Universidad de Cantabria

Diseño y Operación de Redes Telemáticas

Práctica 2: Implementación de un simulador por eventos

Autores : ***Grupo 2***	*E-mail :*
Betegón garcía, Miguel	mbg51@alumnos.unican.es
Sierra Menéndez, Sergio	ssm86@alumnos.unican.es
Alonso González, Pablo	pag65@alumnos.unican.es

Diciembre 2018

Índice

1. Introducción **2**

2. Implementación del simulador de eventos **3**
2.1. Explicación del programa 3
2.2. Comprobación del funcionamiento del sistema 5

3. Análisis de un sistema de pérdida pura con fuentes finitas. **7**
3.1. Comprobación del funcionamiento del sistema 10
3.1.1. Probabilidad de pérdida 10
3.1.2. Ocupación de todos los recursos del sistema 16

Anexos

A. $M/M/\infty$ **19**
A.1. `main.c` 19
A.2. `library2.h` 21
A.3. `graficas.m` 23

B. Sistema de pérdida pura con fuentes finitas **24**
B.1. `main.c` 24
B.2. `library2.h` 27
B.3. `graficas.m` 30

1. Introducción

Esta segunda práctica de la asignatura *Diseño y Operación de Redes Telemáticas* trata de implementar el código en C del núcleo de un simulador de enventos. [1]. En este caso, el proyecto a realizar ha sido el *Proyecto B. Análisis de un sistema de pérdida pura con fuentes finitas.*

Esta práctica esta estructurada en dos partes, que corresponden con los dos ejercicios a realizar:

(i) Implementación del Simulador de Eventos.

- Emplear el lenguaje de programación C para el motor principal de la herramienta de simulación
- Uso de una lista enlazada donde se introducen los eventos de forma ordenada según el tiempo de ejecución correspondiente.
- El Simulador debe ejecutarse una vez llega el primer paquete hasta que se cumpla una condición dada.
- Analizar el correcto funcionamiento del motor de eventos, para lo que se utiliza un sistema $M/M/\infty$, en el que los paquetes llegan según un proceso de Poisson, y nadaás llegar se transmiten, asumiendo un tiempo de servicio exponencial negativo.
- Se comprueba el modelado anterior mostrando gráficamente el tiempo entre llegadas y el tiempo en el sistema.

(ii) Utilización del Simulador para el análisis de sistemas de comunicaciones.

- Análisis de un sistema de pérdida pura con fuentes finitas.
- Utilizar el simulador para estudiar el comportamiento de una pico-célula de un sistema de comunicaciones móviles.
- Representar la probabilidad de pérdida en función del número de fuentes en el sistema
- Representar el porcentaje del tiempo en el que todos los recursos están ocupados.

El código *Matlab* generado al responder a los ejercicios propuestos se encuentra en el repositorio de Github, [2].

2. Implementación del simulador de eventos

En esta primera parte, Con el fin de llevar a cabo este apartado se proporcionan los siguientes parámetros, que corresponden este grupo y son mostrados en la tabla 1:

Grupo	$\bar{\boldsymbol{\lambda}}[pkt/s]$	$\boldsymbol{T_s}[ms]$
Grupo 2	10	100

Cuadro 1: Parámetros $M/M/\infty$.

Los parámetros del cuadro 1 son en referencia a un sistem $M/M/\infty$, sistema que modela el simulador de eventos.

Con el fin de realizar este ejercicio se han desarrollado tres ficheros, que se encuentran en el Anexo A: `main.c`, donde figura el código fuente del programa. `library2.h`, librería de la que hace uso `main.c` y donde se encuentran todos los parámetros, constantes y funciones utilizadas y `graficas.m`, un script de *Matlab* donde se realizan los cálculos para conseguir las gráficas del tiempo entre llegadas y el tiempo medio en el sistema.

2.1. Explicación del programa

Antes de comenzar el ejercicio se explica la estructura de un evento, que se programa en C como un tipo de dato `struct` formado por tres miembros, escrito en las siguientes líneas de `library2.h`, código situado en el Anexo A.2:

```
18 typedef struct evento
19 {
20   float tiempo;
21   int tipo;
22   struct evento *next; // puntero al siguiente evento.
23 } evento;
```

El significado de cada uno de los tres miembros que conforman la estructura `evento` se explica a continuación:

`tiempo` – Tiempo en el que se debe procesar el evento.

`tipo` – Existen tres tipos de paquetes:

```
8  #define PKT_END 0 // tipo  final
9  #define PKT_IN 1  // tipo llegada
10 #define PKT_OUT 2 // tipo salida
```

`evento *next` – Puntero al siguiente evento de la lista, en caso de que no haya siguiente evento, apunta a `NULL`.

Una vez explicado esto, se comienza el ejercicio generando el paquete con el que dará comienzo la simulación, *paquete génesis o de inicio*. Uno de los requisitos es que los paquetes lleguen según un proceso de Poisson, por lo que el tiempo de llegada del primer paquete se calcula como una variable aleatoria de Poisson de media $\bar{\lambda}$.

Este primer evento se inserta en la lista enlazada, hasta ahora vacía, añadiendo en ella el tiempo y el tipo de paquete (`PKT_IN` al ser el primer paquete que llega al sistema). La creación e inserción de este primer está en las siguientes líneas, pertenecientes a `main.c`:

```
14 tiempo_corriendo = TIEMPO_INICIAL;// Se inicializa el tiempo,
       se actualiza tras cada pkt nuevo.
15
16 // Generacion e insercion del evento genesis:
17 tiempo_primera_llegada = random_poisson(LAMBDA);
18 tiempo_corriendo += tiempo_primera_llegada;
19 InsertarEvento(&lista_eventos,CrearEvento(tiempo_corriendo,
      PKT_IN)); // Creacion e insercion del evento en la lista.
```

Se observa que primero se crea el tiempo en el que llega el paquete, siendo este tiempo creado por la función `random_poisson` que genera una variable aleatoria de Poisson de media la tasa de llegadas por fuente libre. Después, se inserta el evento en la lista de eventos, siendo esta `lista_eventos` con el tiempo previamente creado y con un tipo de paquete `PKT_IN`.

Asimismo, se inserta un evento para que finalice la simulación tras un tiempo específico. Con ese fin, se el tipo de paquete a insertar es `PKT_END` y el tiempo del mismo es en el que se desea que termine la simulación.

En este punto, la lista cuenta con dos eventos: el evento de inicio y el evento de fin de simulación. Con ellos dos, ya es posible comenzar a recorrer la lista.

El modo en el que se recorre la lista es analizar que tipo de evento es el que se situa en primera posición de esta. Como se ha mencionado anteriormente, existen 3 tipos de eventos, y por tanto, tres procedimientos diferentes:

`PKT_IN` Este tipo de evento simula la llegada de un paquete al sistema, es el primer tipo de evento que se procesa ya que para comenzar la simulación se crea un evento de llegada como se ha explicado anteriormente.

Cuando llega un paquete al sistema, se generan dos nuevos eventos nuevos automáticamente, uno de salida y otro nuevo de entrada.

En cuanto al evento de salida (`PKT_OUT`) que se genera, su tiempo es el tiempo en el que se está procesando el evento de entrada sumado al tiempo que va a permanecer el paquete en el sistema, calculado usando una variable aleatoria exponencial negativa de media μ (Tasa de muerte), que se calcula a partir del Tiempo de servicio proporcionado para este grupo como $\mu = 1/T_s$. Este nuevo evento se guarda en la posición que le corresponda en la lista enlazada.

El otro evento que se genera es de tipo `PKT_IN`, lo significa que el tiempo que el tiempo se calcula como el tiempo obtenido mediante una variable aleatoria de Poisson con media λ sumado al tiempo del evento de entrada que se está procesando.

Una vez generados los dos eventos y por tanto finalizado el procesado del evento, se borra el evento de la lista y por tanto el próximo paso es procesar el siguiente evento.

`PKT_OUT` Cuando se procesa un evento de salida solo se ha de elminar ese evento de la lista. Este evento no genera ningun otro evento.

`PKT_END` Es el último evento que se va a procesar, una vez se procese este evento se finaliza el programa.

2.2. Comprobación del funcionamiento del sistema

Ya implementada esta lógica de procesado, se comprueba el funcionamiento del sistema aprovechando los tiempos de llegada y de permanencia en el sistema, cerciorandose de que ambos tiempos siguen una distribución exponencial.

Con este objetivo, se escribe en ficheros los tiempos en los que llega un paquete nuevo al sistema y eltiempo que permanece cada paquete en el sistema. Cuando se analiza un evento `PKT_IN` se guarda su tiempo y cuando se crea el evento `PKT_OUT` que surge del evento de llegada, se guarda el tiempo en el que se procesa.

Con dichos tiempos es posible comprobar si su distribución es o no exponencial negativa con el código de *Matlab* adjunto en el Anexo A.3 y mostrados en la figura 1.

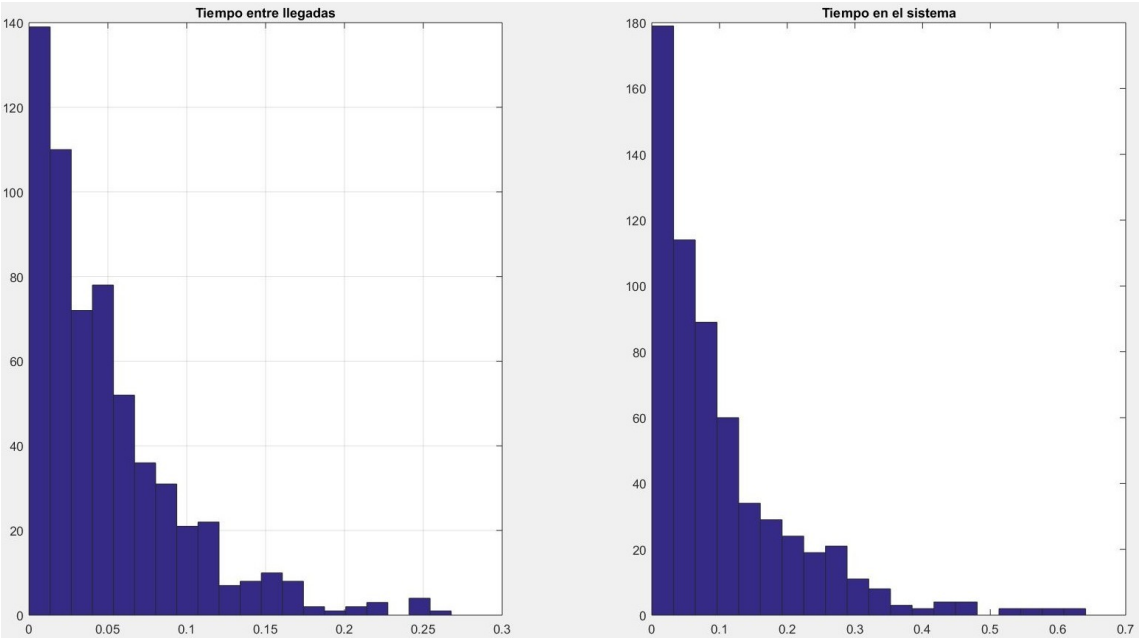

Figura 1: Tiempo entre llegadas y tiempo en el sistema.

En cuanto a la gráfica de la izquierda, el tiempo entre llegada, se tienen los tiempos de cada llegada por lo que debe restar el tiempo de una llegada y la anterio y guardar el resultado en un vector que es representado con un histograma en la gráfica de la izquierda de la figura 1, y que sigue una distribución exponencial negativa.

En referencia a la gráfica de la derecha de la figura 1, el tiempo en el sistema, basta con representar el vector almacenado con los tiempos que han permanecido los paquetes en el sistema. Estos datos se obtienen guardando cada vez que se calculaba la variable aleatoria exponencial negativa de media μ. Se aprecia que sigue también una distribución exponencial negativa.

Con estos resultados se da por finalizada esta primera parte de la práctica, teniendo como punto de partida, el simulador de eventos conseguido.

3. Análisis de un sistema de pérdida pura con fuentes finitas.

Una vez lograda la implementación del simulador de eventos para un sistema $M/M/\infty$, se adapta el simulador para conseguir desarrollar un sistema de comunicaciones, en concreto, un sistema de pérdida pura con fuentes finitas.

Los parámetros correspondientes a este sistema se pueden observar en el cuadro 2. Una forma de entender el número de recursos, es como el número de llamadas capaces de ser cursadas simultáneamente.

Grupo	**Proyecto**	$\#\boldsymbol{Recursos}$	$\#\boldsymbol{fuentes}$	$\bar{\boldsymbol{\lambda}}[pkt/s]$	$\bar{\boldsymbol{T_s}}[s]$
Grupo 2	Proyecto B	10	[1...40]	180	0.001111111

Cuadro 2: Parámetros grupo 2.

Con el propósito de entender fácilmente el funcionamiento de este sistema, se ofrece la figura 2, sacada de [1]. En ella se expone un diagrama de estados con el que se observa globalmente el sistema.

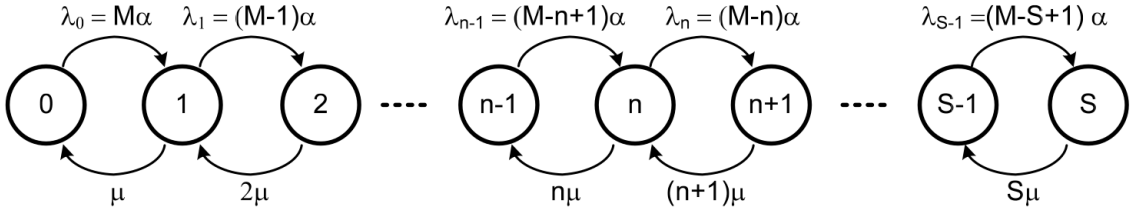

Figura 2: Diagrama de estados del sistema de fuentes finitas con pérdida pura. Fuente [1]

En el diagrama se advierte que el número de estados es S, lo que equivale al número de recursos del sistema, es decir, el número de llamadas que se pueden cursar simultamente. Por lo tanto, $S = 10$ en este sistema. M son los recursos que quedan libres en cada estado. Por último, n es el número de servidores ocupados en ese estado.

Una vez que conocido el funcionamiento del sistema se procede a comentar lo que se requiere para superar la práctica. En este proyecto, se debe representar la probabilidad de pérdida en función del número de fuentes en el sistema. Además, se debe representar el porcentaje del tiempo en el que todos los recursos están ocupados en función también del número de fuentes en el sistema.

Es importante afianzar el concepto de la probabilidad de pérdida. En este sistema para que se produzca una pérdida se debe estar en el estado S de la Figura 2. Una vez llega un paquete en este estado se produce la pérdida del mismo ya que no se tienen recursos para gestionarlo ni para mantenerlo en espera.

Una vez que se sabe en que momento se produce una pérdida se procede a comentar el código utilizado para la resolución del sistema. En primer lugar, cabe mencionar que está basado en el simulador de eventos de la primera parte de la práctica. En esta parte, se usan los mismos tipos de eventos anteriormente mencionados.

Sin embargo, al tener varías fuentes que pueden producir eventos se crea un miembro más en la estructura `evento` que identifica a la fuente que ha generado dicho paquete.

El código de esta parte contiene los ficheros mostrados en el Anexo B: `main.c`, `library2.h` y `graficas.m`. Aunque tengan los mismos nombres que los ficheros de la primera parte no contienen el mismo código, pero si que el propósito de cada fichero coincide con los de la primera parte.

El programa `main.c` se puede explicar a grandes rasgos con las líneas mostoradas a continuación, que constituyen los ciclos principales del código de `main.c` que figura en el Anexo B.1:

```
33 for(j=0;j<NMEDIDAS;j++){ // numero de medidas para calcular la media en Matlab
34
35   for (k=1;k<=NFUENTES;k++){ // numero de fuentes
36       pkt_numero = 0;   // Se inicializa el numero de pkts.
37       tiempo_corriendo = TIEMPO_INICIAL;// Se inicializa el tiempo, se actualiza
             tras cada pkt nuevo.
38
39     // Generacion e insercion del evento genesis de cada fuente.
40     for(i = 1; i<=k; i++){ // k es el numero de fuentes que hay.
41       tiempo_primera_llegada = random_poisson(LLEGADAS_FUENTE_LIBRE*1.0);
42       // llegada del primer pkt. 0 + v.a poisson(usando LLEGADAS_FUENTE_LIBRE)
43       tiempo_corriendo = TIEMPO_INICIAL + tiempo_primera_llegada;
44       //i representa la fuente que genera el pkt
45       InsertarEvento(&lista_eventos,CrearEvento(tiempo_corriendo,PKT_IN,i));
46     }
47     //generacion e insercion del evento final. lo genera la fuente 0 (no importa la
           fuente que lo genere).
48     InsertarEvento(&lista_eventos,CrearEvento(TIEMPO_FINAL,PKT_END, 0));
49       puts("LISTA DE EVENTOS:");
50       PintarLista (&lista_eventos);
```

El primer ciclo `for` se corresponde con el número de medidas que se desea realizar para después calcular la media de la probabilidad de pérdida y del porcentaje del tiempo que están todos los recursos ocupados. El siguiente `for` (línea 35), se encarga de ejecutar lo que sería puramente el simulador de eventos, tantas veces como el rando de fuentes que se desee, en este caso, se realizan las medidas desde 1 a 40

fuentes. El último `for` de la línea 40, se encarga de crear todos los paquetes de inicio que se necesiten. Un ejemplo claro de este último ciclo se muestra en la figura 3, referida a la simulación con 4 fuentes en el sistema.

```
LISTA DE EVENTOS:
tiempo 592.837891 tipo 1 fuente 3
tiempo 649.126404 tipo 1 fuente 2
tiempo 878.423706 tipo 1 fuente 1
tiempo 972.993347 tipo 1 fuente 4
tiempo 10000.000000 tipo 0 fuente 0
```

Figura 3: Lista inicial correspondiente a $k = 4$, es decir, con 4 fuentes.

Poniendose en situación, si se detuviera el programa en el momento de la figura 3, el programa estaría a punto de analizar el sistema con 4 fuentes hasta que se llegue al tiempo 10000.00000.

Todas las fuentes han generado un evento inicial y se ha ordenado en la lista según el tiempo en el que han de ser procesados los eventos. Despues el programa seguiría su curso, simulando eventos hasta llegar al tiempo final.

Una vez termine con esta iteración, se procedería a analizar el sistema con 5 fuentes y así sucesivamente hasta llegar a las 40 fuentes (estas iteraciones corresponden al ciclo de la línea 35). Por último, se volverían a ejecutar estos dos ciclos tantas veces como número de medidas se quieran obtener.

Denotar que, la `fuente 0` que aparece en la figura 3 no es una fuente real, sino que se le da un valor 0 ya que ese paquete no ha sido generado por ninguna fuente, sino que es una condición para que finalize el simulador.

Centrándose en la ejecución del simulador, el modo en el que se recorre la lista es similar al de la primera parte, analizando que tipo de evento es el que se situa en primera posición de esta. La diferencia reside en que acciones se realizan con la llegada de cada tipo de evento:

`PKT_IN` Cuando se analiza un evento de este tipo, el primer paso es comprobar si existen recursos disponibles para poder procesarlo.

Si no hay recursos se pierde el paquete, y esa misma fuente, genera un nuevo paquete de tipo `PKT_IN` con la variable aleatoria de Poisson de media la tasa de llegadas por fuente libre sumado al tiempo en el que se ha pérdido el paquete. s Suponiendo que existen recursos disponibles, se resta un recurso de los disponibles ya que se utiliza para procesar este evento y se genera un evento de tipo `PKT_OUT`, con el tiempo que va

a permanecer el paquete en el sistema, calculado usando una variable aleatoria exponencial negativa de media μ (Tasa de muerte) sumado al tiempo en el que se está procesando el paquete de tipo `PKT_IN` que genera el evento de salida.

Una vez generados los dos eventos y por tanto finalizado el procesado del evento, se borra el evento de la lista y por tanto el próximo paso es procesar el siguiente evento.

`PKT_OUT` Para un evento de este tipo siempre hay recursos disponibles ya que el recurso que se utiliza es el que se restó cuando llego un tipo `PKT_IN`. Cuando se procesa un evento de salida, se genera un paquete de tipo `PKT_IN` ya que la fuente se libera.

El paquete que se genera es de tipo `PKT_IN` con un tiempo igual a la suma del tiempo del paquete tipo salida que se está procesando sumado al tiempo calculado mediante la variable aleatoria de Poisson con media la tasa de llegadas por fuente libre.

Una vez procesado, se elimina el evento de salida de la lista.

`PKT_END` Es el último evento que se va a procesar, una vez se procese este evento se finaliza el programa. También se elimina este evento de la lista, por lo que en este punto, en la lista figuran eventos con un tiempo mayor al de este evento y que jamás serán procesados.

3.1. Comprobación del funcionamiento del sistema

Una vez modelado el sistema, se recaban los datos necesarios para comprobar el corroborar el correcto funcionamiento del sistema.

Se analizan dos métricas interesantes del sistema, la probabilidad de pérdida de un paquete para un determinado número de fuentes y el porcentaje del tiempo que se tienen todos los recursos ocupados, en función del tiempo total que está activo el simulador.

3.1.1. Probabilidad de pérdida

Cuando se pierde un paquete, se aumenta en 1 un contador de paquetes pérdidos, que junto con un contador de paquetes generados, ayudan a calcular la probabilidad

de pérdida como la siguiente fórmula:

$$P_{\text{pérdida}} = \frac{\#pkts_{perdidos}}{\#pkts_{totales}} \tag{1}$$

Para comparar los resultados obtenidos en relación a la probabilidad de pérdida, se aprovecha la gráfica de la figura 4, sacada de [1].

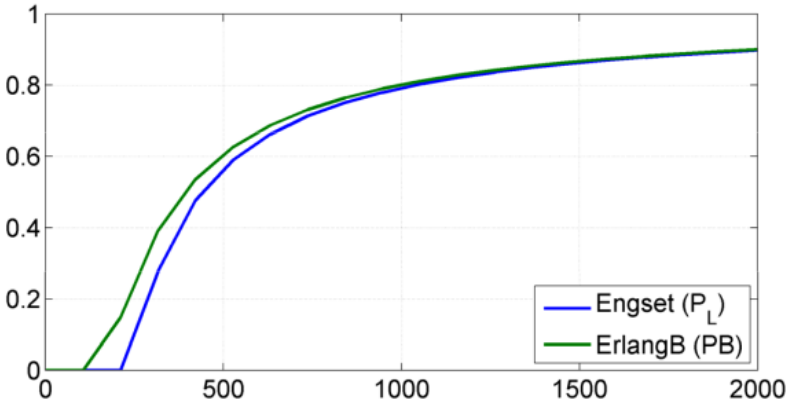

Figura 4: Probabilidad de pérdida teórica. Fuente: [1]

Donde:

a – Se asume que el tráfico por fuente libre es $a = 2$

M – Número de fuentes, $M = 1...2000$

TO – En *ErlangB* el tráfico ofrecido será: $TO = a \times M$

Para estos datos, y programando la fórmula de Engset en *Matlab*, se consigue la gráfica de la figura 5:

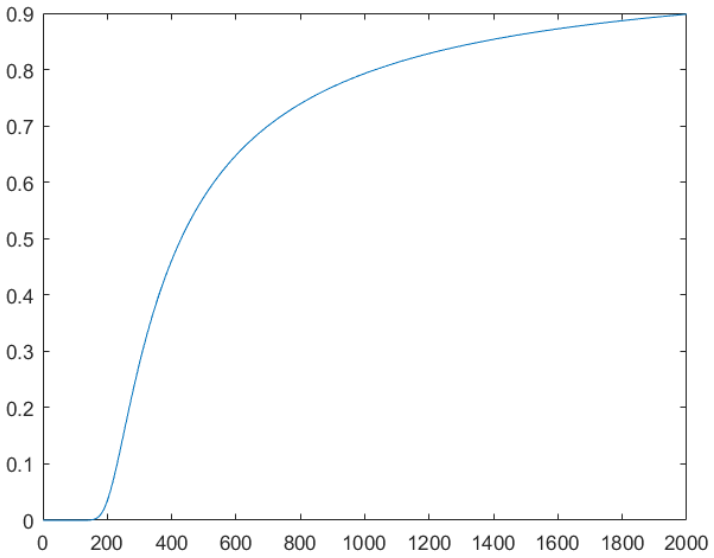

Figura 5: Probabilidad de pérdida Engset en *Matlab*, con 40 recursos ($S = 40$).

Como se puede apreciar aún no estando en la misma escala, las gráficas de las figuras 4 y 5 son similares y por tanto el funcionamiento es óptimo comparado con el teórico. La probabilidad de pérdida para un sistema con 40 recursos es 0 o muy cercana a 0 hasta que como mínimo hay 200 llamadas al sistema.

Visto que la implementación de Engset es correcta, se representa gráficamente para los datos de este proyecto, resultando la gráfica de la figura 6:

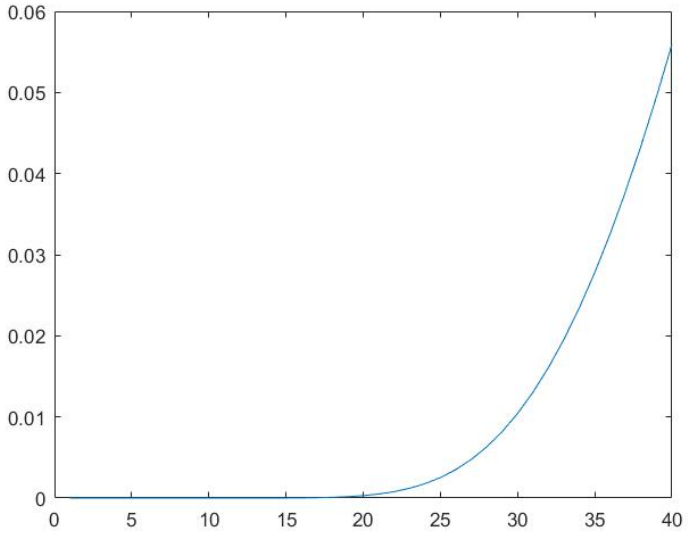

Figura 6: Probabilidad de pérdida, Engset, en función del número de fuentes (de 1 a 40).

La representación de la probabilidad de pérdida conseguida en este proyecto debe asemejarse a la gráfica de la figura 6. Se parte de la gráfica obtenida al calcular la probabilidad de périda, sin realizar ningún tipo de aproximación.

En las siguientes gráficas se muestran en el eje X el número de fuentes del sistema y en el eje Y la probabilidad de pérdida de un paquete. Por ejemplo: si para $M = 37$ se obtiene un valor porcentual de 3, significa que si el sistema cuenta con 37 fuentes, hay una probabilidad del 3 % de que se pierda un páquete.

En gráfica 7 se puede apreciar los valores medios de la probabilidad de pérdida obtenida para cada número de fuentes, realizando 10 medidas.

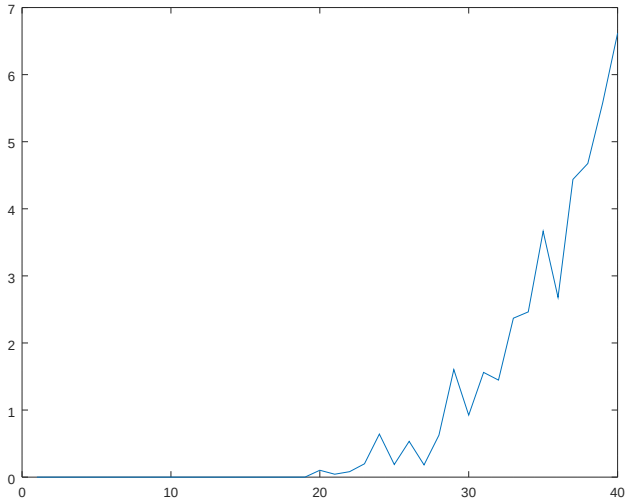

Figura 7: Probabilidad de perdida (%) en función del número de fuentes (1 a 40).

Obtenido este resultado, inmediatamente surge la idea de aproximar la distribución. En la figura 8, se apróxima mediante regresión exponencial, donde cada punto rojo es la probabilidad de pérdida que se obtuvo del simulador para cada número de fuentes.

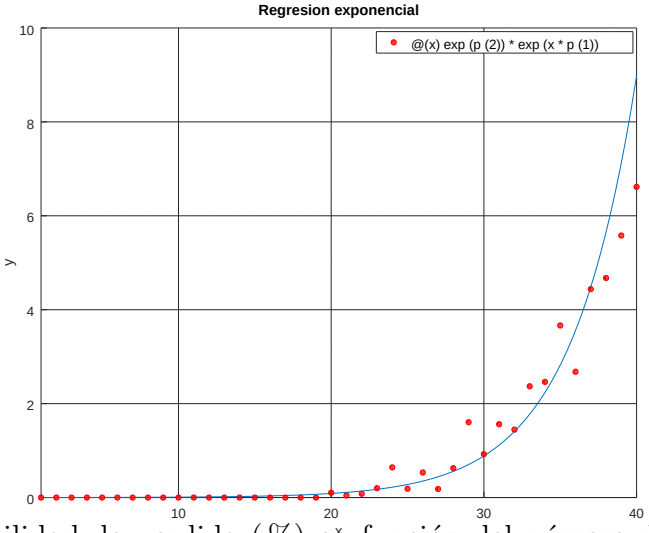

Figura 8: Probabilidad de perdida (%) en función del número de fuentes (1 a 40) aproximada exponencialmente.

Si bien es cierto que mediante la regresión exponencial se consigue una aproximación de la probabilidad de péridda en función del número de fuentes, no es la mejor forma

de aproximarlo. En la figura 8, se observa que el valor de la aproximación cuando se tienen 40 fuentes, es aproximadamente 9 % mientras que el punto rojo (probabilidad de pérdida obtenida en el sistema) tiene un valor orientativo de 6.5 %.

Por este motivo, se aproxima usando varios polinomios mediante el comando conocido de *Octave*, `splinefit`. El resultado obtenido aproximando de esta forma se expone en la gráfica de la figura 9.

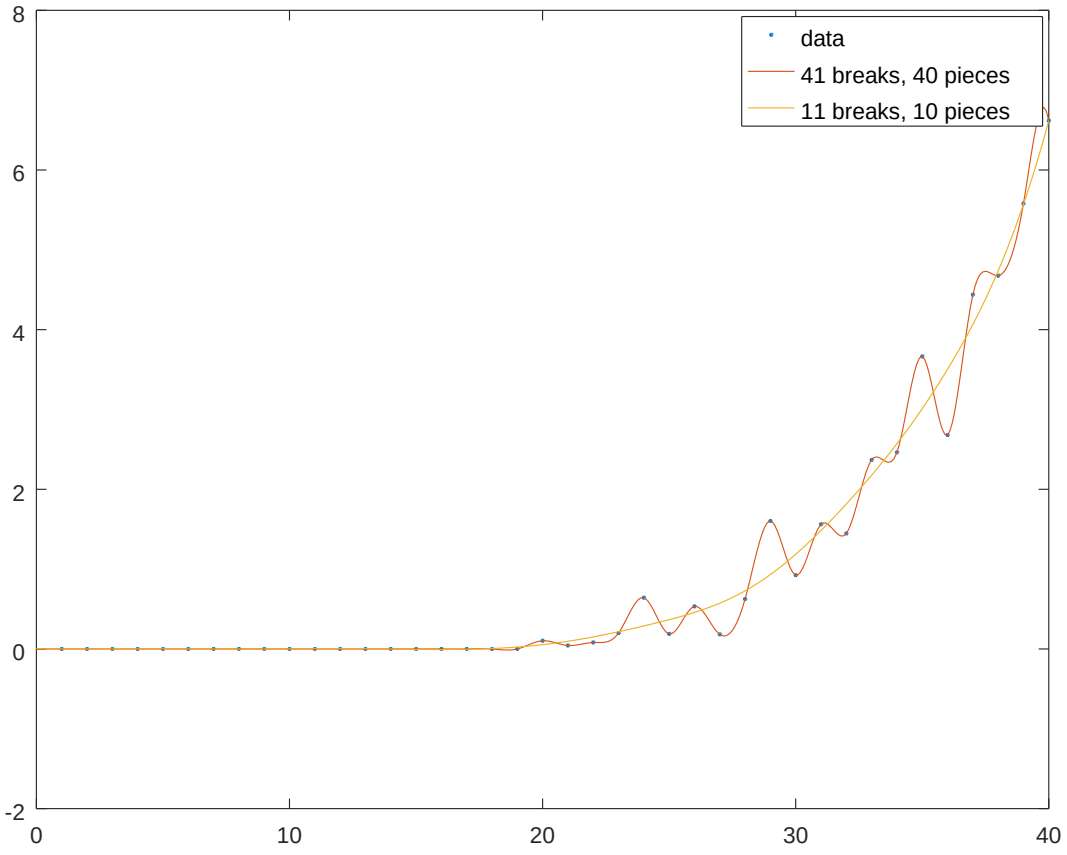

Figura 9: Probabilidad de perdida (%) en función del número de fuentes (1 a 40) aproximada con varios polinomios.

En la figura 9 se ha aproximado de la forma detallada anteriormente. Se ha realizado dos veces esta aproximación, con diferentes parámetros. Por un lado, los puntos azules corresponden directamente con los valores medios obtenidos de la probabilidad de pérdida. Por otro lado, las líneas roja y amarilla son las aproximaciones.

La línea roja corresponde con una aproximación sobreajustada. Esto quiere decir

que se intenta que todos los puntos que se usan para la aproximación esten incluidos dentro de la función de aproximación, lo que causa un sobreajuste y una función de aproximación difícil de modelar de forma sencilla. Además, la aproximación no es fiel puesto que al ir uniendo los puntos, no tienes un valor medio entre ellos.

La línea amarilla, por su parte, corresponde con una aproximación más estática. Aunque no todos los puntos de la probabilidad de pérdida obtenidos con el simulador estén acogidos por la función de aproximación, a la hora de aproximar puntos se conseguiran mejores resultados, con mayor exactitud. Esta sería, la mejor aproximación conseguida.

En ambas funciones, se observa que para 40 fuentes en el sistema la probabilidad de pérdida tiene un valor de 6.5 %, en la línea roja porque se ajusta a todos los puntos y en la amarilla porque es una aproximación mejor que la exponencial.

3.1.2. Ocupación de todos los recursos del sistema

En cuanto al porcentaje del tiempo que están todos los recursos ocupados, se anota el tiempo en el momento que se ocupan todos los recursos hasta que se liberan. Se suman todos estos lapsos de tiempo en los que los recursos se encuentran ocupados y se divide entre el tiempo total de ejecución del programa, resultado la ecuación:

$$\%tiempo(Recursos_{ocupados}) = \frac{\sum_{i=1}^{n} Recursos_{ocupados}(i)}{T_{total}} \tag{2}$$

Donde:

$\boldsymbol{n}$ − Número de veces que han sido ocupados todos los recursos.

$\boldsymbol{T_{total}}$ − Tiempo total de ejecución del programa.

Se implementa la ecuación 2 en *Octave*. Se muestra el resultado obtenido en la figura 10, que a priori si sabe que debería tener forma exponencial.

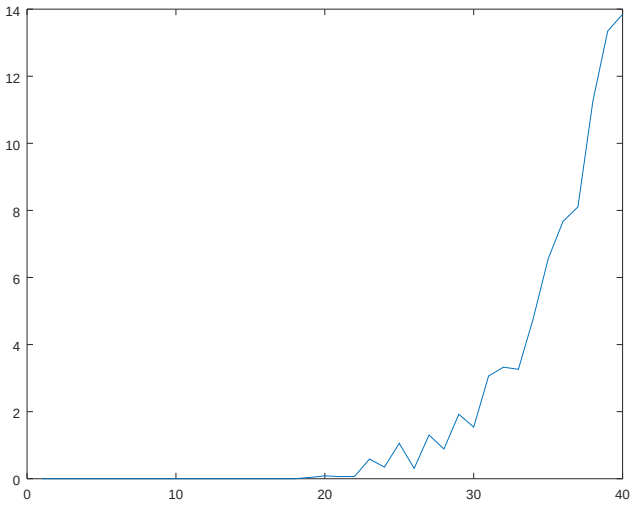

Figura 10: Porcentaje del tiempo que están todos los recursos ocupados en función del número de fuentes del sistema.

Como cabe esperar tiene forma exponencial. La gráfica representa el procentaje del tiempo que todos los recursos están ocupados en función del número de fuentes del sistema. Así para 40 fuentes en el sistema, están todos los recursos ocupados un 14 % del tiempo total.

Al igual que se ha hecho para el caso de la probabilidad de pérdida, se aproximan los datos obtenidos en el simulador en relación al porcentaje del tiempo total que están todos los recursos ocupados mediante el uso de varios polinomios. Se implementa de igual forma en *Octave* y se obtiene el resultado de la figura 11.

En dicha figura (11), los datos obtenidos por el simulador son representados como puntos azules. La línea amarilla constituye una aproximación con varios polinomios estática en comparación con la línea roja, siendo esta una aproximación con varios polinomios sobreajustada.

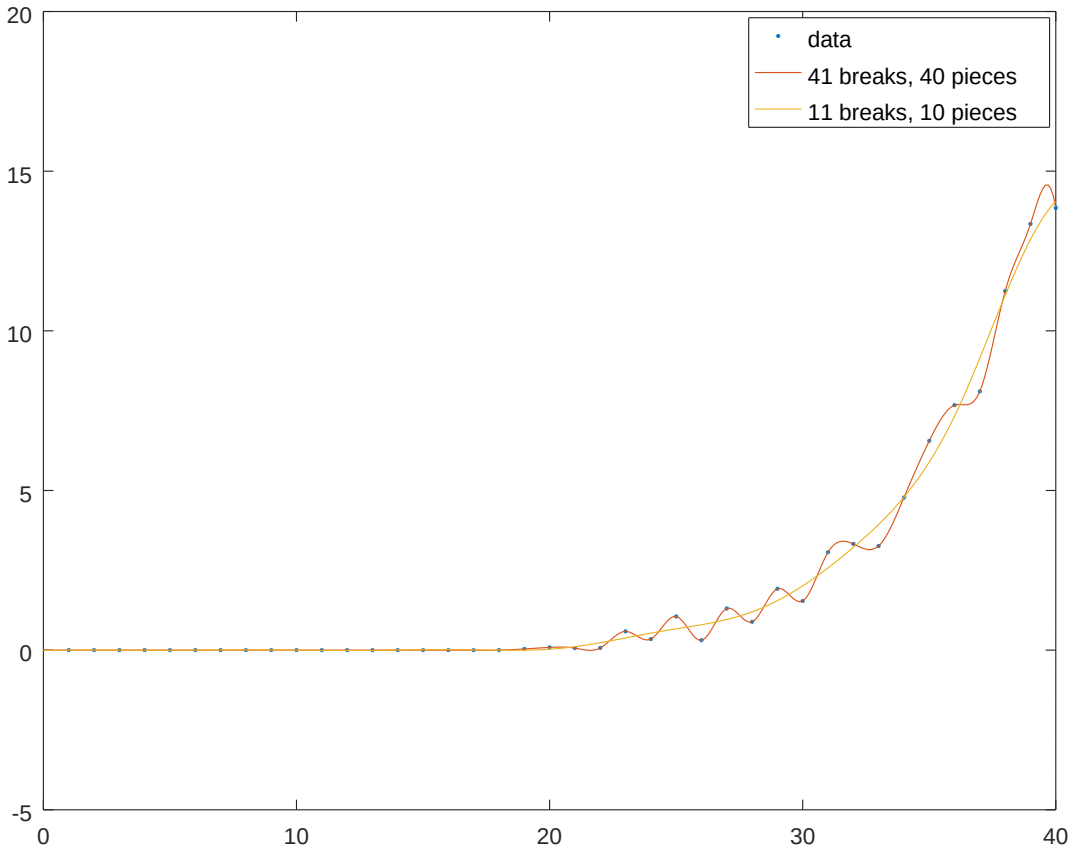

Figura 11: Porcentaje del tiempo que están todos los recursos ocupados en función del número de fuentes del sistema aproximado con varios polinomios.

Se observa en la figura 11 que la línea amarilla constituye la mejor aproximación realizada de los datos obtenidos en el simulador. Refleja que tienen un comportamiento exponencial y que para 40 fuentes en el sistema, se obtiene que un 14 % del tiempo total de ejecución del programa se tienen todos los recursos ocupados. Esto dará lugar a que se produzcan pérdidas de paquetes, que como se ha expresado anteriormente, existen pérdidas y se deben a estar todos los recursos ocupados.

Por tanto, se podrán producir pérdidas en este sistema cuando los recursos estén ocupados.

ANEXOS

A. $M/M/\infty$

El código de la primera parte de la práctica, correspondiente a un sistema $M/M\infty$ está formado por los siguientes archivos:

`main.c` En este archivo se encuentra el código del programa principal.

`library2.h` Librería en la figuran todas las constantes y funciones que usa `main.c`.

`graficas.m` Script de *Matlab* donde se realizan los cálculos necesarios para representar los resultados requeridos en esta primera parte.

A.1. `main.c`

```
1  #include "library2.h"
2
3  int main(int argc, char **argv){
4
5    int pkt_numero, i;
6      float tiempo_primera_llegada, tiempo_corriendo, tiempo, tiemposistema;
7
8    FILE*t_entre_llegadas;  // fichero para almacenar el Tiempo entre llegadas.
9    FILE*t_en_sistema;    // fichero para almacenar el Tiempo en el sistema.
10     t_entre_llegadas = fopen("tllegadas.txt","w");  // Fichero de escritura
11   t_en_sistema = fopen("tsistema.txt","w");
12
13   pkt_numero = 0;   // Se inicializa el numero de pkts.
14   tiempo_corriendo = TIEMPO_INICIAL;// Se inicializa el tiempo, se actualiza tras
         cada pkt nuevo.
15
16   // Generacion del evento genesis:
17   tiempo_primera_llegada = random_poisson(LAMBDA);
18   tiempo_corriendo += tiempo_primera_llegada;
19     InsertarEvento(&lista_eventos,CrearEvento(tiempo_corriendo,PKT_IN)); //
           Creacion e insercion del evento en la lista.
20
21   // Generacion del evento final:
22   InsertarEvento(&lista_eventos,CrearEvento(TIEMPO_FINAL, PKT_END)); // el tiempo
         en el que llega este pkt es 30, marcando el fin de la simulacion.
23
24   // La lista debe contenter el primer y ultimo evento:
25   PrintList (&lista_eventos);
26
27   // Se muestra al usuario que ha llegado un pkt (sin tener en cuenta el final):
28   printf("llega 1 pkt.\n");
29
30   // se pausa el programa esperando interaccion del usuario.
31     getchar();
32
```

```
33    // Bucle infinito usando i como contador del numero de iteraciones que se
          realizan.
34    for (i=1; i>=1; i++){
35
36      // Dependiendo del tipo de pkt se realiza una u otra accion.
37      switch ( lista_eventos->tipo) {
38
39      case PKT_IN:
40
41        // Se escribe el tiempo en el que llega el pkt para conseguir el tiempo entre
              llegadas:
42        fprintf(t_entre_llegadas,"%f \n",lista_eventos -> tiempo);
43
44        //Creacion e insercion de un evento nuevo de llegada:
45        tiempo = (lista_eventos->tiempo) + random_poisson(LAMBDA);
46        InsertarEvento(&lista_eventos,CrearEvento(tiempo, PKT_IN));
47        tiemposistema = random_poisson(MU); // tiempo que va a permanecer en el
              sistema.
48          fprintf(t_en_sistema,"%f \n",tiemposistema); //Se escribe el tiempo que va
                a permanencia del pkt en el sistema.
49
50        // Creacion e insercion de un evento nuevo de salida:
51          tiempo = (lista_eventos->tiempo) + tiemposistema;
52          InsertarEvento(&lista_eventos,CrearEvento(tiempo,PKT_OUT));
53          break;
54
55      case PKT_OUT:
56
57        puts("Paquete Transmitido.");
58        tiemposistema = lista_eventos->tiempo;
59        fprintf(t_en_sistema,"%f \n",tiemposistema); // se escribe en fichero el
              tiempo.
60          break;
61
62      case PKT_END:
63        puts("Programa Finalizado");
64        exit(1);
65        break;
66      }
67      printf("iteracion %i \n",(i));
68      PrintList(&lista_eventos);
69
70      // se elimina de la lista el evento puesto que ya ha sido procesado.
71      BorrarEvento(&lista_eventos,lista_eventos->tiempo);
72    }
73  }
```

A.2. library2.h

```
#ifndef LIBRARY2_H_
#define LIBRARY2_H_
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
// tipos de evento
#define PKT_END 0 // tipo  final
#define PKT_IN 1  // tipo llegada
#define PKT_OUT 2 // tipo salida
// tiempo fijados para la simulacion.
#define TIEMPO_INICIAL 0
#define TIEMPO_FINAL 30
// parametros del grupo
#define LAMBDA 20 // Tasa de llegadas.
#define MU 10     // 1/Ts= 1/(100*10^(-3)) = Tasa muerte o tasa de salida.

typedef struct evento
{
  float tiempo;
  int tipo;
  struct evento *next;  // puntero al siguiente evento.
  } evento;


// VARIABLE GLOBAL
evento *lista_eventos = NULL; //lista_eventos es un puntero a una estructura de
    tipo evento.


// Crear un nuevo elemento de tipo evento en la lista dados los miembros de la
    estructura.
evento *CrearEvento(float tnuevo, int tipo){
        evento *newp;
        newp = (evento *) malloc (sizeof(evento)); // reservar espacio para el
            evento.
        newp -> tiempo = tnuevo;
        newp -> tipo = tipo;
    newp -> next = NULL;
        return newp;
}
/*
Las funciones de insercion y borrado no necesitan devolver un evento *. por eso son
     de tipo void.
head apunta a la variable global lista_eventos. el efecto es que el contenido
de lista_eventos de mantiene actualizado automaticamente.
por tanto no hace falta devolver un puntero  evento* al inicio
de la lista en las funciones de insertar y borrar.
*/

//Insertar elemento nuevo en la lista
/* No se contemplan entradas con el mismo tiempo, se supone
el tiempo continuo y por tanto diferente para cada evento.*/
void InsertarEvento(evento **head, evento *newp)
//head: puntero a puntero del inicio de la lista
// preparamos *newp evento que se insertara
{
  evento **tracer = head; // tracer apunta al puntero de inicio de la lista.
```

```
55    //while(*tracer !=NULL) es lo mismo que while(*tracer)
56    while((*tracer) && (*tracer)->tiempo < (*newp).tiempo){
57      tracer = &(*tracer)->next;
58    }
59    newp -> next = *tracer;
60    *tracer = newp;
61  }
62
63  // Borrar el elemento de la lista que coincida con el tiempo que se desee
64  // si no existe un evento con ese tiempo, no ocurre nada.
65  void BorrarEvento(evento **head, float tborrar)
66  {
67    evento *old;
68    evento **tracer = head;
69
70    while((*tracer) && !(tborrar == (*tracer)->tiempo))
71      tracer = &(*tracer)->next;
72
73    if(tborrar == (*tracer)->tiempo)
74    {
75      old = *tracer;
76      *tracer = (*tracer)->next;
77      free(old); // Liberar el espacio del evento
78    }
79  }
80
81  // Pintar La lista
82  void PrintList(evento **head)
83  {
84    evento **tracer = head;
85    while (*tracer) {
86      printf("tiempo %f tipo %i \n",(*tracer)->tiempo,(*tracer)->tipo);
87      tracer = &(*tracer)->next;
88    }
89  }
90
91  // Exponencial negativa
92  double random_poisson(float lambda){
93      double x, poisson;
94      int i;
95      time_t t;
96      srand((unsigned)time(&t));
97          x =(double)(rand()%(int)100);
98          x = x/100; // hasta aqui es una variable aleatoria uniforme entre 0 y 1.
99          poisson = (-(1/lambda)*log(x))*1;
100     return poisson;
101 }
102 #endif
```

A.3. graficas.m

```
clear all
close all
clc

N=20; %Puntos para el histograma
% Importamos el archivo que contiene los tiempos en que llega el paquete al
% sistema
A = importdata('tllegadas.txt');

for i=1:length(A)-2
    dif(i)=abs(A(i)-A(i+1));    % Diferencia entre cada llegada para calcular el
        tiempo entre llegadas
end
subplot(1,2,1)
hist(dif,N);
grid on
title('Tiempo entre llegadas')

% Importamos los tiempos que estan los paquetes en el sistema
B = importdata('tsistema.txt');

subplot(1,2,2)
hist(B,N)
title('Tiempo en el sistema')
```

B. Sistema de pérdida pura con fuentes finitas

El código de la segunda parte de la práctica, correspondiente a un sistema de périda pura con fuente finitas está formado por los siguientes archivos:

`main.c` En este archivo se encuentra el código del programa principal.

`library2.h` Librería en la figuran todas las constantes y funciones que usa `main.c`.

`graficas.m` Script de *Octave* donde se realizan los cálculos necesarios para representar los resultados relevantes en esta segunda parte de la práctica.

B.1. `main.c`

```
1  #include "library2.h"
2
3  // Exponencial negativa.
4  double random_poisson(double lambda){
5      double x, poisson;
6      int i;
7      for (i=0;i<5;i++){
8          x =(double)(rand()*1.0/RAND_MAX);
9          poisson = (-(1/lambda)*log(x))*1;
10         //printf ("%lf",x);
11     }
12   return poisson;
13 }
14
15
16 int main(int argc, char **argv){
17
18     int pkt_numero, i, recursos, fuente,k,x,cont,j;
19   float tiempo_primera_llegada, tiempollegadas, tiemposistema,tiempo2,tiempo3,
         p_perdida,pkts_perdidos,pkts_generados, tiempo_corriendo;
20
21   FILE*t_entre_llegadas;  // fichero para el Tiempo entre llegadas.
22   FILE*t_en_sistema;    // fichero para el Tiempo en el sistema.
23   FILE*fich_perdida;    // fichero para la probabilidad de perdida.
24   FILE*recursos_ocupados; // Fichero para los recursos ocupados con cada fuente.
25   t_entre_llegadas = fopen("tllegadas.txt","w");
26   t_en_sistema = fopen("tsistema.txt","w");
27   fich_perdida = fopen("prob_perdida.txt","w");
28   recursos_ocupados = fopen("re_ocu.txt","w");
29
30   time_t t;
31   srand((unsigned) time(&t));
32
33   for(j=0;j<NMEDIDAS;j++){ // numero de medidas para calcular la media en Matlab
34
35     for (k=1;k<=NFUENTES;k++){ // numero de fuentes
36         pkt_numero = 0;   // Se inicializa el numero de pkts.
37         tiempo_corriendo = TIEMPO_INICIAL;// Se inicializa el tiempo, se actualiza
               tras cada pkt nuevo.
```

```
38
39        // Generacion e insercion del evento genesis de cada fuente.
40        for(i = 1; i<=k; i++){ // k es el numero de fuentes que hay.
41          tiempo_primera_llegada = random_poisson(LLEGADAS_FUENTE_LIBRE*1.0);
42          // llegada del primer pkt. 0 + v.a poisson(usando LLEGADAS_FUENTE_LIBRE)
43          tiempo_corriendo = TIEMPO_INICIAL + tiempo_primera_llegada;
44          //i representa la fuente que genera el pkt
45          InsertarEvento(&lista_eventos,CrearEvento(tiempo_corriendo,PKT_IN,i));
46        }
47        //generacion e insercion del evento final. lo genera la fuente 0 (no importa
              la fuente que lo genere).
48        InsertarEvento(&lista_eventos,CrearEvento(TIEMPO_FINAL,PKT_END, 0));
49        //puts("LISTA DE EVENTOS:");
50        //PintarLista (&lista_eventos);
51        printf("\n Ha/n llegado %i pkts.\n",k);
52        //getchar(); // pausa el programa hasta que el usuario pulse una tecla.
53        // comentar la linea anterior para mayor rapidez.
54
55        /*
56        Tras la generacion del evento inicial y final
57        se entra en un bucle del que se sale al llegar al evento final
58        */
59        pkts_perdidos = 0; // de momento no hemos perdido ningun pkt.
60        pkts_generados = k; // hemos generado el primer pkt de cada fuente, el ultimo
               no le contamos porque se analiza difernete. (pkt_end)
61        recursos = RECURSOS_TOTALES;
62        x=0; // variable auxiliar para salir de while
63        cont = 0;
64
65        while (x!=10){ //while hasta que llegue un pkt tipo PKT_END y se acabe.
66          cont++;
67
68          if(recursos!=0){ // if para guardar en fichero si todos los recursos estan
                ocupados
69            fprintf(recursos_ocupados,"%f 1 \n",lista_eventos->tiempo); //Si hay
                  recursos guardamos el tiempo y un 1
70          }else{
71            fprintf(recursos_ocupados,"%f 0 \n",lista_eventos->tiempo);//Si no hay
                  recursos guardamos el tiempo y un 0
72          }
73          switch (lista_eventos->tipo) {
74
75          case PKT_IN:
76            pkts_generados++; // Se genera un pkt mas
77            if(recursos!=0){
78              recursos--; // Se tiene un recurso disponible menos
79              tiemposistema = random_poisson(MU*1.0);
80              fprintf(t_en_sistema,"%f \n",tiemposistema);  //Guardamos el tiempo que
                    va a permanecer el paquete en el sistema
81              //Crear evento de Tx del paquete que llega
82              tiempo3 = lista_eventos->tiempo + tiemposistema;
83              InsertarEvento(&lista_eventos,CrearEvento(tiempo3,PKT_OUT,lista_eventos
                    ->fuente));
84            } else{ // Si no hay recursos disponibles
85              pkts_perdidos++; // se pierde el pkt
86              tiemposistema = random_poisson(LLEGADAS_FUENTE_LIBRE); // la fuente
                    perdio el pkt y generara otro cuando toque.
87              tiempo3 = lista_eventos->tiempo + tiemposistema;
88              InsertarEvento(&lista_eventos,CrearEvento(tiempo3,PKT_IN,lista_eventos
                    ->fuente));
89              //printf("se ha perdido 1 pkt, en total: %f pkts perdidos\n",
```

```
                pkts_perdidos);
90            }
91            break;
92          case PKT_OUT:
93            recursos++;
94            fuente = lista_eventos->fuente;
95            fprintf(t_entre_llegadas,"%f \n",lista_eventos->tiempo);
96            // Tiempo de paquete que llega para calcular el tiempo entre llegadas.
97            // Creacion e insercion  de un evento nuevo de llegada
98            tiempo2 = lista_eventos->tiempo + random_poisson(LLEGADAS_FUENTE_LIBRE);
99            InsertarEvento(&lista_eventos,CrearEvento(tiempo2,PKT_IN,fuente));
100           //printf("Paquete Transmitido\n");
101           break;
102         case PKT_END:
103           p_perdida = (pkts_perdidos/pkts_generados)*100;
104           fprintf(fich_perdida,"%f\n",p_perdida); //Guardamos el tiempo que va a
                  permanecer el paquete en el sistema
105           x = 10; // variable auxiliar para final el while.
106           break;
107         }
108         //Avanza al siguiente evento de la lista
109         //PintarLista(&lista_eventos);
110         BorrarEvento(&lista_eventos,lista_eventos->tiempo);
111       }
112       printf("Sistema analizado con %i fuente/s, se borra la lista de eventos.\n\n"
              ,k);
113       BorrarLista(&lista_eventos);
114     }
115     printf("Finalizada la medida %i \n",j+1);
116   }
117   puts("Programa Finalizado");
118 }
```

B.2. library2.h

```
1  #ifndef LIBRARY2_H_
2  #define LIBRARY2_H_
3  #include <stdio.h>
4  #include <math.h>
5  #include <stdlib.h>
6  #include <time.h>
7  // Tipos de evento
8  #define PKT_END 0 // tipo  final
9  #define PKT_IN 1  // tipo llegada
10 #define PKT_OUT 2 // tipo salida
11 // Tiempo fijados para la simulacion.
12 #define TIEMPO_INICIAL 0
13 #define TIEMPO_FINAL 10000
14 // Parametros del grupo 2
15 #define MU 0.0055555  // 1/Ts= 1/(180seg) = 0.0055555 (PKTS/SEG)
16 #define RECURSOS_TOTALES  10 //engset 40
17 #define LLEGADAS_FUENTE_LIBRE 0.001111111 // engset 0.00166667 TASA LLEGADAS POR
       FUENTE LIBRE  (PKTS/SEGUNDO)
18 #define NFUENTES  40
19 #define NMEDIDAS  10 //   Numero de medidas que se quieren realizar para calcular
       la media.
20
21 // Ejemplo Tema 5 DPR:
22 /*
23 a =   lambda        *  Ts
24 a = llegadas fuente libre * tiempo servicio
25    0.1 (llamadas/minuto)* 2min
26 */
27 typedef struct evento
28 {
29   float tiempo;
30   int tipo;
31   int fuente;
32   struct evento *next; // puntero al siguiente evento.
33 } evento;
34
35 // VARIABLE GLOBAL
36 evento *lista_eventos = NULL; //lista_eventos es un puntero a una estructura de
       tipo evento.
37
38
39 // Crear un nuevo elemento de tipo evento en la lista dados los miembros de la
       estructura.
40 evento *CrearEvento(float tnuevo, int tipo, int fuente){
41         evento *newp;
42         newp = (evento *) malloc (sizeof(evento)); // reservar espacio para el
               evento.
43         newp -> tiempo = tnuevo;
44         newp -> tipo = tipo;
45     newp -> fuente = fuente;
46     newp -> next = NULL;
47         return newp;
48 }
49
50 /*
51 para las funciones de insercion y borrado no se necesita devolver un evento *, no
       es necesario.
52 por eso son de tipo void.
```

```
53
54  head apunta a la variable global lista_eventos. el efecto es que el contenido
55  de lista_eventos de mantiene actualizado automaticamente.
56  por tanto no hace falta devolver un puntero  evento* al inicio
57  de la lista en las funciones de insertar y borrar.
58  */
59
60
61  //Insertar elemento nuevo en la lista
62  /* No se contemplan entradas con el mismo tiempo, se supone
63  el tiempo continuo y por tanto diferente para cada evento.*/
64  void InsertarEvento(evento **head, evento *newp)
65  //head: puntero a puntero del inicio de la lista
66  // preparamos *newp evento que se insertara
67  {
68    evento **tracer = head; // tracer apunta al puntero de inicio de la lista.
69    //while(*tracer !=NULL) ES LO MISMO QUE while(*tracer)
70    while((*tracer) && (*tracer)->tiempo < (*newp).tiempo){
71      tracer = &(*tracer)->next;
72    }
73    newp -> next = *tracer;
74    *tracer = newp;
75  }
76
77  // Borrar el elemento de la lista que coincida con el tiempo que se desee
78  // si no existe un evento con ese tiempo, no pasa nada. Se puede poner un print o
        devolver un error...
79  void BorrarEvento(evento **head, float tborrar)
80  {
81    evento *old;
82    evento **tracer = head;
83
84    while((*tracer) && !(tborrar == (*tracer)->tiempo))
85      tracer = &(*tracer)->next;
86
87    if(tborrar == (*tracer)->tiempo)
88    {
89      old = *tracer;
90      *tracer = (*tracer)->next;
91      free(old); // Liberar el espacio del evento
92    }
93  }
94
95  // Pintar la Lista.
96  void PintarLista(evento **head)
97  {
98    evento **tracer = head;
99    while (*tracer) {
100     printf("tiempo %f tipo %i fuente %i \n",(*tracer)->tiempo,(*tracer)->tipo,(*
            tracer)->fuente);
101     tracer = &(*tracer)->next;
102   }
103 }
104
105 // Borrar la lista, destruirla.
106 void BorrarLista(evento **head) {
107   evento *current = *head;
108   evento* next;
109
110     while (current != NULL)
111     {
```

```
112         next = current->next;
113         free(current);
114         current = next;
115     }
116     *head = NULL;
117 }
118
119
120 #endif
```

B.3. graficas.m

```
clear all
close all

%% se carga el fichero con las probabilidades de perdida
filename = 'prob_perdida.txt';
[A,delimiterOut]=importdata(filename);

% Parametros del grupo
M = 40; %numero de fuentes
n = 10; % numero de medidas, cantidad de veces que se ha realizado el experimento
a=1;    % inicializacion

for i=1:M:M*n
  B(:,a)=A(i:i+(M-1));
  a=a+1;
 end

for i = 1:M
  prob_medias(i) = mean(B(i,:));
end

% grafica prob perdida
plot(prob_medias);

x=[1:40]; % fuentes
y=prob_medias;
figure(1)
plot(x,y)
figure(2)
%% aproximacion con varios polinomios (piecewise polynomial).
breaks = linspace (0, 40, 41);
pp1 = splinefit (x, y, breaks);
pp2 = splinefit (x, y, 10);
xx = linspace (0, 40, 400);
y1 = ppval (pp1, xx);
y2 = ppval (pp2, xx);
%plot (x, y,'ro','markersize',4,'markerfacecolor','r', xx, [y1; y2])
plot (x, y,'.', xx, [y1; y2])
% para ver mejor las aproximaciones, se pone un "."
% en cada punto, en vez de circulos rellenos.
axis tight
ylim auto
legend ({"data", "41 breaks, 40 pieces", "11 breaks, 10 pieces"})

x=[1:40];
y=prob_medias(20:end);
x=x(20:end);
p=polyfit(x,log(y),1);
x=[1:40];
y =prob_medias;
fprintf('exponente a= %2.3f\n',p(1));
fprintf('coeficiente c = %3.3f\n',exp(p(2)));
figure(3)
hold on
plot(x,y,'ro','markersize',4,'markerfacecolor','r')
z=@(x) exp(p(2))*exp(x*p(1)); % aproximacion con una exponencial
fplot(z,[x(1),x(end)])
xlabel('x')
```

```
ylabel('y')
grid on
title('Regresion exponencial')
hold off


%% se carga el fichero.
data=importdata('re_ocu.txt');
tiempo=data(:,1);
estado=data(:,2);
t_total=10000; % tiempo total.
ocupado=0;
tiempoocupado=0;
itera=0;
j=0;
for k=1:length(tiempo)
    if tiempo(k)==t_total
        itera=itera+1;
        porocu=(tiempoocupado/t_total)*100; %porcentaje del tiempo ocupado
        tiempos(itera)=porocu;
        tiempoocupado=0;
    end

    if estado(k)==0;% Ocupado
        tiempoini=tiempo(k);
        j=0;
        while j~=1
            %Problema cuando el que esta ocupado es el ultimo
            k=k+1;
            j=estado(k); %comprobamos si se ha liberado

            if tiempo(k)<tiempo(k-1)
             tiempoliberacion=tiempo(k-1); %Tiempo en que se ha liberado el recurso
             j=1;
             k=k-1;
            else
            tiempoliberacion=tiempo(k); %Tiempo en que se ha liberado el recurso
            end
        end
        tiempoocupado=(tiempoliberacion-tiempoini)+tiempoocupado;
    end
end


%% se recuerdan los valores del grupo:
M = 40; %numero de fuentes
n = 10; % numero de medidas, cantidad de veces que se ha realizado el experimento
A=tiempos;
total=zeros(1,40);
for g=0:9
    total=tiempos(((g*40)+1):((g*40)+40))+total;

end
total=total/10;
N=1:40;
figure(4)
plot(N,total)
%xlabel('Numero de Fuentes');
%ylabel('% Tiempo con todos los recursos ocupados')


```

```
120
121 y=total;
122 x=1:40;
123 figure(5)
124 breaks = linspace (0, 40, 41);
125 pp1 = splinefit (x, y, breaks);
126 pp2 = splinefit (x, y, 10);
127 xx = linspace (0, 40, 400); % 10 medidas x 40 fuentes
128 y1 = ppval (pp1, xx);
129 y2 = ppval (pp2, xx);
130 plot (x, y,'.', xx, [y1; y2])
131 % para ver mejor las aproximaciones, se pone un "."
132 % en cada punto, en vez de circulos rellenos.
133 axis tight
134 ylim auto
135 legend ({"data", "41 breaks, 40 pieces", "11 breaks, 10 pieces"})
```

Referencias

[1] R. Agüero. *Dimensionado y Planificación de Redes: TEMA 5 - Análisis de Técnicas de Acceso al Medio*. Grupo de Ingeniería Telemática, Universidad de Cantabria, 2016.

[2] M. Betegón. Event Simulator for Communitacion Networks. *https://github.com/betegon/Event-Simulator-for-Communication-Networks*, 2018.