LABORATORIO DE SISTEMAS OPERATIVOS
(Curso 2013/2014)

PRÁCTICA 3
Aplicación Multiproceso sobre HTTP – Ejemplo -

Objetivos del documento

Este documento presenta un ejemplo, en el contexto de desarrollo planteado en el enunciado de la práctica 3, de cómo implementar un servidor web sencillo capaz de atender a una única petición HTTP efectuada a través de una conexión con un cliente.

El ejemplo se desarrolla mediante un proceso incremental en el que cada una de las fases tiene como objetivo que el alumno se familiarice con el uso y funcionalidad de las principales funciones del módulo `http` y comprenda el comportamiento básico de un proceso encargado de gestionar una conexión con un cliente.

Se recomienda que el alumno escriba, compile y ejecute cada una de las fases.

Paso 1: Lectura de parámetros

```
#include <stdio.h>
#include <stdlib.h>

#include "param.h"
#include "http.h"

int main(int argc, char** argv)
{
    ParametrosServidor param;

    /* Procesar los parámetros de configuración */
    analizar_linea_mandatos ( argc, argv, &param );
    if ( param.nivel_depuracion > 0 )
      fprintf ( stderr, "\nPuerto del Servidor:: %d\n", param.puerto_escucha );

    /* comprobar que param.directorio_base existe y es un directorio */

    return(0);
}
```

Paso 2: Creación de GestorHTTP

```
#include <stdio.h>
#include <stdlib.h>

#include "param.h"
#include "http.h"

int main(int argc, char** argv)
{
   ParametrosServidor param;
   GestorHTTP* gHttp;

   /* Procesar los parámetros de configuración */
   analizar_linea_mandatos ( argc, argv, &param );
   if ( param.nivel_depuracion > 0 )
     fprintf ( stderr, "\nPuerto del Servidor:: %d\n", param.puerto_escucha );

   /* comprobar que param.directorio_base existe y es un directorio */

   /* Crear gestorHTTP */
   if ( ( gHttp = http_crear ( param.puerto_escucha ) ) == NULL ){
        fprintf( stderr, "\n....ERROR: No se ha podido crear el gestor" );
        exit(1);
   }

   /* descriptor asociado al puerto de escucha */
   if ( param.nivel_depuracion > 0 )
     fprintf ( stderr, "\nDescriptor del gestor :: %d\n",
             http_obtener_descriptor_escucha ( gHttp ) );

   /* destrucción del gestor */
   http_destruir ( gHttp );
   return(0);
}
```

Paso 3: Esperar conexión de cliente

```
#include <stdio.h>
#include <stdlib.h>

#include "param.h"
#include "http.h"

int main(int argc, char** argv)
{
   ParametrosServidor param;
   GestorHTTP* gHttp;
   ConexionHTTP* cliente;

   /* Procesar los parámetros de configuración */
   analizar_linea_mandatos ( argc, argv, &param );
   if ( param.nivel_depuracion > 0 )
     fprintf ( stderr, "\nPuerto del Servidor:: %d\n",param.puerto_escucha);

   /* comprobar que param.directorio_base existe y es un directorio */

   /* Crear gestorHTTP */
   if ( (gHttp = http_crear ( param.puerto_escucha ) ) == NULL){
        fprintf( stderr, "\n....ERROR: No se ha podido crear el gestor");
        exit(1);
   }

   /* Esperar conexión de cliente */
   switch ( http_esperar_conexion ( gHttp, &cliente, 1 ) )
   {
        case HTTP_OK:
          if ( param.nivel_depuracion > 0 )
            fprintf ( stderr, "\nConexión de Cliente con IP %s establecida.\n",
                   http_obtener_ip_cliente ( cliente ) );

          /* cerrar conexión */
          http_cerrar_conexion ( cliente );
          break;

        case HTTP_SEGUIR_ESPERANDO:
        case HTTP_ESPERA_INTERRUMPIDA:
          break;

        case HTTP_ERROR:
          fprintf ( stderr, "\n...ERROR: http_esperar_conexión" );
          break;
   }

   /* destrucción del gestor */
   http_destruir ( gHttp );
   return(0);
}
```

Prueba: establecer una conexión con el servidor utilizan alguno de los mecanismos presentados en el enunciado de la práctica 3.

Paso 4: Esperar petición de cliente

```
#include <stdio.h>
#include <stdlib.h>

#include "param.h"
#include "http.h"

void atender_cliente ( ConexionHTTP* cliente );

/* variables globales */
ParametrosServidor param;

int main(int argc, char** argv)
{
   GestorHTTP * gHttp;
   ConexionHTTP* cliente;

   /* Procesar los parámetros de configuración */
   analizar_linea_mandatos ( argc, argv, &param );
   if ( param.nivel_depuracion > 0 )
     fprintf ( stderr, "\nPuerto del Servidor:: %d\n",param.puerto_escucha);

   /* comprobar que param.directorio_base existe y es un directorio */

   /* Crear gestorHTTP */
   if ( (gHttp = http_crear ( param.puerto_escucha ) ) == NULL){
        fprintf( stderr, "\n....ERROR: No se ha podido crear el gestor");
        exit(1);
   }

   /* Esperar conexión de cliente */
   switch ( http_esperar_conexion ( gHttp, &cliente, 1 ) )
   {
        case HTTP_OK:
          if ( param.nivel_depuracion > 0 )
            fprintf ( stderr, "\nConexión de Cliente con IP %s establecida.\n",
                   http_obtener_ip_cliente(cliente) );

          /* atender cliente */
          atender_cliente ( cliente );

          /* cerrar conexión */
          http_cerrar_conexion( cliente );
          break;

        case HTTP_SEGUIR_ESPERANDO:
        case HTTP_ESPERA_INTERRUMPIDA:
          break;

        case HTTP_ERROR:
          fprintf ( stderr, "\n...ERROR: http_esperar_conexión" );
          break;
   }

   /* destrucción del gestor */
   http_destruir ( gHttp );
   return(0);
}
```

```
void atender_cliente ( ConexionHTTP* cliente )
{
   Peticion* peticion;
   int estado;

   /* leer peticion */
   switch ( estado = http_leer_peticion ( cliente, &peticion, 1 ) )
   {
        case HTTP_OK:
          if ( param.nivel_depuracion > 0 )
            fprintf ( stderr, "Petición: IP=%s, RECURSO: %s\n",
                   peticion->ip_cliente, peticion->fichero );

          http_destruir_peticion ( peticion );
          break;

        case HTTP_CLIENTE_DESCONECTADO:
          if ( param.nivel_depuracion > 0 )
            fprintf ( stderr, "\n....ERROR: Detectado cliente desconectado\n");
          break;

        case HTTP_ESPERA_INTERRUMPIDA:
          break;

        case HTTP_ERROR:
          fprintf ( stderr, "\n...ERROR: Error http_leer_peticion \n");
          fflush ( stderr );
          break;
   }
}
```

Prueba: ejecutar el servidor utilizando como directorio base el directorio `paginas` referenciado en el enunciado de la práctica 3 y solicitar el recurso ‘basica.htm’.

Paso 5: Procesar petición de cliente

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/param.h>

#include "param.h"
#include "http.h"

void atender_cliente ( ConexionHTTP* cliente );
void atender_peticion (ConexionHTTP* cliente, Peticion* peticion );

/* variables globales */
ParametrosServidor param;

int main(int argc, char** argv)
{
    GestorHTTP * gHttp;
    ConexionHTTP* cliente;

    /* Procesar los parámetros de configuración */
    analizar_linea_mandatos ( argc, argv, &param );
    if ( param.nivel_depuracion > 0 )
      fprintf ( stderr, "\nPuerto del Servidor:: %d\n",param.puerto_escucha );

    /* comprobar que param.directorio_base existe y es un directorio */

    /* Crear gestorHTTP */
    if ( (gHttp = http_crear ( param.puerto_escucha ) ) == NULL){
         fprintf( stderr, "\n....ERROR: No se ha podido crear el gestor");
         exit(1);
    }

    /* Esperar conexión de cliente */
    switch ( http_esperar_conexion ( gHttp, &cliente, 1 ) )
    {
         case HTTP_OK:
           if ( param.nivel_depuracion > 0 )
             fprintf ( stderr, "\nConexión de Cliente con IP %s establecida.\n",
                    http_obtener_ip_cliente ( cliente) );

           /* atender cliente */
           atender_cliente ( cliente );

           /* cerrar conexión */
           http_cerrar_conexion( cliente );
           break;

         case HTTP_SEGUIR_ESPERANDO:
         case HTTP_ESPERA_INTERRUMPIDA:
           break;

         case HTTP_ERROR:
           fprintf ( stderr, "\n...ERROR: http_esperar_conexión" );
           break;
    }

    /* destrucción del gestor */
    http_destruir ( gHttp );
```

```
    return(0);
}

void atender_cliente ( ConexionHTTP* cliente )
{
    Peticion* peticion;
    int estado;

    /* leer peticion */
    switch ( estado = http_leer_peticion ( cliente, &peticion, 1 ) )
    {
         case HTTP_OK:
           if ( param.nivel_depuracion > 0 )
             fprintf ( stderr, "Petición: IP=%s, RECURSO: %s\n",
                    peticion->ip_cliente, peticion->fichero );

           /* servir el recurso si es un fichero y se tienen permisos
            de lectura */
           atender_peticion( cliente, peticion );

           http_destruir_peticion ( peticion );
           break;

         case HTTP_CLIENTE_DESCONECTADO:
           if ( param.nivel_depuracion > 0 )
             fprintf ( stderr, "\n....ERROR: Detectado cliente desconectado\n");
           break;

         case HTTP_ESPERA_INTERRUMPIDA:
           break;

         case HTTP_ERROR:
           fprintf ( stderr, "\n...ERROR: Error en http_leer_peticion \n");
           break;
    }
}
```

```
void atender_peticion ( ConexionHTTP* cliente, Peticion* peticion )
{
    char nombre_fichero[MAXPATHLEN];
    const char* CGI_EXT = ".cgi";


    /* path incorrecto - intento de violación de seguridad */
    if ( strstr ( peticion->fichero, "/../" ) ||
         ! strncmp ( peticion->fichero, "../", 3 ) ){
         http_enviar_codigo ( cliente, peticion, 403, "No autorizado" );

         if ( param.nivel_depuracion > 0 )
           fprintf ( stderr, "... PETICION_ERROR: fichero no autorizado:: %s\n",
                   peticion->fichero );

         return;
    }

    /* composición del path físico como concatenación de directorio_base
       y nombre del recurso */
    strcpy ( nombre_fichero, param.directorio_base );
    strcat ( nombre_fichero, peticion->fichero );

    /* procesar solicitud de recurso o de cgi */
    if ( strcmp ( peticion->fichero + strlen(peticion->fichero) –
                  strlen ( CGI_EXT ), CGI_EXT ) ){
         fprintf ( stderr, "... PETICION_RECURSO: recurso solicitado:: %s\n",
                    nombre_fichero);
         fflush ( stderr );

    }
    else {
         if ( param.nivel_depuracion > 0 )
           fprintf ( stderr, "... PETICION_CGI: cgi solicitado:: %s\n",
                   nombre_fichero);
    }

}
```

Prueba: ejecutar el servidor utilizando como directorio base el directorio `paginas` referenciado en el enunciado de la práctica 3 y solicitar el recurso ‘basica.htm’. Probar a solicitar otro tipo de recursos como CGIs, imágenes, directorios,…

Paso 6: Enviar respuesta al cliente

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/param.h>
#include <fcntl.h>
#include <unistd.h>

#include "param.h"
#include "http.h"

void atender_cliente ( ConexionHTTP* cliente );
void atender_peticion ( ConexionHTTP* cliente, Peticion* peticion );
void enviar_recurso ( ConexionHTTP* cliente, Peticion* peticion,
                      char* nombre_fichero);

/* variables globales */
ParametrosServidor param;

int main(int argc, char** argv)
{
   GestorHTTP * gHttp;
   ConexionHTTP* cliente;

   /* Procesar los parámetros de configuración */
   analizar_linea_mandatos ( argc, argv, &param );
   if ( param.nivel_depuracion > 0 )
     fprintf ( stderr, "\nPuerto del Servidor:: %d\n",param.puerto_escucha );

   /* comprobar que param.directorio_base existe y es un directorio */

   /* Crear gestorHTTP */
   if ( (gHttp = http_crear ( param.puerto_escucha ) ) == NULL){
        fprintf( stderr, "\n....ERROR: No se ha podido crear el gestor" );
        exit(1);
   }

   /* Esperar conexión de cliente */
   switch ( http_esperar_conexion ( gHttp, &cliente, 1 ) )
   {
        case HTTP_OK:
          if ( param.nivel_depuracion > 0 )
            fprintf ( stderr, "\nConexión de Cliente con IP %s establecida.\n",
                   http_obtener_ip_cliente ( cliente ) );

          /* atender cliente */
          atender_cliente ( cliente );

          /* cerrar conexión */
          http_cerrar_conexion ( cliente );
          break;

        case HTTP_SEGUIR_ESPERANDO:
        case HTTP_ESPERA_INTERRUMPIDA:
          break;

        case HTTP_ERROR:
          fprintf ( stderr, "\n...ERROR: http_esperar_conexión" );
```

```
            break;
    }

    /* destrucción del gestor */
    http_destruir ( gHttp );
    return(0);
}

void atender_cliente ( ConexionHTTP* cliente )
{
    Peticion* peticion;
    int estado;

    /* leer peticion */
    switch ( estado = http_leer_peticion ( cliente, &peticion, 1 ) )
    {
         case HTTP_OK:
           if ( param.nivel_depuracion > 0 )
             fprintf ( stderr, "Petición: IP=%s, RECURSO: %s\n",
                     peticion->ip_cliente, peticion->fichero );

           /* servir el recurso si es un fichero y se tienen permisos
            de lectura */
           atender_peticion( cliente, peticion );
           http_destruir_peticion ( peticion );
           break;

         case HTTP_CLIENTE_DESCONECTADO:
           if ( param.nivel_depuracion > 0 )
             fprintf ( stderr, "\n....ERROR: Detectado cliente desconectado\n");

           break;

         case HTTP_ESPERA_INTERRUMPIDA:
           break;

    case HTTP_ERROR:
           fprintf ( stderr, "\n...ERROR: Error en http_leer_peticion\n");
           break;
    }
}
```

```
void atender_peticion (ConexionHTTP* cliente, Peticion* peticion )
{
   char nombre_fichero[MAXPATHLEN];
   const char* CGI_EXT = ".cgi";


   /* path incorrecto - intento de violación de seguridad */
   if ( strstr ( peticion->fichero, "/../" ) ||
        ! strncmp ( peticion->fichero, "../", 3 ) ){
        http_enviar_codigo ( cliente, peticion, 403, "No autorizado" );

        if ( param.nivel_depuracion > 0 )
          fprintf ( stderr, "... PETICION_ERROR: fichero no autorizado:: %s\n",
                  peticion->fichero);

        return;
   }

   /* composición del path físico como concatenación de directorio_base
      y nombre del recurso */
   strcpy ( nombre_fichero, param.directorio_base );
   strcat ( nombre_fichero, peticion->fichero );

   /* procesar solicitud si no es cgi */
   if ( strcmp ( peticion->fichero + strlen(peticion->fichero) –
                 strlen ( CGI_EXT ), CGI_EXT ) ){
        enviar_recurso ( cliente, peticion, nombre_fichero );
   }
   else {
        if ( param.nivel_depuracion > 0 )
          fprintf ( stderr, "... PETICION_CGI: cgi solicitado:: %s\n",
                  nombre_fichero);
   }
}
```

```
/*
 * Enviar el recurso estático solicitado o el fichero por defecto
 * si se trata de un directorio (NO implementado)
 */
void enviar_recurso( ConexionHTTP* cliente, Peticion* peticion,
                     char* nombre_fichero )
{
    int fd;
    struct stat st;

    if ( stat ( nombre_fichero, &st ) == -1 ) {
         http_enviar_codigo ( cliente, peticion, 500,
                              "Problema indeterminado" );
         return;
    }

    if ( S_ISREG(st.st_mode) ) {
         if ( ( fd = open(nombre_fichero, O_RDONLY) ) == -1 ){
           http_enviar_codigo ( cliente, peticion, 404, "No encontrado" );

           if ( param.nivel_depuracion > 0 )
             fprintf ( stderr, "...ERROR: Error en el acceso al recurso:: %s
                                    \n", nombre_fichero );
           return;
         }

         http_enviar_respuesta ( cliente, peticion, fd, 1 );
         close(fd);

         if ( param.nivel_depuracion > 0 )
           fprintf ( stderr, " Recurso %s enviado\n", nombre_fichero );
    }
    else if ( S_ISDIR(st.st_mode) ){
         if ( param.nivel_depuracion > 0 )
           fprintf ( stderr, "...INFO: Se ha solicitado un directorio:: %s \n",
                    nombre_fichero );
    }
    else{
         if ( param.nivel_depuracion > 0 )
           fprintf ( stderr, "...ERROR: Se ha solicitado un recurso no permitido
                   :: %s \n", nombre_fichero );

         http_enviar_codigo ( cliente, peticion, 403, "No autorizado" );
    }

}
```

Prueba: ejecutar el servidor utilizando como directorio base el directorio `paginas` referenciado en el enunciado de la práctica 3 y solicitar el recurso 'basica.htm'. Probar a solicitar otro tipo de recursos como CGIs, imágenes, directorios,…