

Facultad de Ingeniería
Escuela de Sistemas
Sistemas Operativos 1

Proyecto 1

Manual de instalación de KVM, Compilación de un nuevo Kernel, Creación de módulos.

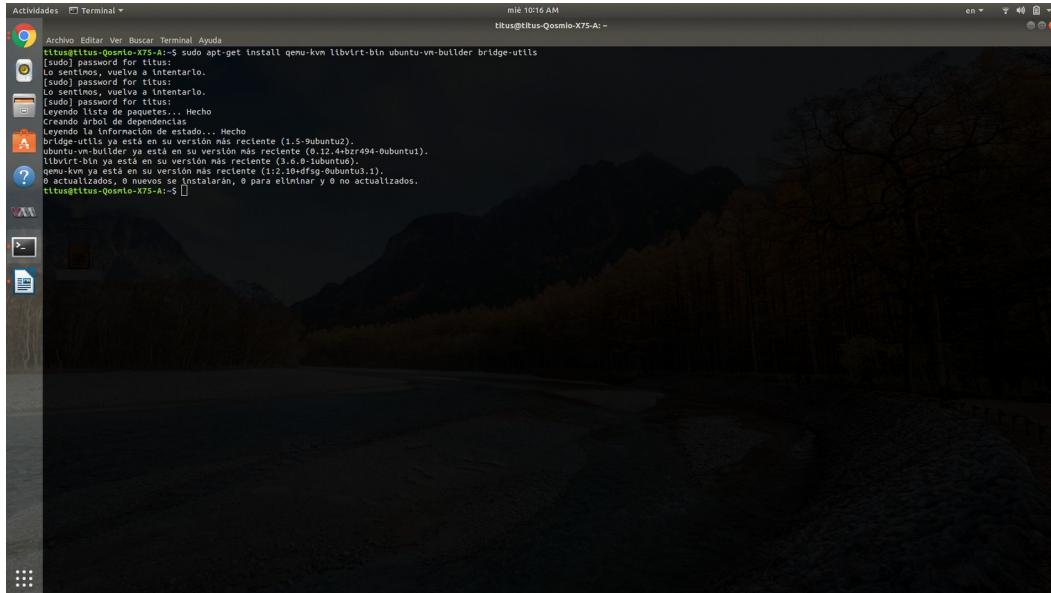
Marvin Emmanuel Pivaral Orellana
201213587

Guatemala 22 de diciembre de 2017

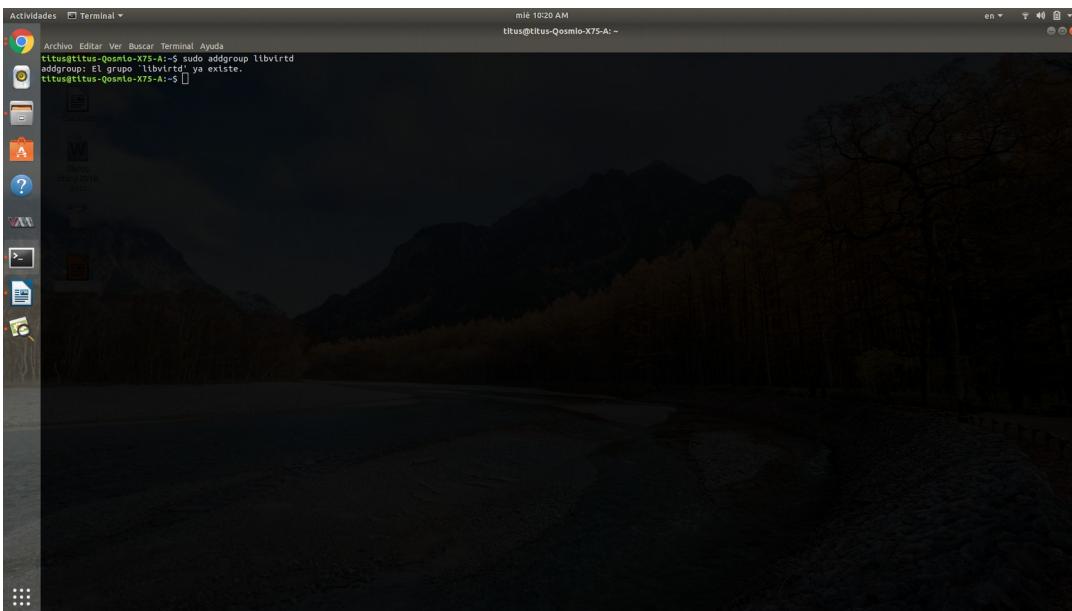
1. Instalación de KVM

1. Instalamos los siguiente paquetes con el siguiente comando “sudo apt-get install qemu-kvm libvirt-bin ubuntu-vm-builder bridge-utils”

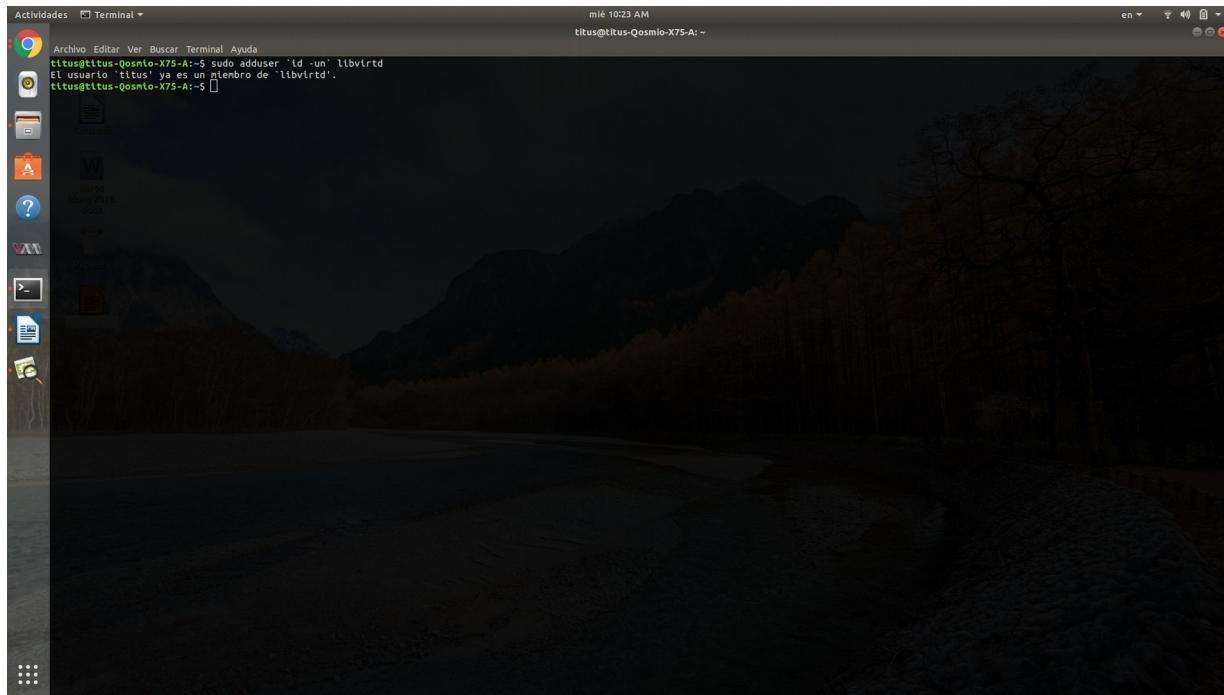
1. qemu-kvm
2. libvirt-bin
3. ubuntu-vm-builder
4. brigde-utils



2. Agregamos el grupo “libvирtd” con el comando “sudo addgroup libvирtd”

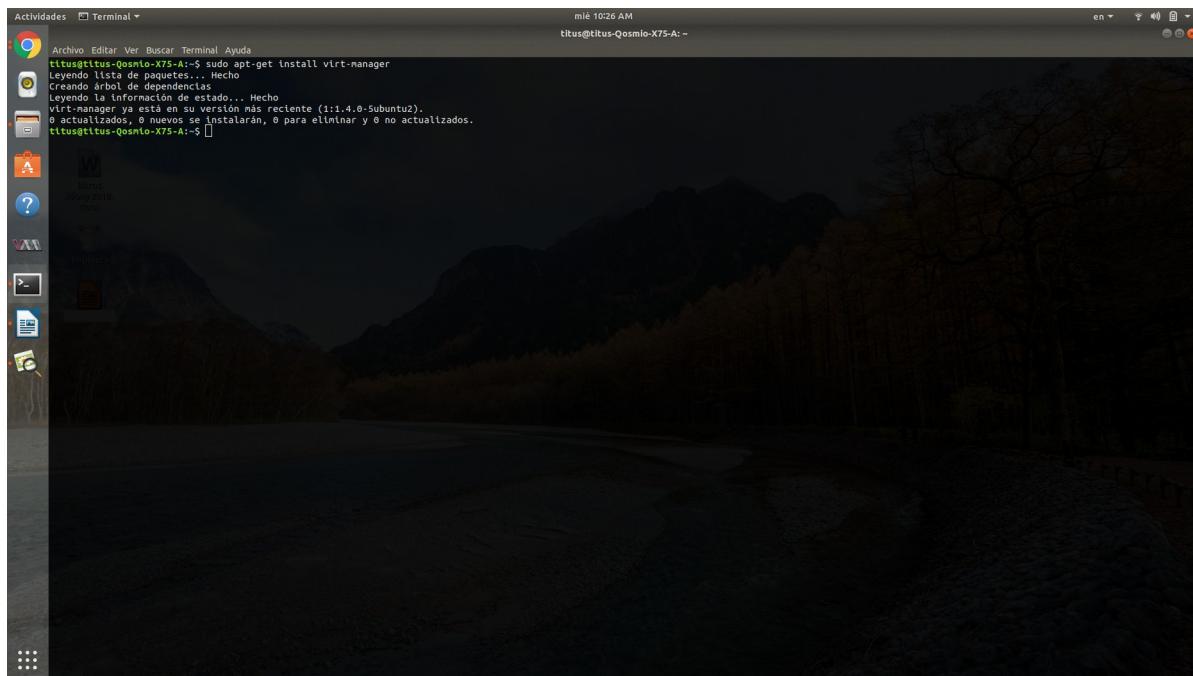


3. Agregamos el usuario actual al grupo que creamos anteriormente con el siguiente comando “sudo adduser `id -un` libvirtd”

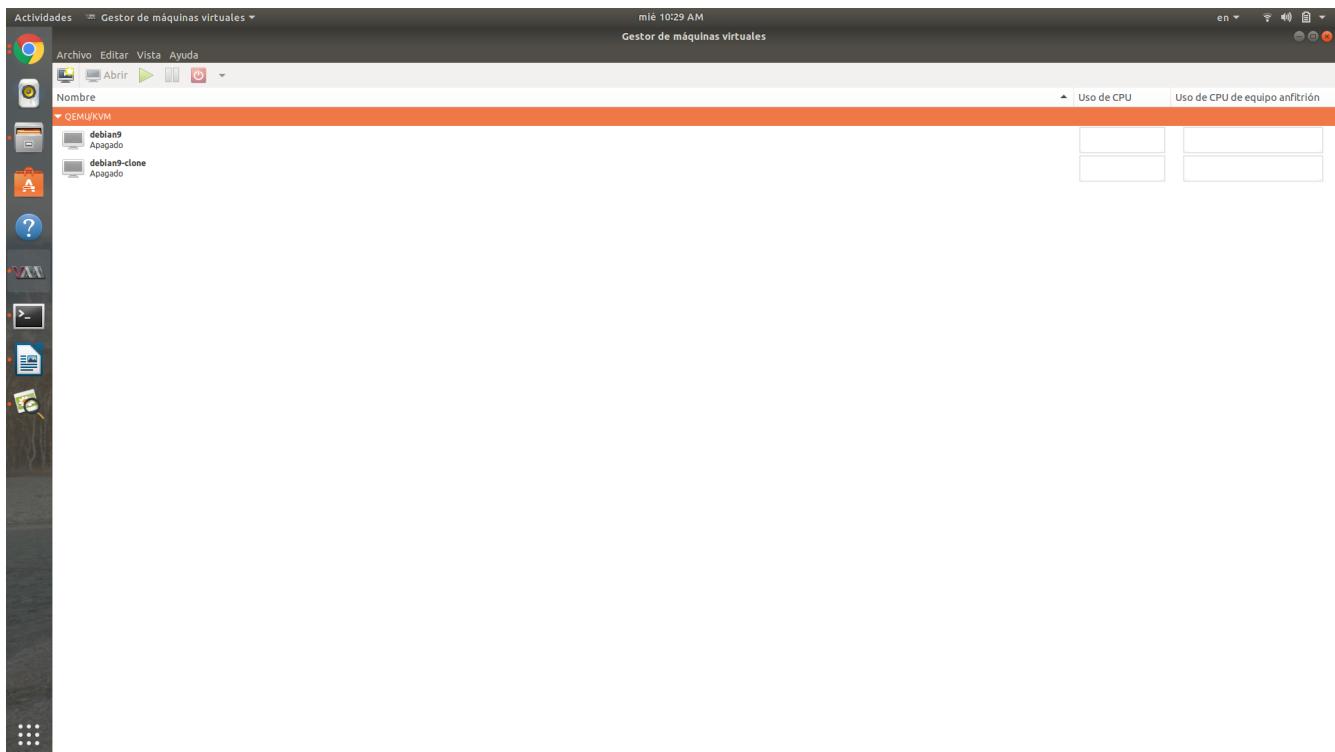


4. Reiniciamos nuestro ordenador para que los cambios tomen efecto.

5. Instalamos la herramienta gráfica “Gestor de maquinas virtuales” para administrar las maquinas virtuales que tendremos en KVM con el siguiente comando “sudo apt-get install virt-manager”

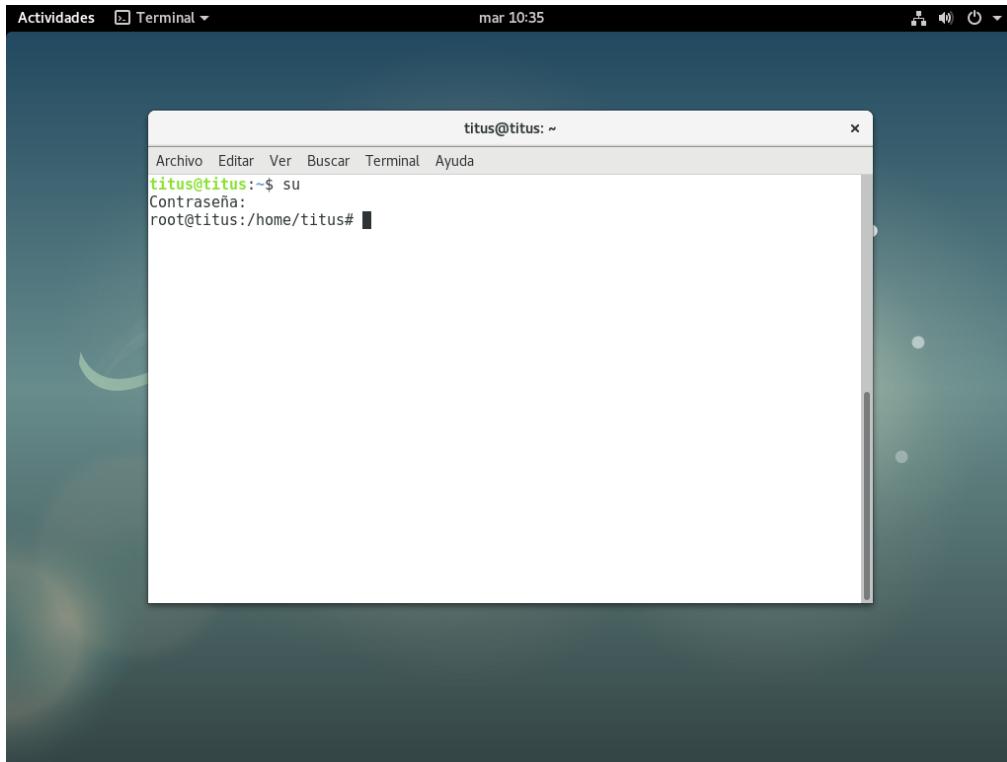


6. Con esto ya tendremos instalado todo lo necesario y lo unico que queda es inicializar el programa y administrar nuestras maquinas virtuales.

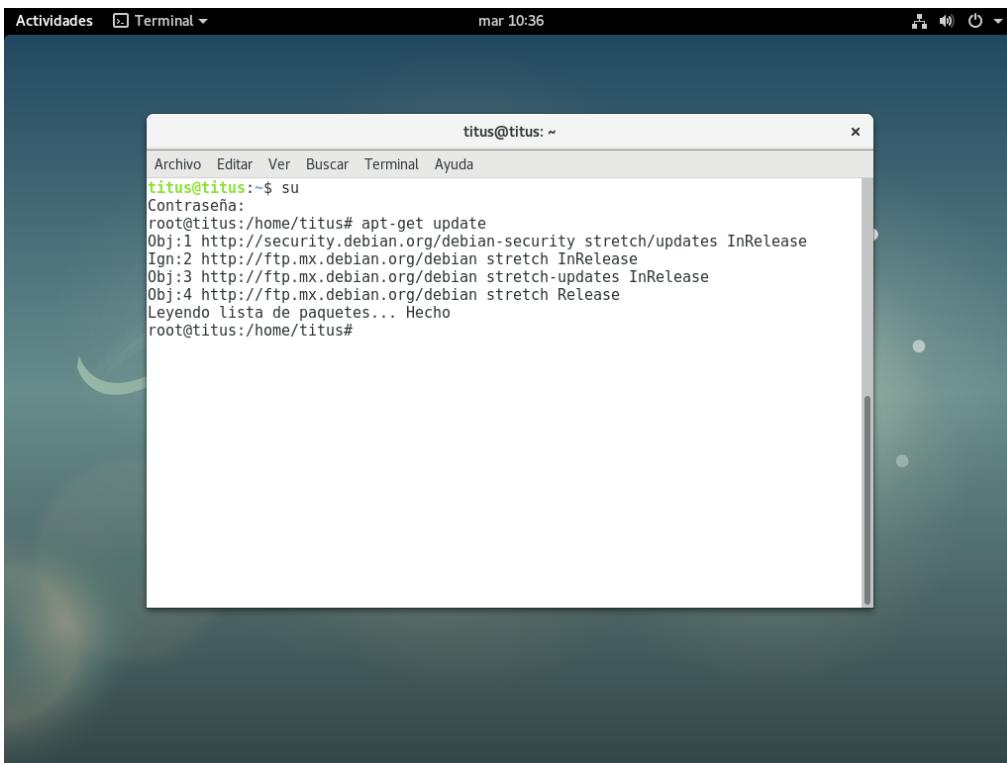


2. Compilacion del Kernel

1. Abrimos una terminal y accedemos como super usuario con el comando “su” e ingresamos nuestra contraseña de super usuario.



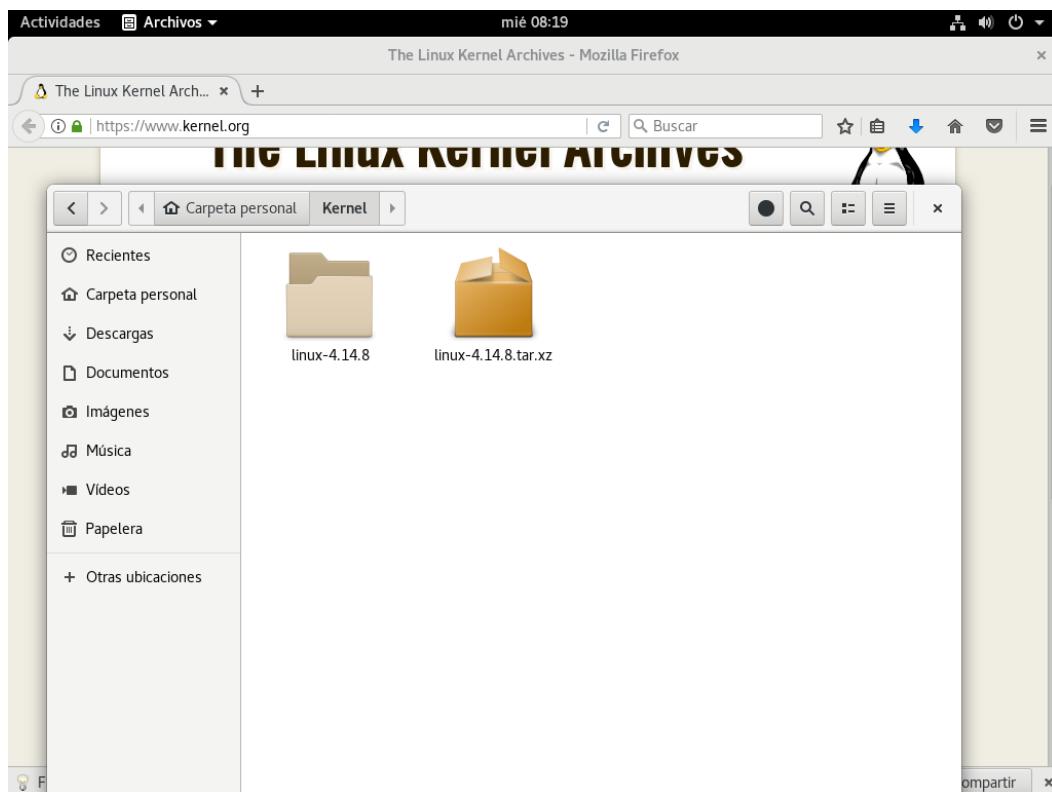
2. Luego actualizamos la cabezarea de nuestro debian con el comando “apt-get update”.



3. Descargamos la ultima version estable del kernel de linux, en este caso usaremos la version 4.14.8.

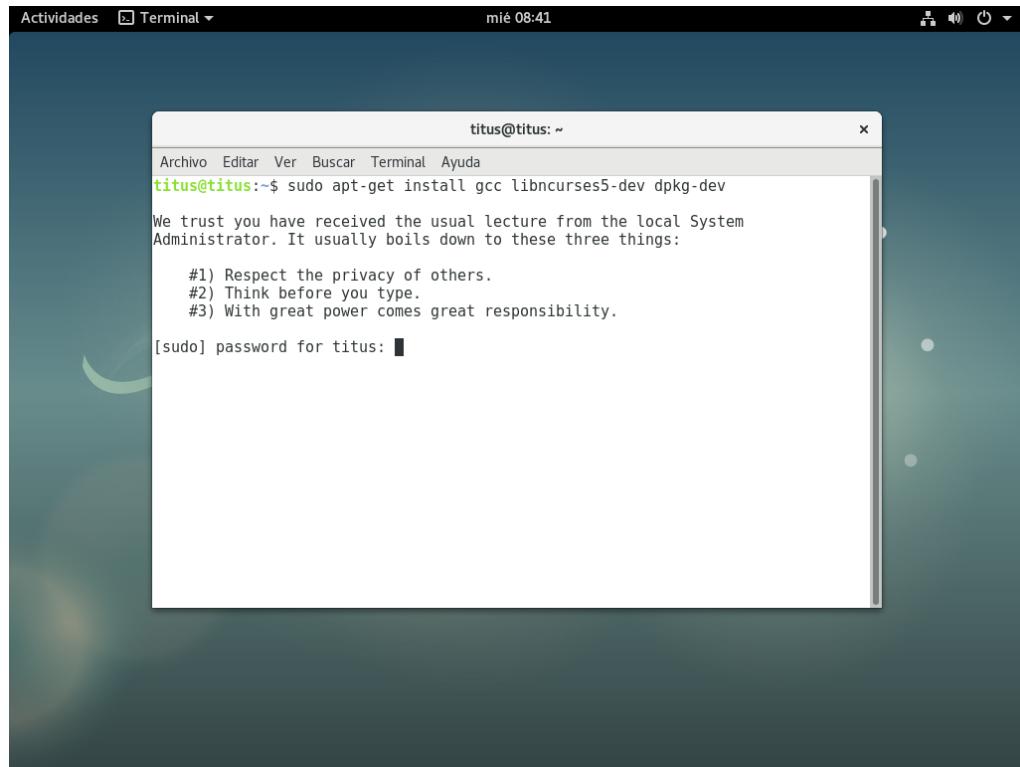


4. Descomprimimos el kernel en un carpeta llamada “kernel” dentro de nuestra carpeta personal

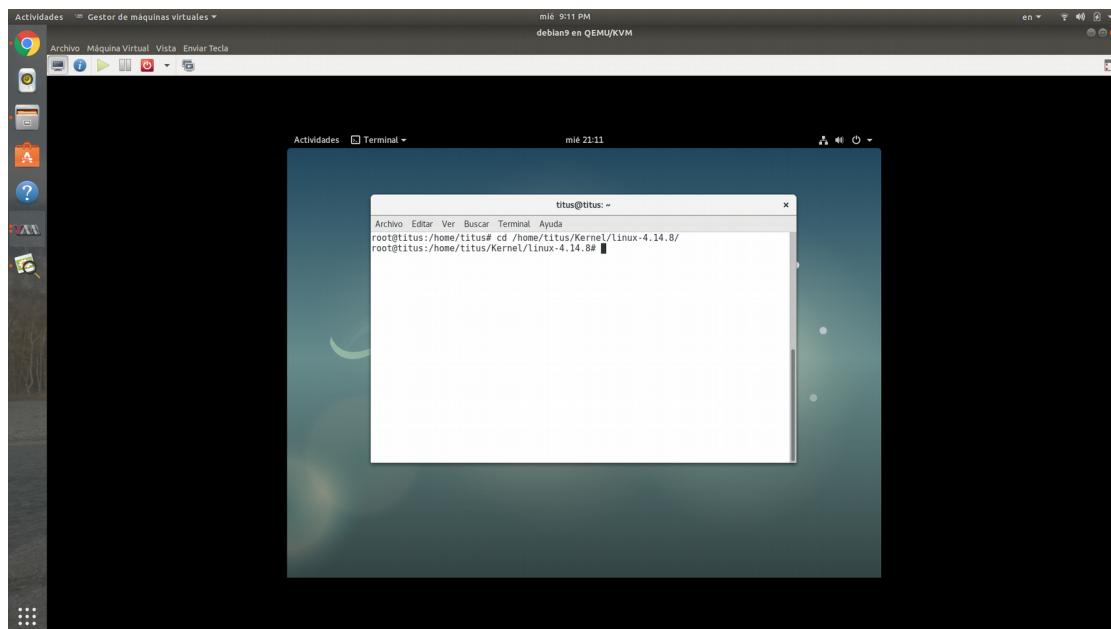


5. Instalamos los siguiente paquetes con el comando “apt-get install gcc libncurses5-dev dpkg-dev”:

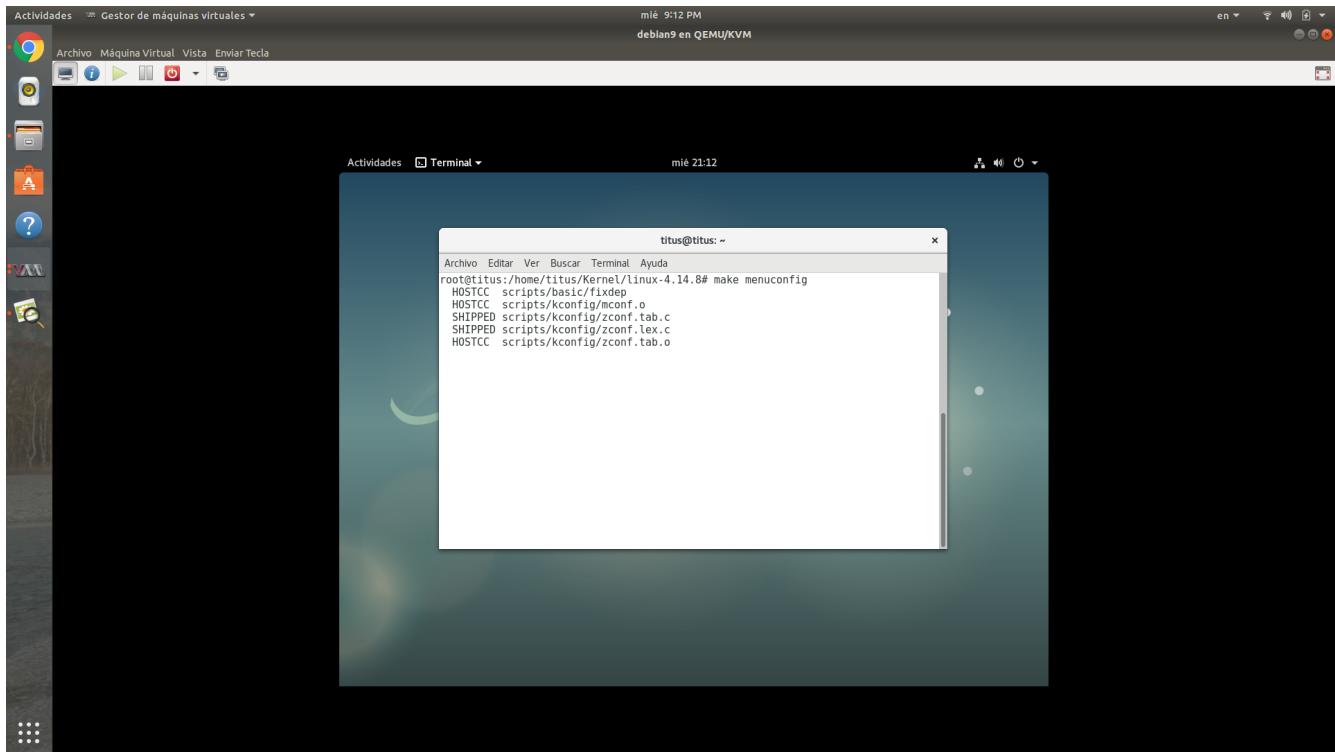
1. gcc
2. libncurses5-dev
3. dpkg-dev



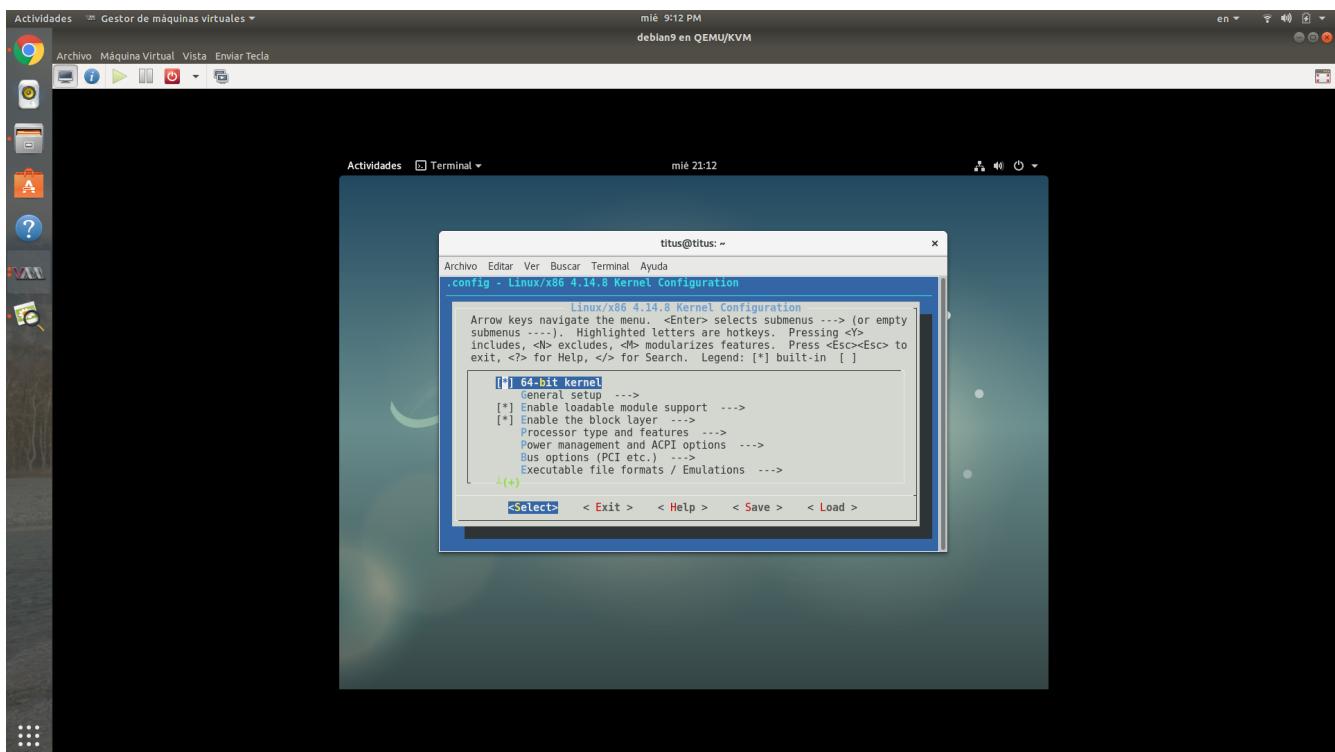
6. Nos colocamos en la ruta donde descomprimimos el kernel.



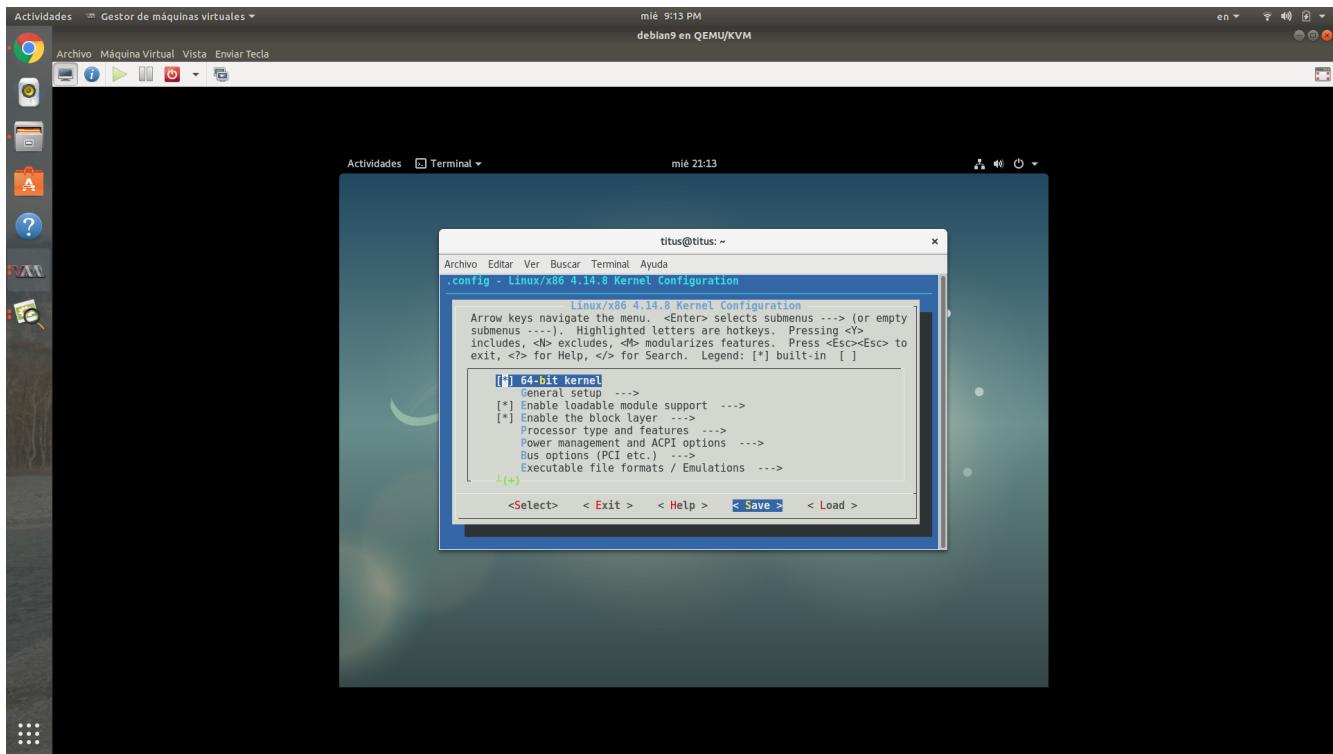
7. Ejecutamos el siguiente comando “make menuconfig”, para configurar el kernel antes de que lo compilemos.



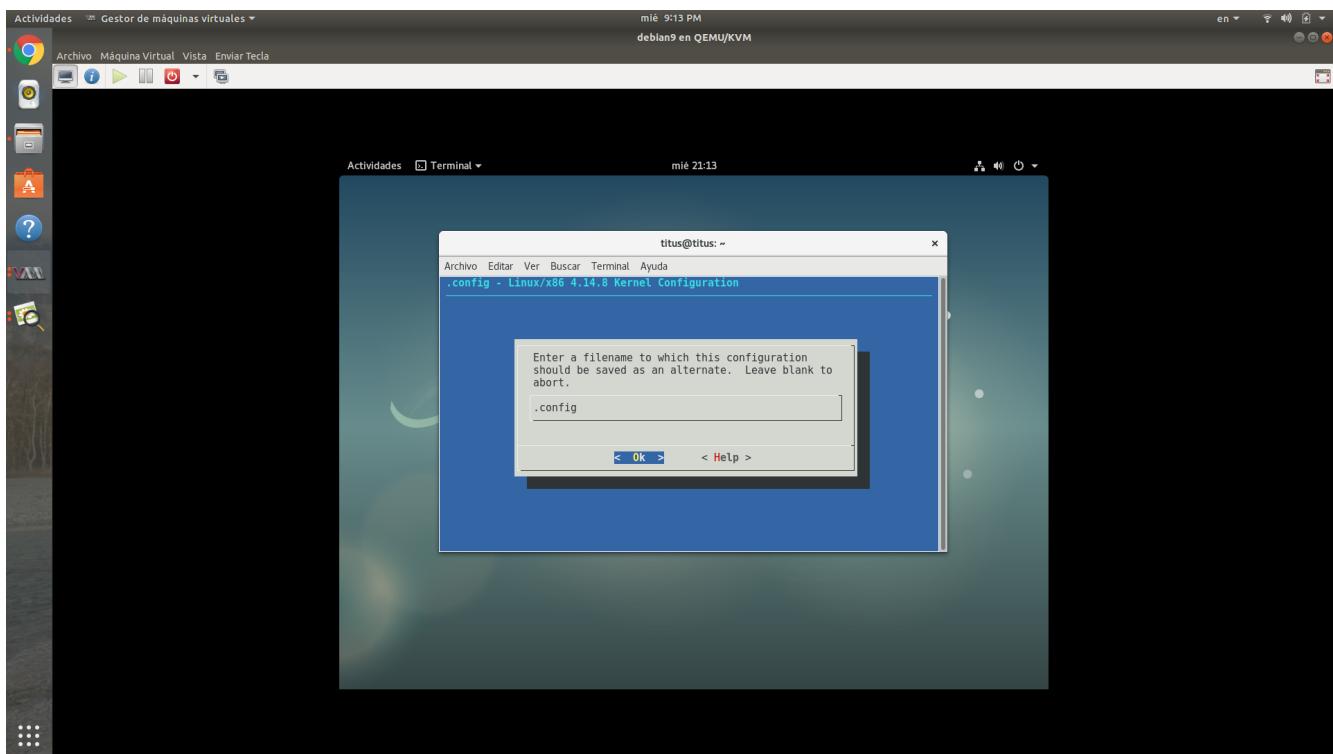
8. Se nos desplegará un menú, lo único que debemos hacer es irnos a la opción “save”.



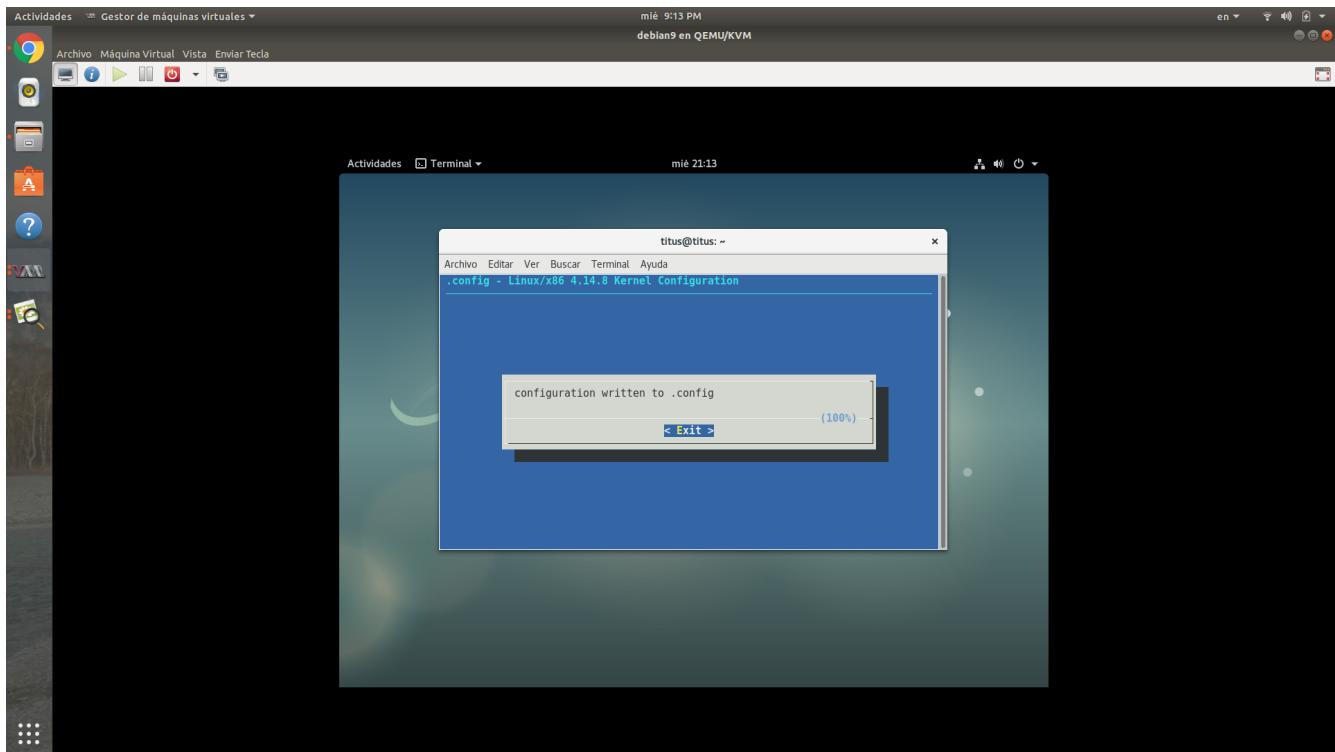
9. Le damos enter.



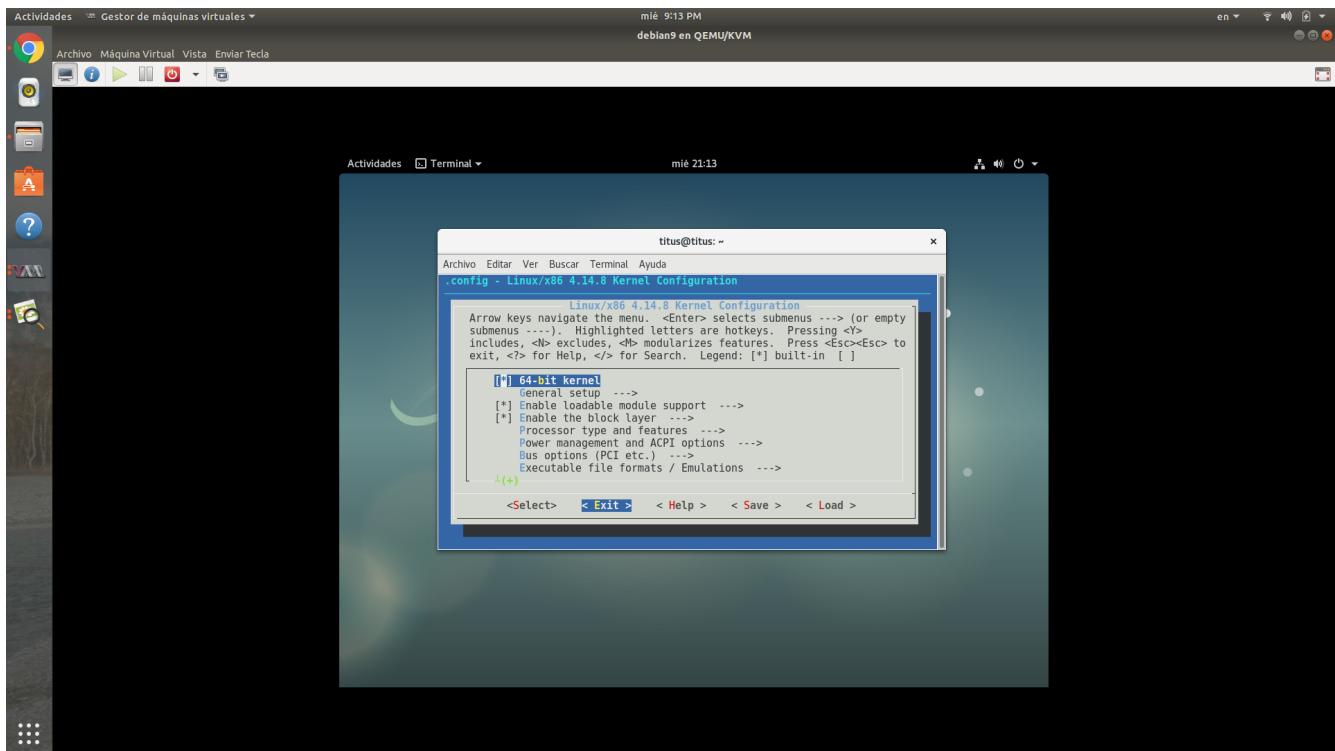
10. Nos preguntara con que nombre guardamos la configuracion, dejamos la que tiene por defecto que seria “.config”.



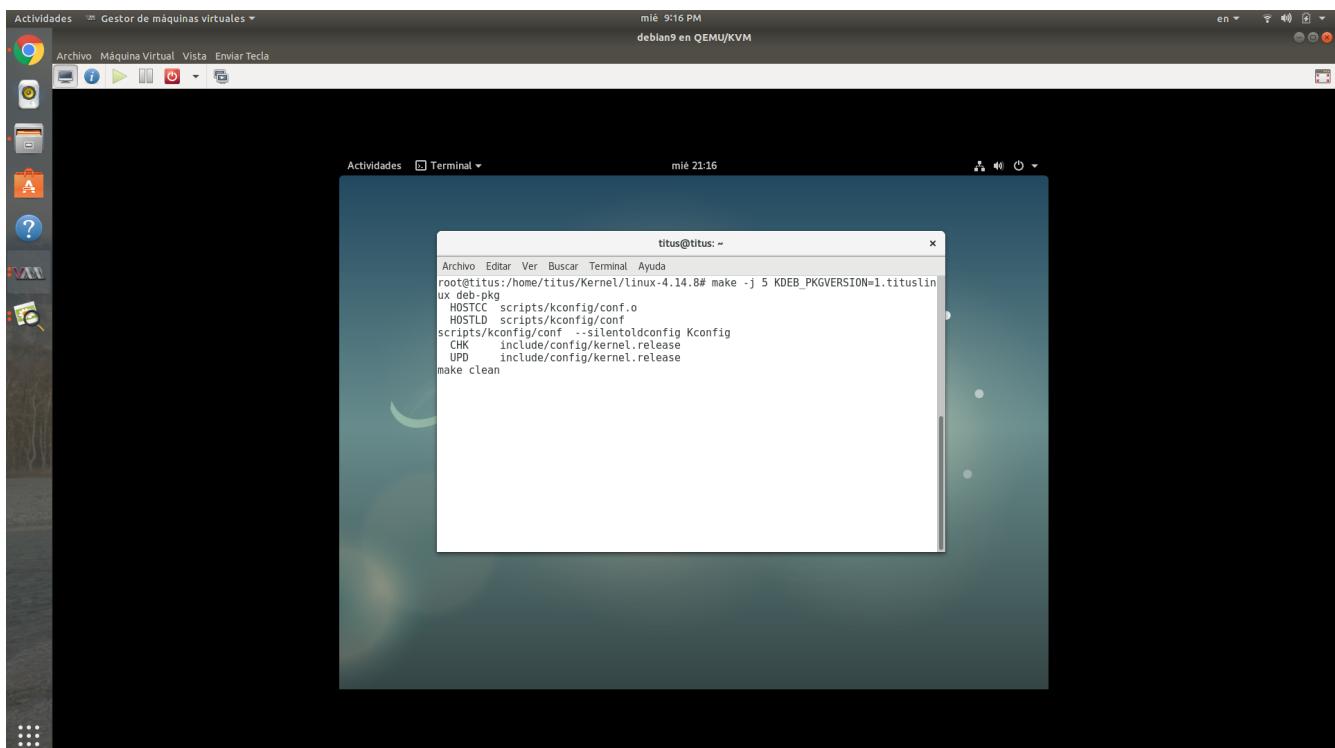
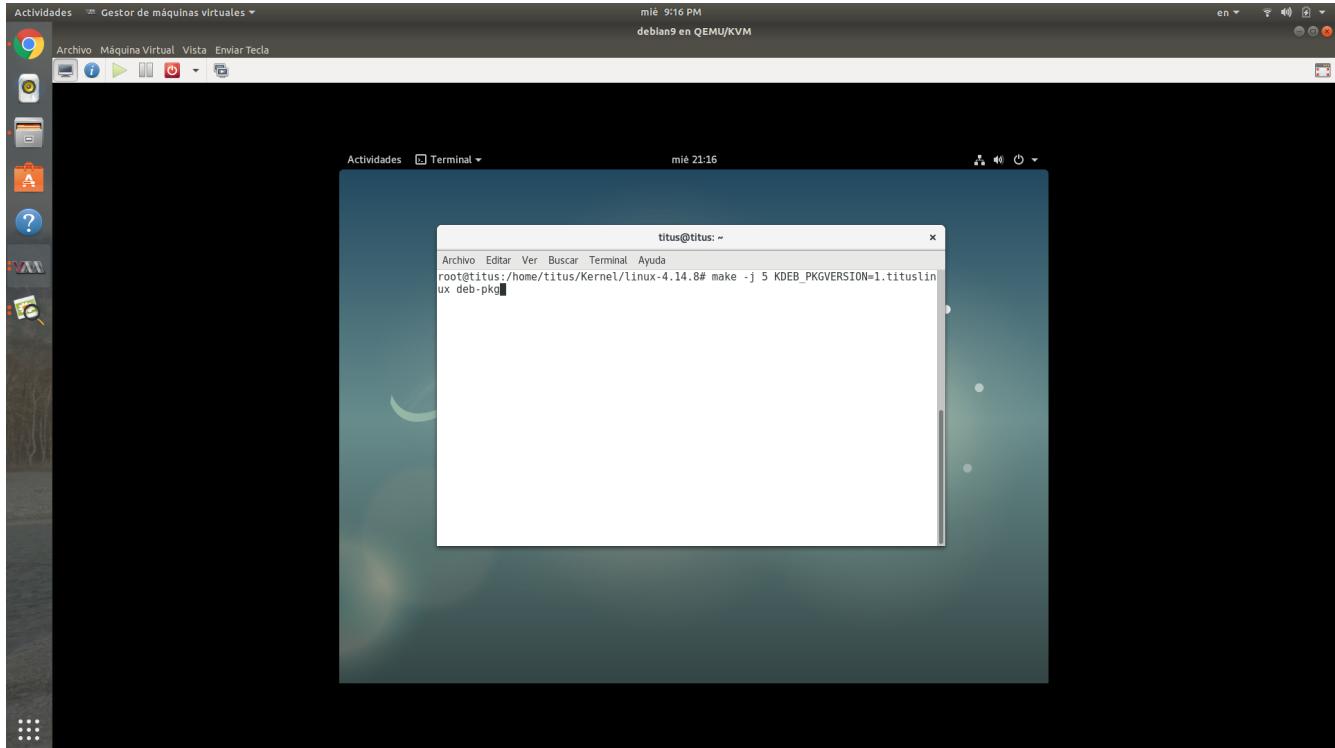
11. Le damos enter en la opcion de “Ok”, luego damos enter en “Exit”.



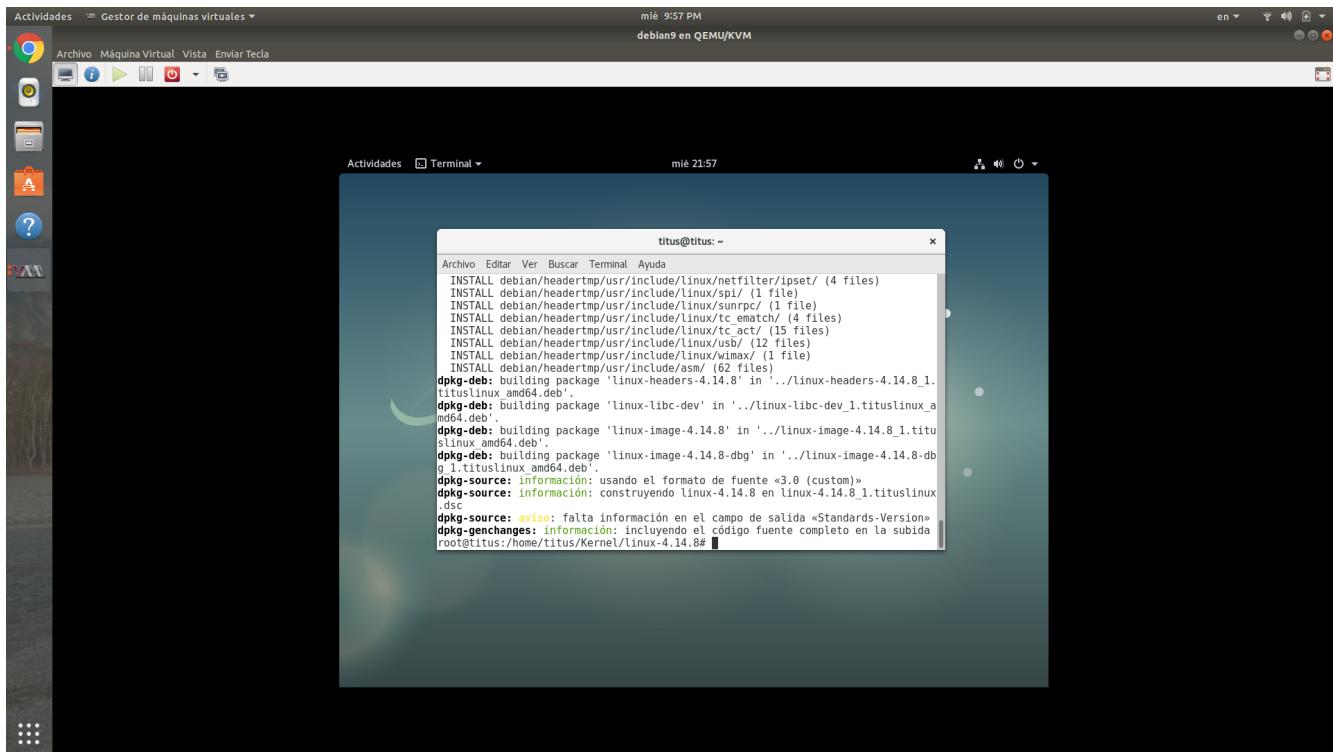
12. y por ultimo salimos de la configuracion dando otro enter en “Exit”.



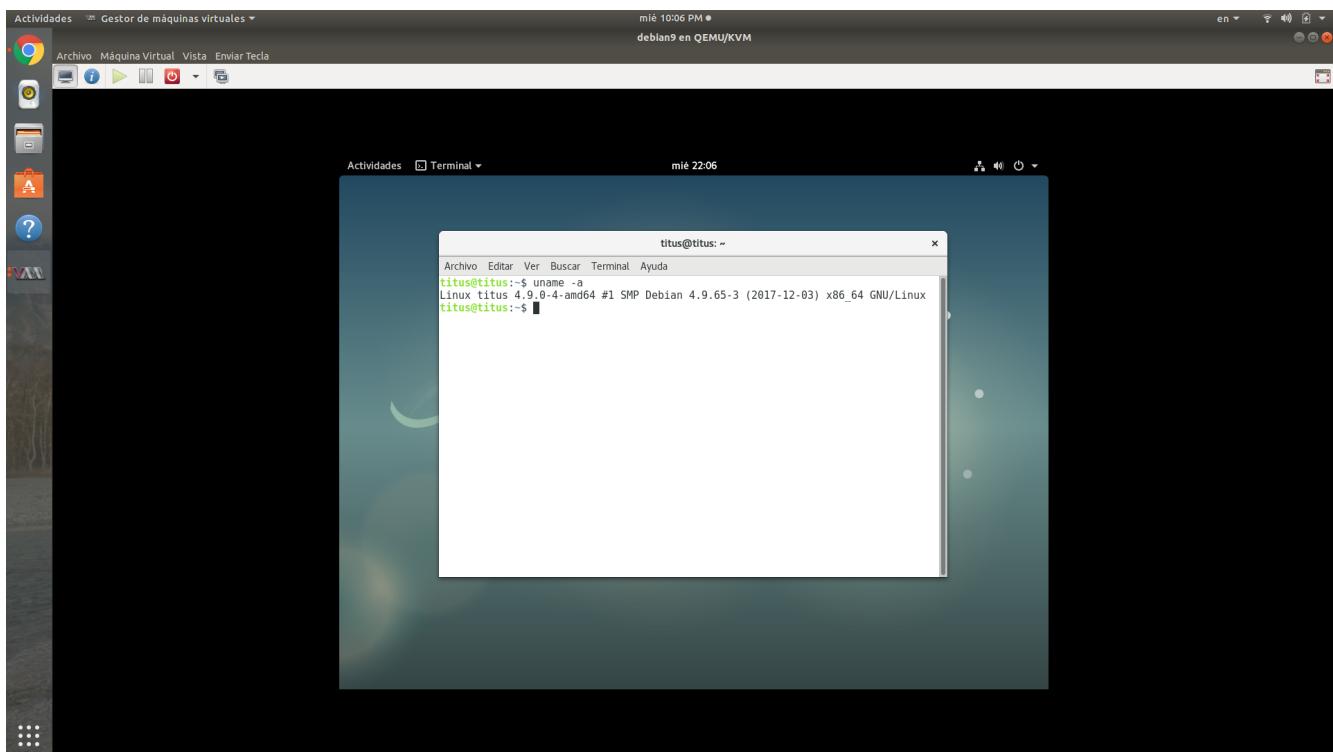
13. Ahora ingresamos el siguiente comando “make -j 5 KDEB_PKGVERSION=1.tituslinux deb-pkg”, para comenzar con la compilación del kernel, esto puede tomar varios minutos, incluso horas, dependiendo de la cantidad de RAM y nucleos que tenga tu procesador.



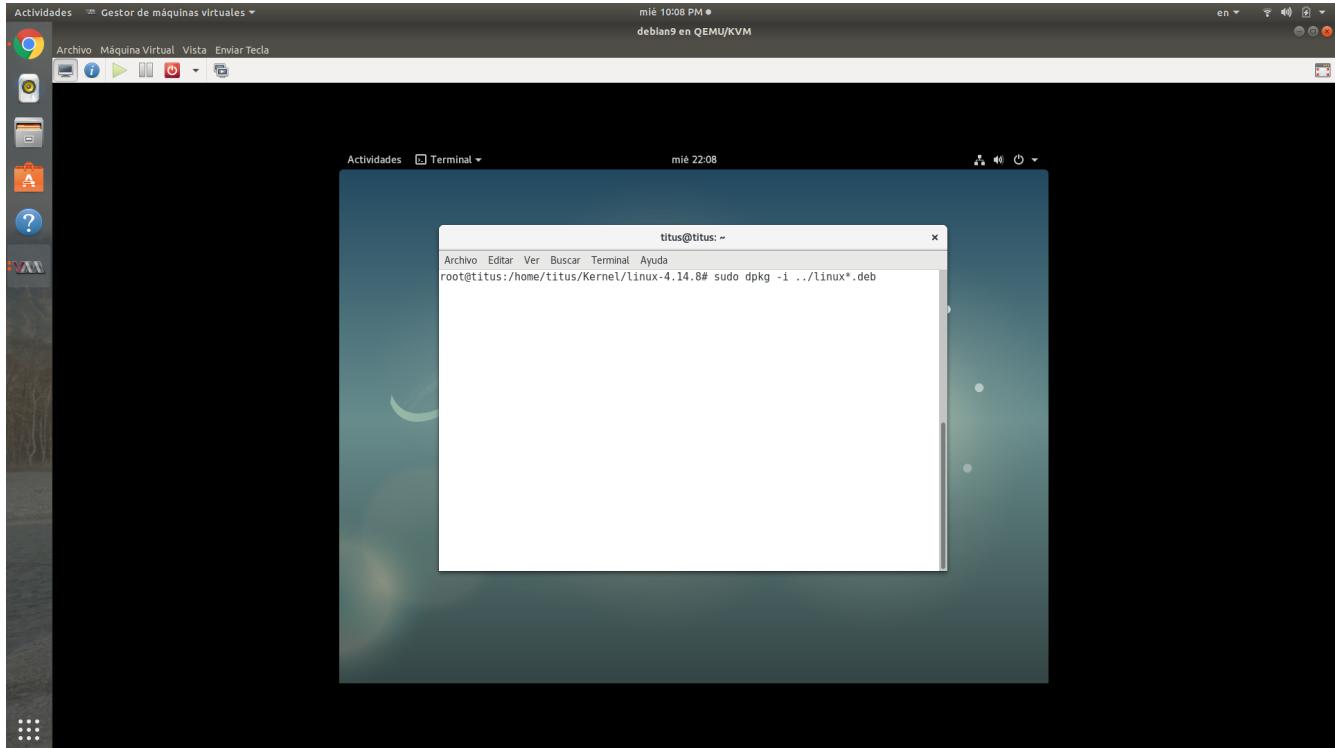
14. Una vez finalizada la compilacion, nos aparecera un mensaje como el siguiente.



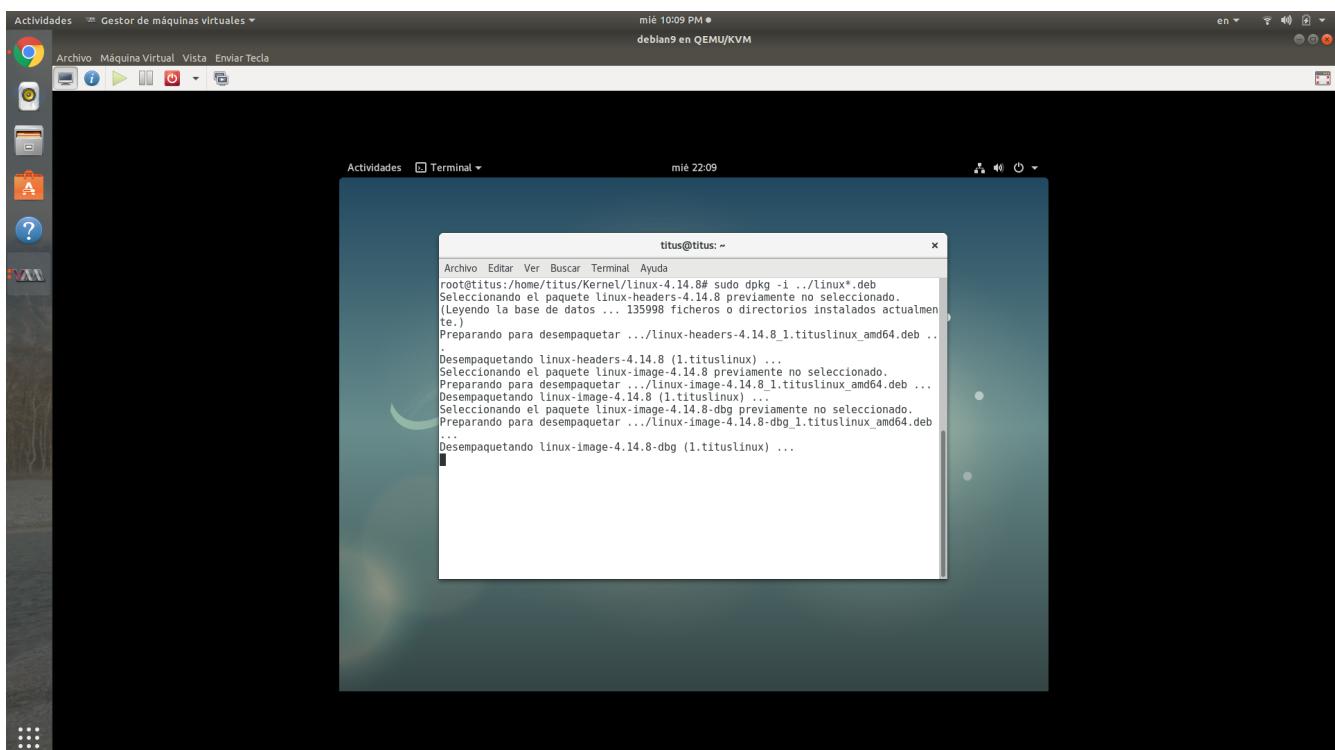
15. Antes de comenzar la instalacion del kernel revisaremos la version que tenemos instalada actualmente que seria la 4.9.0 con el comando “uname -a”.



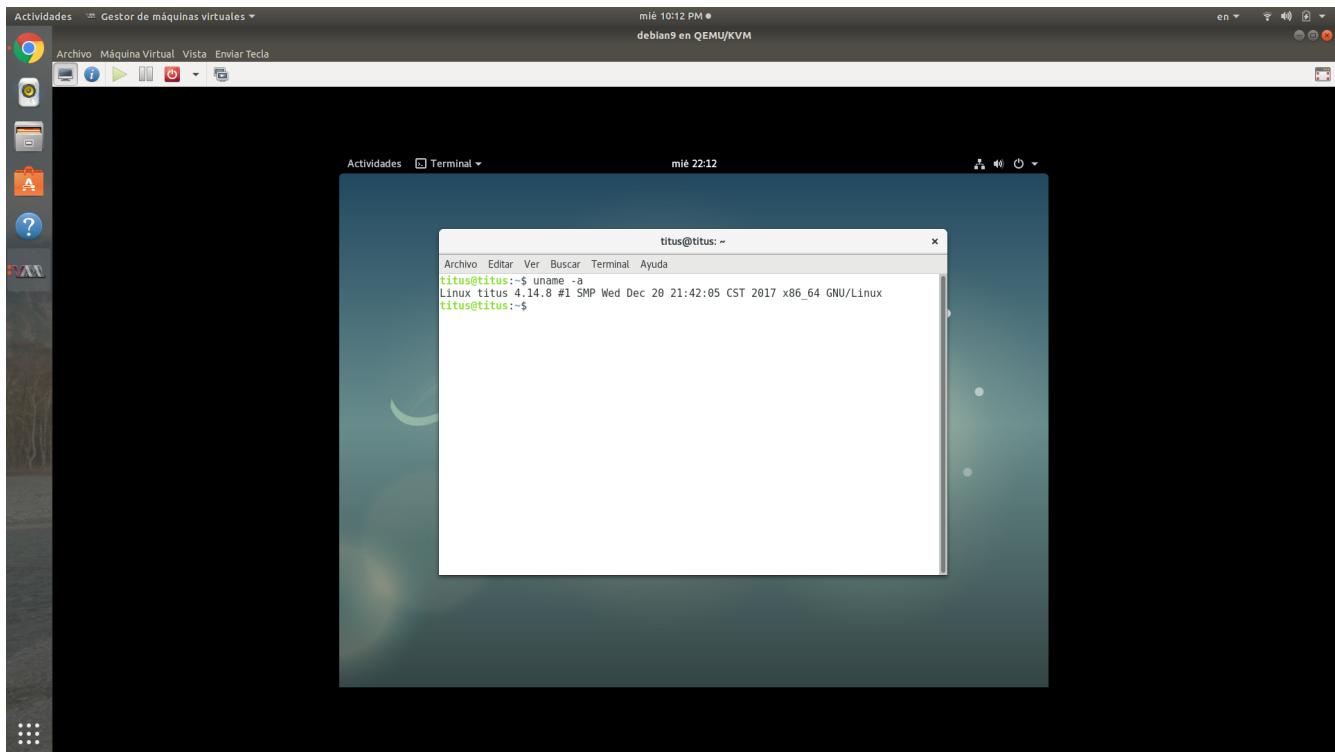
16. Antes de comenzar con este paso, aconsejo reiniciar el ordenador ya que la compilación consume muchos recursos del mismo y con esto liberaremos esos recursos, ahora para instalar el nuevo kernel ingresamos el siguiente comando “sudo dpkg -i ../linux*.deb”.



17. Comenzara la instalacion del nuevo kernel e instalara todos los paquetes y cabezeras que se hayan compilado.



18. Una vez terminada la instalacion reiniciamos, si todo funcione, volvera a iniciar el sistema operativo, y por ultimo verificaremos nuevamente la version del kernel que tenemos instalada con el comando “uname -a”, en nuestro caso se mostrara que ahora tenemos la version 4.14.8.



3. Creacion de modulos

Antes de comenzar con el codigo para crear modulos, hablaremos sobre los structs que utilizaremos para leer la informacion del kernel.

Sysinfo

```
struct sysinfo {  
    long uptime;          /* Seconds since boot */  
    unsigned long loads[3]; /* 1, 5, and 15 minute load averages */  
    unsigned long totalram; /* Total usable main memory size */  
    unsigned long freeram; /* Available memory size */  
    unsigned long sharedram; /* Amount of shared memory */  
    unsigned long bufferram; /* Memory used by buffers */  
    unsigned long totalswap; /* Total swap space size */  
    unsigned long freeswap; /* Swap space still available */  
    unsigned short procs; /* Number of current processes */  
    char _f[22];           /* Pads structure to 64 bytes */  
};
```

Retorna informacion sobre la memoria RAM y la memoria SWAP

List_head

Lista vinculada que acepta cualquier estructura.

task_struct

```
struct task_struct {  
/* these are hardcoded - don't touch */  
    volatile long      state;      /* -1 unrunnable, 0 runnable, >0 stopped */  
    long              counter;  
    long              priority;  
    unsigned          long signal;  
    unsigned          long blocked; /* bitmap of masked signals */  
    unsigned          long flags;   /* per process flags, defined below */  
    int               errno;  
    long              debugreg[8]; /* Hardware debugging registers */  
    struct exec_domain *exec_domain;  
/* various fields */  
    struct linux_binfmt *binfmt;  
    struct task_struct *next_task, *prev_task;  
    struct task_struct *next_run, *prev_run;  
    unsigned long      saved_kernel_stack;  
    unsigned long      kernel_stack_page;  
    int               exit_code, exit_signal;  
/* ??? */  
    unsigned long      personality;
```

```

int      dumpable:1;
int      did_exec:1;
int      pid;
int      pgrp;
int      tty_old_pgrp;
int      session;
/* boolean value for session group leader */
int      leader;
int      groups[NGROUPS];
/*
 * pointers to (original) parent process, youngest child, younger sibling,
 * older sibling, respectively. (p->father can be replaced with
 * p->p_pptr->pid)
 */
struct task_struct  *p_opptr, *p_pptr, *p_cptr,
                    *p_ysptr, *p_osptr;
struct wait_queue  *wait_chldexit;
unsigned short    uid,euid,suid,fsuid;
unsigned short    gid,egid,sgid,fsgid;
unsigned long     timeout, policy, rt_priority;
unsigned long     it_real_value, it_prof_value, it_virt_value;
unsigned long     it_real_incr, it_prof_incr, it_virt_incr;
struct timer_list real_timer;
long      utime, stime, cutime, cstime, start_time;
/* mm fault and swap info: this can arguably be seen as either
   mm-specific or thread-specific */
unsigned long    min_flt, maj_flt, nswap, cmin_flt, cmaj_flt, cnswap;
int swappable:1;
unsigned long    swap_address;
unsigned long    old_maj_flt; /* old value of maj_flt */
unsigned long    dec_flt;    /* page fault count of the last time */
unsigned long    swap_cnt;   /* number of pages to swap on next pass */
/* limits */
struct rlimit    rlim[RLIM_NLIMITS];
unsigned short   used_math;
char      comm[16];
/* file system info */
int      link_count;
struct tty_struct *tty;      /* NULL if no tty */
/* ipc stuff */
struct sem_undo   *semundo;
struct sem_queue   *semsleeping;
/* ldt for this task - used by Wine. If NULL, default_ldt is used */
struct desc_struct *ldt;
/* tss for this task */
struct thread_struct tss;
/* filesystem information */
struct fs_struct   *fs;
/* open file information */

```

```

struct files_struct *files;
/* memory management info */
struct mm_struct *mm;
/* signal handlers */
struct signal_struct *sig;
#ifndef __SMP__
int processor;
int last_processor;
int lock_depth; /* Lock depth.
We can context switch in and out
of holding a syscall kernel lock... */
#endif
};

```

Describe un proceso o tarea en el sistema.

1. Para crear un modulo se definen basicamente dos funciones importantes que son las que se utilizan para su carga y descarga, esto son el init y exit que se declararian asi

```

module_init(nombre_funcion)
module_exit(nombre_funcion)

```

2. Ahora dejare unos ejemplo para un modulo que se encarga de leer informacion basica de la memoria RAM y lo escribira en un archivo en el directorio proc, y un modulo que mostrara el listado de procesos y su informacion de la misma forma, estos se llamaran memo_201213587 y cpu_201213587

4. Para poder ejecutar estos modulos debemos de crear un Makefile para cada uno de ellos con el siguiente codigo, en el nombre del objeto salida variaremos cpu por memo:

```
obj-m += cpu_201213587.o
```

all:

```
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

clean:

```
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

5. Una vez guardamos los Makefile ingresamos el siguiente comando “make” y generara un archivo con extension ko que sera el que cargaremos al kernel.

6. Para cargar los modulos al kernel basta con introducir el siguiente comando “insmod nombre.ko”.

7. Una vez ya no sea necesario el modulo podemos descargarlo con el siguiente comando “rmmod nomre.ko”

8. Para ver los mensajes de salida al momento de la carga y descarga utilizaremos el comando “dmesg | tail -1”, el 1 nos indica cuantos mensajes obtendra de la cola para mostrar.

Memo 201213587

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
#include <asm/uaccess.h>
#include <linux/hugetlb.h>
#include <linux/mm.h>
#include <linux/mman.h>
#include <linux/mmzone.h>
#include <linux/quicklist.h>
#include <linux/syscalls.h>
#include <linux/swap.h>
#include <linux/swapfile.h>
#include <linux/vmstat.h>
#include <linux/atomic.h>

struct sysinfo info;

static int readMemorie(struct seq_file *m, void *v){
    #define Convert(x) ((x) << (PAGE_SHIFT - 10))
    si_meminfo(&info);
    seq_printf(m, "Carnet: 201213587\n");
    seq_printf(m, "Nombre: Marvin Emmanuel Pivaral Orellana\n");
    seq_printf(m, "Sistema Operativo: Debian GNU/Linux 9.3 (stretch)\n");
    seq_printf(m, "Memoria Total: %8lu kB\n", Convert(info.totalram));
    seq_printf(m, "Memoria Libre: %8lu kB\n", Convert(info.freeram));
    seq_printf(m, "Memoria Usada: %ld %%\n", (((Convert(info.totalram)-Convert(info.freeram))*100) / (Convert(info.totalram))*100)/100);
    #undef K
    return 0;
}

static int Meminfo_open(struct inode *inode, struct file *file){
    return single_open(file, readMemorie, NULL);
}

static const struct file_operations Meminfo_fops = {
    .owner = THIS_MODULE,
    .open = Meminfo_open,
    .read = seq_read,
    .llseek = seq_lseek,
    .release = single_release,
};
```

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Titus");
MODULE_DESCRIPTION("Modulo de memoria - Sistemas Operativos 1");

static int __init memo_201213587_init(void)
{
    printk(KERN_INFO "201213587\n");
    proc_create("memo_201213587", 0, NULL, &Meminfo_fops);
    return 0;
}

static void __exit memo_201213587_cleanup(void)
{
    remove_proc_entry("memo_201213587", NULL);
    printk(KERN_INFO "Sistemas Operativos 1\n");
}

module_init(memo_201213587_init);
module_exit(memo_201213587_cleanup);
```

cpu_201213587

```
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/list.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/sched.h>
#include <linux/seq_file.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/types.h>

void readProcess(struct seq_file *m, struct task_struct *s)
{
    struct list_head *list;
    struct task_struct *task;

    char estado[50];

    switch(s->state){
        case TASK_RUNNING:
            strcpy(estado,"Ejecucion");
            break;

        case TASK_STOPPED:
            strcpy(estado,"Detenido");
            break;

        case TASK_INTERRUPTIBLE:
            strcpy(estado,"Interrumpible");
            break;

        case TASK_UNINTERRUPTIBLE:
            strcpy(estado,"Ininterrumpible");
            break;

        case EXIT_ZOMBIE:
            strcpy(estado,"Zombi");
            break;

        default:
            strcpy(estado, "Desconocido");
    }

    seq_printf(m,"PID: %d\tNombre: %s\tEstado:%s\n",s->pid, s->comm, estado);
```

```

list_for_each(list, &s->children) {
    task = list_entry(list, struct task_struct, sibling);
    readProcess(m, task);
}
}

static int pstree(struct seq_file *m, void *v)
{
    struct task_struct *parent = current;
    while (parent->pid != 1)
        parent = parent->parent;
    readProcess(m, parent);
    return 0;
}

static int meminfo_proc_open(struct inode *inode, struct file *file)
{
    return single_open(file, pstree, NULL);
}

static const struct file_operations meminfo_proc_fops = {
    .open     = meminfo_proc_open,
    .read     = seq_read,
    .llseek   = seq_lseek,
    .release  = single_release,
};

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Titus");
MODULE_DESCRIPTION("Modulo de cpu - Sistemas Operativos 1");

static int __init cpu_201213587_init(void)
{
    printk(KERN_INFO "Marvin Emmanuel Pivaral Orellana\n");
    proc_create("cpu_201213587", 0, NULL, &meminfo_proc_fops);
    return 0;
}

static void __exit cpu_201213587_cleanup(void)
{
    remove_proc_entry("cpu_201213587", NULL);
    printk(KERN_INFO "Sistemas Operativos 1\n");
}

module_init(cpu_201213587_init);
module_exit(cpu_201213587_cleanup);

```