

EXAMEN PARCIAL
ARQUITECTURAS CLIENTE SERVIDOR
FECHA DE ENTREGA: 03 DE ABRIL DEL 2018
NOMBRE: CHÁVEZ DELGADO JORGE LUIS
GRUPO: 02

1.-INVESTIGA QUÉ SIGNIFICA QUE UN PROCESO ESTÉ EN BACKGROUND Y QUÉ SIGNIFICA QUE UN PROCESO ESTÉ EN FOREGROUND.

Foreground: El proceso se está ejecutando en primer plano, un programa cuando en la terminal, no se puede ejecutar otro comando hasta que el proceso finalice.

Background: El proceso se está ejecutando en segundo plano, cuando un programa se ejecuta en segundo plano, no se usa la terminal donde se ejecutó hasta que se termina, para ejecutar un comando o programa en segundo plano a la instrucción se le agrega al final un &, el sistema operativo lo ejecuta en segundo plano

INVESTIGA QUÉ COMANDO PONE UN PROCESO EN BACKGROUND Y QUÉ COMANDO PONE UN PROCESO EN FOREGROUND.

Background:

comando **&** o ejecutar el programa, presionar ctrl+Z y **bg**

Foreground:

fg comando

fg número_proceso

ESCRIBE EL SIGUIENTE CÓDIGO Y EJECÚTALO. APLICA EL COMANDO QUE INVESTIGASTE PARA PONER EL PROCESO EN BACKGROUND Y APLICA EL OTRO COMANDO PARA PONERLO EN FOREGROUND. EN ESTE INCISO RESPONDE:

```

132.248.59.6 - PuTTY
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Proceso a dormir para que lo pongas en background\n");
    sleep(120);
    exit(0);
}

"sleep.c" 10L, 143C

```

LAS 2 PREGUNTAS Y AGREGA LA CAPTURA DE PANTALLA DE LA EJECUCIÓN DEL CÓDIGO, CUANDO COLOCASTE EL PROCESO EN BACKGROUND Y EN FOREGROUND.

- Ejecutamos el programa

```

zaira@zaira-Satellite-L55Dt-C: ~/Escritorio/arqui
Archivo Editar Ver Buscar Terminal Ayuda
zaira@zaira-Satellite-L55Dt-C:~/Escritorio/arqui$ gcc sleep.c -o sleep
sleep.c: In function 'main':
sleep.c:6:2: warning: implicit declaration of function 'sleep' [-Wimplicit-funct
ion-declaration]
    sleep(120);
    ^~~~~~
zaira@zaira-Satellite-L55Dt-C:~/Escritorio/arqui$ ./sleep
Proceso a dormir para que lo pongas en background
^Z
[1]+  Detenido                  ./sleep
zaira@zaira-Satellite-L55Dt-C:~/Escritorio/arqui$

```

- Para ponerlo en background tecleamos *ctrl* + *Z* y tecleamos *bg*

```

zaira@zaira-Satellite-L55Dt-C: ~/Escritorio/arqui
Archivo Editar Ver Buscar Terminal Ayuda
zaira@zaira-Satellite-L55Dt-C:~/Escritorio/arqui$ gcc sleep.c -o sleep
sleep.c: In function 'main':
sleep.c:6:2: warning: implicit declaration of function 'sleep' [-Wimplicit-funct
ion-declaration]
    sleep(120);
    ^~~~~~
zaira@zaira-Satellite-L55Dt-C:~/Escritorio/arqui$ ./sleep
Proceso a dormir para que lo pongas en background
^Z
[1]+  Detenido                  ./sleep
zaira@zaira-Satellite-L55Dt-C:~/Escritorio/arqui$ bg
[1]+ ./sleep &
zaira@zaira-Satellite-L55Dt-C:~/Escritorio/arqui$

```

- Para revisar que se este ejecutando tecleamos *jobs*

```
zaira@zaira-Satellite-L55Dt-C: ~/Escritorio/arqui
Archivo Editar Ver Buscar Terminal Ayuda
zaira@zaira-Satellite-L55Dt-C:~/Escritorio/arqui$ gcc sleep.c -o sleep
sleep.c: In function 'main':
sleep.c:6:2: warning: implicit declaration of function 'sleep' [-Wimplicit-funct
ion-declaration]
    sleep(120);
    ^~~~~~
zaira@zaira-Satellite-L55Dt-C:~/Escritorio/arqui$ ./sleep
Proceso a dormir para que lo pongas en background
^Z
[1]+  Detenido                  ./sleep
zaira@zaira-Satellite-L55Dt-C:~/Escritorio/arqui$ bg
[1]+ ./sleep &
zaira@zaira-Satellite-L55Dt-C:~/Escritorio/arqui$ jobs
[1]+  Ejecutando                ./sleep &
zaira@zaira-Satellite-L55Dt-C:~/Escritorio/arqui$
```

- Cuando queremos regresar un proceso a primer plano debemos teclear *fg* num_proceso, en este caso es el 1

2. ESCRIBE EL SIGUIENTE CÓDIGO DE SERVIDOR

socket-server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>

int server (int client_socket)
{
    while (1)
    {
        int length;
        char* text;
        if (read (client_socket, &length, sizeof (length)) == 0)
            return 0;

        text = (char*) malloc (length);
        read (client_socket, text, length);
        printf ("%s\n", text);
        int cmp = strcmp (text, "quit");

        if (!cmp)
            return 1;
    }
}
```

```

int main (int argc, char* const argv[])
{
    int socket_fd;
    struct sockaddr_un name;
    int client_sent_quit_message;
    const char* const socket_name = argv[1];
    socket_fd = socket (AF_UNIX, SOCK_STREAM, 0);
    name.sun_family = AF_UNIX;
    strcpy (name.sun_path, socket_name);
    bind (socket_fd, (struct sockaddr *)&name, SUN_LEN (&name));
    listen (socket_fd, 1);

do
{
    struct sockaddr_un client_name;
    socklen_t client_name_len;
    int client_socket_fd;
    client_socket_fd = accept (socket_fd,
                              (struct sockaddr *)&client_name,
                              &client_name_len);

    client_sent_quit_message = server (client_socket_fd);
    close (client_socket_fd);
}while (!client_sent_quit_message);

close (socket_fd);
unlink (socket_name);
return 0;
}

```

ESCRIBE EL SIGUIENTE CÓDIGO DE CLIENTE

socket-cliente.c

```

#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>

void write_text (int socket_fd, const char* text)
{
    int length = strlen (text) + 1;
    write (socket_fd, &length, sizeof (length));
    write (socket_fd, text, length);
}

```

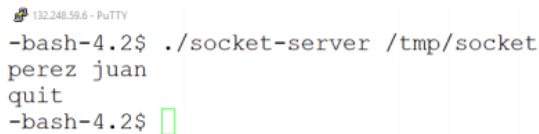
```

int main (int argc, char* const argv[])
{
    const char* const socket_name = argv[1];
    const char* const message = argv[2];
    int socket_fd;
    struct sockaddr_un name;
    socket_fd = socket (PF_LOCAL, SOCK_STREAM, 0);

    name.sun_family = AF_LOCAL;
    strcpy (name.sun_path, socket_name);
    connect (socket_fd, (struct sockaddr *)&name, SUN_LEN (&name));
    write_text (socket_fd, message);
    close (socket_fd);
    return 0;
}

```

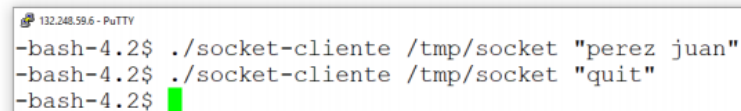
HAY QUE COMENTAR LOS 2 CÓDIGOS SENTENCIA POR SENTENCIA EJECUTALOS DE TAL MANERA QUE EL CLIENTE MANDE TU NOMBRE Y APELLIDO ENTRE COMILLAS Y OBSERVA QUE EL SERVIDOR LOS RECIBE Y LOS IMPRIME. CUANDO EL CLIENTE LANZA LA PALABRA “quit”, EL SERVIDOR LA IMPRIME Y TERMINA SU EJECUCIÓN.



```

132.248.59.6 - PuTTY
-bash-4.2$ ./socket-server /tmp/socket
perez juan
quit
-bash-4.2$ █

```



```

132.248.59.6 - PuTTY
-bash-4.2$ ./socket-client /tmp/socket "perez juan"
-bash-4.2$ ./socket-client /tmp/socket "quit"
-bash-4.2$ █

```

EN ESTE INCISO HAY ENTREGAR LOS 2 CÓDIGOS IMPRESOS Y COMENTADOS SENTENCIA POR SENTENCIA Y LA CAPTURA DE PANTALLA DE LA EJECUCIÓN DEL CLIENTE Y DEL SERVIDOR COMO EN EL EJEMPLO, DONDE PASES TU NOMBRE Y DESPUÉS PASES LA PALABRA “quit”.

socket_server.c

```

#include <stdio.h> //Biblioteca estándar de entrada y salida
#include <stdlib.h> //Biblioteca estándar de C
#include <string.h> //Biblioteca para el manejo de cadenas
#include <sys/socket.h> //Biblioteca para manejar sockets
#include <sys/un.h> //Biblioteca para manejar las familias de sockets
#include <unistd.h> //Biblioteca para utilizar POSIX
#define p printf //Definimos la letra p como si fuera un printf

int server(int client_socket){ //Se declara una función que recibe el socket
cliente
    while(1){ //Entramos a un ciclo infinito

```

```

        int length; //Variable para el tamaño del mensaje
        char* text; //Apuntador donde se guardará el mensaje
        if(read(client_socket,&length,sizeof(length)) == 0) /*Si la lectura
del cliente
sale.*/
            return 0;
        text = (char*) malloc (length); //Se reserva la memoria para el
mensaje
        read(client_socket,text,length); // Se lee del socket el mensaje y
su tamaño
        p("%s\n",text); //Se muestra en pantalla el mensaje
        int cmp = strcmp(text,"quit"); /*Se guarda en una variable la
comparación entre
el texto y la palabra "quit"*/
        if(!cmp) /*Si el mensaje no es igual a quit se cicla nuevamente,
en caso contrario sale del ciclo*/
            return 1;
    }
}

```

```

int main(int argc, char* const argv[]){

    int socket_fd; //File descriptor para el socket servidor

    struct sockaddr_un name; //Estructura socket servidor

    int client_sent_quit_message; /*Variable que almacena el 1 o 0
regresado por la función server*/

    //Cadena que almacena el nombre del socket
    const char* const socket_name = argv[1];

    socket_fd = socket(AF_UNIX,SOCK_STREAM,0); //Se crea el socket
AF_UNIX
    //Se inicializa el atributo sun_family de la struct
    name.sun_family = AF_UNIX;

    //Se inicializa el atributo sun_path con el nombre del archivo
    strcpy(name.sun_path,socket_name);
    //Se asocia el socket al archivo, para que sea un socket nombrado
    bind(socket_fd,(struct sockaddr *)&name,SUN_LEN (&name));
    //Se pone al servidor a escuchar conexiones entrantes de un cliente
    listen(socket_fd,1);
}

```

```

//Ciclo infinito para recibir mensajes
do{
    //Estructura para el socket cliente
    struct sockaddr_un client_name;
    //Variable para el tamaño del mensaje del cliente
    socklen_t client_name_len;
    //File descriptor del socket cliente
    int client_socket_fd;
    //El servidor acepta la conexión del cliente
    client_socket_fd = accept(socket_fd,(struct sockaddr
*)&client_name,&client_name_len);
    //Se guarda el valor de retorno de la función server
    client_sent_quit_message = server(client_socket_fd);
    //Se cierra el file descriptor del socket servidor
    close(socket_fd);
}while(!client_sent_quit_message); //Mientras el valor de retorno sea
falso, seguira
    //dentro del ciclo

//Se cierra el file descriptor del socket servidor
close(socket_fd);
//Se desliga el archivo del socket servidor
unlink(socket_name);
//Buena ejecución
return 0;
}

```

socket_client.c

```

#include <stdio.h> //Biblioteca estándar de entrada y salida
#include <string.h> //Biblioteca estándar de C
#include <sys/socket.h> //Utilizamos esta biblioteca para sockaddr y len
#include <sys/un.h> //Para poder usar sockaddr_un
#include <unistd.h> //Biblioteca para utilizar POSIX

void write_text (int socket_fd, const char* text) //creamos la función para
almacenar parámetros
{
    //del mensaje que mandaremos al
servidor

```

```

        int length = strlen (text) + 1;                //primero capturamos la
longitud del mensaje
        write (socket_fd, &length, sizeof (length)); //apartamos la memoria
necesaria
        write (socket_fd, text, length);              //y tenemos los parámetros
necesarios
    }

int main (int argc, char* const argv[]) //creamos la función para el envío del
mensaje
{
    const char* const socket_name = argv[1]; //tenemos un apuntador para
el socket
    const char* const message = argv[2]; //y uno mas para el mensaje
    int socket_fd; //creamos el file descriptor
    struct sockaddr_un name; //almacenamos las direcciones
    socket_fd = socket (PF_LOCAL, SOCK_STREAM, 0); //socket devolvera
al file descriptor

    name.sun_family = AF_LOCAL; //inicializamos sun_family
    strcpy (name.sun_path, socket_name); //inicializamos sun_path con el
nombre del archivo
    connect (socket_fd, (struct sockaddr *)&name, SUN_LEN (&name));
//conectamos el socket con los argumentos
    write_text (socket_fd, message);                    //file descriptor,
estructura y longitud
    close (socket_fd);                                  //escribe el mensaje y
finaliza el proceso
    return 0;
}

```



The screenshot shows a terminal window with the following text:

```

kubos@kubos:~/Desktop/ArquitecturaClienteServidor/CódigosExamen$ ./socket-server /tmp
/socket
Chávez Jorge
quit
kubos@kubos:~/Desktop/ArquitecturaClienteServidor/CódigosExamen$

```



```
kubos@kubos:~/Desktop/ArquitecturaClienteServidor/CódigosExamen$ ./socket-client /tmp  
/socket "Chávez Jorge"  
kubos@kubos:~/Desktop/ArquitecturaClienteServidor/CódigosExamen$ ./socket-client /tmp  
/socket "quit"  
kubos@kubos:~/Desktop/ArquitecturaClienteServidor/CódigosExamen$
```