Práctica 3. Simulación de Sistemas

Objetivos generales

En esta práctica se busca tomar contacto con el modo de funcionamiento interno de muchos simuladores mediante el desarrollo de dos simuladores que permitan de paso verificar resultados presentados en la teoría del tema.

El desarrollo puede servir en otros temas al evaluar sistemas, como por ejemplo servidores de aplicaciones, cuyo comportamiento se asemeje al modelo presentado.

Objetivos específicos

Los resultados esperados en esta práctica son 3:

- Un simulador para un sistema G/G/n/c cuyo tiempo de servicio venga definido por un parámetro de la unidad de servicio.
- Un simulador más complejo que emplee el simulador anterior para simular un escenario de servicio de video bajo demanda.
- Una memoria en la que se describan los métodos empleados para testear y validar los simuladores anteriores.

En caso de necesitar cambiar los formatos de entrada y/o salida de los programas se debe comentar la circunstancia con el profesor para obtener aprobación.

Pautas de desarrollo

A continuación se dan una serie de pautas que deberían aplicar en caso de querer desarrollar algo profesional. **No es obligatorio emplearlas para el desarrollo de esta práctica y se dejan como algo opcional por si se quiere ahondar más en el tema.**

Se podrán emplear los lenguajes de programación que se deseen siempre que el software se pueda probar en el laboratorio a partir del código fuente. Dicho esto se recomienda emplear Java debido a que:

- Hay herramientas IDE preinstaladas en el laboratorio (Eclipse, Netbeans).
- Se dispone de una herramienta de testeo unitario (JUnit) que permite ir comprobando partes del código de forma automatizada.
- Hay mucha documentación disponible y librerías que pueden facilitarnos la tarea.
- Es un lenguaje relativamente sencillo y fácil de utilizar.

Considerar lo siguiente:

- Leer y comprender toda la práctica. Algunas partes del desarrollo del objetivo 1 se deben reutilizar para el objetivo 2.
- No esperéis a testear y validar el sistema hasta el final. Ir testeando las funciones, clases y módulos de forma **incremental**.
- Emplear algún método para testear y validar de forma automática. Por ejemplo con JUnit si trabajáis con Java. **Se ahorra mucho tiempo.**

Objetivo 1. Simulador G/G/n/c (30%)

Se debe desarrollar un simulador de un sistema CPD con m servidores y cola con capacidad para c-n clientes, de forma que se puedan obtener resultados para un sistema G/G/n/c (Figura 1). El programa se puede dividir en tres módulos: la fuente que genera las llegadas partiendo del fichero de entrada, el propio CPD, y el sumidero que puede ser empleado para recoger estadísticas de las salidas del sistema y escribirlas al fichero de salida.

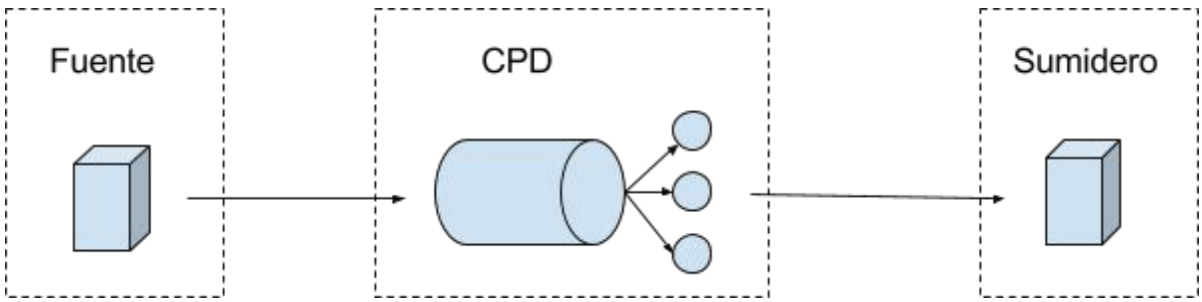

Figura 1. CPD a simular para el objetivo 1.

La llamada al programa se realizará de la siguiente forma:

```
% simula <fichero de entrada> <procesadores> <espacio cola>
```

El simulador obtendrá los tiempos entre llegadas, así como los tiempos de servicio, del fichero que se le pasa como argumento. Cada una de las líneas del fichero contendrá dos valores

```
<instante de peticion> <tiempo de servicio>
```

Donde:

- **instante de peticion** es el instante de tiempo absoluto en que la petición llega
- **tiempo de servicio** será el tiempo de servicio de la misma en el servidor (duración de la película, esto viene con la propia llegada)

El programa deberá retornar una línea para cada una de las llegadas. Cada línea tendrá los siguientes parámetros, separados por un espacio:

```
<instante de petición> <tiempo de servicio> <rechazada>
```

Donde:

- **instante de petición** (viene de la petición)
- **tiempo de servicio** (viene de la petición)
- **rechazada** valdrá 0 si la petición se ha cursado con éxito. Valdrá 1 si la petición ha sido rechazada

No es necesario que el fichero esté ordenado por la primera columna.

El programa continuará hasta que no queden más llegadas y el sistema se quede completamente vacío.

Un ejemplo de ejecución, con 1 procesador y 4 espacios en la cola, sería el siguiente:

```
% simula entrada.txt 1 4
0.057903 0.0 0
0.086235 0.0 0
0.611401 3.0 1
0.235151 1.0 0
0.268361 2.0 0
0.455198 3.0 0
0.508149 4.0 0
0.516904 4.0 0
```

donde el archivo entrada.txt podría contener lo siguiente:

```
0.057903 0.0
0.086235 0.0
0.235151 1.0
0.268361 2.0
0.455198 3.0
0.508149 4.0
0.516904 4.0
0.611401 3.0
```

Pautas de desarrollo

Internamente el programa tiene que manejar una estructura para almacenar los datos relativos a la unidad de servicio. Se recomienda que al menos que se consideren los siguientes parámetros de modo que este simulador pueda ser reutilizado para el desarrollo del segundo simulador (objetivo 2):

- instante de petición

- tiempo de servicio
- tipo de usuario: “premium” o “basico”. Para el segundo objetivo.
- reentrada. Para el segundo objetivo

La clase que contiene la unidad de servicio sería algo así como lo siguiente:

```
public class UnidadServicio {
	public final double llegada;
	public final double tiempoServicio;
	public final String tipo;
	public final boolean reentrada;

	otros datos útiles...
}
```

Se recomienda separar la fuente, el servidor y el sumidero en clases o módulos separados. Para conectarlos emplear alguna interfaz que permita conectarlos a posteriori. Por ejemplo:

```
public interface Entrada {
	void enviar(UnidadServicio unidad);
}

public interface Salida {
	void connectar(Entrada entrada);
}
```

De esta forma el módulo con la fuente, el del CPD y el del sumidero podrían implementarse de forma independiente y conectarlos a posteriori:

```
public class Fuente {
	public final Salida salida = new Salida();

	resto de implementación...
}

public class CPD {
	public final Entrada entrada = new EntradaCPD();
	public final Salida salida = new Salida();
	public final Salida salidaRechazado = new Salida();

	resto de implementacion...
}
```

```
public class Sumidero {
      public final Entrada entrada = new EntradaSumidero();

      resto de implementación
}

public static void main(String args[]) {
      FEL fel = new FEL();

      Fuente fuente = new Fuente(fel, args[0],...);
      CPD cpd = new CPD(fel,...);
      Sumidero sumidero = new Sumidero(fel, System.out,...);

      fuente.salida.conectar(cpd.entrada);
      cpd.salida.conectar(sumidero.entrada);

      fel.run();
}
```

Para validar los resultados que da el simulador se recomienda emplear simulaciones tipo M/M/n/c empleando tests automatizados.

```
class Validador {
      void test() {
            FEL fel = new FEL();

            Fuente fuente = new FuenteExponencial(fel,...);
            CPD cpd = new CPD(fel,...);
            Sumidero sumidero = new SumideroContador(fel,...);
            Sumidero sumideroRechazada
                  = new SumideroContador(fel,...);

            fuente.salida.conectar(cpd.entrada);
            cpd.salida.conectar(sumidero.entrada);

      cpd.salidaRechazada.conectar(sumideroRechazada.entrada);

            fel.run();

            assertEquals(sumidero.estadisticos[0],  0.123,  0.05);
// valor esperado y error
            assertEquals(sumidero.estadisticos[1], 0.582, 0.05);
            assertEquals(sumidero.estadisticos[2], 0.024, 0.05);
      }
}
```

Objetivo 2. Simulador de un CPD de vídeo bajo demanda más completo (30%)

El simulador a desarrollar en el objetivo 2 es un CPD más completo (Figura 2). Tiene varias características avanzadas, como tipos de usuarios diferentes (“premium” y “basicos”), reentradas de usuarios, y balanceo de carga, las cuales son habituales en sistemas reales.

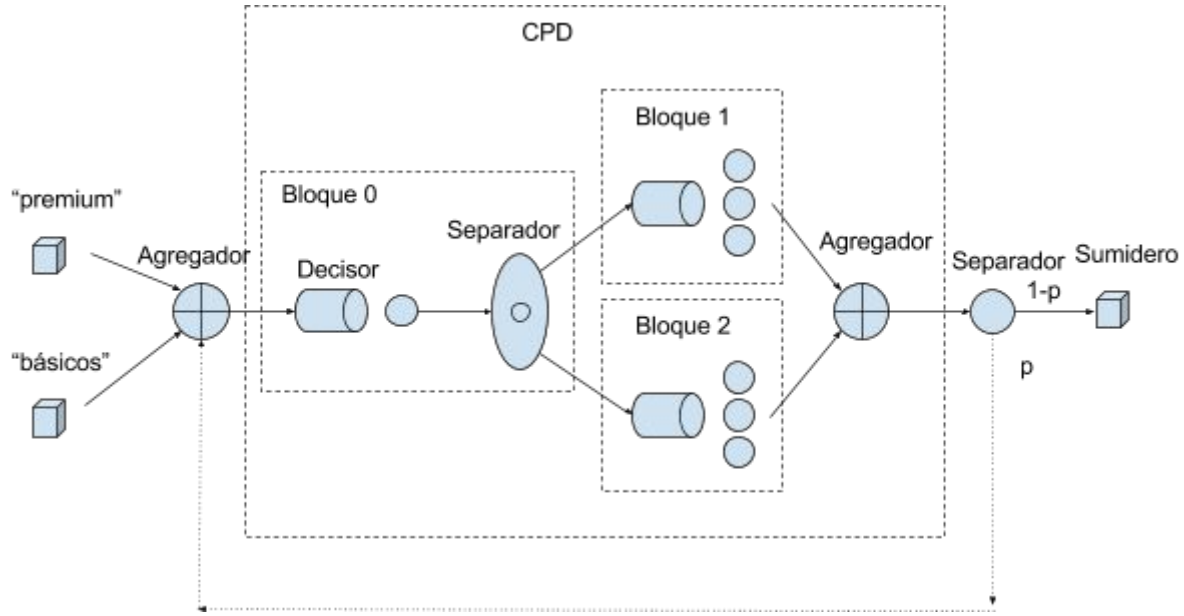

Figura 2. CPD a simular para el objetivo 2.

Llegan 2 tipos de peticiones:

- Peticiones de alta prioridad o peticiones de usuarios "premium"
- Peticiones de baja prioridad o peticiones de usuarios "básicos"

Las peticiones de cada tipo vienen de una fuente diferente que representan las poblaciones de usuarios. Cada petición tiene un tiempo de servicio que es la duración de la película solicitada.

Cada una de las fuentes tiene un fichero de entrada. Cada fichero debe contener líneas de texto, cada una para una llegada, con 2 datos en columnas separadas por un espacio: instante de llegada y duración del servicio que pide (la duración del vídeo). Las llegadas están ordenadas en cada fichero por instante de llegada y ambas columnas miden en la misma unidad de tiempo. En caso de no estar ordenadas el programa puede dar un error o comportarse de forma impredecible.

La llamada al programa simulador se tiene que poder realizar de la siguiente forma:

```
% simula <ficheroLlegadasPremium.txt> <ficheroLlegadasBasicos.txt>
<tiempoServicioBloque0> <maxColaBloque0> <numServidoresCPD1>
<maxColaCPD1> <numServidoresCPD2> <maxColaCPD2>
<parametroDeRetorno> [<instantesDeSalida.txt>]
```

Los parámetros son los siguientes:

- **ficheroLlegadasPremium.txt:** nombre del fichero de llegadas “premium”: Un fichero de llegadas de usuarios de tipo premium con el mismo formato que para el simulador del objetivo 1.
- **ficheroLlegadasBasico.txt:** nombre del fichero de llegadas “basicos”: Un fichero de llegadas de usuarios de tipo básico con el mismo formato que el simulador del objetivo 1.
- **tiempoServicioBloque0**: tiempo de servicio constante en el bloque 0
- **maxColaBloque0**: máximo número de peticiones que puede haber en la cola del bloque 0 (no cuenta la petición en servicio, así que puede valer 0, lo cual significaría que no hay posibilidad de quedarse a la espera en cola)
- **numServidoresCPD1**: número de procesadores en el CPD1.
- **maxColaCPD1**: número de posiciones en la cola del CPD1.
- **numServidoresCPD2:** número de procesadores en el CPD2.
- **maxColaCPD2:** número de posiciones en la cola del CPD2.
- **parametroDeRetorno:** probabilidad de que una petición completada se convierta en una nueva petición en ese mismo instante, de ese mismo tipo y de mitad de duración
- **instantesDeSalida.txt:** nombre de fichero que el programa creará y en el cual guardará las estadísticas de la simulación.

El parámetro instantesDeSalida es opcional (no de implementarse sino de ponerse en una ejecución). En caso de estar, debe contener un nombre de fichero que el programa creará y en el cual guardará una línea de texto por cada petición con las siguientes columnas:

```
<instantePeticion> <tipoUsuario> <retornada> <tiempoServicio>
<CPD> <rechazada>
```

- **instantePeticion:** es el instante de tiempo absoluto en que la petición llega al bloque 0. No es necesarios que el fichero esté ordenado por la primera columna
- **tipoUsuario:** valdrá “premium” o “basico”
- **retornada:** valdrá 1 si esta petición proviene de una petición anterior completada y que ha retornado y en caso contrario valdrá 0
- **tiempoServicio:** será el tiempo de servicio de la misma en el CPD (duración de la película, esto viene con la propia llegada)

- **CPD:** valdrá 1 ó 2 según qué CPD haya atendido esta petición
- **rechazada:** valdrá 0 si la petición se ha cursado con éxito. Valdrá 1 si la petición ha sido rechazada en el bloque 0 y valdrá 2 si es rechazada por no encontrar hueco en los CPDs (si es de un usuario básico es que no se ha encontrado hueco en el CPD2 y si es de un usuario premium es que no se ha encontrado hueco en ninguno). En este caso las columnas que no tienen sentido (por ejemplo la columna CPD) valdrán todas “-”.

El servicio es de vídeo bajo demanda. Se sirven los vídeos desde dos CPDs diferentes.

- El bloque 0 representa un repartidor de carga que recibe todas las peticiones y las reparte entre los CPDs.
- El bloque 1 (CPD1) representa un CPD dedicado a atender las peticiones de los usuarios "premium".
- El bloque 2 (CPD2) representa un CPD dedicado a atender las peticiones de los usuarios "básicos" y de los usuarios "premium" que no puedan ser atendidos en el bloque 1. Cada CPD es capaz de atender a múltiples peticiones simultáneas y posee una cola de espera para almacenar peticiones mientras todos su servidores están en uso.

El bloque 0 tiene un conocimiento global del estado de ambos CPDs. Según el tipo de petición y el estado de los CPDs la reenviará a uno u otro. Tarda un tiempo constante en tomar la decisión, por ello se puede crear una cola de peticiones (que tiene una ocupación máxima). Este tiempo es un parámetro y se dará en las mismas unidades de tiempo que las llegadas. Si es una llegada de un usuario básico la encaminará al CPD2 (bloque 2), salvo que la cola del bloque 2 esté llena, en cuyo caso la descarta. Si es una llegada de un usuario premium la manda al bloque 1 salvo que la cola de espera del bloque 1 esté llena, en cuyo caso la envía al bloque 2, salvo que ésta también esté llena, en cuyo caso la descarta. Consulta el estado de los CPDs tras procesar la petición. Eso quiere decir que primero transcurre el tiempo de procesado en este bloque y cuando este tiempo se completa se mira en ese instante el estado de los CPDs. Si en ese instante hay hueco en alguno de los CPDs se reenviará; si no encuentra hueco en ningún CPD que pueda utilizar se rechaza.

Se debe llevar cuenta del número de peticiones de cada tipo de usuario que son descartadas en el bloque 0 para cada tipo, así como del número de peticiones de usuarios premium que son reenviadas al bloque 2. Las llegadas al bloque 1 ó 2 ocuparán uno de los servidores disponibles durante el tiempo que trae indicado la llegada y que se supone que es la duración de la película.

Una vez que se complete un servicio del bloque 1 o del bloque 2, dicho usuario puede que abandone el sistema o que continúe con otra película. Esto se modela con una probabilidad de que al completarse el servicio se convierta en una nueva llegada (elemento Separador). Por simplicidad se modelará esta probabilidad con una binomial con parámetro indicado en el argumento parametroDeRetorno de ejecución del programa. Es decir, el argumento dirá por ejemplo que la probabilidad de que un usuario retorne es de 0.05 (probabilidad, no porcentaje, es decir, en este caso un 5%).

En caso de retorno (decide ver otra película) lo hace volviendo al bloque 0 a través del elemento Agregador de entrada del CPD completo y en este caso la duración del siguiente vídeo (el nuevo tiempo de servicio en el CPD que le atienda) supondremos por simplicidad que es la mitad de la duración del vídeo anterior. Por supuesto no cambia su tipología (básico o premium).

La simulación debe continuar hasta que se agoten las llegadas y se haya vaciado el sistema. Es decir, no debe detenerse por ejemplo porque se agoten las llegadas de usuarios básicos, si aún quedan llegadas de usuarios premium en el fichero ni debe detenerse cuando se agoten ambos ficheros hasta que se haya calculado lo que sucede con las llegadas que se encuentran en el sistema. Hay que recordar que una llegada puede retornar.

Pautas de desarrollo

A primera vista es un problema mucho más complejo que el desarrollado en el objetivo 1. Sin embargo nada más lejos de la realidad puesto que el problema se puede descomponer en bloques elementales más simples que cumplen tareas muy sencillas. Los bloques que lo componen son los siguientes:

- 2 fuentes de tráfico: una fuente para tráfico de usuarios “premium” y otra fuente para usuarios “básicos”. Las fuentes de tráfico generan unidades de servicio. Se pueden reutilizar del ejercicio 1.
- 2 agregadores: tienen un número de entradas y simplemente combinan el tráfico procedente de sus entradas.
- Bloque 0: se puede modelar como un servidor G/G/1/c del ejercicio 1 y un decisor que decide si se asigna al bloque 0 o al 1.
- Bloque 1 y Bloque 2: Se pueden modelar como G/G/n/c del objetivo 1.
- 1 separador para las reentradas.
- 1 sumidero para imprimir las estadísticas.

Para ahorrar tiempo se debería reutilizar el servidor del objetivo 1 para construir los Bloques 1 y 2. Para comunicar los diferentes bloques es aconsejable emplear la estructura de unidad de servicio y de entrada/salida de cada uno de los bloques definida en el objetivo 1.

El resultado de la simulación debe incluir los siguientes estadísticos:

- nº de vídeos pedidos por usuarios premium (incluye los debidos a retornos)
- nº de vídeos pedidos por usuarios básicos (idem)
- nº de peticiones de usuarios premium rechazadas en el bloque 0 por saturación de su cola
- nº de peticiones de usuarios básicos rechazadas en el bloque 0 por saturación de su cola
- nº de peticiones de usuarios premium enviadas al CPD 2
- nº de peticiones de usuarios premium rechazadas por saturación de la cola de ambos CPDs
- nº de peticiones de usuarios básicos rechazadas en el bloque 2 por saturación de su cola

Ejemplo:

```
% simula ficheroLlegadasPremium.txt ficheroLlegadasBasicos.txt tiempoServicioBloque0 maxColaBloque0 numServidoresCPD1 maxColaCPD1 numServidoresCPD2 maxColaCPD2 parametroDeRetorno [instantesDeSalida]
```

En fichero de nombre ficheroLlegadasPremium.txt podría contener:

```
0.1 20
12 50
20 15
40 105
50 203
51 204
52 206
53 207
400 1
```

Eso quiere decir que en el instante 0.1 desde el comienzo de la simulación llega una petición de un usuario “premium” que solicita un vídeo de duración 20 unidades. En el instante 12 llega otro usuario “premium” que pide un vídeo de duración 50 unidades, etc
Todos los valores pueden ser decimales.

El fichero de nombre ficheroLlegadasBasicos.txt podría contener:

```
0.2 40
30 50
45 80
50 60
51 61
52 62
53 67
55 88
500 1
```

El parámetro tiempoServicioBloque0 puede valer 0.123564

El parámetro maxColaBloque0 puede valer 0, lo cual significaría que no hay posibilidad de quedarse a la espera en cola)

El parámetro numServidoresCPD1 podría valer 10, lo cual significa que el CPD1 puede atender a 10 peticiones a la vez

El parámetro maxColaCPD1 podría valer 5, lo cual significa que puede tener 5 peticiones encoladas

El parámetro numServidoresCPD2 podría valer 10, lo cual significa que el CPD2 puede atender a 10 peticiones a la vez

El parámetro maxColaCPD2 podría valer 5, lo cual significa que puede tener 5 peticiones encoladas

El parámetro parametroDeRetorno podría valer 0.1. Esto quiere decir que la probabilidad de que una petición completada se convierta en una nueva petición en ese mismo instante, de ese mismo tipo y de mitad de duración es del 10%

El parámetro instantesDeSalida podría valer instantesDeSalida.txt. El contenido resultante de la ejecución de ese fichero sería algo así como:

```
0.2 basico 0 40.0 - 1
20.0 premium 0 15.0 - 1
0.1 premium 0 20.0 1 0
30.0 basico 0 50.0 - 1
45.0 basico 0 80.0 - 1
50.0 basico 0 60.0 - 1
50.0 premium 0 203.0 - 1
51.0 basico 0 61.0 - 1
51.0 premium 0 204.0 - 1
52.0 basico 0 62.0 - 1
52.0 premium 0 206.0 - 1
```

```
53.0 basico 0 67.0 - 1
53.0 premium 0 207.0 - 1
55.0 basico 0 88.0 - 1
12.0 premium 0 50.0 2 0
40.0 premium 0 105.0 1 0
400.0 premium 0 1.0 1 0
500.0 basico 0 1.0 2 0
```

Las peticiones de usuarios que sean retornadas no se mostrarán en el fichero de salidas.

Objetivo 3. Memoria con los resultados de testeo y validación (30%)

Analizar los dos simuladores realizados con diferentes parámetros. Comparar (si se puede) con resultados teóricos. Se pueden extraer resultados de probabilidades de pérdida en función de parámetros de tráfico de entrada y de las colas. Incluir la validación de ambos sistemas exponiéndolos a escenarios en los que se pueda analizar de forma teórica.

Los objetivos 1 y 2 se valorarán únicamente de acuerdo a los análisis y la validación que se presenten en la memoria.

Extensión máxima 2000 palabras.