Fila única

Sistemas Operativos (SO)
Grado en Ingeniería Informática

Universidad de León

Resumen

El objetivo de la práctica es realizar un programa que simule el sistema de fila única para el cobro a clientes en un pequeño supermercado. La práctica tiene una parte básica donde se define el funcionamiento básico del sistema de fila única y que es obligatorio implementar (supondrá como mucho el 80 % de la nota) y, además, se proponen una serie de mejoras opcionales para aumentar la nota (como mucho supondrá el 20 % de la nota).

1. Parte Básica (80 % de la nota)

El sistema de fila única para el cobro a clientes del supermercado funciona de la siguiente forma:

- El supermercado cuenta con 3 cajeros para atender a sus clientes. 3 clientes pueden ser atendidos a la vez, el resto esperará formando una fila única.
- Cuando los clientes terminan sus compras se colocan en la fila única. A cada cliente se le asignará un identificador único y secuencial (cliente_1, cliente_2, ..., cliente_N). El número máximo de clientes que pueden ponerse a la cola es de 20.
- Los clientes pueden cansarse de esperar y abandonar la cola dejando sus compras.
- El supermercado cuenta con 1 reponedor al que los cajeros pueden llamar para consultar un precio.

Cada cliente será atendido por un cajero que:

- Tiene un identificador único (cajero_1, cajero_2 y cajero_3).
- Solamente cuando haya terminado con un cliente pasará a atender al siguiente.
- Cada vez que un cajero atiende a 10 clientes se toma un descanso (duerme 20 segundos), y no atiende al siguiente hasta que vuelve.
- De los clientes atendidos el 70 % no tiene problemas. El 25 % tiene algún problema con el precio de alguno de los productos que ha comprado y es necesario que el reponedor vaya a comprobarlo, si está ocupado cliente y cajero deberán esperar. El último 5 % no puede realizar la compra por algún motivo (no tiene dinero, no funciona su tarjeta, etc.).

Toda la actividad quedará registrada en un fichero plano de texto llamado `registroCaja.log`:

- Cada vez que se atiende a un cliente se registrará en el log las horas de inicio y final de la atención al mismo.
- Se registrará el precio total de la compra.
- Se registrará si hay algún problema con algún precio o si el cliente no finalizó la compra.
- Se registrará la entrada y salida del descanso por parte del cajero.
- Al finalizar el programa se debe registrar el número de clientes atendidos por cada cajero.

Consideraciones prácticas:

- Simularemos la llegada de clientes a la fila única mediante la señal SIGUSR1. Se enviarán desde fuera del programa mediante el comando *kill* (`kill -SIGUSR1 PID`).

    - Los clientes pueden irse sin ser atendidos tras 10 segundos de espera. Se simulará esta posibilidad haciendo que el 10 % lo hagan.
- Simularemos el tiempo de atención de cada cliente con un valor aleatorio entre 1 y 5 segundos.
- El precio total de cada compra será un número aleatorio entre 1 y 100.
- Todas las entradas que se hagan en el log deben tener el siguiente formato: `[YYYY-MM-DD HH:MI:SS] identificador: mensaje.`, donde `identificador` puede ser el identificador del cliente, el identificador del cajero, el identificador el reponedor *o el identificador del departamento de atención al cliente* y `mensaje` es una breve descripción del evento ocurrido.
- Las entradas del log deben quedar escritas en orden cronológico.
- Hay que controlar el acceso a los recursos compartidos (al menos la cola de clientes y al fichero de log) utilizando semáforos (*mutex*).
- Para comunicar a los cajeros con el reponedor, utilizaremos una variable de condición. Los cajeros señalizarán la condición cuando necesiten que el reponedor consulte un precio. El reponedor estará a la espera de que esto ocurra.
- Simularemos el tiempo de atención del reponedor durmiendo un valor aleatorio entre 1 y 5 segundos.

2. Partes opcionales (20 % de la nota)

- Asignación estática de recursos (10 %):
    - Modifica el programa para que el número de clientes que pueda acceder al la fila única sea un parámetro que reciba el programa al ser ejecutado desde la línea de comandos.
    - Modifica el programa para que el número de cajeros que atienden sea un parámetro que reciba el programa al ser ejecutado desde la línea de comandos.
- Asignación dinámica de recursos I (5 %):
    - Modifica el programa para que el número de clientes que puedan acceder a la fila única pueda modificarse en tiempo de ejecución.
    - Solamente es necesario contemplar un incremento en el número de clientes. No es necesario contemplar la reducción.
    - Cada vez que se cambie el número de clientes tiene que reflejarse en el log.
- Asignación dinámica de recursos II (5 %):
    - Modifica el programa para que el número de cajeros que atiende se pueda modificar en tiempo de ejecución.
    - Solamente es necesario contemplar un incremento en el número de cajeros. No es necesario contemplar la reducción.
    - Cada vez que se produce un cambio en este sentido debe quedar reflejado en el log.

3. Diseño

En los siguientes apartados se detalla un posible diseño para la práctica. las zonas coloreadas en rojo, azul y verde son zonas de exclusión mutua que deben ser controladas. En las zonas verdes se deben usar variables condición.

3.1. Variables globales

Para implementar el diseño propuesto, el programa debería contar, al menos, con las siguientes variables globales:

- Ruta al archivo de log.
- Archivo de log (FILE *logFile).
- Lista de 20 clientes:
    - id
    - estado (0=esperado, 1=en cajero, 2=atendido)
- Semáforo para controlar el acceso al archivo de log.

- Semáforo para controlar el acceso a la lista de clientes.
- Variable de condición para interactuar con el reponedor.
- Semáforo asociado a la variable anterior.

3.2. Función main

A continuación, se enumeran la secuencia de acciones que se deberían realizar en el hilo principal.

1. `sigaction` de SIGUSR1 para crear clientes cuando se reciba la señal.
2. Inicializar recursos (¡Ojo! Inicializar no es lo mismo que declarar):
   - Semáforos.
   - Variables de condición.
   - Creamos el fichero de log.
   - Lista de clientes.
3. Creamos 3 hilos cajero.
4. Creamos 1 hilo reponedor.
5. Esperamos la recepción de señales (`pause`).

3.3. Llegada de nuevos clientes

Cada vez que se reciba la señal SIGUSR1 deberían realizarse las siguientes acciones. Estas tienen que implementarse en la función manejadora de la señal.

1. Buscamos una posición vacía en la lista de clientes.
2. Si hay hueco en la lista de clientes.
   - Creamos un nuevo hilo cliente.
   - Rellenamos los datos del cliente (id, estado=0).
3. Si no lo hay ignoramos la llamada.

3.4. Cajeros

Cada hilo cajero debería realizar la siguiente secuencia de acciones en bucle:

1. Buscamos el cliente que más tiempo lleva en la cola (el de menor id).
2. Si hay un cliente que atender:
   a) Cambiamos su estado (1).
   b) Calculamos el tiempo de atención.
   c) Escribimos la hora de la atención de la compra.
   d) Esperamos el tiempo de atención.
   e) Generamos un aleatorio de 1 a 100:
      - Si está entre 71 y 95, avisamos al reponedor y esperamos a que vuelva.
      - Si está entre 96 y 100, el cliente tiene algún problema (no tiene dinero suficiente, no funciona su tarjeta, etc.) y no puede realizar la compra.
   f) Escribimos la hora de finalización de la compra, si está ha terminado correctamente –en ese caso registramos también el importe– o ha habido algún problema.
   g) Cambiamos el estado del cliente (2).
   h) Si ha atendido a 10 clientes, el cajero descansa 20 segundos.

3.5. Clientes

Cada hilo cliente debería realizar la siguiente secuencia de acciones:

1. Escribimos la hora de llegada del cliente.
2. Calculamos el tiempo de espera máximo del cliente (aleatorio).
3. Esperamos a que expire el tiempo de espera o a que un agente nos atienda.
4. Esperamos si un agente nos ha atendido, esperamos a que termine (comprobar estado).
5. Guardamos en el log la hora de finalización.
6. Borramos la información del cliente de la lista.

3.6. Reponedor

El hilo reponedor debería realizar la siguiente secuencia de acciones en bucle:

1. Esperamos a que algún cajero me avise.
2. Calculamos el tiempo de trabajo (aleatorio).
3. Esperamos el tiempo.
4. Avisamos de que ha terminado el reponedor.

3.7. Escritura de mensajes en log

Es recomendable utilizar una función parecida a esta para evitar repetir líneas de código. Recibe como parámetros dos cadenas de caracteres, una para el identificador y otra para el mensaje (la fecha la calcula la propia función):

```
void writeLogMessage(char *id, char *msg) {
  // Calculamos la hora actual
  time_t now = time(0);
  struct tm *tlocal = localtime(&now);
  char stnow[25];
  strftime(stnow, 25, "%d/%m/%y %H:%M:%S", tlocal);

  // Escribimos en el log
  logFile = fopen(logFileName, "a");
  fprintf(logFile, "[%s] %s: %s\n", stnow, id, msg);
  fclose(logFile);
}
```