SERVICIO DE GESTIÓN DE IMÁGENES PUBLICITARIAS EN BASE A UN SISTEMA DE SUBASTA

PSCD – TP6

AUTORES:

EDUARDO ALONSO MONGE (728502)

PABLO NOEL CARRERAS AGUERRI (718743)

MIGUEL BENTUÉ BLANCO (719378)

INTRODUCCIÓN

Una empresa de publicidad nos pide desarrollar un sistema para gestionar sus vallas electrónicas y mostrar imágenes de sus clientes en ellas.

La empresa dispone de dos vallas publicitarias idénticas, y su negocio consiste en ofertas subastas de un tiempo determinado a los clientes para que puedan insertar su publicidad en las vallas.

SECCIÓN PRINCIPAL

El trabajo consiste en el desarrollo de un sistema de gestión como el que se esquematiza en la figura:

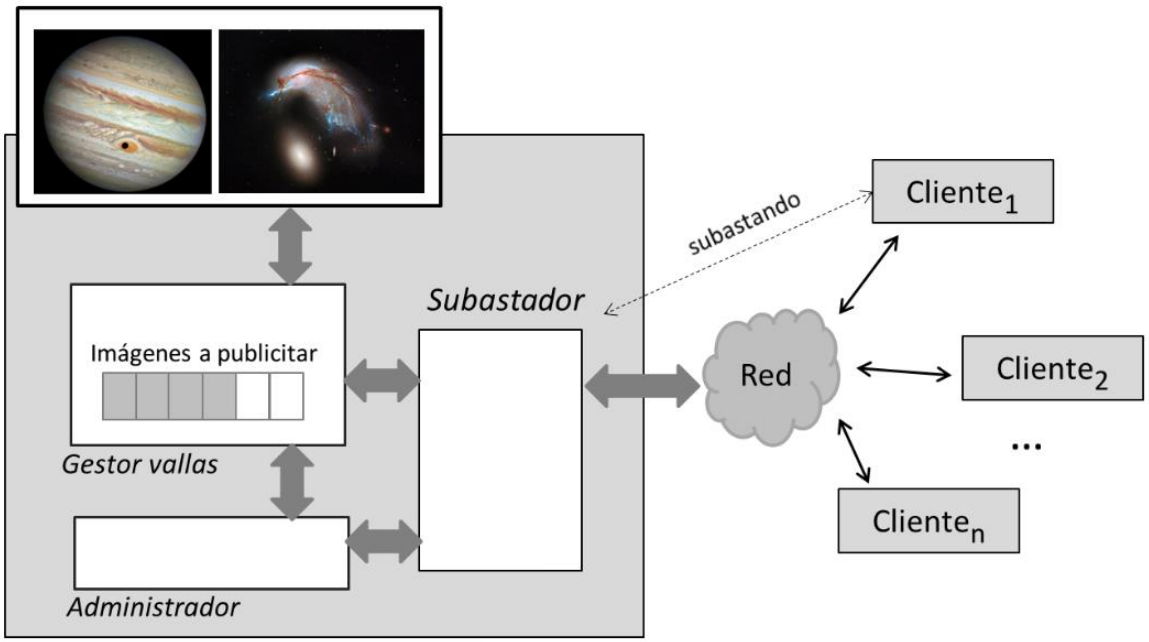


Figura 1: Esquema del sistema

La empresa tiene dos vallas en las que se van mostrando las imágenes que los clientes solicitan tras ganar la subasta.

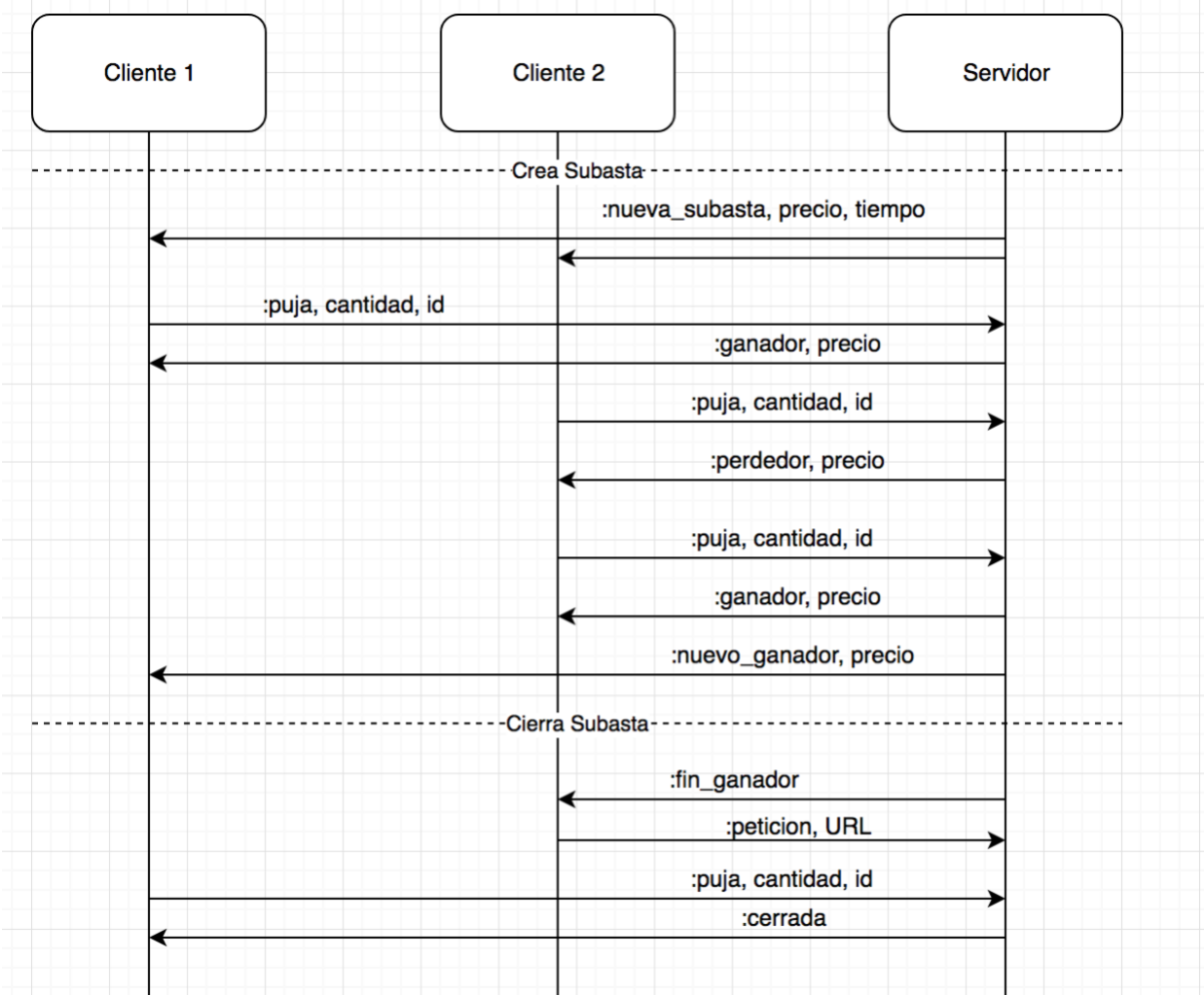
Para ello el sistema consta de tres módulos: el módulo subastador, encargado de crear y cerrar las subastas; el módulo gestor de vallas, encargado de atender las peticiones encoladas y mostrando las imágenes; y el módulo administrador, encargado de atender órdenes del administrador y de la terminación del sistema.

COMUNICACIÓN CLIENTE-SERVIDOR

La comunicación entre los procesos distribuidos del sistema se realizará mediante sockets TCP síncronos, esto implica que las llamas *send* y *recv* son bloqueantes.

Todos los mensajes se codifican para poder enviar múltiples datos en uno único. El proceso que envíe un mensaje utilizará el caracter separador '#', de esta forma, al recibir el mensaje tenemos que decodificarlo, obteniendo un vector con todos los datos.

El siguiente diagrama de secuencias muestra el proceso que siguen los clientes/servidor cuando se crea una nueva subasta.



La finalización de los procesos se lleva a cabo al introducir un comando ('EOS') por la entrada estándar. Al recibir dicho mensaje se envía y se procede a cerrar el socket y finalizar el proceso.

CLIENTE

Al lanzar un cliente se le especifican por la entrada estándar la dirección IP y puerto en la que está alojado el servidor y la URL de la imagen que desea mostrar. Opcionalmente se le puede añadir el parámetro "AUTO" para que funcione puje en las subastas de manera automática.

Inicialmente crea el socket e intenta conectarse a través de él al servidor. Una vez hemos establecido la conexión, queda a la espera de recibir una subasta por parte del servidor.

Cuando recibe la subasta muestra los datos por pantalla y espera a recibir la puja del usuario por la entrada estándar, enviandola al servidor.

Una vez enviada la puja, recibe un mensaje que le puede indicar:

{"PERDEDOR"#Precio_subasta}

La puja enviada por el cliente no supera la actual, recibe el precio actual de la subasta

{"GANADOR"#Precio_subasta}

La puja enviada por el cliente es la más alta hasta el momento.

El cliente queda bloqueado hasta recibir otro mensaje por parte del servidor en el que le puede indicar:

{"FIN_GANADOR"} .El usuario ha ganado la subasta actual y le envía al servidor la URL de la imagen que desea mostrar.

{"NUEVO_GANADOR"#Precio_subasta}. La puja ha sido superada por otro cliente, se recibe el precio actual de la puja.

El usuario también puede pasar de la subasta actual si no le interesa, para ello debe introducir el comando 'EOB' (End of Bid), enviándole un mensaje al servidor y quedando a la espera de que se abra una nueva subasta.

Por otro lado, si desea terminar el servicio porque ya ha acabado de realizar sus pujas, debe introducir el comando 'EOS' (End of Service), enviándole al servidor el mensaje para que cierre la comunicación.

SERVIDOR

Al lanzar el servidor se le indica por la entrada estándar como parámetro el número de puerto en el que se va a alojar.

Inicialmente crea el socket, descarga una imagen por defecto para mostrar en las vallas cuando no haya peticiones e inicializa las clases *Subasta* y *Valla.*

Antes de comenzar a atender a los clientes lanzamos los módulos del sistema (gestor de valla, subastador y administrador) pasándoles los parámetros necesarios a cada uno de ellos.

Finalmente comenzamos a aceptar conexiones de los clientes, las cuales atenderemos lanzando un thread nuevo (dispatcher) para atender simultáneamente a distintos clientes conectados para pujar en las subastas.

DISPATCHER

Se trata de un proceso que ejecuta el servidor por cada cliente que se conecte al mismo. De esta forma, hay un proceso dispatcher atendiendo a cada cliente. Así pues, este opera hasta que se ponga a fin la variable de fin del servidor, o se reciba un mensaje "EOS" (End Of Service) por parte del cliente al que atiende.

En primer lugar, el proceso se queda bloqueado en el monitor subasta hasta que empiece una subasta, para enviar al cliente los datos de la misma {Precio#Tiempo} y recibir su petición.

Al recibir la puja del cliente comprobamos que la subasta esté abierta y que esté pujando sobre la subasta actual, en caso contrario se desecha la petición y se le comunica al cliente.

Si la subasta está abierta, hacemos la puja y comprobamos si somos la puja más alta hasta el momento. En caso hacer una puja inferior, notificamos al cliente que su puja no ha superado la actual {"SUBASTA_CERRADA"}, y en caso de ser el ganador, enviamos la notificación al cliente {"GANADOR"#Precio_ganador} y quedamos bloqueados hasta que se termine la puja o alguien nos supere. Si alguien nos supera la puja, simplemente avisamos al cliente {"NUEVO_GANADOR"#Precio_ganador} y volvemos a esperar recibir una nueva petición. Si por el contrario, somos el ganador final de la subasta, notificamos al cliente {"FIN_GANADOR"#Precio_ganador} y esperamos recibir la URL de la imagen que desea mostrar para añadir una petición en el monitor de las vallas con el tiempo de la subasta y la URL de la imagen.

MÓDULO ADMINISTRADOR

Es el encargado de mostrar información del sistema y de la terminación ordenada del servicio.

Para interactuar con él debemos introducir comandos por la entrada estándar, los comandos pueden ser:

> **START :** Da permiso al subastador para comenzar a crear subastas al inicio del sistema, solo lo reconoce una vez.
>
> **PRINTH:** Obtiene y muestra los datos estadísticos de la valla: número de imágenes, tiempo de las imágenes en las vallas, etc.
>
> **PRINTS:** Obtiene y muestra la información actual del estado del sistema: número de peticiones recibidas, tiempo contratado, tiempo total, etc.
>
> **EOS:** Inicia la terminación ordenada del sistema. Activa la variable FIN_SERVICIO para que el resto de módulos no permitan nuevas peticiones y espera a que estos finalicen. Primero cierra el módulo subastas y espera hasta que acabe la subasta actual, después cierra el módulo de las vallas publicitarias, esperando a que se muestren todos las imagenes que queden en la cola del monitor. Finalmente cierra el socket y termina el sistema.

MÓDULO SUBASTADOR

Es el encargado de crear y cerrar las subastas. Para ello, mientras no reciba la señal de fin de servicio, crea una subasta para poder mostrar una imagen en una de las vallas por un tiempo aleatorio con un un precio inicial establecido.

Una vez creada la subasta, el proceso duerme para simular un tiempo de duración de la subasta y al despertar la cierra, impidiendo a los clientes hacer nuevas pujas sobre la subasta y quedando bloqueado a la espera de que el ganador de la puja notifique su petición y le despierte, momento en el que vuelve a crear una nueva subasta.

MÓDULO GESTOR DE VALLAS

Es el encargado de mostrar las imágenes en las vallas. Inicialmente crea dos vallas con la imagen por defecto descargada previamente y quedando a la espera de recibir una petición y que haya hueco en alguna valla.

Una vez obtiene la petición, descarga la imágen y lanza un thread encargado de imprimirla por la valla durante un tiempo que recibimos en los datos de la petición, finalmente el thread termina mostrando la imagen por defecto nuevamente.

BIBLIOTECAS

VALLA

Se trata de un monitor que tiene el fin de administrar las dos vallas que se inician en el sistema y de la misma manera el acceso a las mismas, así pues, el monitor realiza acciones como retener encolados los clientes que esperan su turno para mostrar su imagen en una de las valla. También realiza bloqueos a los clientes que están esperando a poder realizar una petición o aquellos que están esperando a que termine su imagen de ser mostrada en la valla.

Se cuenta con las siguientes funciones básicas para el tratamiento de la valla:

addPeticion(): Se añade una petición a la valla sí y solo sí la valla se encuentra en servicio y el número de peticiones en la valla es menor que el máximo de peticiones aceptadas por el monitor, Para añadirla, tras cumplir el requisito anterior, se encola la tupla formada por el string (la URL de la imagen) y el entero (que será el tiempo de muestra de la imagen) a la cola "*peticion*", se actualiza los datos para las estadísticas, tanto "*num_peticiones*" como "*tiempo_total*". Por último, se notifica al monitor que hay una petición para ser atendida.

atenderPeticion(): Se va a explicar el funcionamiento del algoritmo de atender peticiones. Al principio, si no hay ninguna petición, se queda bloqueado a la espera de que lleguen peticiones mediante la variable condición "*espera_peticion*". Luego, si no hay ventana disponible para procesar la petición, se vuelve a quedar bloqueado a la espera de que haya una ventana libre para ser utilizada mediante la variable condición "espera_ventana". Tras, la espera, si fuera necesaria, para tener una ventana, la actualiza como que esa ventana va a ser ocupada. Después, se saca la petición de la cola a la que se había añadido en la funcion *addPeticion,* y actualiza las variables estadísticas. Por último, se devuelve mediante una tupla el número de ventana utilizada, y los parámetro de la tupla desencolada, string(URL) y int(time).

finPeticion(): La principal acción de esta función, es liberar la ventana ocupada, por el parámetro de entrada "*n_ventana*", y actualizar todas las variables estadísticas. Por último, se notifica a que estén esperando una ventana mediante la variable condición "*espera_ventana*". Si se ha terminado el servicio de las vallas y no hay más peticiones encoladas, notificamos al cierre total del servicio mediante la variable condición "*espera_fin*".

***cerrarServicio()*:** Este proceso actualiza la variable booleana "*fin_servicio*" a true, y espera a que si hay todavía procesos encolados, éstos hayan notificado su salida, y den paso al cierre total del servicio mediante la variable condición "*espera_fin*".

```
class Valla {

    public:

        /* --------- CONSTRUCTORES ------------------------------------------------*/
        Valla();

        /* --------- GETTERS ------------------------------------------------------*/
        int getNum_peticiones();
        int getNum_imagenes();
        int getTiempo_estimado();
        time_t getTiempo_total();
        time_t getTiempo_imagenes_mostradas();

        /* --------- FUNCIONES ----------------------------------------------------*/

        // Añade una peticion {img, tmp} a la cola de peticiones
        void addPeticion(const string img,const int tmp);

        // Devuelve la peticion de la cola de peticiones y el número de ventana
        // libre en el que se debe mostrar {n_ventana, img, tmp}
        tuple<int, string, int> atenderPeticion();

        // Avisa de la finalización de la petición y libera la ventana
        void finPeticion(const int tmp, const int n_ventana);

        // Cierra el servicio y no permite atender nuevas peticiones
        void cerrarServicio();



    private:
        /* ---------  ATRIBUTOS ---------------------------------------------------*/
        mutex mtx;
        bool ventanas_libres[MAX_VENTANAS];     // indica el estado de cada ventana
        bool fin_servicio;                      // indica la finalización del servicio
        int n_libres;                           // nÚmero de ventanas libre
        queue<tuple<string, int>> peticiones;   // cola con tuplas {img, tiempo}
        time_t tiempoEspera;

        // Estadisticas
        int num_peticiones;
        int num_imagenes;
        time_t tiempo_total;
        time_t tiempo_imagenes_mostradas;

        condition_variable espera_peticion, espera_ventana, espera_fin;

};
```

SUBASTA

Se trata de un monitor que tiene el fin de administrar las subastas que se inician en el sistema y de la misma manera el acceso a las mismas, así pues, realiza acciones como retener encolados los clientes que esperan el inicio de una subasta, o al proceso que es ganador esperando el fin de la subasta o una subasta mayor.El módulo emplea las siguientes funciones básicas:

> ***iniciarSubasta()***: función la cual se llama con un tiempo para la subasta y un precio inicial y cuya función es iniciar una puja con esos parámetros y avisar a todos los procesos que estén esperando una nueva subasta.

cerrarSubasta(): como su nombre indica, cierra la subasta, devolviendo los valores a los por defecto, y avisa al ganador de que se ha cerrado la subasta y el es que ha ganado. Una vez despertado, se duerme para esperar a que el ganador termine con sus trámites de ganador.

cerrarServicio(): si hay una subasta activa, se duerme hasta que esta termine, y una vez terminada pone a true la variable de fin de las subastas "fin_servicio".

***entrarSubasta()*:** un proceso que entra a esta función del monitor, espera a que se inicie la puja cuyo "id" se pasa como parámetro, y una vez que empieza recoge sus valores.

pujar(): a través de esta función, un proceso puede realizar una puja, de forma que si no mejora el precio devuelve los valores de la puja más alta, y en caso de superarlos, despierta al ganador y se marca el mismo como el ganador.

dormirLider(): la cual ejecuta un proceso cuando supera el precio de la puja más alta, para bloquearse mientras siga siendo el ganador.

avisarSubastador(), con la cual el ganador avisa al subastador que ha terminado de llevar a cabo sus trámites como ganador, y por tanto puede empezar otra subasta.

```
class Subasta{

    public:
        //* ------- CONSTRUCTORES -----------------------------------------------------*/
        Subasta();

        //* ------- GETTER ------------------------------------------------------------*/
        int getPrecio_subasta();
        int getNum_subastas();
        int getTiempo_subasta();
        bool getActiva();

        //* ------- FUNCIONES ---------------------------------------------------------*/
        //inicia una subasta
        void iniciarSubasta(const int precio, const int tiempo);
        //Un proceso intenta acceder una subasta y se bloquea hasta que se inicia la
        //subasta deseada
        int entrarSubasta(const int i);
        //cierra la subasta actual
        void cerrarSubasta();
        //cuando la ultima subasta acabe pone a true la variable de fin de subastas
        void cerrarServicio();
        //realiza una puja en la subasta actual
        int pujar(const int id, const int precio);
        //duerme un proceso
        void dormirLider();
        //avisa al subastador de que debe continuar
        void avisarSubastador();
        //despierta un proceso
        void despertar();
        //indica si se ha alcanzado un numero de subastas determinado
        bool maxSubastas(const int max);

    private:
        //* ------- ATRIBUTOS ---------------------------------------------------------*/
        mutex mtx;
        bool fin_servicio;  //True -> servicio terminado y por tanto no se permiten
                            //más subastas
        bool activa;        //False -> Subasta sin empezar True -> Subasta empezada
        int precio_subasta;     //precio maximo por el que va la subasta
        int id_ganador;         //id del ganador en ese momento de la subasta
        int num_subastas;       //numero de subastas realizadas hasta el momento
        int tiempo_subasta;     //tiempo que tiene el cliente para pujar en la subasta
        bool ganador_pendiente; //indica si hay un ganador de la subasta en este
                                //momento

        condition_variable espera, espera_ganador;//var.cond para quedarte bloqueado
};
```

BIBLIOTECAS ADICIONALES

Socket e ImageDonwnloader no modificados respecto a los proporcionados en la asignatura.