

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE CIENCIAS
REDES DE COMPUTADORAS



Práctica 4
**Implementación de algoritmos de detección
de errores de capa de enlace**

Equipo 4

Integrantes :

Adrián Aguilera Moreno Num.C:421005200

Francisco Contreras Ibarra Num.C: 316083786

Aldo Daniel Licona Gómez Num.C: 316263863

1. Cálculos.

Paridad Par de 1010111

Bit de Paridad	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
1	1	0	1	0	1	1	1

Paridad Par de 1000010

Bit de Paridad	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
0	1	0	0	0	0	1	0

Paridad Par de 1001101 1111000

Bit de Paridad	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
0	1	0	0	1	1	0	1	1	1	1	1	0	0	0

Paridad Par de 1111111 0000001

Bit de Paridad	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
0	1	1	1	1	1	1	1	0	0	0	0	0	0	1

Paridad Impar 0101011

Bit de Paridad	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
1	0	1	0	1	0	1	1

Paridad Impar 0000011

Bit de Paridad	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
1	0	0	0	0	0	1	1

Paridad Impar 0010111 0011111

Bit de Paridad	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
0	0	0	1	0	1	1	1	0	0	1	1	1	1	1

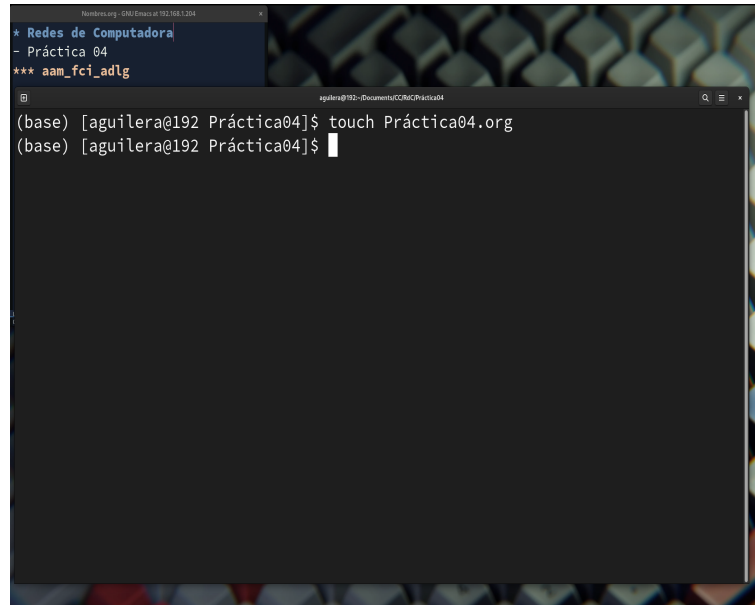
Paridad Impar 1111110 0111111

Bit de Paridad	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
1	1	1	1	1	1	1	0	0	1	1	1	1	1	1

2. Checksum.

Cada equipo tendrá que realizar las siguientes actividades explicando y con capturas de pantalla de lo siguiente:

1. Crear un archivo (del tipo, tamaño y extensión que prefieran).



```
Remnux.org - GNU Emacs at 192.168.1.204
* Redes de Computadora
- Práctica 04
*** aam_fci_adlg

aguilera@192-~(Documents/CC/RdC/Práctica04)
(base) [aguilera@192 Práctica04]$ touch Práctica04.org
(base) [aguilera@192 Práctica04]$
```

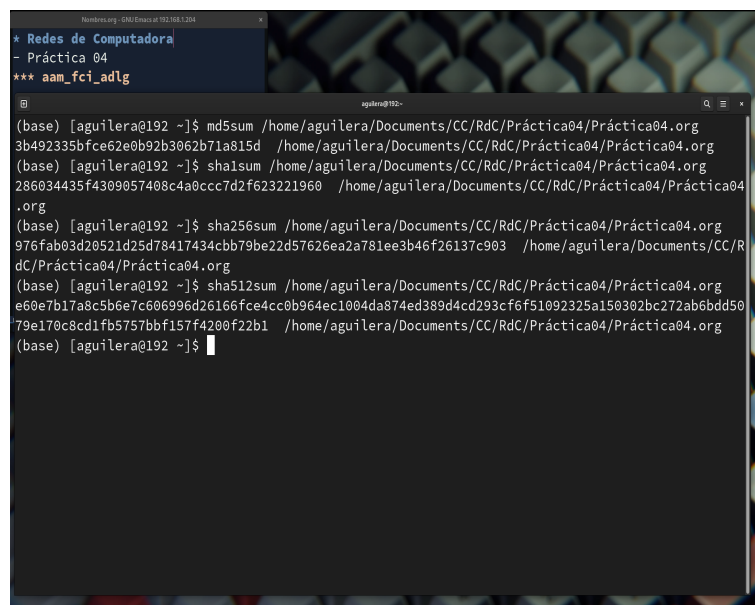
- Investigar cuál es el procedimiento (comando, aplicación, etc..) a realizar para obtener el checksum de su archivo mediante consola/terminal.

En el caso de Fedora y sistemas linux, basta con invocar alguno de los siguientes algoritmos:

- md5sum fichero.
- sha1sum fichero.
- sha256sum fichero.
- sha512sum fichero.

Cómo podemos observar, cada función hash va seguido del fichero del cuál queremos obtener su checksum.

- Obtener el checksum de su archivo con al menos 2 algoritmos diferentes.

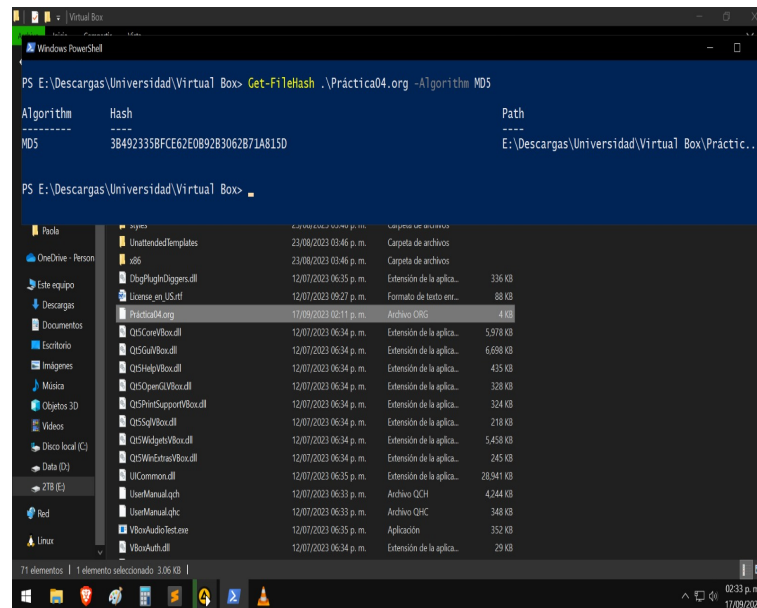


```
Remnux.org - GNU Emacs at 192.168.1.204
* Redes de Computadora
- Práctica 04
*** aam_fci_adlg

aguilera@192-~(Documents/CC/RdC/Práctica04)
(base) [aguilera@192 ~]$ md5sum /home/aguilera/Documents/CC/RdC/Práctica04/Práctica04.org
3b492335bfce62e0b92b3062b71a815d /home/aguilera/Documents/CC/RdC/Práctica04/Práctica04.org
(base) [aguilera@192 ~]$ sha1sum /home/aguilera/Documents/CC/RdC/Práctica04/Práctica04.org
286034435f4309057408c4a0ccc7d2f623221960 /home/aguilera/Documents/CC/RdC/Práctica04/Práctica04.org
(base) [aguilera@192 ~]$ sha256sum /home/aguilera/Documents/CC/RdC/Práctica04/Práctica04.org
976fab03d20521d25d78417434cbb79be22d57626ea2a781ee3b46f26137c903 /home/aguilera/Documents/CC/RdC/Práctica04/Práctica04.org
(base) [aguilera@192 ~]$ sha512sum /home/aguilera/Documents/CC/RdC/Práctica04/Práctica04.org
e60e7b17a8c5b6e7c606996d26166fce4cc0b964ec1004da874ed389d4cd293cf6f51092325a150302bc272ab6bdd5079e170c8cd1fb5757bbf157f4200f22b1 /home/aguilera/Documents/CC/RdC/Práctica04/Práctica04.org
(base) [aguilera@192 ~]$
```

Cómo podemos notar, se obtuvieron los hash del checksum con 4 algoritmos diferentes (funciones hash distintas).

- Enviar el archivo a través de internet a otro integrante del equipo, obtener el checksum del archivo recibido y compararlo con el original.



El archivo Práctica04.org fue enviado por whatsapp desde un dispositivo con Fedora como sistema operativo y fue recibido por uno con windows como sistema operativo.

5. Investigue y explique al menos 3 ejemplos de la implementación de esta comprobación.

Cómo mencionamos en el punto 2 de esta sección (checksum) los sistemas linux tienen implementados 4 funciones hash que realizan este procedimiento (algoritmos), los cuáles detallaremos:

- **md5sum** (Message Digest Algorithm 5). El valor de resumen MD5 es una cadena alfanumérica de 32 caracteres que representa de manera única el contenido de un archivo. Esto puede ser útil para verificar la integridad de un archivo o comparar dos archivos para ver si son idénticos. Cuando se ejecuta el comando, lee el contenido del archivo especificado y luego aplica el algoritmo MD5 para calcular el valor de resumen MD5. Este valor se muestra en la salida estándar (generalmente en la terminal) como una cadena de 32 caracteres hexadecimales.
- **sha1sum** (Secure Hash Algorithm 1). Genera un valor hash que representa de manera única el contenido de un archivo. Sin embargo, el algoritmo SHA-1 es más seguro que MD5, aunque también se considera obsoleto para aplicaciones críticas de seguridad debido a vulnerabilidades conocidas.
- **sha256sum** (Secure Hash Algorithm 256-bit). Genera un valor hash que representa de manera única el contenido de un archivo. El algoritmo SHA-256 es considerablemente más seguro que SHA-1 y MD5 y se utiliza comúnmente en aplicaciones de seguridad y verificación de integridad.
- **sha512sum** (Secure Hash Algorithm 512-bit). Genera un valor hash que representa de manera única el contenido de un archivo. El algoritmo SHA-512 es una versión más segura y robusta que SHA-256, ya que utiliza un hash de 512 bits en lugar de 256 bits, lo que lo hace aún más resistente a ataques criptográficos.

6. Realice un programa en Python donde se ejemplifique la implementación del checksum. El programa debe de ser capaz de simular una comunicación Emisor-Receptor, es decir, de calcular el checksum de una cadena de datos y luego verificar si se ha producido un error durante la transmisión calculando nuevamente el checksum y comparándolo con el valor recibido.

```

1 def checksum(bits):
2     # Verificamos que la cadena tenga 16 bits
3     if len(bits) != 32: # Modificar en caso de querer pasar 64 bits.
4         raise ValueError("La cadena debe contener exactamente 32 bits")
5
6     # Convertimos la cadena de bits en una lista de enteros
7     cadena = list(map(int, bits))
8
9     # Separamos dos segmentos de bits (nuestras "palabras"):
10    P_1 = cadena[:len(cadena) // 2]
11    P_2 = cadena[len(cadena) // 2:]

```

```

12     complemento = [(1 - bit) for bit in sumar2(P_1, P_2)]
13
14     checksum = cadena + complemento
15
16     nueva_cadena = ''.join(map(str, checksum))
17     return nueva_cadena
18
19
20 def sumar2(P_1, P_2):
21     # Verificamos que ambas cadenas sean del mismo tamaño.
22     if len(P_1) != len(P_2):
23         raise ValueError("Error, las cadenas tienen tamaño distinto.")
24
25     lista = []
26     valor_extra = 0
27     P_3 = reversed(P_1)
28     P_4 = reversed(P_2)
29     for elem1, elem2 in zip(P_3, P_4):
30         if ((elem1 == 0) and (elem2 == 0)):
31             if (valor_extra == 1):
32                 lista.insert(0, 1)
33                 valor_extra = 0
34             else:
35                 lista.insert(0, 0)
36         elif ((elem1 == 1) and (elem2 == 1)):
37             if (valor_extra == 1):
38                 lista.insert(0, 1)
39             else:
40                 lista.insert(0, 0)
41                 valor_extra = 1
42         elif ((elem1 == 1) and (elem2 == 0)):
43             if (valor_extra == 1):
44                 lista.insert(0, 0)
45                 valor_extra = 1
46             else:
47                 lista.insert(0, 1)
48         elif ((elem1 == 0) and (elem2 == 1)):
49             if (valor_extra == 1):
50                 lista.insert(0, 0)
51                 valor_extra = 1
52             else:
53                 lista.insert(0, 1)
54
55     if (valor_extra == 1):
56         lista.insert(0, 1)
57
58     return lista
59
60
61 def sumar3(bits):
62     cadena = list(map(int, bits))
63     tercios = (len(cadena) // 3)
64     P_1 = cadena[:tercios]
65     P_2 = cadena[tercios:(2*tercios)]
66     P_3 = cadena[(2*tercios):len(bits)]
67
68     R = sumar2(P_1, P_2)
69     RP = sumar2(R, P_3)
70     RT = ''.join(map(str, RP))
71     return RT

```

Un ejemplo de ejecución sería

```

Nombres.org - GNU Emacs at 192.168.1.204
* Redes de Computadora
- Práctica 04
*** aam_fci_adlg

(base) [aguilera@192 Práctica04]$ emacs -nw
(base) [aguilera@192 Práctica04]$ python3 Checksum.py
Orden:      Cadena original      :      Checksum añadido      :      Comprobación.
Checksum para 011001100110011001010101010101010100010001000100 : 1111111111111111
(base) [aguilera@192 Práctica04]$

```

3. CRC.

3.1. Calcula el CRC con los siguientes datos y realiza la comprobación

3.1.1. Ejercicio a : Trama: $x^8 + x^7 + x^6 + x^5 + x^2 + 1$, $G(x) = x^4 + x^2 + 1$

a)

$$M = x^8 + x^7 + x^6 + x^5 + x^2 + 1 = 111100101$$

$$G(x) = x^4 + x^2 + 1$$

1) Observar el grado del polinomio generador y asignarlo a r
 $r = 4$

2) Añadir el número de 0 a la trama de acuerdo con el valor de r
 $x^r M(x) = 1111001010000$

3) Convertir el polinomio generador a binario

x^4	x^3	x^2	x^1	x^0
1	0	1	0	1

$$= 10101$$

4) Para obtener CRC se debe dividir $\frac{x^r M(x)}{G(x)}$

$$CRC = \frac{1111001010000}{10101}$$

$$\begin{array}{r}
 101011111001010000 \\
 \underline{1010110} \\
 0101010 \\
 \underline{10101} \\
 00011101 \\
 \underline{10101} \\
 010000 \\
 \underline{10101} \\
 0010100 \\
 \underline{10101} \\
 000010
 \end{array}$$

0010 CRC de 4 bits

5) El mensaje se envia al receptor T se obtiene de unir la trama con el residuo de la division (CRC)
 $T = 111100101010$

6) Para comprobar que el mensaje no tuvo errores se realiza la division de T y el residuo obtenido deber ser 0

$$\begin{array}{r}
 101011111001010010 \\
 \underline{1010110} \\
 0101010 \\
 \underline{10101} \\
 00011101 \\
 \underline{10101} \\
 010000 \\
 \underline{10101} \\
 0010101 \\
 \underline{10101} \\
 000000
 \end{array}$$

al mensaje no tiene errores

3.1.2. Ejercicio b : Trama: 110101111, $G(x) = x^4 + x + 1$

b)

$M = 110101111$
 $G(x) = x^4 + x + 1$

1) Observar el grado del polinomio generador y asignarlo a r

$r = 4$

2) Añadir el número de 0 a la trama de acuerdo con el valor de r

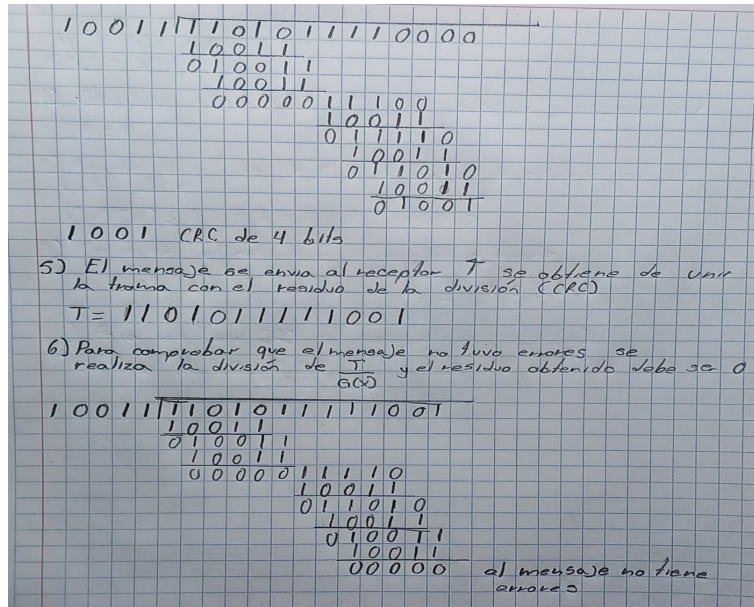
$x^r M(x) = 1101011110000$

3) Convertir el polinomio generador a binario

x^4	x^3	x^2	x^1	x^0	
1	0	0	1	1	$= 10011$

4) Para obtener CRC se debe dividir $x^r M(x)$ por $G(x)$

$$\text{CRC} = \frac{1101011110000}{10011}$$



3.2. Ejercicios resueltos en python

3.2.1. Ejercicio resuelto en el pdf

```

> pwsh -> PracRed 2ms
> python crc.py
Ingresa la trama que se desea enviar en binario : 111101
Ingresa el polinomio generador (ingresa el polinomio de las 2 siguientes posibles maneras y
sin espacios: x4+x2+x1+1 o x^4+x^2+x^1+1): x^3+x^2+1
_ _ _ _ _ Resultados _ _ _ _ _
M : 111101
G(x) : x^3+x^2+1
r : 3
x^rM(x) : 111101000
Polinomio generado en binario : 1101
resuido de la division : 0011
CRC : 011
Trama unido con CRC : 111101011
Comprobación : 0000
_ _ _ _ _

```


3.2.2. Ejercicio a

```
> pwsh -> PracRed 2ms
>> python crc.py
Ingresa la trama que se desea enviar en binario : 111100101
Ingresa el polinomio generador (ingresa el polinomio de las 2 siguientes posibles maneras y
sin espacios: x4+x2+x1+1 o x^4+x^2+x^1+1): x^4+x^2+1
-_-_-_-_- Resultados -_-_-_-_-
M : 111100101
G(x) : x^4+x^2+1
r : 4
x^rM(x) : 1111001010000
Polinomio generado en binario : 10101
resido de la division : 000010
CRC : 0010
Trama unido con CRC : 1111001010010
Comprobación : 000000
-_-_-_-_-
```

3.2.3. Ejercicio b

```
> pwsh -> PracRed 2ms
>> python crc.py
Ingresa la trama que se desea enviar en binario : 1101011111
Ingresa el polinomio generador (ingresa el polinomio de las 2 siguientes posibles maneras y
sin espacios: x4+x2+x1+1 o x^4+x^2+x^1+1): x^4+x^1+1
-_-_-_-_- Resultados -_-_-_-_-
M : 1101011111
G(x) : x^4+x^1+1
r : 4
x^rM(x) : 11010111110000
Polinomio generado en binario : 10011
resido de la division : 000010
CRC : 0010
Trama unido con CRC : 11010111110010
Comprobación : 000000
-_-_-_-_-
```

3.3. Capturas de código CRC

```
1 def polinomioMas(gxC): "polinomio": Unknown word.
2     gxSinX = gxC.replace('x','')
3     gxSinElevado = gxSinX.replace('^','') "Elevado": Unknown word.
4     return gxSinElevado "Elevado": Unknown word.
5
6 def polinomioGrado(gxNM): "polinomio": Unknown word.
7     return gxNM[0:(gxNM.find('+'))]
8
9 def xOr(gxD,mD):
10     if gxD == mD :
11         return "0"
12     else :
13         return "1"
14
15 def addCero(r,m):
16     for i in range(r):
17         m = m+'0'
18     return m
19
20 def polinomioLista(gxNM): "polinomio": Unknown word.
21     cantidadNumeros = gxNM.count('+') "cantidad": Unknown word.
22     lista = [] "lista": Unknown word.
23     for i in range(cantidadNumeros): "cantidad": Unknown word.
24         lista.append(gxNM[0:(gxNM.find('+'))]) "lista": Unknown word.
25         gxNM = gxNM[gxNM.find('+')+1:len(gxNM)]
26     lista.append(gxNM) "lista": Unknown word.
27     return lista "lista": Unknown word.
28
29 def polinimioGenerado (lista): "polinimio": Unknown word.
30     n = int(lista[0])
31     u = lista.pop()
32     b = ""
33     for j in range(n):
34         if(esElemento(str(j+1), lista)): "Elemento": Unknown word.
35             b = '1' + b
36         else:
37             b = '0' + b
38     b = b + u
39     return b
40
41 def esElemento(s,lista): "Elemento": Unknown word.
42     e = False
43     for i in range(len(lista)):
44         if(s == lista[i]):
45             e = True
46     return e
47
```

```

48 def division(gx,m):
49     gxT = len(gx)
50     mm = m[0:gxT]
51     mC = m[gxT:len(m)]
52     rf = miniXOr(gx,mm,gxT)
53     f = 0
54     c = 0
55     r = ""
56     while c == 0 :
57         r = rf
58         rf = quitaCeros(rf) "quita": Unknown word.
59         mm = rf + mC[0:falta(gxT,rf)] "falta": Unknown word.
60         if(len(gx)>len(mm)):
61             if (f == 1):
62                 r = r + mC
63                 break
64         mC = mC[falta(gxT,rf):len(mC)] "falta": Unknown word.
65         rf = miniXOr(gx,mm,gxT)
66         if(len(mC) == 1):
67             f = f + 1
68     return r
69
70
71 def miniXOr(gx,mm,gxT):
72     r = ""
73     for i in range(gxT):
74         r = r + xOr(gx[i],mm[i])
75     return r
76

```

```

77 def quitaCeros(r): "quita": Unknown word.
78     return r[r.find('1'):len(r)]
79
80 def falta(gxT,mm): "falta": Unknown word.
81     return gxT - len(mm)
82
83 def crc(r,div):
84     return div[len(div)-r:len(div)]
85
86 def agregaCRC(m,crc): "agrega": Unknown word.
87     return m + crc
88
89 m = input ("Ingresa la trama que se desea enviar en binario : ") "Ingresa": Unknown word.
90 print("Ingresa el polinomio generador (ingresa el polinomio de las 2 siguientes posibles maneras y ") "Ingresa": Unknown word.
91 gxC = input ("sin espacios: x4+x2+x1+1 o x^4+x^2+x^1+1): ") "espacios": Unknown word.
92 gxNM = polinomioMas(gxC)
93 gxlista = polinomioLista(gxNM) "Lista": Unknown word.
94 r = polinomioGrado(gxNM) "Grado": Unknown word.
95 mC = addCero(int(r),m)
96 gxB = polinomioGenerado( gxlista) "polinomio": Unknown word.
97 div = division(gxB,mC)
98 crc = crc(int(r),div)
99 mCRC = agregaCRC(m,crc) "agrega": Unknown word.
100 com = division(gxB,mCRC)
101 print("-_-_-_- Resultados -_-_-_-") "Resultados": Unknown word.
102 print("M : " + m)
103 print("G(x) : " + gxC)
104 print("r : " + r)
105 print("x*rM(x) : " + mC)
106 print("Polinomio generado en binario : " + gxB) "Polinomio": Unknown word.
107 print("resuido de la division : " + div) "resuido": Unknown word.

```

4. Conclusiones

4.1. Adrián

Existen varios algoritmos de detección de errores para archivos, estos usan distintas técnicas (paridad, ...). Durante la práctica exploramos algunos algoritmos en distribuciones Linux y Windows, encontramos la paridad de bits de algunas cadenas, y escribimos programas que implementaran los algoritmos Checksum y CRC. También investigamos acerca de estos últimos y elaboramos algunos ejemplos.

4.2. Francisco

Los algoritmos de detección de errores son de gran utilidad ya que estos permiten saber cuando un mensaje llego correctamente o incorrectamente para así saber cómo resolver este problema y así asegurar la integridad de los datos. Cada uno de los diferentes algoritmos tiene diferentes utilidades como por ejemplo el bit de paridad usualmente se usa en aplicaciones en donde la detección de errores de un solo bit es suficiente pero cuando se requiere algo más preciso se utiliza otro tipo de algoritmo, por otra parte tenemos el CRC el cual es más utilizado en redes de computadoras pero solo puede detectar errores no corregirlos.

4.3. Aldo

En esta practica se hizo uso de la paridad de los bits (se usó tanto de la paridad par como la paridad impar) para detectar errores, también se hizo uso de MD5 en archivos para obtener su hash y comprobarlo con los miembros del equipo y por último el tema con los códigos de redundancia cíclica que fueron desarrollados tanto en programa como ejercicio de teoría.

5. Referencias

- Fedora Keeps You Safe | The Fedora Project. (s.f.). <https://fedoraproject.org/security/>
- Fernández, R. P. (2023). ¿Cómo generar y comprobar Checksums en GNU/Linux?. Ingeniero Técnico Industrial Mecánico & Administrador de Sistemas. <https://www.raulprietofernandez.net/blog/gnu-linux/como-generar-y-comprobar-checksums-en-gnu-linux>
- How to check MD5, SHA1 and SHA256? [ArCHIVE] - FedoraForum.org. (s.f.). <https://forums.fedoraforum.org/archive/index.php/t-250700.html>