



Facultad de Ingeniería

Escuela de Ingeniería en Bioinformática

---

# Informe Proyecto

## Administración de Sistemas

---

*Alumno:*

Nicolás Rojas Poblete

*Matrícula:*

2016430002

*E-mail:*

nicrojas16@alumnos.utalca.cl

*Profesor:*

Alejandro Valdés

*Módulo:*

Administración de Sistemas

*Minor:*

Desarrollo de Software y  
Administración de Sistemas

# Índice

<b>1. Introducción</b>	<b>5</b>
<b>2. Compilando programas, librerías, kernel</b>	<b>6</b>
2.1. Compilando el compilador GCC . . . . .	6
2.1.1. Prerequisitos GCC 9.3 . . . . .	6
2.1.2. Descarga y Configuración GCC 9.3 . . . . .	7
2.1.3. Instalación GCC 9.3 . . . . .	8
2.2. Compilando Librería KD-tree . . . . .	8
2.2.1. Creando Programa con la librería KD-tree . . . . .	9
2.2.2. Compilación y Prueba del programa . . . . .	11
2.3. Compilando el Kernel . . . . .	12
2.3.1. Descarga y Configuración . . . . .	12
2.3.2. Screen y Compilación . . . . .	14
2.3.3. Instalación de la imagen . . . . .	15
<b>3. Acceso Remoto</b>	<b>16</b>
3.1. Telnet . . . . .	16
3.2. SSH . . . . .	16
3.2.1. Configuración SSH . . . . .	17
3.2.2. Conexión por túnel . . . . .	18

3.2.3. SSHFS: Montar archivos remotos . . . . .	18
---	----

## Índice de figuras

1.	Makefile importando librería KD-Tree . . . . .	11
2.	Ejecución del programa con KD-tree . . . . .	11
3.	Menú de Configuración del Kernel . . . . .	13
4.	Configuración de la familia del CPU . . . . .	13
5.	Archivo de Configuración del Kernel . . . . .	13
6.	Archivo config de ssh para establecer conexión remota mediante un túnel . .	18

# 1. Introducción

## 2. Compilando programas, librerías, kernel

### 2.1. Compilando el compilador GCC

El compilador es muy importante dentro de los sistemas informáticos, se encargan de traducir el código fuente de los programas a un lenguaje que pueda ejecutar la máquina. En sistemas Linux por lo general viene incluido el compilador GCC en una versión defecto. Para ver la versión por defecto se puede ejecutar el siguiente comando.

```
$ gcc --version
gcc (Debian 8.3.0-6) 8.3.0
Copyright (C) 2018 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.
```

En caso de que no se encuentre instalado, se puede instalar desde los repositorios de su distribución por apt u otro gestor de paquetes.

```
$ sudo apt install gcc
```

#### 2.1.1. Prerequisitos GCC 9.3

En este caso se procederá a instalar la versión 9.3 de GCC a partir de la versión 8.3 que viene en nuestro sistema instalada.

Según el sitio web de [manuallinux.eu](http://manuallinux.eu)<sup>[1]</sup> los requisitos para compilar gcc 9.3 son los siguientes:

```
Gawk - (5.0.1)
M4 - (1.4.18)
Libtool - (2.4.6)
Make - (4.3)
Bison - (3.5.3)
Flex - (2.6.4)
Automake - (1.16.1)
Autoconf - (2.69)
Gettext - (0.20.1)
Gperf - (3.1)
Texinfo - (6.7)
```

Librerías en Desarrollo

```
Gmp - (6.2.0)
Mpfr - (4.0.2)
Mpc - (1.1.0)
ISL - (0.18)
```

### 2.1.2. Descarga y Configuración GCC 9.3

Una vez revisadas las dependencias se procede a descargar el código fuente de GCC 9.3, esto en el ftp oficial.

```
$ wget ftp://ftp.mirrorservice.org/sites/sourceware.org/pub/gcc/releases/gcc-9.3.0/gcc-9.3.0.tar.gz
```

Para descomprimir

```
$ tar -xzf gcc-9.3.0.tar.gz
```

Luego es necesario entrar en la carpeta y dentro de los archivos viene incluido un script que descarga las principales dependencias(Gmp,Mpfr,Mpc,ISL).

```
$ cd gcc-9.3.0/
$ contrib/download_prerequisites
```

Luego se puede crear una carpeta aparte en dónde se configure y construya el nuevo compilador.

```
$ cd ~
$ mkdir build-gcc9.3
$ cd build-gcc9.3/
```

Ahora se procede a ejecutar el ./configure con los siguiente parámetros.

```
$ ../gcc-9.3.0/configure --build=x86_64-linux-gnu --host=x86_64-linux-gnu \
--target=x86_64-linux-gnu --enable-shared --enable-threads=posix \
--enable-__cxa_atexit --enable-clocale=gnu --enable-languages=c,c++,fortran \
--prefix=/usr/local/gcc-9.3 --disable-multilib --program-suffix=-9.3
```

### 2.1.3. Instalación GCC 9.3

Una vez terminada la configuración y creado el archivo make file se procede a compilar

```
$ make -j 16
```

En este caso -j 16 indica la cantidad de CPUs disponibles en el sistema para la compilación. Dependiendo de la potencia puede tardar unos pocos minutos hasta un par de horas.

En este caso un CPU con 8 núcleos/16 hilos a 5.0 GHz la compilación solo tardó 15 minutos aproximadamente.

```
real    15m41.040s
user    140m26.060s
sys     3m52.618s
```

Luego como superusuario se copian los binarios al prefix indicado anteriormente en la configuración

```
$ sudo make install-strip
```

Por último es necesario exportar el directorio a la variable de entorno PATH

```
$ export PATH=$PATH:/usr/local/gcc-9.3/bin/
$ export export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/gcc-9.3/lib64/
```

Ahora se puede comprobar la version instalada

```
$ gcc-9.3 --version
gcc-9.3 (GCC) 9.3.0
Copyright (C) 2019 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.
```

## 2.2. Compilando Librería KD-tree

El primer paso es descargar el código fuente desde el repositorio oficial.

```
$ wget http://nuclear.mutantstargoat.com/sw/kdtree/files/kdtree-0.5.7.tar.gz
$ tar -xzvf kdtree-0.5.7
$ cd kdtree-0.5.7/
```



Una vez listo se procede a configurar la librería y hacer la compilación.

```
$ ./configure
$ make
```

### 2.2.1. Creando Programa con la librería KD-tree

Se usa como base el ejemplo test2 incluido en el código fuente y se modifica para que solicite un punto inicial y el radio de búsqueda en el árbol. Quedando un programa como el siguiente.

```
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>
#include "kdtree.h"

#define DEF_NUM_PTS 10

/* entrega la distancia euclidiana entre 2 puntos */
static double dist_sq( double *a1, double *a2, int dims );

/* numero random entre -10 y 10 */
static double rd( void );

int main(int argc, char **argv) {
    printf("Kd-tree_de_3_dimensiones\n");
    int i, num_pts = DEF_NUM_PTS;
    void *ptree;
    char *data, *pch;
    struct kdres *presults;
    double pos[3], dist;
    double pt[3];
    printf("Ingrese_un_numero_de_3_dimensiones:(Separado_por_un_espacio)_");
    scanf("%lf_%lf_%lf", &pt[0], &pt[1], &pt[2]);
    double radius;
    printf("Ingrese_el_radio_de_búsqueda:_");
    scanf("%lf", &radius);
    if(argc > 1 && isdigit(argv[1][0])) {
        num_pts = atoi(argv[1]);
    }
    if(!(data = malloc(num_pts))) {
        perror("malloc_failed");
        return 1;
    }
}
```

```

srand( time(0) );
/* Se crea el arbol kd-tree con 3 dimensiones */
ptree = kd_create( 3 );
/* Se agregan nodos con datos random al arbol */
for( i=0; i<num_pts; i++ ) {
    data[i] = 'a' + i;
    assert( 0 == kd_insert3( ptree, rd(), rd(), rd(), &data[i] ) );
}
/* busca los nodos mas cercanos al punto entregado en el radio entregado
   por e

   1 usuario */
presults = kd_nearest_range( ptree, pt, radius );
/* imprime los resultados */
printf( "found_%d_results:\n", kd_res_size(presults) );

while( !kd_res_end( presults ) ) {
    /* se obtiene las coordenadas de cada nodo con resultados */
    pch = (char*)kd_res_item( presults, pos );

    /* calcula la distancia euclidiana entre el resultado del arbol y punto
       ingr

       esado por el usuario */
    dist = sqrt( dist_sq( pt, pos, 3 ) );

    /* Se imprimen los datos calculados */
    printf( "node_at_(%.3f,_%%.3f,_%%.3f)_is_%.3f_away_and_has_data=%c\n",
        pos[0], pos[1], pos[2], dist, *pch );
    /* Se recorre el siguiente resultado */
    kd_res_next( presults );
}
/* libera la memoria utilizada de las estructuras definidas anteriormente
   */
free( data );
kd_res_free( presults );
kd_free( ptree );

return 0;
}

static double dist_sq( double *a1, double *a2, int dims ) {
    double dist_sq = 0, diff;
    while( --dims >= 0 ) {
        diff = (a1[dims] - a2[dims]);
        dist_sq += diff*diff;
    }
    return dist_sq;
}

static double rd( void ) {
    return (double)rand()/RAND_MAX * 20.0 - 10.0;
}

```

Con el código listo se crea el archivo Makefile para hacer su compilación, en dónde se debe considerar la ruta en donde se encuentra la librería kd-tree. Quedando como se muestra en la siguiente imagen.

```
prefix=/usr/local
CC = gcc

CFLAGS = -std=c89 -pedantic -Wall -g -I..
SRC = test2.c
OBJ = test2.o
APP = test2
LDFLAGS= ../libkdtree.a -lm

all: $(OBJ)
    $(CC) $(CFLAGS) -o $(APP) $(OBJ) $(LDFLAGS)

clean:
    $(RM) $(OBJ) $(APP)

install: $(APP)
    install -m 0755 $(APP) $(prefix)/bin

uninstall: $(APP)
    $(RM) $(prefix)/bin/$(APP)

.PHONY: install
```

Figura 1: Makefile importando librería KD-Tree

### 2.2.2. Compilación y Prueba del programa

Se procede a compilar con make y ejecutar el el programa

```
$ make
$ ./test2
```

Como se mencionó anteriormente el programa solicita datos al usuario y luego realiza la búsqueda en un árbol kd-tree generado con números al azar. En la siguiente imagen se muestra su ejecución.

```
Kd-tree de 3 dimensiones
Ingrese un numero de 3 dimensiones: (Separado por un espacio) 1 1 3
Ingrese el radio de busqueda: 7
found 4 results:
node at (0.338, -4.283, 3.609) is 5.359 away and has data=j
node at (-0.699, 3.482, 2.057) is 3.152 away and has data=e
node at (-2.216, -3.809, -0.008) is 6.520 away and has data=c
node at (1.476, 7.077, 6.068) is 6.824 away and has data=b
```

Figura 2: Ejecución del programa con KD-tree

## 2.3. Compilando el Kernel

### 2.3.1. Descarga y Configuración

Primero es necesario saber que versión del kernel viene instalada en nuestro sistema, para ello usamos el siguiente comando.

```
$ uname -a
Linux admSist6 4.19.0-10-amd64 #1 SMP Debian 4.19.132-1 (2020-07-24) x86_64
GNU/Linux
```

Ahora se procede a descargar la última versión estable del kernel

```
$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.8.14.tar.xz
```

Se descomprime el paquete

```
$ tar axvf linux-5.8.14.tar.xz
```

En caso de haber realizado otra configuración en el actual directorio, es necesario limpiar con los siguientes comandos.

```
$ make distclean
$ make clean
```

Luego se debe copiar la configuración con la cuál se configuró el kernel instalado en nuestra máquina.

```
$ cp /boot/config-4.19.0-10-amd64 .config
```

Ahora se carga esta configuración anterior para el nuevo kernel

```
$ yes "" | make oldconfig
```

En este caso puede arrojar algunos errores de dependencias, en este caso fue necesario instalar flex y bison

```
$ apt install flex
$ apt install bison
```

También se puede desplegar un menú en la consola para ver todas las opciones de configuración

```
$ make menuconfig
```

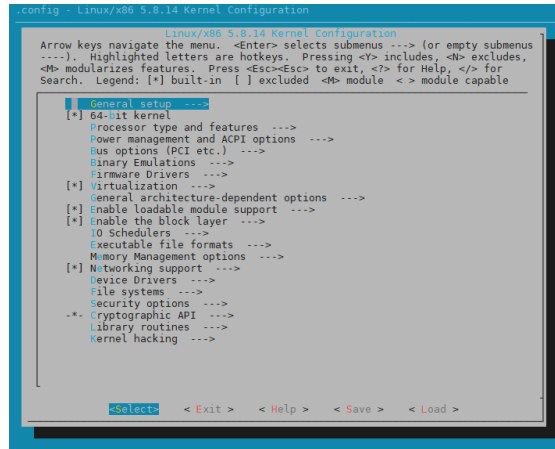


Figura 3: Menú de Configuración del Kernel

Se deben seleccionar las opciones dependiendo de cada sistema y los requerimientos. Para este caso se cambió la configuración específica al tipo de CPU y se desactivó drivers de audio que no serán utilizados.

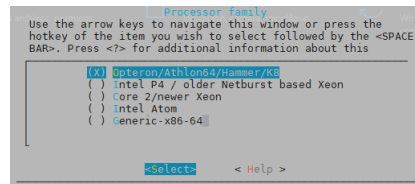


Figura 4: Configuración de la familia del CPU

Una vez configurado se guarda todo y si vemos el contenido del archivo .config se encuentran todos los parámetros de configuración para instalar el nuevo kernel.

```
# Automatically generated file; DO NOT EDIT.
# Linux/x86 5.8.14 Kernel Configuration
#
CONFIG_CC_VERSION_TEXT="gcc (Debian 8.3.0-6) 8.3.0"
CONFIG_CC_IS_GCC=y
CONFIG_GCC_VERSION=80300
CONFIG_LTO_VERSION=231010000
CONFIG_CLANG_VERSION=0
CONFIG_CC_CAN_LINK=y
CONFIG_CC_CAN_LINK_STATIC=y
CONFIG_CC_HAS_ASM_GOTO=y
CONFIG_CC_HAS_ASM_INLINE=y
CONFIG_IRQ_WORK=y
CONFIG_BUILDTIME_TABLE_SORT=y
CONFIG_THREAD_INFO_IN_TASK=y
#
# General setup
#
CONFIG_INIT_ENV_ARG_LIMIT=32
# CONFIG_COMPILE_TEST is not set
CONFIG_LOCALVERSION=""
# CONFIG_LOCALVERSION_AUTO is not set
CONFIG_BUILD_SALT="4.19.0-10-amd64"
CONFIG_HAVE_KERNEL_GZIP=y
CONFIG_HAVE_KERNEL_BZIP2=y
CONFIG_HAVE_KERNEL_LZMA=y
CONFIG_HAVE_KERNEL_XZ=y
CONFIG_HAVE_KERNEL_LZO=y
CONFIG_HAVE_KERNEL_LZ4=y
# CONFIG_KERNEL_GZIP is not set
# CONFIG_KERNEL_BZIP2 is not set
# CONFIG_KERNEL_LZMA is not set
CONFIG_KERNEL_XZ=y
# CONFIG_KERNEL_LZO is not set
# CONFIG_KERNEL_LZ4 is not set
.config 94361, 229152C
```

Figura 5: Archivo de Configuración del Kernel

Dentro de este archivo es importante modificar la línea que hace referencia al certificado del kernel anterior.

```
$ nano .config
CONFIG_SYSTEM_TRUSTED_KEYS=""
```

Además se puede deshabilitar la línea que crea una imagen dbg con la información del debug del kernel, esto permite que la imagen sea mucho más pequeña y así no quedarse sin espacio durante la compilación.

```
$ nano .config
CONFIG_DEBUG_INFO=n
```

### 2.3.2. Screen y Compilación

Ahora quedaría compilar el kernel, para lo cual se hizo una prueba en dos máquinas , una alojada en un servidor remoto y otra localmente.

- La máquina remota cuenta con 2 CPUs con 2 hilos de AMD a 2.7 GHz y 2 GB RAM
- La máquina local cuenta con 8 CPUs con 16 hilos de Intel a 5.0 GHz y 16 GB RAM

Para la máquina remota se configuró el uso de screen que es un software de terminales virtuales que permiten dejar el proceso de compilación corriendo sin necesidad de estar conectado todo el tiempo al servidor remoto.

Instalación de screen

```
$ apt install screen
```

Se crea una terminal en screen con nombre

```
$ screen -S kernel
```

Esto nos deja lista la terminal para lanzar un proceso de larga duración en nuestro servidor remoto. Para salir de esta terminal se debe apretar CTR + A + D. Para volver a la terminal creada se pueden listar las terminales de screen creadas

```
$ screen -ls
There is a screen on:
      30906.kernel      (10/10/2020 07:47:05 PM)      (Detached)
1 Socket in /run/screen/S-root.
```

Con esto podemos volver a conectarnos a esa terminal

```
$ screen -r 30906.kernel
```

Finalmente dejamos corriendo la compilación del kernel

```
$ make -j2 deb-pkg
```

Para la máquina local se usó

```
$ make -j16 deb-pkg
```

Los tiempos de ejecución en la máquina remota fueron los siguientes

```
real    96m49.182s
user    168m48.511s
sys     20m3.484s
```

Mientras que en la máquina local

```
real    11m7.837s
user    129m13.261s
sys     16m43.976s
```

### 2.3.3. Instalación de la imagen

Una vez terminado el proceso de compilación solo quedaría instalar la imagen .deb generada

```
$ cd ..
$ dpkg -i dpkg -i linux-image-5.8.14_5.8.14-1_amd64.deb
```

Terminada la instalación se reinicia la máquina para que arranque con el nuevo kernel.

```
$ shutdown -r now
```

Si la máquina vuelve arrancar se puede verificar el kernel instalado

```
$ uname -a
Linux admSist6 5.8.14 #1 SMP Sun Oct 11 04:00:29 -03 2020 x86_64 GNU/Linux
```

## 3. Acceso Remoto

### 3.1. Telnet

Primero es necesario la instalación de telnet

```
$ sudo apt install telnetd
```

Una vez instalado se debe correr el servicio para poder utilizarlo

```
$ sudo systemctl start inetd
```

Antes de establecer una conexión a otra máquina de forma remota, es necesario crear un nuevo usuario y asignándole un directorio y una contraseña.

Para crear un usuario y asignarle un directorio en /home usamos el siguiente comando.

```
$ sudo useradd -m -s /bin/bash nirojas
```

Luego para asignarle una contraseña

```
$ sudo passwd nirojas
New password:
Retype new password:
passwd: password updated successfully
```

Ahora ya podremos conectarnos mediante telnet

```
$ telnet 10.1.1.33 -l nirojas
Trying 10.1.1.33...
Connected to 10.1.1.33.
Escape character is '^]'.
Password:
```

### 3.2. SSH

SSH es un protocolo que permite al igual que telnet permite la conexión remota a otra máquina, la diferencia es que SSH utiliza un canal seguro, usando un sistema de cifrado para enviar y recibir la información.<sup>[2]</sup>



### 3.2.1. Configuración SSH

Para conectarse por ssh a un host remoto es necesario especificar el usuario seguido de la dirección ip o hostname de la máquina remota.

```
$ ssh nirojas@bioinfo.otalca.cl
nirojas@bioinfo.otalca.cl's password:
```

Para facilitar el ingreso se puede modificar el archivo config ( crear si no existe), para crear un alias a la conexión remota.

```
$ vim .ssh/config

Host bioinfo
user nirojas
hostname bioinfo.otalca.cl
```

Además es posible generar una llave encriptada rsa para acceder al host remoto sin indicar la contraseña cada vez que se intente conectar.

Primero se genera una llave rsa que permitirá identificar a nuestro usuario y máquina local.

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/yanrri/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/yanrri/.ssh/id_rsa.
Your public key has been saved in /home/yanrri/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:OUkuYjpu7B/wQDdGJoWH84L0P+oBUfTHCprmp6UMTlQ yanrri@CEBOYAN
The key's randomart image is:
+---[RSA 2048]-----+
|  o*+                |
|  .+=+o .           |
|  .ooE+. o.         |
|  ..*+o.oo o        |
|  *o.+.. S          |
|+  .* + . .         |
|  .+o++ .           |
|=.Bo..              |
|  B+o.               |
+-----[SHA256]-----+
```

Luego es necesario crear el directorio .ssh/ en el host remoto en caso de que no haya sido creado con anterioridad.

```
$ ssh bioinfo mkdir -p .ssh
```

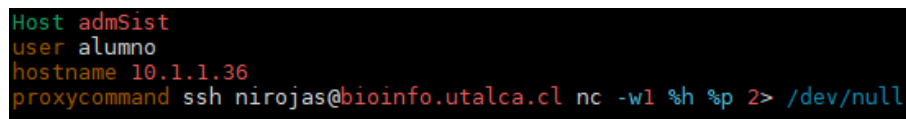
Finalmente se copia la llave generada anteriormente en el archivo que almacena las claves autorizadas para realizar conexión sin password.

```
$ cat ~/.ssh/id_rsa.pub | ssh bioinfo 'cat >> .ssh/authorized_keys'
```

### 3.2.2. Conexión por túnel

En algunos casos para conectarse a un equipo, se requiere un host remoto intermedio que haga un túnel al host remoto final.

El host intermedio se puede agregar a la configuración ssh quedando como la siguiente imagen.



```
Host admSist
  user alumno
  hostname 10.1.1.36
  proxycommand ssh nirojas@bioinfo.utalca.cl nc -w1 %h %p 2> /dev/null
```

Figura 6: Archivo config de ssh para establecer conexión remota mediante un túnel

La imagen muestra en 'proxycommand' el host que actúa como túnel para establecer la conexión con el host remoto final. En este caso se usa el servidor bioinfo.utalca.cl para poder conectarse a la máquina con dirección IP 10.1.1.36 con el usuario alumno.

Finalmente si se desea conectar a través de un túnel sin indicar contraseña por cada conexión, es necesario copiar la clave rsa tanto en el host intermedio como el host final.

### 3.2.3. SSHFS: Montar archivos remotos

Otro aspecto interesante de las conexiones remotas es la posibilidad de acceder a los archivos de un host remoto directamente desde nuestra máquina local, para ello existen varias opciones una de ellas es usar el protocolo seguro de SSH con el software SSHFS que permite montar un directorio remoto como se muestra en el siguiente comando.

```
sshfs alumno@10.1.1.36:RemoteFolder/ ~/Documents/ -o ssh_command='ssh -J nirojas@bioinfo.utalca.cl'
```

En este caso se agregó la opción *ssh\_command* para lograr la conexión mediante un túnel.

## Referencias

- [1] Manual Linux <https://manuallinux.eu/gcc.html>
- [2] SSH [https://es.wikipedia.org/wiki/Secure\\_Shell](https://es.wikipedia.org/wiki/Secure_Shell)