### Ejercicio 1

Con este tipo de registro y para 70 millones de votantes, ¿cuánto espacio de almacenamiento será necesario?

R: $(11 + 19 + 4)\ bytes\ *\ 7000000\ de\ votantes = 238000000\ bytes\ =\ 238\ Megabytes$

¿Es posible almacenarlo en su disco duro?

R: Sí

¿En una memoria USB?

R: Sí

¿Y en un archivo virtual dentro de la RAM?

R: Sí

## Ejercicio 2

Para este ejercicio, solamente se agregó una línea de código, que indica que si no hay un archivo con ese nombre, lo crearemos y lo editaremos por completo. Y en la función write (cómo viene en la lectura) el primer parámetro es el número que se genera al momento de abrir el documento. Por lo que el código quedaría de la siguiente forma:

```
1. //Programa para crear registros de votos [celular, CURP, partido, separador], con
   el campo "celular" como clave
2.
3. #include <stdio.h>
4. #include <string.h>
5. #include <stdlib.h>
6. #include <fcntl.h>
7. #include <unistd.h>
8. #include <iostream>
9. #include <iterator>
10. #include <sys/types.h>
11. #include <sys/stat.h>
12. #include <cstring>
13.
14. #include <algorithm>    // std::random_shuffle
15. #include <vector>       // std::vector
16. #include <cstdlib>      // std::rand, std::srand
17.
18. using namespace std;
19.
20. struct registro{
21.     char celular[11];
22.     char CURP[19];
23.     char partido[4];
24. };
25.
26. char buffer[BUFSIZ];
27.
28. int main(int argc, char *argv[]){
```

```
29.     char telefono[11], curp[19], t[11], sexo;
30.     int i, n, destino, j, opcion, inicial, elemento;;
31.     struct registro reg1;
32.     vector<struct registro> registros_vector;
33.
34.     //Partidos disponibles 2018
35.     char const partidos[9][4] = {"PRI", "PAN", "PRD", "P_T", "VDE", "MVC", "MOR",
   "PES", "PNL"};
36.
37.     //Entidades federativas
38.     char const entidad[32][3] =  {"AS", "BC", "BS", "CC", "CS", "CH", "CL", "CM",
   "DF", "DG", "GT", "GR", "HG", "JC", "MC", "MN", "MS", "NT", "NL", "OC", "PL",
   "QT", "QR", "SP", "SL", "SR", "TC", "TL", "TS", "VZ", "YN", "ZS"};
39.
40.     if(argc != 3){
41.         printf("Forma de uso: programa numero_registros nombre_archivo\n");
42.         exit(0);
43.     }
44.
45.     //Numero de registros n
46.     n = atoi(argv[1]);
47.
48.     //Genera un numeros telefonicos inicial de 9 digitos y despues se obtendran su
   secuenciales para evitar repeticion
49.     inicial = 500000000 + rand()%100000000;
50.
51.     //Crea todos los registros con numero de telefono consecutivo y los almacena
   en un vector
52.     for(j=0; j<n; j++){
53.         sprintf(telefono, "5%9d", inicial);
54.         inicial++;
55.         strcpy(reg1.celular, telefono);
56.
57.         if(rand()%2 == 0)
58.             sexo = 77;
59.         else
60.             sexo = 72;
61.
62.         i = rand()%32;
63.         sprintf(curp, "%c%c%c%c%c%c%c%c%c%c%s%c%c%c%c%c", 65 + rand()%25 , 65 +
   rand()%25, 65 + rand()%25, 65 + rand()%25, rand()%10 + 48, rand()%10 + 48,
   rand()%10 + 48, rand()%10 + 48, rand()%10 + 48, rand()%10 + 48,
64.             sexo, entidad[i], 65 + rand()%25, 65 + rand()%25, 65 + rand()%25,
   rand()%10 + 48, rand()%10 + 48);
65.         strcpy(reg1.CURP, curp);
66.
67.         i = rand()%9;
68.         strcpy(reg1.partido, partidos[i]);
69.         registros_vector.push_back(reg1);
```

```
70.     }
71.
72.     //Abre un archivo para escritura, si no existe lo crea, si existe lo trunca,
   con permisos rw-
73.     if((destino = open(argv[2], O_WRONLY|O_TRUNC|O_CREAT, 0666)) == -1)
74.     {
75.         perror(argv[2]);
76.         exit(-1);
77.     }
78.
79.     //Aleatoriza el vector de registros e imprime el resultado
80.     random_shuffle(registros_vector.begin(), registros_vector.end());
81.     for (std::vector<struct registro>::iterator it=registros_vector.begin();
   it!=registros_vector.end(); ++it){
82.         reg1 = *it;
83.         write(destino,&1, sizeof(reg1));
84.         write(destino,"\n",1);
85.     }
86.     close(destino);
87.     exit(0);
88. }
```

Poniendo a prueba la generación de archivos, queda de la siguiente forma:

| Número de registros | Tiempo | | |
|---|---|---|---|
| | real | user | sys |
| 7000 | 0.048s | 0.016s | 0.032s |
| 70000 | 0.304s | 0.108s | 0.196s |
| 700000 | 2.840s | 1.012s | 1.828s |
| 7000000 | 29.010s | 10.184s | 18.827s |

Como podemos ver, y podemos imaginar, mientras más datos se generen, más tiempo tarda, pero, lo curioso, es que no se lleva los “minutos” guardando los datos, esto es por el tipo de dato que se está manejando, y en cómo se guardan, que son los bytes, entonces esto hace que sea el proceso mucho más rápido.

## Ejercicio 3

Para el último ejercicio, se utilizó las interfaces de *PaqueteDatagrama* y *SocketDatagrama*, que para crear un socket UDP para poder hacer el intercambio de información, para poder hacer un poco más rápida la lectura del documento, se usó la librería *fstream*, además de que se leyó

linea por linea el documento para poder mandar el archivo. Los códigos del cliente y servidor, se presentan a continuación.

**Cliente:**

```
1.  #include "PaqueteDatagrama.h"
2.  #include "SocketDatagrama.h"
3.  #include <fstream>
4.  #include <stdio.h>
5.  #include <vector>
6.  #include <stdlib.h>
7.  #include <string.h>
8.  #include <iostream>
9.  using namespace std;
10.
11. int main(int argc, char* argv[]){
12.     if(argc != 3){
13.         cout<<"Forma de usa: nombre_programa ip nombre_archivo"<<endl;
14.     }
15.
16.     ifstream archivo(argv[2], ios::binary);
17.     SocketDatagrama sockClient(0);
18.     char cadena[256];
19.     string s;
20.     int res=0;
21.
22.     while(getline(archivo,s)){
23.         memcpy(cadena,s.c_str(),s.size());
24.         PaqueteDatagrama package00(cadena, sizeof(cadena), argv[1], 7200);
25.         int resp;
26.         sockClient.envia(package00);
27.         PaqueteDatagrama package01(3);
28.         resp=sockClient.recibe(package01);
29.         if(resp)
30.             continue;
31.         else{
32.             cout<<"Algo salio mal"<<endl;
33.             archivo.close();
34.             exit(0);
35.         }
36.
37.         sleep(1);
38.     }
39.     cout<<"Archivo terminado de leer"<<endl;
40.     memcpy(cadena,"ya",3);
41.     PaqueteDatagrama package(cadena, sizeof(cadena), argv[1], 7200);
42.     sockClient.envia(package);
43.     archivo.close();
44.
```

```
45.     return 0;
46. }
```

### Servidor

```
1.  #include "SocketDatagrama.h"
2.  #include "PaqueteDatagrama.h"
3.  #include <cstring>
4.  #include <iostream>
5.  #include <fstream>
6.
7.  #include <fcntl.h>
8.
9.  using namespace std;
10.
11. int main(int argc, char* argv[]){
12.
13.     if(argc != 2){
14.         cout<<"Forma de usa: nombre_programa nombre_archivo"<<endl;
15.         exit(0);
16.     }
17.
18.     SocketDatagrama sockServer(0);
19.     PaqueteDatagrama p00 = PaqueteDatagrama(4000);
20.     int con=123;
21.     //Respuesta.h
22.     SocketDatagrama *socketlocal;
23.     socketlocal = new SocketDatagrama(7200);
24.
25.     int destino;
26.     //Abre un archivo para escritura, si no existe lo crea, si existe lo trunca,
    con permisos rw-
27.     if((destino = open(argv[1], O_WRONLY|O_TRUNC|O_CREAT, 0666)) == -1){
28.         perror(argv[1]);
29.         exit(-1);
30.     }
31.
32.     cout <<"Servidor iniciado....\n"<<endl;
33.     while(1){
34.         //getRequest
35.         int tam=socketlocal->recibe(p00);
36.         if(tam==-1){
37.             perror("Recvfrom fallo");
38.             break;
39.         }
40.
41.         char msjRecib[1024];
42.         memcpy(msjRecib,p00.obtieneDatos(),32);
43.         if(strcmp(msjRecib,"ya")==0)
```

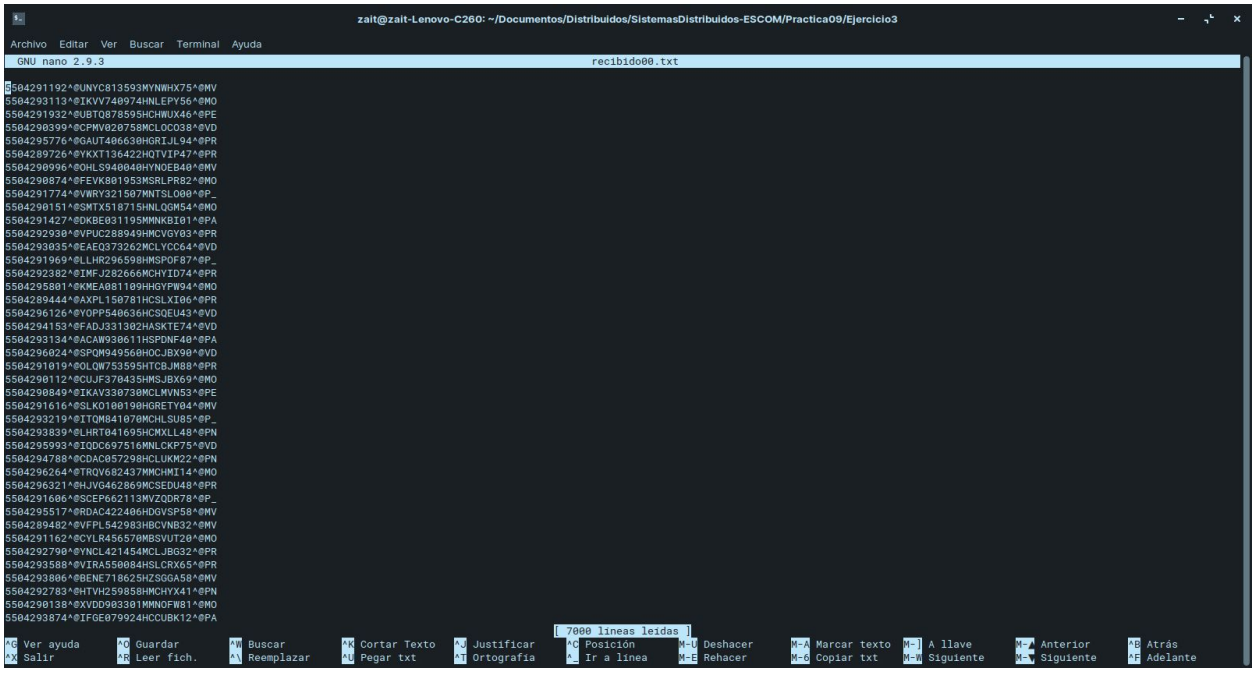
```
44.             break;
45.         PaqueteDatagrama
   p01((char*)&(con),3,p00.obtieneDireccion(),p00.obtienePuerto());
46.         socketlocal->envia(p01);
47.
48.         write(destino,msjRecib, 32);
49.         write(destino,"\n",1);
50.     }
51.     cout<<"Servidor terminado"<<endl;
52.     close(destino);
53.
54.     return 0;
55. }
```

Se hicieron pruebas con un archivo de 7000 registros de votación, el cual el cliente lee dicho archivo, linea por linea, los guarda en una cadena y este los envía al servidor, recibiendo los datos y guardando todo en una cadena para después guardarlo en un documento.

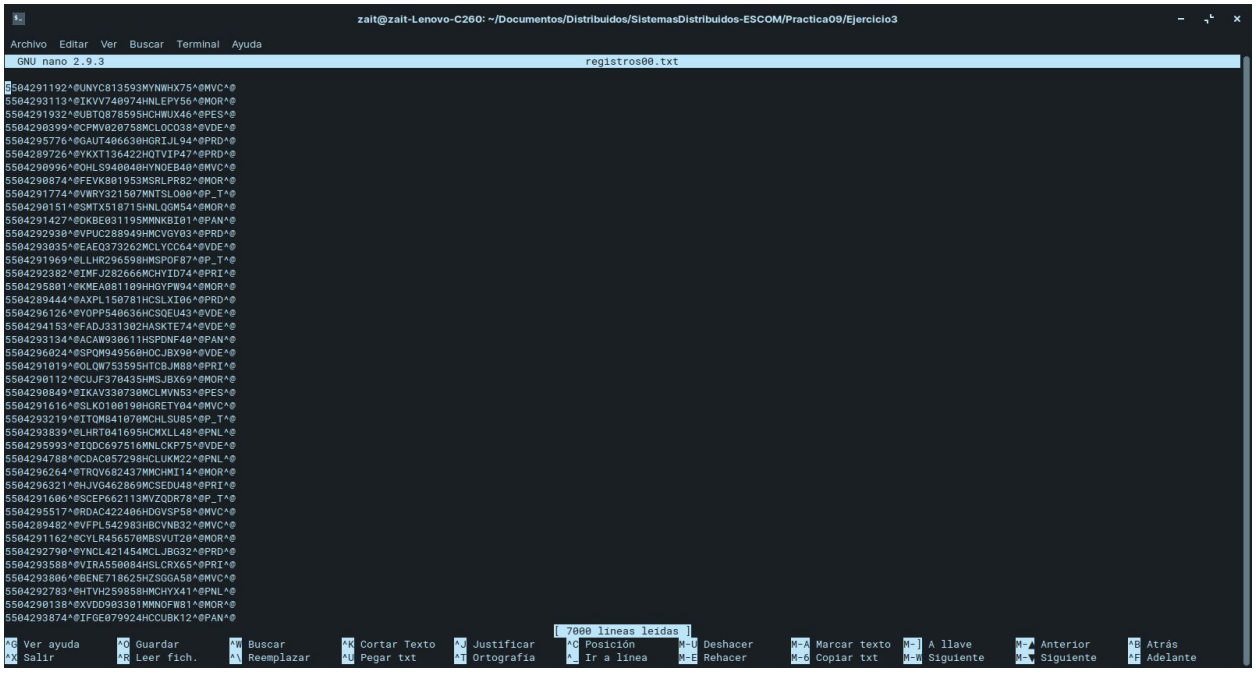
```
9/Ejercicio3$ time ./Server recibido00.txt
Servidor iniciado....

Servidor terminado

real    0m2.109s
user    0m0.129s
sys     0m0.276s
```

```
time ./Client 127.0.0.1 registros00.txt
Archivo terminado de leer

real    0m0.596s
user    0m0.116s
sys     0m0.126s
```

Como podemos notar, el servidor acaba al mismo tiempo que el cliente, es decir que el cliente mandará el archivo completo y para saber que este acabó, le manda un mensaje al servidor para que este acabe y a su vez cierre el archivo para dejar de editarlo. También se agregó al apunte los datos que se mandaron y recibieron, es decir, el archivo de lectura y el de almacenamiento.




Archivo de lectura (Enviado)




Archivo de escritura (Recibido)