PROGRAMACIÓN PARALELA Y DISTRIBUIDA

PRACTICA Nº 2: ASPECTOS DE LA PROGRAMACIÓN PARALELA Y DISTRIBUIDA

TEMAS: Herramientas y conceptos para lograr concurrencia: Threads, Semáforos, Variables condicionales, Procesos, Pipes y Sockets.
OBJETIVOS: Que los alumnos identifiquen los posibles usos de los threads y procesos para el paralelismo y la concurrencia, que reconozcan los elementos que los componen, y las herramientas que brindan soporte a la sincronización y comunicación.

Ejercicio Nº1 – Escribir un programa en C que reciba como argumento dos números enteros, N y M. Dichos números son utilizados para crear N threads de *tipo 1* y M threads de *tipo 2.* La funcionalidad de los threads es la siguiente:

THREAD *tipo 1*: Debe mostrar por pantalla el siguiente mensaje: "Thread1 instancia *x*", siendo *x* el número de thread creado (entre 0 y N-1). Luego se suspenderá por un tiempo aleatorio entre 0 y 1 segundo (llamada *usleep* o *sleep*) y posteriormente incrementará la variable global "Compartida".

THREAD *tipo 2*: Debe mostrar por pantalla el siguiente mensaje: "Thread2 instancia *y*", siendo *y* el número de thread creado (entre 0 y M-1). Posteriormente se suspenderá por un tiempo aleatorio entre 0 y 1 segundo. Finalmente leerá y mostrará por pantalla el valor de la variable global "Compartida".

Cuando los threads finalicen su ejecución, el padre debe mostrar por pantalla "Fin del programa".

Nota: Para realizar este ejercicio deben utilizarse las llamadas pthread_create y pthread_join

Ejercicio Nº2 – Realizar el ejercicio anterior, pero esta vez utilizando procesos en vez de threads. Es decir, ahora se deberán crear, por medio de la llamada **fork**, N PROCESOS tipo 1 y M PROCESOS tipo 2. ¿Qué comportamiento puede observarse? ¿Qué diferencia existe entre esta implementación y la del ejercicio anterior?

Ejercicio Nº 3 – Se deberá desarrollar una aplicación que simule partidas de siete y medio entre N jugadores. Esta aplicación estará compuesta por varios procesos, y seguirá un esquema cliente/servidor, donde los procesos clientes serán los jugadores, y el proceso servidor, se encargará de distribuir las cartas, recoger las decisiones de los jugadores, y así hasta que se declare un ganador.

> Algunas normas básicas del juego: Se juega con As, 2, 3, 4, 5, 6, 7, Sota, Caballo, y Rey. Las primeras tienen su valor numérico como puntos, y la sota, caballo, y rey tienen un valor de medio (1/2 punto). En cada ronda del repartidor, se da una carta a cada jugador (que continua en la partida), este en función del valor de sus cartas previas decide si se planta o abandona (se ha pasado de 7 y medio). La repartición de cartas continua mientras no hayan llegado todos a una situación de plante o abandono, en este caso se avalúan las cartas de los jugadores plantados, y se decide el ganador (el que más se aproxima a 7 y medio), en caso de empate se escoge a uno de ellos como ganador (por ejemplo al primero).

La ejecución de la partida será mediante el comando: *sieteymedio <N>*

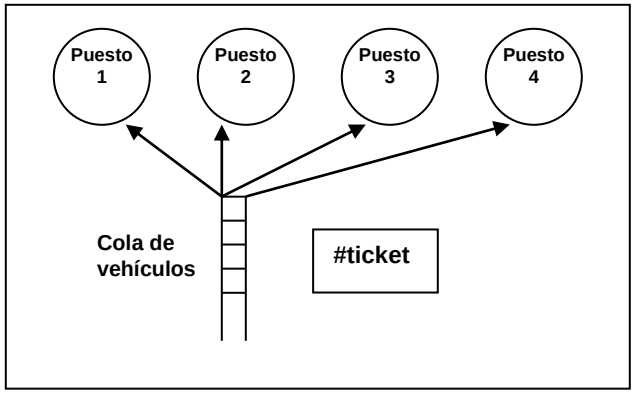
Donde N será el número de jugadores que participan en la primera partida. Como resultado de la ejecución se publica el ganador de la partida, así como una tabla describiendo la situación de cada jugador (cuántos puntos posee, se plantó o abandonó).

El proceso *sieteymedio* será el encargado de gestionar la partida, será el que repartirá las cartas por turnos, mantendrá el estado de la partida a partir de las decisiones que le pasen los jugadores, y decidirá finalmente el ganador de la partida

El proceso *jugador* será el que implementará la lógica del jugador en la partida, en base a las cartas que reciba del repartidor (*sieteymedio*), y de su propio estado (puntos actuales) decidirá si continúa en el juego (pide más cartas), se planta, o abandona (porque se pasó). Esta lógica de juego se deja abierta, de forma que se implemente como se quiera, pudiendo llegar a ser simplemente aleatoria: según un número aleatorio el jugador decide plantarse, pedir carta, o abandonar.

a) La generación de los N procesos jugadores deberá realizarse a través del uso de **forks**, y la comunicación repartidor/jugador se efectuará mediante **pipes**.
b) Implementar una segunda versión mediante **fork** y comunicación a través de **sockets**.

Ejercicio Nº 4 – En este ejercicio simularemos un puesto de peaje de una autopista.



La estación de peaje deberá atender a los vehículos que están en una única cola de espera de longitud fija. Cada vehículo está descrito por un registro compuesto por los siguientes campos:

- Identificador: un número entero que identifica al vehículo.
- Categoría: tipo de vehículo del que se trata {motocicleta; coche; utilitario; pesado}.
- Dinero: cantidad de dinero del que dispone el conductor del vehículo.

Esta información se encontrará en el archivo de entrada *autopista.in* con el siguiente formato:

id:<número entero>;cat:<texto>;din:<número real>

Un ejemplo de una entrada válida sería:

id:34;cat:motocicleta;din:30,50
id:35;cat:pesado;din:430,70

id:12;cat:coche;din:7,20
id:64;cat:motocicleta;din:50

La tarea que realizará cada puesto de peaje consiste en decidir el importe a pagar en función de la categoría de vehículo ($1 para motocicletas; $2,50 para coches; $3,25 para utilitarios; $5 para pesados). Una vez efectuado el cobro (para lo cual deberá deducirse el importe del campo dinero), se entregará a cada vehículo un número de ticket que sirve como comprobante de pago. A su vez, de acuerdo a la categoría del vehículo, éste se demorará más o menos tiempo en salir del puesto de peaje y continuar su viaje (1 segundo para motocicletas, 2 segundos para coches, 3 segundos para utilitarios y 4 segundos para pesados).

Como salida de la estación de peaje se obtendrá un nuevo archivo con el mismo formato del archivo de entrada, con dos salvedades: el campo dinero deberá estar actualizado según el pago efectuado, y cada registro deberá tener un campo adicional que indique el número de ticket asociado.

Un ejemplo de salida para el caso anterior sería:

id:34;cat:motocicleta;din:29,50;tick:1
id:12;cat:coche;din:4,70;tick:2
id:64;cat:motocicleta;din:49;tick:3
id:35;cat:pesado;din:425,70;tick:4

La estación de peaje deberá crear 4 threads que representen los 4 puestos de cobro. También será la encargada de colocar los vehículos en la cola de espera. Si durante este proceso encuentra que la cola está llena, deberá bloquearse hasta que vuelva a haber lugar en la cola.
Los threads atenderán los vehículos que estén en dicha cola compartida. La cola se implementará como un arreglo de 15 elementos. El número de ticket a asignar a cada vehículo será obtenido de la variable compartida *num_tick*, la cual comenzará en cero y se irá incrementando sucesivamente. Los puestos de cobro escribirán los resultados directamente en el archivo de salida *autopista.out*. Deberá prestarse particular atención al uso y acceso de los recursos compartidos.

NOTA: Para compilar un programa que utilice la librería pthreads debe utilizarse la opción de enlace –lpthread.