LABORATORIO DE TELEMÁTICA

Prácticas de la asignatura

Autores: D. Morató, M. Izal
Fecha: 20 de Febrero de 1998

INTRODUCCIÓN

La asignatura de *Laboratorio de Telemática* de cuarto curso de Ingeniería Superior de Telecomunicación tiene como objetivo profundizar en la programación sobre el sistema operativo UNIX, aplicando los conocimientos adquiridos respecto a programación en sistemas multiproceso. Asimismo se pretende que el alumno se familiarice con el paradigma de programación orientada a objeto y que tome contacto con el lenguaje de programación Java.

Se proponen las siguientes prácticas:

- Empleo de Streams
- Programación concurrente
- Comunicación entre procesos
- Programación de clases en Java
- Programación de Applets Java.
- Construccion de aplicaciones con interfaz gráfico en Java

Las prácticas se realizarán en el Laboratorio de Ingeniería Telemática del Dpto. de Automática y Computación, segunda planta del Edificio de Ingenieros. Se supone que el alumno conoce el lenguaje de programación C y tiene conocimientos de UNIX, impartidos en la asignatura de Arquitectura de Computadores. Por otro lado, se darán clases teóricas en la asignatura para dar los conocimientos teóricos necesarios de streams y técnicas de comunicación entre procesos, asi como una introduccion a la programación orientada a objeto y al lenguaje Java.

EVALUACIÓN

Las prácticas suponen un 50% de la nota de la asignatura, es decir cinco puntos sobre diez. Las puntuaciones de cada práctica aparecen en el enunciado. La evaluación se hará de manera automática de manera que es *imprescindible seguir las convenciones de nombres y directorios* que se citan en este documento. Si usted no sigue estas convenciones puede ocurrir que el script de UNIX que evalúa las prácticas no detecte las suyas.

NORMAS DE ETIQUETA EN EL LABORATORIO

Asegúrese de que hace *exit* en todas las shell y conexiones Telnet que abra. Esto es importante por la seguridad del sistema y por su propio interés. Si usted deja shells abiertas puede ocurrir que otra persona entre en su cuenta y borre o copie sus practicas. Esta circunstancia le colocaría en una situación muy comprometida a la hora de la evaluación de su trabajo. Así mismo, cambie inmediatamente su password la primera vez que entre en su cuenta, no comunique su password a

nadie, el primer interesado en que su password sea desconocida es usted por las mismas razones.

Finalmente, no apague su máquina, piense que está en un entorno multiusuario real y que por tanto puede haber otra persona trabajando en su máquina. Es posible que, sin quererlo, haga que esa persona pierda todo su trabajo.

COPIAS

Intentar obtener el aprobado por métodos fraudulentos (copias de software de sus compañeros) es una estafa a la universidad, a la empresa y a la sociedad en general.

Las copias de prácticas se detectarán mediante analizadores léxicos de muy alta fiabilidad. Una copia supone el suspenso automático de la asignatura en la convocatoria de Junio y Septiembre de 1998 y un informe al Rectorado de la Universidad para que se le abra un expediente académico, con las gravísimas consecuencias que esto puede acarrear. Es preferible suspender la asignatura que copiar. Ni lo intente.

BIBLIOGRAFÍA

Para UNIX/LINUX:

W. R. Stevens, *Advanced Programming in the UNIX environment,* Addison-Wesley, 1992.

G. Glash, *UNIX For Programmers and Users, a complete guide*. Prentice Hall 1993.

C. Brown, *UNIX Distributed Programming*. Prentice Hall 1994

Neil Matthew and Richard Stones, *Beginning Linux Programming*, Wrox Press, 1996.

Existe una abundante bibliografía sobre Linux, ademas de este título, incluyendo CD-ROM para instalación de LINUX. La versión de Linux que se utiliza en el Laboratorio de Telemática es *Linux Red Hat*.

Para programción orientada a objeto y Java:

Laura Lemay, Charles L. Perkins *Teach yourself Java in 21 days*. Prentice Hall Hispanoamericana, 1996

G. Booch, *Object-oriented analysis and design with applications* Benjamin/Cummings, cop. 1994

WWW

http://www.tlm.upna.es/~daniel/asignaturas/labtel/labtel.html

Práctica 1

Duración: 3 horas
Puntuación: 1 pto.

Objetivos:

El objetivo de esta práctica es iniciarse en el empleo de streams desde el punto de vista del volcado de datos binarios en vez de texto. Para ello se evaluará la interacción de dos parámetros decisivos en el volcado de datos a través de un stream: el tamaño del buffer de éste y la cantidad de bytes que se desea almacenar en cada llamada a fwrite.

Enunciado:

Cree un fichero de datos y almacene en él datos a la máxima velocidad hasta alcanzar un tamaño prefijado constante (1 MB). Emplee para ello distintos tamaños del buffer del stream y para cada uno de ellos distintos tamaños de volcado de datos (cantidad de bytes a volcar por el fwrite).

El resultado debe ser una tabla con el tiempo total empleado para completar el fichero. Las líneas horizontales de datos poseerán el mismo tamaño de buffer y las verticales el mismo tamaño de volcado del fwrite (Diríjalo a la salida estándar).

Cree el fichero de datos en el directorio /tmp. Emplee un nombre que difícilmente vaya a coincidir con otros y tras terminar la simulación borre el fichero.

Recorra tamaños de buffer y de volcado entre 1 y 100000 bytes (por ejemplo una tabla de 50x50).

Cuestiones:

• Mida el tiempo transcurrido tanto empleando la estructura que provee times(2) como el valor que retorna. ¿Hay diferencias importantes? ¿Por qué?. Pruebe de nuevo, esta vez simultaneando la ejecución del simulador con un proceso que se limite a consumir ciclos de reloj (que haga un simple bucle infinito). ¿Hay diferencias importantes? ¿Por qué?

• Repita la simulación creando esta vez el fichero en su directorio home. ¿Hay diferencias importantes? Si las hay, ¿a qué pueden deberse? (Guarde esta nueva tabla en un fichero llamado *resultados2*).

Otras funciones de utilidad:

times(2), tmpfile(3), tempnam(3)

daniel.morato@upna.es

Presentación:

En el directorio practica1 en su home deben existir uno o varios ficheros *.c* y *.h* así como un *makefile* que compile el programa mencionado. Para la corrección de la práctica se borrarán todos los ejecutables, se hará un *touch* a todos los ficheros fuente y se recompilará mediante el makefile, así que no servirá de nada dejar en el directorio ejecutables o ficheros objeto. Igualmente debe haber un fichero llamado *resultados* con la tabla que genera el programa.

daniel.morato@upna.es

Práctica 2

Duración: 5 horas
Puntuación: 1.5 ptos.

Objetivos:

El objetivo de esta práctica es comprender ciertos aspectos del empleo del multiproceso en UNIX, así como de su implementación. Para ello se ha elegido el scheduler como proceso estrella del problema.

Enunciado:

Cree un programa que responda a la siguiente página de manual:

NOMBRE

ownsched - Realiza un scheduling Round Robin entre varios procesos.

SYNOPSIS

ownsched [-s segs] [file...]

DESCRIPCION

Ownsched es una utilidad que permite repartir el uso de la CPU entre varios comandos que se ejecutan concurrentemente. Para ello suspende y reanuda la ejecución de los mismos simulando a alto nivel un algoritmo RR de asignación de recursos.

Los programas no han de requerir parámetros de entrada ni precisar el empleo de la entrada estándar.

OPCIONES

-s segs

Especifica la duración en segundos del segmento de tiempo de ejecución que se le concede a un proceso ante de ser suspendido en favor de otro.

(FIN)

Pruebe el comando con segmentos de unos 5 segundos de duración.

daniel.morato@upna.es

Cuestiones:

• ¿Qué sucede si el segmento de tiempo es muy pequeño, del orden de un par de segundos?

• Modifique el programa para que responda a la estructura:

ownsched2 [-s segs] [-n nanosegs] [file...]

Donde en este caso se puede especificar el segmento de tiempo con precisión de nanosegundo.
¿Tiene sentido intentar especificar tanta precisión en el segmento de tiempo? ¿En qué medida es real el scheduling que realiza el comando?

• ¿Podría implementarse otra estrategia de scheduling diferente a Round Robin? De ser así, ¿cómo?

Otras funciones de utilidad:

kill(2), sleep(3), nanosleep(2)

Presentación:

En el directorio practica2 en su home deben existir uno o varios ficheros *.c* y *.h* así como un *makefile* que compile el programa mencionado. Para la corrección de la práctica se borrarán todos los ejecutables, se hará un *touch* a todos los ficheros fuente y se recompilará mediante el makefile, así que no servirá de nada dejar en el directorio ejecutables o ficheros objeto.

daniel.morato@upna.es

Práctica 3

Duración: 7 horas
Puntuación: 2.5 ptos.

Objetivo:

Crear un caso práctico de comunicación entre varios procesos. Estudiar los diferentes problemas que puede presentar la comunicación entre procesos concurrentes.

Enunciado:

Cree un programa que sea capaz de permanecer ejecutándose permanentemente aunque el propietario se desconecte. Este proceso debe esperar datos por una cola de mensajes para a continuación añadir el contenido del mensaje a un fichero. Su nombre será *p3d*.

Cree otro programa que responda a la siguiente especificación:

NOMBRE

saverr - Permite ejecutar comandos de forma que su salida de error se dirija al demonio p3d.

SYNOPSIS

saverr [-n] [file [opciones]]

DESCRIPCION

Saverr es una utilidad que ejecuta el programa especificado en sus opciones de forma que los avisos que éste dirija a la salida estándar de error (stderr) son enviados mediante mensajes al demonio p3d para ser almacenados en un fichero global de errores. De esta forma los mensajes de error de diferentes programas que se ejecuten simultáneamente pueden ser recogidos en el fichero de p3d secuencialmente.

El programa ejecutado no pierde la interactividad con su terminal, es decir, puede emplear su entrada estándar con normalidad.

OPCIONES

-n

Elimina la salida de error del programa, de forma que ésta no aparece en el terminal pero sí se envía al demonio p3d. Debe ir antes del fichero ejecutable para que no se confunda con las posibles opciones de éste.

daniel.morato@upna.es

El fichero de errores de p3d, llamado p3d.errors sigue la siguiente estructura:

Fecha Programa : Mensaje_de_error

Donde:

Fecha
Día_semana Mes Día hora:min:segs Año

ej: Thu Feb 19 13:34:38 MET 1998

Programa
Nombre del ejecutable que dio ese error.

Mensaje_de_error
Texto que el proceso envió a la salida de error.

(FIN)

Cree algún programa interactivo que saque mensajes de error para probar *saverr* y *p3d*.

Cuestiones:

• Supongamos que se intenta leer el contenido de p3d.errors justo cuando *p3d* intenta actualizarlo. ¿Podría haber algún problema? De ser así, ¿cómo podría resolverse?.

Otras funciones de utilidad:

pipe(2), gettimeofday(2), time(2), ctime(3)

Presentación:

En el directorio practica3 en su home deben existir uno o varios ficheros *.c* y *.h* así como **un** *makefile* que compile tanto el demonio *p3d* como *saverr*. Para la corrección de la práctica se borrarán todos los ejecutables, se hará un *touch* a todos los ficheros fuente y se recompilará mediante el makefile, así que no servirá de nada dejar en el directorio ejecutables o ficheros objeto.

daniel.morato@upna.es

Práctica 4

Duración: 4 horas
Puntuación: 1.5 ptos

Objetivos:

Conceptos básicos de la programación orientada a objeto: Crear clases, invocar métodos y reutilizar código.

Enunciado:

Implementar el famoso "juego de la vida" en un lenguaje orientado a objeto como Java. En esta primera práctica se programará el juego y en las siguientes prácticas se programará un interfaz gráfico.

El "juego de la vida" se desarrolla en un tablero rectangular dividido en MxN celdas. Cada celda puede estar ocupada(viva) o desocupada(muerta) Cada turno las celdas nacen, mueren o se mantienen vivas según el número de celdas vivas que tengan alrededor. Las reglas para saber el estado de una celda en el turno siguiente son:

- Si una celda está viva y tiene 0 ó 1 vecino morirá de soledad.
- Si una celda está viva y tiene 2 ó 3 vecinos sobrevivirá.
- Si una celda está viva y tiene 4, 5, 6 ,7 ó 8 vecinos morirá debido a la superpoblación.
- Si una celda está vacía y tiene 3 vecinos exactamente vivirá en el siguiente turno. Con cualquier otro numero de vecinos seguirá desocupada.

Los vecinos de una celda son las celdas situadas a 1 celda de distancia vertical, horizontal o diagonalmente. Por convenio consideraremos que el tablero es continuo, las celdas del borde derecho tienen por vecinas a las del borde izquierdo y la del borde superior a las del inferior.

Para programar esto se utilizarán dos clases. La primera se llamará *celda* y definirá el estado de una celda (ocupado / desocupado) así como el estado que tendrá la celda en el próximo turno. También llevará la cuenta del tiempo (numero de turnos) que lleva viva esa celda.
La otra clase se llamará *tablero* y contendrá un conjunto de *celdas* a las que hará evolucionar siguiendo las reglas que hemos definido antes.

mikel.izal@upna.es

La clase *celda* debe tener los siguientes métodos:

Constructores:

• `celda()` : construye una celda desocupada.

• `celda(boolean estado)` : construye una celda ocupada o desocupada según el valor del parámetro estado.

Métodos:

• `void preparaSiguienteTurno(int n_vecinos)` : hace que la celda calcule cual será su estado en el siguiente turno a partir del número de vecinos que se le pasa en el parámetro.

• `void actualizaTurno()` : hace que la celda avance un turno.

La clase *tablero* debe tener los siguientes métodos:

Constructores:

• `tablero(int w,int h)` : construye un tablero de w x h celdas. El contenido de las celdas puede estar vacío, ser aleatorio o lo que quiera cada uno.

• `tablero(boolean[][] inicial)` : construye un tablero a partir de un array de booleanos que indican si la celda en esa posición esta viva o muerta en el primer turno. El tablero es del mismo tamaño que el array.

Métodos:

• `void actualizaTurno()` : hace que el tablero haga avanzar un turno a todas sus celdas.

• `boolean getEstado(int x, int y)` : devuelve el estado de la celda que está en la posición x,y

• `boolean getAntiguedad(int x,int y)` : devuelve la antigüedad de la celda que esta en la posición x,y

• `String toString()` : devuelve una cadena que representa el tablero separando las líneas con `\n` de forma que al imprimir la cadena se vea el tablero más o menos así:

```
......XX......      ......21......
.....XX..X....      .....22..3....
......XX..X...      ......11..1...
..............      ..............

   X viva              n antigüedad
   . muerta            . muerta
```

Se aconseja realizar también una aplicación java que calcule varios turnos del juego para comprobar que funciona, aunque no hace falta entregarla.

Presentación:

Se deberán entregar los ficheros fuente y los bytecodes de estas dos clases.
En el directorio practica4 se dejaran los ficheros:

```
celda.java, celda.class
tablero.java, tablero.class
```

mikel.izal@upna.es

Práctica 5

Duración: 4 horas
Puntuación: 1.5 ptos

Objetivos:

Familiarizarse con la programación de applets, los métodos de dibujo del AWT y la animación mediante un Thread.

Enunciado:

Utilizando las clases de la práctica anterior construir un Applet que dibuje el juego de la vida de forma continua. Para ello construir una clase que se llame *lifegameCanvas* y que sea una subclase de java.awt.Canvas. La clase dibujara cada segundo un turno del juego. La representación gráfica se hará de forma que se aprecie la antigüedad de cada celda por ejemplo dibujando en diferente color según los turnos que lleve viva cada celda.

La clase *lifegameCanvas* debe admitir los siguientes métodos:

Constructores:
• `lifegameCanvas(int w, int h)` : utiliza un tablero de wxh
• `lifegameCanvas(boolean[][] inicial)` : utiliza el tablero inicial indicado

Métodos:
• `void start()` : empieza a correr el juego de la vida.
• `void suspend()` : suspende la ejecución del juego.
• `void resume()` : continua la ejecución del juego.

Una vez construida la clase *lifegameCanvas* programar el Applet es muy sencillo, basta con añadir el canvas en la inicialización del Applet.

Presentación:

Se presentará la clase *lifegameCanvas* con el fichero fuente y el bytecode:
`lifegameCanvas.java, lifegameCanvas.class`
Se presentará también el Applet al que pondremos de nombre *lifegameApplet* con su código fuente y el bytecode así como un fichero html que permita verlo.
`lifegameApplet.java, lifegameApplet.class, lifegame.html`

La clase *lifegameCanvas* se probará independientemente del applet por lo que debe cumplir fielmente los métodos expuestos.

mikel.izal@upna.es

En el directorio practica5 dejad los ficheros

`lifegameCanvas.java, lifegameCanvas.class`
`lifegameApplet.java, lifegameApplet.class, lifegame.html`

más las clases de la práctica anterior necesarias para que funcione.

mikel.izal@upna.es

Práctica 6

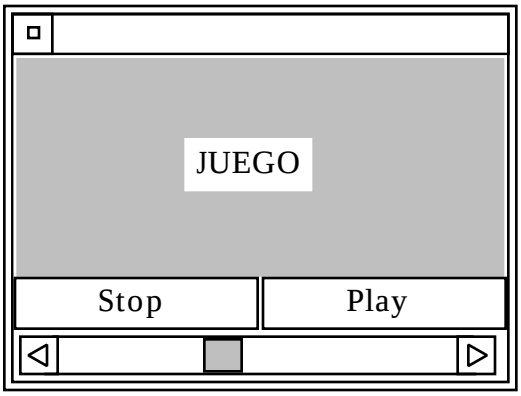
Duración: 7 horas
Puntuación: 2 ptos

Objetivos:

Familiarizarse con los elementos básicos del AWT utilizando Layouts y respondiendo a eventos, programar una aplicación java con interfaz gráfico.

Enunciado:

Ahora intentaremos colocar el canvas de la práctica anterior en una aplicación java. Al ejecutar la aplicación debe abrirse una ventana en la que tendremos un tablero para el juego, botones para detenerlo y reanudarlo y una barra para controlar la velocidad a la que pasan los turnos. Algo así:



Además cuando el juego está parado se puede editar pinchando con el ratón en la zona de juego. Si pinchamos en una casilla vacía se llenara y si lo hacemos sobre una celda viva desaparecerá.

En esta práctica se deja al alumno que diseñe las clases nuevas que necesite para conseguir los requisitos que se piden en el enunciado.

Presentación:

Puesto que para realizar esta práctica habrá que modificar ligeramente las clases de las prácticas anteriores se entregarán todas las clases con sus códigos fuente y bytecodes. La clase que forme la aplicación debe llamarse *lifegame*.

En el directorio practica6 dejad todos los ficheros con los códigos fuente y las clases que hacen falta para que funcione.

mikel.izal@upna.es