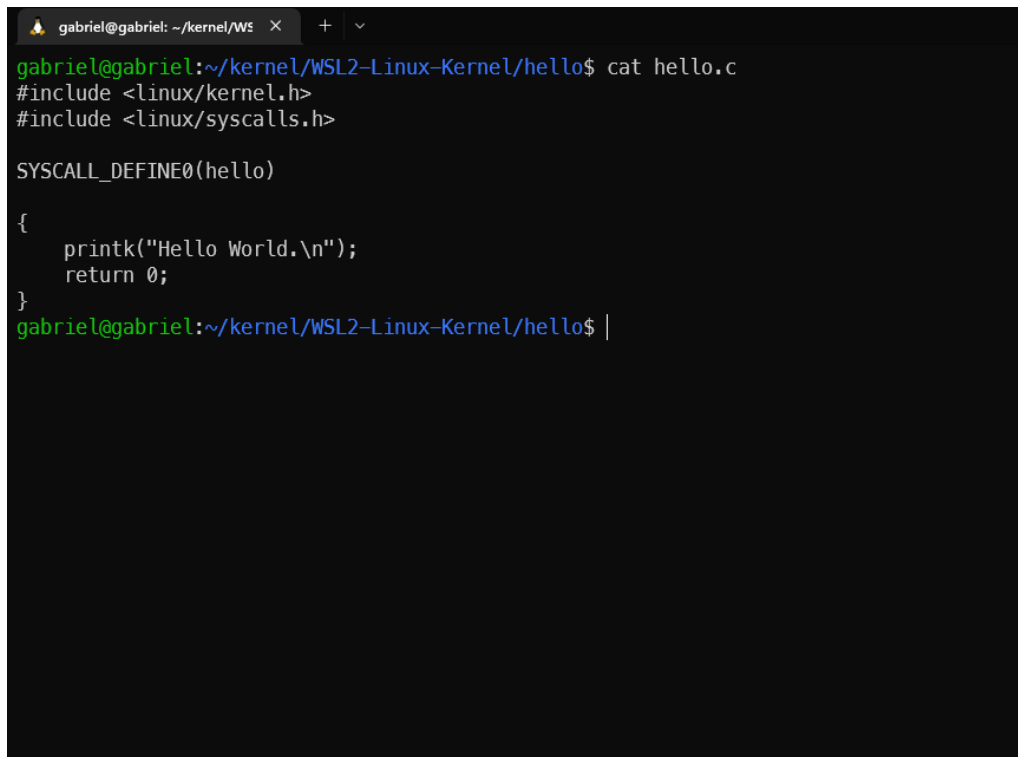
## 1.2 Hello World

O módulo adicionado será o Hello World e para isso cria-se um diretório dentro desse kernel com um código em c, como apresentado abaixo:

```
#include <linux/kernel.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE0(hello)

{
    printk("Hello World.\n");
    return 0;
}
```

A terminal window with a dark background. The title bar shows 'gabriel@gabriel: ~/kernel/WSL' and a tab icon. The prompt is 'gabriel@gabriel:~/kernel/WSL2-Linux-Kernel/hello\$'. The command 'cat hello.c' has been executed, displaying the following C code:

```
#include <linux/kernel.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE0(hello)
{
    printk("Hello World.\n");
    return 0;
}
```

The prompt is now 'gabriel@gabriel:~/kernel/WSL2-Linux-Kernel/hello\$' followed by a cursor.

Figura 2: Código em C

Onde está o arquivo em c, deve-se criar um Makefile apresentando um arquivo objeto obtido após a compilação e além disso no Makefile do kernel deve-se mostrar quais são os módulos pertencentes ao mesmo e portanto é necessário alterar adicionando o hello aos módulos no core-y.

```
gabriel@gabriel: ~/kernel/WSL2-Linux-Kernel/hello$ cat hello.c
#include <linux/kernel.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE0(hello)

{
    printk("Hello World.\n");
    return 0;
}

gabriel@gabriel:~/kernel/WSL2-Linux-Kernel/hello$ touch Makefile
gabriel@gabriel:~/kernel/WSL2-Linux-Kernel/hello$ echo "obj-y := hello.o" > Makefile
gabriel@gabriel:~/kernel/WSL2-Linux-Kernel/hello$ cd ..
gabriel@gabriel:~/kernel/WSL2-Linux-Kernel$ nano Makefile
gabriel@gabriel:~/kernel/WSL2-Linux-Kernel$ cat Makefile | grep core-y
core-y      := init/ usr/ arch/$(SRCARCH)/
core-y      += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ hello/
              $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
              $(core-y) $(core-m) $(libs-y) $(libs-m) \
KBUILD_VMLINUX_OBJS := $(head-y) $(patsubst %/,%/built-in.a, $(core-y))
gabriel@gabriel:~/kernel/WSL2-Linux-Kernel$ |
```

Figura 3: Makefile

Agora, o hello deve ser conhecido como chamada de sistema e para isso no diretório de inclusão que tem o syscalls.h deve-se adicionar o

```
asmlinkage long sys_hello(void)
```

e além disso, essa chamada deve ser adicionada na tabela de chamadas.

```

gabriel@gabriel: ~/kernel/WS
+ -

/* for __ARCH_WANT_SYS_IPC */
long ksys_semtimedop(int semid, struct sembuf __user *tsops,
                    unsigned int nsops,
                    const struct __kernel_timespec __user *timeout);
long ksys_semget(key_t key, int nsems, int semflg);
long ksys_old_semctl(int semid, int semnum, int cmd, unsigned long arg);
long ksys_msgget(key_t key, int msgflg);
long ksys_old_msgctl(int msqid, int cmd, struct msqid_ds __user *buf);
long ksys_msgrcv(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                long msgtyp, int msgflg);
long ksys_msgsnd(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                int msgflg);
long ksys_shmget(key_t key, size_t size, int shmflg);
long ksys_shmctl(int shmid, int cmd, struct shmid_ds __user *buf);
long ksys_old_shmctl(int shmid, int cmd, struct shmid_ds __user *buf);
long compat_ksys_semtimedop(int semid, struct sembuf __user *tsems,
                            unsigned int nsops,
                            const struct old_timespec32 __user *timeout);
long __do_semtimedop(int semid, struct sembuf *tsems, unsigned int nsops,
                    const struct timespec64 *timeout,
                    struct ipc_namespace *ns);

int __sys_getsockopt(int fd, int level, int optname, char __user *optval,
                    int __user *optlen);
int __sys_setsockopt(int fd, int level, int optname, char __user *optval,
                    int optlen);
asmlinkage long sys_hello(void);
#endif
gabriel@gabriel:~/kernel/WS-Linux-Kernel$

```

Figura 4: Chamadas

522	x32	rt_sigpending	compat_sys_rt_sigpending
523	x32	rt_sigtimedwait	compat_sys_rt_sigtimedwait_time64
524	x32	rt_sigqueueinfo	compat_sys_rt_sigqueueinfo
525	x32	sigaltstack	compat_sys_sigaltstack
526	x32	timer_create	compat_sys_timer_create
527	x32	mq_notify	compat_sys_mq_notify
528	x32	kexec_load	compat_sys_kexec_load
529	x32	waitid	compat_sys_waitid
530	x32	set_robust_list	compat_sys_set_robust_list
531	x32	get_robust_list	compat_sys_get_robust_list
532	x32	vmsplice	sys_vmsplice
533	x32	move_pages	sys_move_pages
534	x32	preadv	compat_sys_preadv64
535	x32	pwritev	compat_sys_pwritev64
536	x32	rt_tgsigqueueinfo	compat_sys_rt_tgsigqueueinfo
537	x32	recvmmsg	compat_sys_recvmmsg_time64
538	x32	sendmmsg	compat_sys_sendmmsg
539	x32	process_vm_readv	sys_process_vm_readv
540	x32	process_vm_writev	sys_process_vm_writev
541	x32	setsockopt	sys_setsockopt
542	x32	getsockopt	sys_getsockopt
543	x32	io_setup	compat_sys_io_setup
544	x32	io_submit	compat_sys_io_submit
545	x32	execveat	compat_sys_execveat
546	x32	preadv2	compat_sys_preadv64v2
547	x32	pwritev2	compat_sys_pwritev64v2
548	common	hello	sys_hello

# This is the end of the legacy x32 range. Numbers 548 and above are  
# not special and are not to be used for x32-specific syscalls.

```
gabriel@gabriel:~/kernel/WSL2-Linux-Kernel$ |
```

Figura 5: Tabela

### 1.3 Compilação no WSL2

Para compilar no wsl se usa

```
make KCONFIG_CONFIG=Microsoft/config-wsl
```

alterando o parâmetro j caso queira utilizar mais cores da sua máquina do que o padrão e o resultado final fica armazenado na pasta do kernel baixado com o nome vmlinux além do bzImage identificado na imagem.



```
gabriel@gabriel: ~/kernel/WSL X gabriel@gabriel: ~/kernel/WSL X + | v
REL0CS arch/x86/boot/compressed/vmlinux.relocs
HOSTCC arch/x86/boot/compressed/mkpiggy
CC arch/x86/boot/compressed/cpuflags.o
CC arch/x86/boot/compressed/early_serial_console.o
CC arch/x86/boot/compressed/kaslr.o
CC arch/x86/boot/compressed/ident_map_64.o
CC arch/x86/boot/compressed/idt_64.o
AS arch/x86/boot/compressed/idt_handlers_64.o
AS arch/x86/boot/compressed/mem_encrypt.o
CC arch/x86/boot/compressed/pgtable_64.o
CC arch/x86/boot/compressed/acpi.o
HOSTCC arch/x86/boot/tools/build
AS arch/x86/boot/compressed/efi_thunk_64.o
CC arch/x86/boot/compressed/misc.o
GZIP arch/x86/boot/compressed/vmlinux.bin.gz
CPUSTR arch/x86/boot/cpustr.h
CC arch/x86/boot/cpu.o
MKPIGGY arch/x86/boot/compressed/piggy.S
AS arch/x86/boot/compressed/piggy.o
LD arch/x86/boot/compressed/vmlinux
ZOFFSET arch/x86/boot/zoffset.h
OBJCOPY arch/x86/boot/vmlinux.bin
AS arch/x86/boot/header.o
LD arch/x86/boot/setup.elf
OBJCOPY arch/x86/boot/setup.bin
BUILD arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready (#3)
gabriel@gabriel:~/kernel/WSL2-Linux-Kernel$ |
```

Figura 6: Compilação terminada

## 1.4 Configuração

Movendo o arquivo executável para o usuário no windows faremos com que a informação do kernel do linux presente no windows seja alterada pelo valor dessa vmlinux com o código:

```
[ wsl2 ]
kernel=C:\\Users\\<Nome_Usu_rio>\\vmlinux
```

Dessa forma o kernel muda como segue a imagem:

```

Windows PowerShell
gabriel@gabriel: ~
gabriel@gabriel:~$ neofetch
      ,--/+00SSSS00/+-.
      `:+SSSSSSSSSSSSSSSS+:`
      -+SSSSSSSSSSSSSSSSSSSS+--
      .0SSSSSSSSSSSSSSSSSSSSdMMMMNySSSS0.
      /SSSSSSSSSSShdmmNNmmyNNMMhSSSSSSS/
      +SSSSSSSSShmydMMMMMMNNdddySSSSSSSS+
      /SSSSSSSSShNNMMMyhhyyyhmNNMMhSSSSSSSS/
      .SSSSSSSSdMMMNhSSSSSSSSShNNMMdSSSSSSSS.
      +SSSShhhyNNMMNySSSSSSSSSSyNNMMYSSSSSSSS+
      ossyNNMMNMyMMhSSSSSSSSSSShmmhSSSSSSSS0
      ossyNNMMNMyMMhSSSSSSSSSSShmmhSSSSSSSS0
      +SSSShhhyNNMMNySSSSSSSSSSyNNMMYSSSSSSSS+
      .SSSSSSSSdMMMNhSSSSSSSSShNNMMdSSSSSSSS.
      /SSSSSSSSShNNMMMyhhyyyhdNNMMhSSSSSSSS/
      +SSSSSSSSSdmydMMMMMMNNdddySSSSSSSS+
      /SSSSSSSSShdmmNNmmyNNMMhSSSSSSS/
      .0SSSSSSSSSSSSSSSSSSdMMMMNySSSS0.
      -+SSSSSSSSSSSSSSSSSSSSySSSS+--
      `:+SSSSSSSSSSSSSSSSSS+:`
      ,--/+00SSSS00/+-.

OS: Ubuntu 20.04.4 LTS on Windows 10 x86_64
Kernel: 5.15.57.1-microsoft-standard-WSL2
Uptime: secs
Packages: 1153 (dpkg)
Shell: bash 5.0.17
Theme: Adwaita [GTK3]
Icons: Adwaita [GTK3]
Terminal: /dev/pts/0
CPU: Intel i5-8250U (8) @ 1.799GHz
GPU: 900e:00:00.0 Microsoft Corporation Device 008e
Memory: 310MiB / 3874MiB

```

Figura 7: Kernel alterado

## 1.5 Teste

Compilando e executando o seguinte código em c obtemos uma chamada de sistema como a imagem posterior descreve.

```

#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
int main()
{
    long int amma = syscall(548);
    printf("System call sys_hello returned %ld\n", amma);
    return 0;
}

```

```
Windows PowerShell
gabriel@gabriel: ~

[ 13.438374] pci 6ed1:00:00.0: [1af4:1049] type 00 class 0x010000
[ 13.439488] pci 6ed1:00:00.0: reg 0x10: [mem 0xbffe04000-0xbffe04fff 64bit]
[ 13.440211] pci 6ed1:00:00.0: reg 0x18: [mem 0xbffe05000-0xbffe05fff 64bit]
[ 13.440925] pci 6ed1:00:00.0: reg 0x20: [mem 0xbffe06000-0xbffe06fff 64bit]
[ 13.445327] pci_bus 6ed1:00: busn_res: [bus 00-ff] end is updated to 00
[ 13.445332] pci 6ed1:00:00.0: BAR 0: assigned [mem 0xbffe04000-0xbffe04fff 64bit]
[ 13.445858] pci 6ed1:00:00.0: BAR 2: assigned [mem 0xbffe05000-0xbffe05fff 64bit]
[ 13.446371] pci 6ed1:00:00.0: BAR 4: assigned [mem 0xbffe06000-0xbffe06fff 64bit]
[ 13.720036] FS-Cache: Duplicate cookie detected
[ 13.720038] FS-Cache: 0-cookie c=0000003b [p=00000002 fl=222 nc=0 na=1]
[ 13.720040] FS-Cache: 0-cookie d=000000005810f652{9P.session} n=00000000c5048437
[ 13.720042] FS-Cache: 0-key=[10] '34323934393338363630'
[ 13.720048] FS-Cache: N-cookie c=0000003c [p=00000002 fl=2 nc=0 na=1]
[ 13.720049] FS-Cache: N-cookie d=000000005810f652{9P.session} n=000000002fb59f9
[ 13.720050] FS-Cache: N-key=[10] '34323934393338363630'
[ 16.725819] hv_pci ab904ef3-c408-4f5d-b495-7a740c082c5e: PCI VMBus probing: Using version 0x10004
[ 16.783820] hv_pci ab904ef3-c408-4f5d-b495-7a740c082c5e: PCI host bridge to bus c408:00
[ 16.783823] pci_bus c408:00: root bus resource [mem 0xbffe08000-0xbffe0afff window]
[ 16.783825] pci_bus c408:00: No busn resource found for root bus, will use [bus 00-ff]
[ 16.784945] pci c408:00:00.0: [1af4:1049] type 00 class 0x010000
[ 16.786049] pci c408:00:00.0: reg 0x10: [mem 0xbffe08000-0xbffe08fff 64bit]
[ 16.786769] pci c408:00:00.0: reg 0x18: [mem 0xbffe09000-0xbffe09fff 64bit]
[ 16.787472] pci c408:00:00.0: reg 0x20: [mem 0xbffe0a000-0xbffe0afff 64bit]
[ 16.791735] pci_bus c408:00: busn_res: [bus 00-ff] end is updated to 00
[ 16.791739] pci c408:00:00.0: BAR 0: assigned [mem 0xbffe08000-0xbffe08fff 64bit]
[ 16.792281] pci c408:00:00.0: BAR 2: assigned [mem 0xbffe09000-0xbffe09fff 64bit]
[ 16.792782] pci c408:00:00.0: BAR 4: assigned [mem 0xbffe0a000-0xbffe0afff 64bit]
[ 49.239703] hv_balloon: Max. dynamic memory size: 4050 MB
[ 127.641408] Hello world.
gabriel@gabriel:~$
```

Figura 8: Resultado final

## 2 Questões

1. A implementação da chamada de sistema e o teste funcionaram corretamente? Em caso de negativo, explique o motivo.

Resposta: Sim, funcionou, como demonstrado na figura 8 desse relatório.

2. Qual é o nome do arquivo executável que corresponde ao Kernel?

Resposta: O arquivo bzImage aparece como executável, bem como o vmlinux que também é gerado por esse procedimento, no entanto apenas o bzImage aparece o nome do kernel.

```
gabriel@gabriel: ~/kernel/WSL2-Linux-Kernel/arch/x86$ ls
cKbuild      Kconfig.debug  boot      entry      include  math-emu  pci      ras      video
Kconfig      Makefile        built-in.a events     kernel    mm         platform realmode xen
Kconfig.assembler Makefile.um     configs  hyperv    kvm      modules.order power  tools
Kconfig.cpu   Makefile.32.cpu crypto   ia32      lib       net        purgatory um

gabriel@gabriel:~/kernel/WSL2-Linux-Kernel/arch/x86$ cd ..
gabriel@gabriel:~/kernel/WSL2-Linux-Kernel/arch$ cd x86
gabriel@gabriel:~/kernel/WSL2-Linux-Kernel/arch/x86$ ls
Kbuild      Kconfig.debug  boot      entry      include  math-emu  pci      ras      video
Kconfig      Makefile        built-in.a events     kernel    mm         platform realmode xen
Kconfig.assembler Makefile.um     configs  hyperv    kvm      modules.order power  tools
Kconfig.cpu   Makefile.32.cpu crypto   ia32      lib       net        purgatory um

gabriel@gabriel:~/kernel/WSL2-Linux-Kernel/arch/x86$ cd boot/
gabriel@gabriel:~/kernel/WSL2-Linux-Kernel/arch/x86/boot$ ls
Makefile      cmdline.c      cpuflags.c      genimage.sh      mkcpustr.c      regs.o      tty.o      video-vesa.o
a20.c         cmdline.o      cpuflags.h      header.S          mtools.conf.in  setup.bin    version.c    video-vga.c
a20.o         compressed    cpuflags.o      header.o          pm.c            setup.elf    version.o    video-vga.o
apm.c         copy.S        cpustr.h        install.sh        pm.o            setup.ld     vesa.h       video.c
bioscall.S    copy.o        ctype.h         main.c            pmjump.S        string.c     video-bios.c video.h
bioscall.o    cpu.o         early_serial_console.c main.o            pmjump.o        string.h     video-bios.o video.o
bitops.h      cpu.o         early_serial_console.o memory.c          printf.c        string.o     video-mode.c vmlinux.bin
boot.h        cpucheck.c   edd.c           memory.o          printf.o        tools        video-mode.o voffset.h
bzImage       cpucheck.o   edd.o           mkcpustr          regs.c          tty.c        video-vesa.c zoffset.h

gabriel@gabriel:~/kernel/WSL2-Linux-Kernel/arch/x86/boot$ file bzImage
bzImage: Linux kernel x86 boot executable bzImage, version 5.15.57.1-microsoft-standard-WSL2+ (gabriel@gabriel) #3 SMP Thu Aug 18 18:25:20 -03 2022, R0-rootFS, swap_dev 0x0, Normal VGA

gabriel@gabriel:~/kernel/WSL2-Linux-Kernel/arch/x86/boot$ file ~/kernel/WSL2-Linux-Kernel/vmlinux
/home/gabriel/kernel/WSL2-Linux-Kernel/vmlinux: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, BuildID[sha1]=27e95ac5f878985ca4395fa8390e67d05cd93d17, with debug_info, not stripped
gabriel@gabriel:~/kernel/WSL2-Linux-Kernel/arch/x86/boot$
```

Figura 9: Vmlinux e bzImage

3. Após a instalação do Kernel, em qual local (diretório) foi armazenado o executável do Kernel?

Resposta: Em arch/x86/boot/ onde podemos encontrar o bzImage e como apresentado pela figura 6.

4. Em qual nível de privilégio a chamada de sistema implementada irá executar (usuário ou kernel/núcleo)?

Resposta: No kernel, tendo em vista de que se trata de um "print" que necessita do auxílio do kernel.

5. Um roteiro típico contendo as etapas da execução de uma chamada de sistema é apresentado nas páginas 23 e 24 do livro (Capítulo 2 do livro do Mazieiro). Você entendeu todos os passos de 1 a 8?

6. O Kernel precisará de ser recompilado toda vez que uma nova funcionalidade for adicionada ao kernel? Explique. Provavelmente, você terá que pesquisar na internet. Não precisa se preocupar em dar a resposta correta, apenas pense sobre o assunto e procure responder a questão.

## Bibliografia

AGUIRRE, L. A. Introdução à Identificação de Sistemas, Técnicas Lineares e Não lineares Aplicadas a Sistemas Reais. Belo Horizonte, Brasil, EDUFMG. 2004.