Simulador COVID-19

Versión serie

Jon Etxauri, José María Irizar

April 2020

Indice

1 Introducción

El proyecto ha relizar es un simulador de evolución del virus Coronavirus (COVID-19) que ha provocado esta nueva situación donde todos los ciudadanos estamos confinados en nuestras casas.

En la evolución de este virus se muestra el número de personas que sobreviviría al virus tras haberse contagiado, el número de ciudadanos que no se contagiaría y el número de ciudadanos que no conseguiría superar el contagio, por cada día y de manera global.

Para la primera parte del proyecto de simulación de la evolución del virus se han escrito seis ficheros, los cuales se dividen en código y en sus headers. Los ficheros son referentes al programa principal (que contiene el main), el objeto persona (contiene el struct y las funciones relacionadas, que se comentan más adelante) y las funciones para los calculos de probabilidades. El programa principal se compone de dos partes: inicializaciones de variables y bucle principal. El bucle principal a su vez se puede dividir en otras tres partes: mover todas las personas, decisión de contagiar y/o morir de cada persona y actualización de variables.

El programa se basa en estructuras del tipo array, en la cual se guardan las personas. Las personas son struct en las cuales se guardan información de los individuos. En un principio se penso en sustiuir el array por una matriz, pero debido a que esto era menos eficiente se decidió que la opción mas óptima era desarrollar mediante el array y que cada struct de cada persona guardará la posición en los ejes X e Y.

2 Uso y ejecución

Para ejecutar el programa se le indican como parametro el número de días que se quiere que transcurran, tamaño del escenario (ancho y alto), radio de contagio, probabilidad de contagio en caso de estar dentro del radio indicado, la población, la edad media que se desea que sea a priori y el batch (engloba tantos dias como el valor de este parametro para ahorrar imprimir por pantalla exceso de información). En caso de no habersele indicado dichos parametros, el programa se interrumpe e imprime un error de ejecución. Esto se controla de la siguiente manera:

```
if(argc!=7){
   fprintf(stderr,"%s <tiempoASimular> <tamanoAncho> <tamanoAlto>
       <radio> <probRadio> <poblacion> <edadMedia> <batch>\n",
       argv[0]);
   exit(1);
}
```

El código para la compilación y ejecución del programa es como sigue:

```
// COMPILACION
gcc main.c persona.c probabilidad.c -o main -lgsl -lm
// EJEMPLO DE EJECUCION
./main 10000 50 50 5 0.7 1000 70 1
```

3 main.c

3.1 Declaración de variables

Al principio del programa, tras comprobar que se le han pasado todos los parámetros esperados, se controla que los valores de éstos estén en un rango aceptable, y se guardan en variables.

```
  int TIEMPO    = atoi(argv[1]);
  int ESCHEIGHT  = atoi(argv[2]);
  int ESCWIDTH  = atoi(argv[3]);
  int RADIO     = atoi(argv[4]);
  float PROBRADIO = atof(argv[5]);
  int POBLACION  = atoi(argv[6]);
  int EDADMEDIA  = atoi(argv[7]);
  int BATX      = atoi(argv[8]);

  if (PROBRADIO > 0.9 || PROBRADIO < 0 || TIEMPO < BATX || TIEMPO < 1
      || RADIO >= ESCWIDTH || RADIO >= ESCHEIGHT) {
      fprintf(stderr,"Error de parametros: \n\t- La probabilidad de
          contagio debe estar comprendido entre 0 y 1.\n\t- El tiempo
          a simular debe ser mayor que 1.\n\t- El batch no puede ser
          mayor que el tiempo a simular.\n\t- El radio de contagio
          debe ser menor que el tamaino del lienzo.\n");
     exit(1);
  }
```

A continuación se declara e inicializan las variables que serán utilizadas en la ejecución del programa. Estas variables son utilizadas para recorrer bucles (i, e, j), para guardar valores de posiciones de una persona, el número de habitantes que quedan vivos, la edad media de los habitantes y para llevar la cuenta de las personas que se han contagiado, recuperado y muerto, tanto en cada ronda como de manera global.

Además se declaran dos variables para controlar dos ficheros (que se crean en caso de que no existan en el directorio) donde se almacenará cada movimiento de las personas como el historial de los datos de cada día.

Por último se declara un array dinámico que guardará todas las personas que quedan vivas en la población.

```
  int i, e, j;
     int rangox, rangoy;
  int muertosRonda, curadosRonda, contagiadosRonda;
  int muertosTotales = 0;
  int curadosTotales = 0;
  int contagiadosTotales = 0;
  int diasTranscurridos = 0;
  int pobActual = POBLACION;
  int edadMedia = EDADMEDIA;

  // INICIALIZACION FICHEROS
  FILE *dias, *posic;
  posic=fopen("historialposic.txt","w+");
  dias=fopen("historialdias.txt", "w+");

  // INICIALIZACION ARRAY PERSONAS
  struct persona *personas;
   personas = malloc(POBLACION*sizeof(struct persona));
```

Posteriormente se llena el array de tantas personas como se indique el tamaño de la población (indicado con un parámetro) y se infecta un individuo aleatoriamente generando un número aleatorio de 0 a POBLACION.

```
   // CREAR POBLACION
   for(i=0; i<POBLACION; i++)
       personas[i] = crearPersona();
   edadMedia = mediaEdad(personas, POBLACION);

   printf("STATUS: PRIMER INFECTADO!\n");
   // PRIMER INFECTADO!
   int aux = rand()%POBLACION;
   personas[aux].estado = 1;
   contagiadosTotales++;
```

Para calcular la edad media se ha creado una función a la que se le pasan como parametro el array de las personas que pertenecen a la población y el número de personas que lo componen en ese momento (será un integer menor o igual a POBLACION). El programa recorre el array simplemente sumando la edad de cada individuo y lo divide por el número de personas que lo componen.

```
int mediaEdad(persona *per, int pobl){
  int i;
  int media = 0;
  for(i=0; i<pobl; i++)
     media += per[i].edad;

  return (media/pobl);
}
```

3.2 Bucle principal

El bucle principal es recorrido tantas veces como se le indica en el parametro de ejecución mediante un while. Al inicio de cada iteración se reinician las variables referentes a una "ronda"; muertosRonda, curadosRonda y contagiadosRonda. Al final de cada iteración se imprime una línea de seguimiento indicando el número de iteración en la que se encuentra el programa, el número de infectados, fallecidos, recuperados, número de personas vivas y edad media de estas.

3.2.1 Movimiento de personas

Lo primero que se realiza es el movimiento y velocidad de cada persona. Para esto se recorre el array de la población y cada persona se le pasa como parametro a la función que controla el movimiento (función definida en el fichero persona.c que se analizará posteriormente). Por cada persona que se mueve se registra su movimiento en el fichero historialposic.txt.

```
// MOVER PERSONA y CAMBIAR VELOCIDAD PARA LA SIGUIENTE RONDA
for(i=0; i<pobActual; i++){
   moverPersona(&personas[i], ESCWIDTH, ESCHEIGHT);
   // FICHERO: GUARDAR CAMBIO DE PERSONA
   if(diasTranscurridos%BATX==0)
      fprintf(posic,"%d,%d,%d:",personas[i].pos[0],personas[i].pos[1],
      personas[i].estado);
}
// FICHERO: SALTAR DE LINEA TRAS MOVER TODAS LAS PERSONAS
if(diasTranscurridos%BATX==0)
   fprintf(posic,"\n");
```

3.2.2 Decisión de muerte y/o contagio

A continuación se toman las dos decisiones más importantes de cada persona: si va a ser infectada o no, y en caso de estar infectada, si se va a morir o se recupera (en caso de haber pasado quince días infectada). Para esto se recorre el array de la población pero se trabaja únicamente con los contagiados (con un if para mirar el estado).

Se guarda la posición de la persona infectada y se recorre toda la población, se le pasa a la función que toma la decisión cada una de las personas la posición del infectado, el radio de contaminación que se ha pasado como parametro y la probabilidad de contagio.

Para cada uno de los infectados se decide si se muere o no (en base a su probabilidad de muerte), y en caso negativo, se mira el número de días que lleva infectado, para pasar a estado 2 si lleva más de cinco días infectado, o para curarse y pasar a estado 3 en caso de pasar quince días infectado. Esto se controla en la funcion matarPersona() en el fichero persona.c. En caso de morir (la función devuelve 1), se recolocan las personas que quedan vivas en el array, y en caso de que se recupere devuelve un 2 y se actualizan las variables.

```
// INFECTADOS: COMPROBAR RADIO DE CONTAGIOS y DECISIONES DE MUERTE o
    SUPERVIVENCIA
for(i=0; i<pobActual; i++){
    if(personas[i].estado == 1 || personas[i].estado == 2){
      rangox = personas[i].pos[0];
      rangoy = personas[i].pos[1];

      // DECIDIR SI SE CONTAGIA CADA INDIVIDUO EN BASE AL RADIO DE UN
          CONTAGIADO
      for(e=0; e<pobActual; e++)
         contagiadosRonda += infecPersona(&personas[e], rangox, rangoy,
             RADIO, PROBRADIO);

      // DECIDIR SI SE MUERE O SE RECUPERA
      int samatao = matarPersona(&personas[i]);
      if(samatao == 0){       // SE MUERE
         for(e=i; e<pobActual-1; e++)
            personas[e] = personas[e+1];
         muertosRonda++;
         contagiadosTotales--;
         pobActual--;
      } else if(samatao == 2){      // SE CURA
         curadosRonda++;
         contagiadosTotales--;
      }
   }
}
```

3.2.3 Actualización de variables

Al final de la iteración se calcula la nueva edad media de la población, se incrementa en uno el número de días que han transcurrido y se actualizan las variables que almacenan el número total de contagiados, fallecidos y recuperados, sumándoles a éstas el valor de las variables que almacenan los datos de la iteración actual.

```
   // ACTUALIZAR EDAD MEDIA
   edadMedia = mediaEdad(personas, pobActual);

   // RULAR TIEMPO
   diasTranscurridos++;

    // ACTUALIZAR VALORES TOTALES
   contagiadosTotales += contagiadosRonda;
   curadosTotales += curadosRonda;
    muertosTotales += muertosRonda;
```

Al acabar cada una de las iteraciones, el programa imprime el resultado obtenido, listando el número de iteraciones que han transcurrido, el total de infectados en ese momento, el número de personas que se han curado, fallecidos, población actual y su media de edad. Los valores de seguimiento se imprimen tanto los datos globales como los de cada iteración, y también se almacenan en el fichero historialdias.txt en caso de que los dias que han transcurrido sean multiplos del batch indicando en los parametros.

```
// VISUALIZAR PROGRESO
if(diasTranscurridos%BATX==0){
    printf("DIA %i: %i INFECTADOS (%i NUEVOS), %i RECUPERADOS (%i
        NUEVOS), %i FALLECIDOS (%i NUEVOS). POBLACION: %i, EDAD MEDIA:
        %i\n", diasTranscurridos, contagiadosTotales, contagiadosRonda,
        curadosTotales, curadosRonda, muertosTotales, muertosRonda,
        pobActual, edadMedia);

   fprintf(dias, "%d:%d,%d,%d\n", diasTranscurridos, contagiadosTotales,
       curadosTotales, muertosTotales);
   }
```

Para finalizar el programa, únicamente se libera la memoria que había sido dinámicamente reservada para el array de perosnas, con el comando free. Hay que destacar que el programa llega a su fin tanto cuando transcurren los días indicados por parametro como cuando no quedan personas con vida o infectados.

```
    // CONTROLAR SI SE DEBE FINALIZAR EL PROGRAMA
    if(contagiadosTotales == 0) break;
    if(pobActual == 0) break;

   // LIBERAR MEMORIA AL ACABAR PROGRAMA
   free(personas);
    printf("STATUS: Fin del programa.\n");
```

4 persona.c

4.1 Declaración de variables

Primero se han definido las variables estaticas mediante defines. En estas variables se ha decidido indicar las probabilidades de muerte en base a la edad del individuo (obtenidos del pdf de presentación del proyecto).

```
#define EDAD1 0.004 // < 50
#define EDAD2 0.013 // 50 - 60
#define EDAD3 0.036 // 60 - 70
#define EDAD4 0.080 // 70 - 80
#define EDAD5 0.148 // > 80
```

Además, en el header (persona.h) se ha creado el struct persona con los campos indicados en la presentación del documento, habiendosele añadido el campo diasContaminado, para llevar una cuenta del número de días que el individuo lleva enfermo.

```
struct persona {
   int edad;
   int estado;
   int diasContaminado;
   float probMuerte;
   int pos[2];
   int vel[2];
};
```

4.2 Funciones relacionadas con las personas

Se han creado un total de cuatro funciones para realizar tareas que se emplean varias veces cuando se ejecuta el programa. Estas tareas son para crear, mover, infectar y matar personas.

4.2.1 Función para crear personas

Para crear una persona se calcula la edad con una función de probabilidad (perteneciente al fichero probabilidad.c, que se estudia a continuación), se pone el estado en 0 (sano, sin haber sido infectado hasta el momento), el número de días contaminado en 0, se le indica la probabilidad de muerte en base a su edad, su posición random en el escenario y la velocidad (entre -5 y 5). Los parámetros que se le indican son la edad media que debe tener la población y el tamaño (ancho y alto) del escenario.

```
struct persona crearPersona(int edadMedia, int escAncho, int escAlto){
   struct persona per;
   per.edad = numeroRandom(edadMedia);
   per.estado = 0;
   per.diasContaminado = 0;

   // PROBABILIDAD DE MUERTE EN BASE A EDAD
   if(per.edad<50)
      per.probMuerte = EDAD1;
   else if(per.edad>=50 && per.edad<60)
      per.probMuerte = EDAD2;
   else if(per.edad>=60 && per.edad<70)
      per.probMuerte = EDAD3;
   else if(per.edad>=80 && per.edad<80)
      per.probMuerte = EDAD4;
   else
      per.probMuerte = EDAD5;
```

```
   //CALCULO DE LA POSICION y VELOCIDAD INICIAL
   per.pos[0] = rand()%escAlto;
   per.pos[1] = rand()%escAncho;
   per.vel[0] = rand()%10+(-5);
   per.vel[1] = rand()%10+(-5);

   return per;
}
```

4.2.2 Función de movimiento

Para mover una persona se controla que la nueva posición que se calcula no sale del escenario y la velocidad para la próxima iteración se calcula también en base a su posición (para así si está en un margen, rebote). Se le pasan por parametro el struct persona y el tamaño del escenario (alto y ancho).

```
void moverPersona(Persona *pers, int escAncho, int escAlto){
   // SE CONTROLA PRIMERO UN EJE, DESPUES EL OTRO, SE CONTROLA
   // QUE NO SE SALGA DE LOS LIMITES DEL ESCENARIO, Y SE ELIGE
   // LA VELOCIDAD DE LA SIGUIENT RONDA EN BASE A SU POSICION:
   // SI ESTA EN EL BORDE REBOTA, SI NO SE MUEVE LIBREMENTE :)

   if(pers->pos[0] + pers->vel[0] >= escAlto){
      pers->pos[0] = escAlto;
      pers->vel[0] = rand()%5+(-5);
   } else if(pers->pos[0] + pers->vel[0] <= 0){
      pers->pos[0] = 0;
      pers->vel[0] = rand()%5;
   } else {
      pers->pos[0] += pers->vel[0];
      pers->vel[0] = rand()%10+(-5);
   }

   if(pers->pos[1] + pers->vel[1] >= escAncho){
      pers->pos[1] = escAncho;
      pers->vel[1] = rand()%5+(-5);
   } else if(pers->pos[1] + pers->vel[0] <= 0){
      pers->pos[1] = 0;
      pers->vel[1] = rand()%5;
   } else {
      pers->pos[1] += pers->vel[1];
      pers->vel[1] = rand()%10+(-5);
   }
}
```

4.2.3 Función de decisión de infección

Se guarda la posición de la persona infectada y se mira el radio de contaminación; en caso de que otra(s) persona(s) esté(n) dentro de este radio, mediante probabilidad se decide si esta(s) persona(s) se infecta(n) o no. Se le indican por parametro el sctruct de la persona sobre la que se tiene que tomar la decisión, la posición de una persona infectada, el radio de contagio y la probabilidad de contagio. La función devuelve 1 si se decide que la persona se contagia, en caso contrario 0.

```
int infecPersona(Persona *per, int rangox, int rangoy, int radio, float
    probRadio){
  // SI NO ESTA INFECTADO y NO LO HA ESTADO
  if(per->estado == 0){
    // SI ESTA DENTRO DEL RANGO DE EJE X DEL INFECTADO
    if(per->pos[0] <= rangox+radio && per->pos[0] >= rangox-radio){
      // SI ESTA DENTRO DEL RANGO DE EJE Y DEL INFECTADO
      if(per->pos[1] <= rangoy+radio && per->pos[1] >= rangoy-radio){
        float deci = (rand()%100) /100.0;
        if(deci>probRadio){
          per->estado = 1;
          return 1;
        }
      }
    }
  }
  return 0;
}
```

4.2.4 Función de decisión de muerte

Para cada uno de los infectados se decide si se muere o no (en base a su probabilidad de muerte), y en caso negativo, se mira el número de días que lleva infectado, para pasar a estado 2 si lleva más de cinco días infectado, o para curarse y pasar a estado 3 en caso de pasar quince días infectado. Se le indica como parametro la persona sobre la que se debe tomar la decisión. Se genera un float con una función de probabilidad (se analiza posteriormente) y en base a ese número se toma la decisión. La función devuelve 0 si la persona fallece, 1 si la persona pasa a estado 2 (muestra sintomas) y 2 si la persona ha pasado 15 días infectada y se recupera.

```
int matarPersona(Persona *per){
   float deci = calcProb();
   if(deci <= per->probMuerte)
      return 0;
   else {
      per->diasContaminado = per->diasContaminado +1;
      if(per->estado == 1 && per->diasContaminado >= 5){
         per->estado = 2;
         return 1;
      } else if(per->estado == 2 && per->diasContaminado >= 15){
         per->estado = 3;
         return 2;
      }
   }
}
```

5 probabilidad.c

5.1 Función para el cálculo de un entero entre 0 y 100

La siguiente funcion es la encargada de calcular las edades de las personas siguiendo una distribución normal. Se ha implementado siguiendo un algoritmo encontrado en internet y cambiandolo para que funcionase de manera correcta en el proyecto. Link: https://stackoverflow.com/questions/19944111/creating-a-gaussian-random-generator-with-a-mean-and-standard-deviation

El algoritmo recibe dos parámetros, uno el de la media con la que se desea sacar el numero y la otra con la desviación. Se ha creado otra función mediante la cual se calcula la desviación estandar deseada dependiendo de la edad media deseada.

El algoritmo utilizado se basa en el metodo Box muller.

```
double rand_normal(double media, double stddev)
{
    static double n2 = 0.0;
    static int n2_cached = 0;
    if (!n2_cached)
    {
        double x, y, r;
        do
        {
            x = 2.0*rand()/RAND_MAX - 1;
            y = 2.0*rand()/RAND_MAX - 1;
            r = x*x + y*y;
        }
        while (r == 0.0 || r > 1.0);
        {
```

```
            double d = sqrt(-2.0*log(r)/r);
            double n1 = x*d;
            n2 = y*d;
            double result = n1*stddev + media;
            n2_cached = 1;
       if(result>100){
      return 100.0;
       }else if(result<0){
      return 0.0;
       }else{
              return result;
            }
   }
    }
    else
    {
        n2_cached = 0;
   double aux=n2*stddev + media;
       if(aux>100){
      return 100.0;
   }else if(aux<0){
      return 0.0;
   }else{
      return n2*stddev + media;
      }
     }
}
```

5.2 Función para el cálculo de un numero float entre 0 y 1

Mediante el siguiente algoritmo se genera un número random, en este caso siguiendo una distribución uniforme. El algoritmo saca un valor aleatorio usando la libreria gsl. El numero aleatorio se consigue mediante la asignación de una semilla basada en el momento del día en el que se esta generando.

Esta función la usamos para calcular la probabilidad de que un individuo fallezca o se infecte

```
float calcProb(){
    const gsl_rng_type * T;
    gsl_rng * r;
    gsl_rng_env_setup();
    struct timeval tv;
    gettimeofday(&tv,0);
    unsigned long mySeed = tv.tv_sec + tv.tv_usec;
    T = gsl_rng_default; // Generar setup
    r = gsl_rng_alloc (T);
    gsl_rng_set(r, mySeed);
```

```
    double u = gsl_rng_uniform(r); // Generar numero entre 0 y 1.
    gsl_rng_free (r);
    return (float)u;
}
```

5.3 Función para el cálculo de la desviación estándar

Mediante la sencilla funcion que a continuación se muestra, se cálcula un valor estimado para la desviación. Se ha decidido y visto oportuno el siguiente calculo de la distribución. Mediante esta distribución se consigue una variación suficiente para ver los cambios que hay en la población.

Este función solo se ejecuta una vez al principio del programa.

```
int calculo_desv(int edad){
   int resul;
   if(edad>=50){
      resul=100-edad;
      resul=resul/2;
      return resul;
   }else{
      resul=edad/2;
      return resul;
   }
}
```

Conclusión: Mediante el desarrollo de este proyecto hemos podido ver como dependiendo de las decisiones adoptadas en un pricipio, puede afectar en gran medida al desarrollo de partes posteriores. Por ello, para adelantarnos a problemas que puede haber en el momento de paralelizar mediante el uso de MPI. Una de estas decisiones ha sido el uso de un array para representar en vez de una matriz, que aunque es mas dificil, ahorraremos ancho de banda en el momento de dividir la carga.

Por otro lado, el uso de la librería GSL ha sido de gran ayuda para el desarrollo de algunas funciones estadísticas, por ejemplo, en la de calcProb(), en la cual usabamos funciones de GSL para generar numeros aleatorios.