I OBJETIVOS

Implementará un sistema de archivos tipo UNIX empleando las estructuras de datos y diseñando las funciones necesarias para que las aplicaciones sean capaces de leer y escribir archivos en un disco donde inicialmente solo pueden accederse sectores, cilindros y superficies.

II BIBLIOGRAFÍA

- Sistemas Operativos Diseño e Implementación, Andrew Tanenbaum, Prentice Hall
- Sistemas Operativos, Silberschatz & Galvin, Prentice Hall.
- Beginniing Linux Programming, Neil Matthew & Richard Stones, Wrox Press
- UNIX Programación Práctica, Key A. Robbins & Steven Robbins, Prentice Hall

III RECURSOS

- Una estación de trabajo con UNIX.
- Un editor de texto
- El compilador de C de UNIX
- El programa `khexedit` existente en ambientes Linux con KDE instalado, para poder explorar el contenido de los discos virtuales
- El archivo `vdisk.zip` donde se encuentra:
    - El programa ejecutable `createvd`
    - El programa `dumpsec.c`
    - Las funciones contenidas en el archivo `vdisk.o`
    - El ejemplo del Shell

IV DESCRIPCIÓN DEL PROYECTO

1 Creación del disco virtual

Con el programa `createvd` podrá crear los discos virtuales necesarios, cada disco virtual será un archivo que existirá en el directorio donde fue ejecutado el programa `createvd`.

El programa `createvd` recibe un parámetro que es un número entero y este indica el número de disco virtual, si un disco virtual ya existe `createvd` indicará un mensaje de error, de manera que si se desea volver a crear un disco virtual, es necesario borrar el archivo correspondiente.

Los archivos generados por `createvd` son `disco`*`N`*`.vd`, donde *N* es el número de disco que puede ser entre 0 y 3.

El tamaño de cada disco virtual es de 21.09 Mb, consta de 43200 sectores de 512 bytes con una geometría de 8 superficies, 200 cilindros y 27 sectores por pista.

2 Funciones de acceso al disco virtual

La forma de acceder un disco virtual es partir de las llamadas definidas para leer y escribir sectores, esas llamadas son `vdreadsector` y `vdwritesector`, los parámetros que reciben se muestran en la Figura 1.

```
int vdreadsector(int unidad, int superficie, int cilindro, int
sector,int numero_de_sectores, char *buffer)

int vdwritesector(int unidad, int superficie, int cilindro, int
sector, int numero_de_sectores, char *buffer)
```

Figura 1. Definición de las llamadas `vdreadsector` y `vdwritesector`.

- Los parámetros `unidad`, `superficie` y `cilindro` se numeran a partir del elemento 0, el parámetro `sector` se numera a partir del elemento 1.
- El `buffer` contendrá los datos a escribir en el bloque de la imagen en el caso de la llamada `vdwritesector` y ahí se almacenará el bloque leído después de la llamada `vdreadsector`, este buffer deberá ser de 512 bytes multiplicado por el número de sectores que se van a leer o escribir.
- Las funciones regresan –1 si hubo error después de realizar alguna operación de lectura y/o escritura o reciben parámetros incorrectos de acuerdo a la geometría del disco.

Estas funciones se encuentran en el archivo `vdisk.o` y para encadenarlas a un programa este tiene que ser indicado durante la compilación, ejemplo:

```
$ gcc –o miprograma miprograma.c vdisk.o
```

Las funciones del disco virtual simulan retardos de acuerdo al movimiento del disco y de las cabezas, considere que el posicionamiento de la cabeza de un cilindro a otro lleva tiempo (1/100 de seg. de un cilindro a otro), de manera que mientras más alejados estén los cilindros, más tiempo se llevará el posicionamiento de las cabezas.

El Ejemplo 1 es un programa que despliega en pantalla el contenido de un sector físico del disco virtual, este programa hace uso de la función `vdreadsector` que está en el archivo `vdisk.o`. Este programa puede utilizarse para estar revisando el contenido de los sectores durante las pruebas del desarrollo del sistema de archivos.

Para ejecutar dumpsec es necesario indicar en los parámetros: unidad, cilindro, superficie y sector como se muestra en la Figura 2 desplegará los bytes que están en el disco virtual 0, cilindro 2, superficie 1 y sector físico 4.

```
$ ./dumpsec 0 2 1 4
```

Figura 2. Ejecución del programa dumpsec.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "vdisk.h"

#define LINESIZE 16
#define SECSIZE 512

int main(int argc,char *argv[])
{
	int drive;
	int ncyl,nhead,nsec;
	int fd;
	unsigned char buffer[SECSIZE];
	int offset;
	int i,j,r;
	unsigned char c;

	if(argc==5)
	{
		drive=atoi(argv[1]);
		ncyl=atoi(argv[2]);
		nhead=atoi(argv[3]);
		nsec=atoi(argv[4]);
		if(drive<0 || drive> 3 || ncyl>CYLINDERS || nhead > HEADS || nsec
> SECTORS || ncyl<0 || nhead<0 || nsec<1)
		{
			fprintf(stderr,"Posición invalida\n");
			exit(1);
		}
		printf("Desplegando de disco%d.vd Cil=%d, Sup=%d,
Sec=%d\n",drive,ncyl,nhead,nsec);

	}
	else
	{
		fprintf(stderr,"Error en los argumentos\n");
		exit(1);
	}

	if(vdreadsector(drive,nhead,ncyl,nsec,1,buffer)==-1)
	{
		fprintf(stderr,"Error al abrir disco virtual\n");
		exit(1);
	}

	for(i=0;i<SECSIZE/LINESIZE;i++)
	{
		printf("\n %3X -->",i*LINESIZE);
```

```
            for(j=0;j<LINESIZE;j++)
            {
                  c=buffer[i*LINESIZE+j];
                  printf("%2X ",c);
            }
            printf("  |  ");
            for(j=0;j<LINESIZE;j++)
            {
                  c=buffer[i*LINESIZE+j]%256;
                  if(c>0x1F && c<127)
                        printf("%c",c);
                  else
                        printf(".");
            }
      }
      printf("\n");
}
```

Ejemplo 1. Programa `dumpsec.c` que muestra los bytes contenidos en un sector de un disco virtual.

3 Diseño de un sistema de archivos para el disco virtual

Para el manejo de archivos en los discos virtuales deberán tener un sistema de archivos tipo UNIX.

En esta parte es necesario definir:

- Tabla de particiones
- El súper bloque.
- La forma como el sistema operativo sabe cuáles son los bloques libres.
- Los nodos i y tamaño de los nodos i.
- El área del disco reservada para nodos i.
- El tamaño de los bloques de datos y apuntadores.
- El área del disco reservada para bloques de datos y apuntadores.
- El tamaño de los bloques de apuntadores.
- El tamaño de los apuntadores de manera que pueda aprovecharse al máximo la capacidad del disco.

El sistema de archivos puede utilizar uno o más discos virtuales (ver punto extra).

3.1 Crear particiones en el disco virtual

Desarrolle un programa que cree las estructuras de datos y tablas necesarias en el primer sector físico del disco, ejemplo:

```
$ ./vdfdisk 0
```

La ejecución del programa anterior generará las estructuras de datos tablas necesarias en el primer sector del disco virtual 0. Para el caso de esta práctica solo se generará una partición la cual abarcará todos los sectores restantes de la unidad.

Una vez hecha la partición en el disco virtual esta podrá formatearse de acuerdo a nuestro sistema de archivos.

3.2 Formateo de la partición y conversión de sectores lógicos a sectores físicos, cilindros y superficies.

3.2.1 Formateo de la partición

Basado en el diseño de su sistema de archivos, desarrolle un programa que cree las estructuras de datos y tablas necesarias en el la partición creada previamente del virtual señalado, ejemplo:

```
$ ./vdformat 0
```

La ejecución del programa anterior generará las estructuras de datos tablas necesarias en la partición 0 del disco virtual 0.

Una vez formateada la partición del disco virtual podrán almacenarse archivos en él, para esto será necesario crear las funciones que se describen en las partes 3.3 y 3.4 para acceder el sistema de archivos.

3.2.2 conversión de sectores lógicos a sectores físicos, cilindros y superficies.

En esta parte también deberás modificar el programa ejemplo `dumpsec.c` y nómbralo `dumpseclog.c` para que ahora muestre un sector lógico de la partición del disco virtual (por default será siempre la partición que creamos en `disco0.vd`). De manera que ahora podamos examinar el contenido de los sectores como se muestra en el ejemplo

```
$ ./dumpseclog n
```

Donde n es el número de sector lógico a examinar.

3.3 Funciones de bajo nivel para acceder las estructuras

Una vez que la partición del disco virtual ha sido formateada vamos a necesitar crear funciones que nos faciliten el diseño de las funciones del sistema de archivos. Ejemplo de algunas de estas funciones serán para:

- Leer uno o varios sectores en función al número de sector lógico indicado como parámetro, esta función convertirá el sector lógico a sector físico, cilindro y superficie. Ejemplo: `vdreadsl(int sec_loc, char *buffer);`.
- Escribir uno o varios sectores en función al número de sector lógico indicado como parámetro, esta función convertirá el sector lógico a sector físico, cilindro y superficie. Ejemplo: `vdwritesl(int sec_loc, char *buffer);`.
- Determinar donde inicia el mapa de bits de la zona en función a la información de las tablas creadas en la parte 3.1.
- Determinar donde inicia el mapa de bits del área de datos en función a la información de las tablas creadas en la parte 3.1.
- Determinar donde el área de nodos i en función a la información de las tablas creadas en la parte 3.1.
- Determinar donde el área de los archivos en función a la información de las tablas creadas en la parte 3.1.
- Determinar donde el área de nodos i en función a la información de las tablas creadas en la parte 3.1.
- Localizar un nodo i disponible en la tabla de nodos i en función al mapa de bits de nodos i.
- Asignar un nodo i disponible para un archivo nuevo, esto implica marcarlo como no disponible en el mapa de nodos i de la zona y crear el nodo i en la tabla de nodos i escribiendo la información del archivo.
- Convertir un bloque de datos a sector o grupo de sectores lógicos.
- Localizar bloques de datos disponibles en función al mapa de bits.
- Asignar bloques de datos para un archivo, esto implica marcarlos como no disponibles en el mapa de bits y asignarlos en el nodo i del archivo.
- Leer y escribir en bloques de datos en función al número de bloque, estas funciones podrán apoyarse de funciones desarrolladas en esta parte para hacer las conversiones necesarias a sectores lógicos y realizar las operaciones de lectura o escritura.

3.4 Acceso al sistema de archivos en la partición disco virtual desde las aplicaciones

Para poder usar archivos en el sistema de archivos que tenemos en la partición del disco virtual es necesario implementar las funciones `vdopen`, `vdread`, `vdwrite`, `vdseek`, `vdclose` y `vdunlink` para los archivos que están en el directorio raíz, y las funciones `vdopendir`, `vdreaddir`, `vdclosedir` para manejo de directorios.

Para implementar las funciones para manejo de archivos y directorios es necesario acceder las áreas críticas del disco que son el sector de arranque donde está la información sobre el formato

del disco, directorio raíz y las tablas donde está la información sobre la asignación de los sectores de la partición disco virtual a los archivos.

Detalles sobre las funciones

- **`vdcreat`**: crea un archivo nuevo en la partición del disco virtual, el archivo queda abierto.
- **`vdopen`**: abrir un archivo de la partición del disco virtual, buscar en el directorio si el archivo existe, si es así ponerlo en una **tabla de archivos abiertos que será necesario definir en esta parte** y regresar el descriptor del archivo. A esta función se le pasará como parámetro el nombre del archivo y el modo en el que lo vamos a abrir (read only, read/write)
- **`vdread`**: leer datos a partir de donde está el apuntador al archivo previamente abierto con la función `vdopen`. A esta función se le debe de indicar como parámetros, el descriptor del archivo abierto, el buffer donde va a guardar lo que lee del archivo y la cantidad de bytes a leer.
- **`vdwrite`**: escribir los datos al archivo que previamente fue abierto con la función open. A está función se le debe de indicar como parámetros, el descriptor del archivo abierto, el buffer donde está lo que va a escribir al archivo y la cantidad de bytes a escribir.
- **`vdseek`**: posicionarse en una parte específica del archivo. A esta función se le debe de indicar como parámetros, el descriptor del archivo abierto, la cantidad de bytes a desplazarse y el desplazamiento a partir de donde va a ser, ya sea desde el inicio, el lugar donde se encuentra el apuntador del archivo o a partir del final.
- **`vdclose`**: cerrar un archivo, eliminarlo de la tabla de archivos abiertos. Solo va a recibir como parámetro el descriptor del archivo a cerrar.
- **`vdunlink`**: borra un archivo del directorio raíz del disco virtual. Solo va a recibir como parámetro el nombre del archivo a eliminar.

Los parámetros y valores que regresarán las funciones serán similares a los de las funciones `open`, `read`, `write`, `seek`, `close`, `opendir`, `readdir`, `closedir` y `unlink` que implanta por default UNIX, se recomienda que consulte el manual de estas funciones ya que se asumirá que los parámetros que reciben y los valores devueltos serán iguales en las llamadas de nuestro sistema de archivos.

3.5 Interfaz con el usuario para el acceso al sistema de archivos virtual

Deberá desarrollarse un Shell que permitirá que el usuario introduzca comandos para poder transferir archivos entre el sistema de archivos de UNIX y la unidad virtual. El Shell hará uso de las funciones desarrolladas en la parte 3.4 para acceder las unidades virtuales y de las funciones que implanta UNIX por default para interactuar con los archivos en el sistema de archivos de UNIX.

En el Ejemplo 2 se muestra parte del código del Shell con ejemplos de cómo pueden implementarse los comandos interactuando entre el sistema de archivos de UNIX y el sistema de archivos implementado para la partición de la unidad virtual.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>

#define MAXLEN 80
#define BUFFERSIZE 512

void locateend(char *cmd);
int executecmd(char *cmd);

int main()
{
	char linea[MAXLEN];
	int result=1;
	while(result)
	{
		printf("vshell > ");
		fflush(stdout);
		read(0,linea,80);
		locateend(linea);
		result=executecmd(linea);
	}
}

void locateend(char *linea)
{
	// Localiza el fin de la cadena para poner el fin
	int i=0;
	while(i<MAXLEN && linea[i]!='\n')
		i++;
	linea[i]='\0';
}

int executecmd(char *linea)
{
	char *cmd;
	char *arg1;
	char *arg2;
	char *search=" ";

	// Separa el comando y los dos posibles argumentos
	cmd=strtok(linea," ");
	arg1=strtok(NULL," ");
	arg2=strtok(NULL," ");

	// comando "exit"
```

```
	if(strcmp(cmd,"exit")==0)
		return(0);

	// comando "copy"
	if(strcmp(cmd,"copy")==0)
	{
		if(arg1==NULL && arg2==NULL)
		{
			fprintf(stderr,"Error en los argumentos\n");
			return(1);
		}
		if(!isinvd(arg1) && !isinvd(arg2))
			copyuu(&arg1[2],&arg2[2]);

		else if(!isinvd(arg1) && isinvd(arg2))
			copyuv(&arg1[2],arg2);

		else if(isinvd(arg1) && !isinvd(arg2))
			copyvu(arg1,&arg2[2]);

		else if(isinvd(arg1) && isinvd(arg2))
			copyvv(arg1,arg2);


	}

	// comando "cat"
	if(strcmp(cmd,"cat")==0)
	{
		if(isinvd(arg1))
			catv(arg1);
		else
			catu(&arg1[2]);
	}

	// comando dir
	if(strcmp(cmd,"dir")==0)
	{
		if(arg1==NULL)
			dirv();
		else if(!isinvd(arg1))
			diru(&arg1[2]);
	}
}

/* Regresa verdadero si el nombre del archivo no comienza con // y por lo
   tanto es un archivo que está en el nuestro sistema de archivos */

int isinvd(char *arg)
{
	if(strncmp(arg,"//",2)!=0)
		return(1);
	else
		return(0);
}
```

```
/* Copia un archivo del sistema de archivos de UNIX a un archivo destino
   en el mismo sistema de archivos de UNIX */

int copyuu(char *arg1,char *arg2)
{
      int sfile,dfile;
      char buffer[BUFFERSIZE];
      int ncars;

      sfile=open(arg1,0);
      dfile=creat(arg2,0640);
      do {
            ncars=read(sfile,buffer,BUFFERSIZE);
            write(dfile,buffer,ncars);
      } while(ncars==BUFFERSIZE);
      close(sfile);
      close(dfile);
      return(1);
}



/* Copia un archivo del sistema de archivos de UNIX a un archivo destino
   en nuestro sistema de archivos */

int copyuv(char *arg1,char *arg2)
{
      int sfile,dfile;
      char buffer[BUFFERSIZE];
      int ncars;

      sfile=open(arg1,0);
      dfile=vdcreat(arg2,0640);
      do {
            ncars=read(sfile,buffer,BUFFERSIZE);
            vdwrite(dfile,buffer,ncars);
      } while(ncars==BUFFERSIZE);
      close(sfile);
      vdclose(dfile);
      return(1);
}


/* Copia un archivo del disco virtual a un archivo destino
   en el sistema de archivos de UNIX */

int copyvu(char *arg1,char *arg2)
{
      int sfile,dfile;
      char buffer[BUFFERSIZE];
      int ncars;

      sfile=vdopen(arg1,0);
      dfile=creat(arg2,0640);
      do {
```

```
            ncars=vdread(sfile,buffer,BUFFERSIZE);
            write(dfile,buffer,ncars);
      } while(ncars==BUFFERSIZE);
      vdclose(sfile);
      close(dfile);
      return(1);
}



/* Copia un archivo de nuestro sistema de archivos a un archivo destino
   en nuestro mismo sistema de archivos */

int copyvv(char *arg1,char *arg2)
{
      int sfile,dfile;
      char buffer[BUFFERSIZE];
      int ncars;

      sfile=vdopen(arg1,0);
      dfile=vdcreat(arg2,0640);
      do {
            ncars=vdread(sfile,buffer,BUFFERSIZE);
            vdwrite(dfile,buffer,ncars);
      } while(ncars==BUFFERSIZE);
      vdclose(sfile);
      vdclose(dfile);
      return(1);
}


/* Despliega un archivo de nuestro sistema de archivos a pantalla */

int catv(char *arg1)
{
      int sfile,dfile;
      char buffer[BUFFERSIZE];
      int ncars;

      sfile=vdopen(arg1,0);
      do {
            ncars=vdread(sfile,buffer,BUFFERSIZE);
            write(1,buffer,ncars);  // Escribe en el archivo de salida
estandard
      } while(ncars==BUFFERSIZE);
      vdclose(sfile);
      return(1);
}


/* Despliega un archivo del sistema de archivos de UNIX a pantalla */

int catu(char *arg1)
{
      int sfile,dfile;
      char buffer[BUFFERSIZE];
```

```
	int ncars;

	sfile=open(arg1,0);
	do {
		ncars=read(sfile,buffer,BUFFERSIZE);
		write(1,buffer,ncars);  // Escribe en el archivo de salida
estandard
	} while(ncars==BUFFERSIZE);
	close(sfile);
	return(1);
}


/* Muestra el directorio en el sistema de archivosd de UNIX */

int diru(char *arg1)
{
	DIR *dd;
	struct dirent *entry;

	if(arg1[0]=='\0')
		strcpy(arg1,".");

	printf("Directorio %s\n",arg1);

	dd=opendir(arg1);
	if(dd==NULL)
	{
		fprintf(stderr,"Error al abrir directorio\n");
		return(-1);
	}

	while((entry=readdir(dd))!=NULL)
		printf("%s\n",entry->d_name);

	closedir(dd);
}

/* Muestra el directorio en el nuestro sistema de archivos */

int dirv()
{
	VDDIR *dd;
	struct vddirent *entry;

	printf("Directorio el sistema de archivos\n");

	dd=vdopendir(".");
	if(dd==NULL)
	{
		fprintf(stderr,"Error al abrir directorio\n");
		return(-1);
	}

	while((entry=vdreaddir(dd))!=NULL)
		printf("%s\n",entry->d_name);
```

```
        vdclosedir(dd);
}
```

Ejemplo 2. Parte del código del shell a implementar.

3.6 Identificación de nuestro sistema de archivos y del sistema de archivos de UNIX

Cuando el nombre del archivo está precedido de `//` este es un archivo que se encuentra en el sistema de archivos de UNIX y su nombre comienza a partir del tercer carácter. La función `isinvd()` que está en regresa verdadero si el nombre del archivo no inicia con dos diagonales y por lo tanto debe ser accedido del nuestro sistema de archivos con las funciones implementadas en la parte 3.4.

Si el nombre del archivo inicia con `//` entonces está en el sistema de archivos de UNIX y debe ser referido a partir del tercer carácter como se muestra en la Figura 3.

```
if(!isinvd(archivo))
     nombre_real=&archivo[2];
```

Figura 3. Ejemplo de cómo identificar un archivo en el sistema de archivos de UNIX.

3.7 Directorios

El comando `dir` mostrará la lista de archivos del directorio raíz del nuestro sistema de archivos o del directorio indicado en el sistema de archivos de UNIX, por ejemplo:

`Shell > **dir**`

Muestra los archivos en el directorio raíz de nuestro sistema de archivos

`Shell > **dir //**`

Muestra los archivos del directorio actual en el sistema de archivos de UNIX

`Shell > **dir ///**`

Muestra los archivos del directorio raíz en el sistema de archivos de UNIX

```
Shell > dir ///etc
```

Muestra los archivos del directorio `/etc` en el sistema de archivos de UNIX

3.8 Transferencia de archivos

El comando `copy` del Shell permitirá transferir entre los mismos sistemas de archivos, por ejemplo:

```
Shell > copy fuente destino
```

Duplica un archivo dentro de nuestro sistema de archivos a otro con diferente nombre.

```
Shell > copy //fuente destino
```

Copia un archivo del sistema de archivos de UNIX a un archivo en nuestro sistema de archivos.

```
Shell > copy fuente //destino
```

Copia un archivo de nuestro sistema de archivos al sistema de archivos de UNIX.

```
Shell > copy //fuente //destino
```

Duplica un archivo dentro del sistema de archivos de UNIX a otro con diferente nombre

3.9 Desplegar archivos

El comando `type` del Shell muestra en pantalla el contenido de un archivo de texto, por ejemplo:

```
Shell > type archivo
```

Despliega un archivo del sistema de archivos en nuestro sistema de archivos.

```
Shell > type //archivo
```

Despliega un archivo del sistema de archivos de UNIX.

3.10 Borrar archivos

El comando `delete` (no está en el Shell ejemplo por lo que **hay que implementarlo**) borrará un archivo del sistema de archivos de UNIX o de nuestro sistema de archivos, por ejemplo:

```
Shell > delete //archivo
```

Borra un archivo del sistema de archivos de UNIX.

```
Shell > delete archivo
```

Borra un archivo de nuestro sistema de archivos.

4 Puntos extras

El sistema de archivos puede abarcar dos o más discos virtuales superando de esta forma la barrera de los 21.09 Mb. de un disco y mejorando el rendimiento minimizando los movimientos de las cabezas.

V ENTREGA Y EVALUACIÓN

1 Entrega y Revisión

Domingo 3 de Diciembre a las 23:55 en Moodle.
La revisión la realizaremos el Lunes 4 y Martes 5 de Diciembre.

2 Equipos

Esta práctica se hará en equipos (máximo 2 integrantes), es necesario que en la revisión esté el equipo completo.

3 Evaluación

Puntualidad en las revisiones	El equipo estuvo completo y puntual en todas las sesiones de revisión.	Si hubo dos o más sesiones con el equipo, el equipo estuvo completo y puntual en casi todas las sesiones de revisión	Si solo hubo una sesión de revisión, el equipo no estuvo completo o no fue puntual. Si fueron dos o más sesiones de revisión, en más de una sesión el equipo no estuvo completo o fue puntual
	+7.5	+3.5	0
Especificaciones de entrega	La entrega del producto cumple con todas las especificaciones indicadas en el documento de la práctica, por ejemplo, los archivos se entregan de acuerdo a las formas indicadas en el documento de la práctica.		La entrega del producto no cumple con al menos una de las especificaciones indicadas en el documento de la práctica
	+7.5		0
Funcionamiento	El producto cumple con todas las especificaciones indicadas en el documento y no tiene fallas	El producto muestra una falla no esperada o el producto está casi completo, puede funcionar excepto la parte no completada	El producto muestra más de una falla inesperada o no funciona, esto puede ser debido a que no esté completo.
	+70	+35	0
Interfaz con el usuario	El producto funciona y pudo ser utilizado sin necesidad de recibir indicaciones por el desarrollador, tiene instrucciones claras para ser utilizado.	El producto funciona, pero hubo necesidad de recibir alguna indicación para su uso por parte del desarrollador del producto	El producto carece de instrucciones claras para ser utilizado y requiere que alguno de los desarrolladores esté presente para su utilización o no puede utilizarse debido a que no está completo
	+7.5	+3.5	0
Claridad en el código	El código es claro, usa nombres de variables adecuadas, está debidamente comentado e indentado. Puede ser entendido por cualquier otra persona que no intervino en su desarrollo.	El código carece de claridad, puede ser entendido por cualquier persona ajena a su desarrollo pero con cierta dificultad.	El código carece de comentarios, está mal indentado, usa nombres de variables no adecuadas.
	+7.5	+3.5	0
Defensa del producto	Todos los que presentan la práctica son capaces de explicar cualquier parte del producto presentado	Uno de los que presenta la práctica muestra dudas sobre alguna parte del desarrollo del producto presentado	Más de un integrante, o si el trabajo fue individual , el desarrollador duda sobre cómo está desarrollado producto.
	x 1(puntos se multiplican por 1)	x 0.5 (puntos se multiplican por 0.5)	x 0 (puntos se multiplican por 0)
Sobresaliente 20 %	Tiene 1 en todos los puntos anteriores. El producto entregado es sobresaliente, muestra tener la calidad para ser expuesto como un producto representativo de la carrera Hay evidencia de que los desarrolladores se documentaron y muestran aprendizajes más allá de lo esperado		No tiene 1 en todos los puntos anteriores, o el producto entregado no es sobresaliente y no muestra tener la calidad para ser expuesto como un producto representativo de la carrera o no hay evidencia de que los desarrolladores se documentaron y muestran aprendizajes más allá de lo esperado
	+20		0