



Universitat d'Alacant
Universidad de Alicante

GRADO EN INGENIERÍA INFORMÁTICA
CURSO ACADÉMICO 2025-2026

Sistemas Operativos

GRUPO 8 - Práctica 2



JULIÁN HINOJOSA GIL - DNI 48795869-N

Índice

1. Introducción	1
2. Objetivos	1
3. Desarrollo	1
3.1. Ejercicio 1	1
3.2. Ejercicio 2	1
4. Código Fuente	1
4.1. Código del Ejercicio 1	1
5. Conclusiones	3
6. Bibliografía	3

1. Introducción

Aquí va la introducción de la memoria. Este documento presenta el desarrollo y resultados de la práctica de Sistemas Operativos, donde se abordan diversos ejercicios relacionados con [completar según el tema de la práctica].

2. Objetivos

Los objetivos de esta práctica son:

- Comprender los conceptos fundamentales de [completar]
- Implementar soluciones prácticas utilizando [completar]
- Analizar el comportamiento y rendimiento de [completar]

3. Desarrollo

3.1. Ejercicio 1

En este ejercicio se nos pide desarrollar una aplicación cliente-servidor para poder transferir un archivo desde el servidor al cliente utilizando sockets en C. Usaremos una versión de la página de inicio de google como archivo de prueba.

El servidor se encargará de escuchar las conexiones entrantes y un proceso hijo dentro del mismo servidor se encargará de enviar el archivo al cliente una vez este se haya conectado. El archivo se enviará en bloques configurados por la macro `BUFFER_SIZE`. El servidor estará constantemente escuchando nuevas conexiones con un máximo de 5 conexiones en cola.

Mientras tanto, el cliente se encargará de conectarse al servidor y recibir el archivo en bloques, escribiendo cada bloque recibido en un archivo local. Una vez completado abrirá el archivo en el navegador por defecto del sistema.

3.2. Ejercicio 2

El ejercicio 2 busca implementar mediante el simulador jBACI el problema del productor-consumidor utilizando semáforos para la sincronización entre procesos. Para este ejercicio he reproducido el ejemplo de la transparencia vista en clase, adaptándolo al entorno de jBACI y añadiendo comentarios explicativos en el código para mayor claridad.

4. Código Fuente

A continuación se presenta el código implementado para los ejercicios:

4.1. Código del Ejercicio 1

Empezando con la parte del servidor, el siguiente código muestra la configuración del socket, del servidor y su enlace:

```
1 // Creación del socket, se gestiona el error si no se puede crear
2 sockfd = socket(AF_INET, SOCK_STREAM, 0);
3 if (sockfd == -1) {
4     fprintf(stderr, "Error al crear el socket");
```

```

5     exit(EXIT_FAILURE);
6 }
7 printf("Socket creado con éxito\n");
8
9 // Configuración de la estructura sockaddr_in
10 serverAddr.sin_port = htons(DEFAULT_PORT); // puerto del servidor
11 serverAddr.sin_family = AF_INET; // familia de direcciones IPv4
12 serverAddr.sin_addr.s_addr = INADDR_ANY; // aceptar conexiones en cualquier interfaz de red
13
14 // Enlazar el socket a la dirección y puerto especificados, con gestión de errores
15 if (bind(sockfd, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) != 0) {
16     fprintf(stderr, "Error al enlazar el socket");
17     exit(EXIT_FAILURE);
18 }
```

Listing 1: Configuracion del servidor

El `sin_port` se configura con la función `htons` para convertir el número de puerto al formato de red adecuado, `sin_family` se establece en `AF_INET` para indicar que se utilizará IPv4, y `sin_addr.s_addr` se configura con `INADDR_ANY` para aceptar conexiones en cualquier interfaz de red disponible.

Ahora el servidor se pone a escuchar conexiones entrantes y acepta una conexión cuando llega una nueva:

```

1 // Poner el socket en modo escucha para aceptar conexiones entrantes, maximo 5
2 // conexiones en cola
3 listen(sockfd, 5);
4
5 printf("Esperando nuevas conexiones...\n");
6
7 // Bucle infinito para aceptar y manejar conexiones entrantes
8 while(1) {
9     // Aceptar una nueva conexión de cliente
10    size = sizeof(clientAddr);
11    clientSocketfd = accept(sockfd, (struct sockaddr *)&clientAddr, &size);
12    if (clientSocketfd == -1) {
13        fprintf(stderr, "Error aceptando la conexión\n");
14        continue;
15    }
16
17    printf("Conexión aceptada!!!\n");
18 }
```

Listing 2: El servidor acepta conexiones

Luego de aceptar la conexión, se crea un proceso hijo para manejar la transferencia del archivo al cliente, y se envía el archivo en bloques:

```

1 // Leer y enviar el archivo por trozos en base al tamaño del buffer
2 while((bytesRead = read(filefd, buffer, sizeof(buffer))) > 0) {
3     bytesSent = write(clientSocketfd, buffer, bytesRead);
4     if (bytesSent == -1) {
5         fprintf(stderr, "Error en la transferencia del archivo\n");
6         break;
7     }
8 }
```

Listing 3: Envío del archivo en bloques

Ya después de eso simplemente se cierra el archivo y el socket del cliente en el proceso hijo, y el servidor vuelve a esperar nuevas conexiones.

5. Conclusiones

Tras la realización de esta práctica, se pueden extraer las siguientes conclusiones:

- Conclusión 1
- Conclusión 2
- Conclusión 3

6. Bibliografía

- Referencia 1: [Autor]. [Título]. [Editorial], [Año].
- Referencia 2: [Autor]. [Título]. [Editorial], [Año].
- Documentación oficial de [herramienta/lenguaje utilizado]