PROCESOS EN C

Sistemas operativos

Universidad Veracruzana
Jarly Hernández Romero
Maximiliano Soto Jiménez

08 de enero de 2024

Introducción

Los procesos en C son ejecuciones independientes de un programa, cada una con su propio espacio de memoria y recursos asociados. Esta independencia proporciona a los programadores una manera efectiva de realizar múltiples tareas simultáneamente, mejorando así la eficiencia y el rendimiento de las aplicaciones.

La comunicación entre procesos es esencial para lograr la cooperación y coordinación entre ellos. En C, existen varios mecanismos para facilitar la comunicación entre procesos, siendo los más comunes:

Pipes (Tubos): Permiten la transferencia de datos entre procesos de manera unidireccional. Un proceso puede escribir en un extremo del tubo, mientras que otro proceso puede leer desde el otro extremo, estableciendo así una comunicación simple y eficiente.

Colas de mensajes: Proporcionan una forma de enviar mensajes entre procesos. Cada mensaje contiene información específica que puede ser compartida y procesada por diferentes partes del sistema.

Memoria compartida: Permite que varios procesos accedan a un segmento de memoria compartida, posibilitando así la transferencia de datos de manera más rápida. Sin embargo, se deben implementar mecanismos de sincronización para evitar conflictos en el acceso a la memoria compartida.

Semáforos y Mutex: Son herramientas de sincronización que permiten a los procesos controlar el acceso a recursos compartidos. Los semáforos y mutex ayudan a prevenir condiciones de carrera y garantizan una ejecución ordenada y segura de los procesos.

En este proyecto, se simulo la conexión entre un usuario y un servidor, conectados por un router, el cual transmite las cadenas que envía el usuario al servidor, el cual devuelve la cadena al router e imprime la cadena al usuario.

Descripción del código

1.Paqueterias importadas

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <ctype.h>
```

Se importan las bibliotecas para los sockets y la comunicación entre procesos

2.Declaracion de variables

```c
int main(int argc, char *argv[]) {
    const char protocolo[] = "tcp";
    int socket_cliente;
    struct sockaddr_in dir_router;  // Información del router
    unsigned short puerto_router = 5000u;
    char *direccion_router = "127.0.0.1";  // Dirección del router
    struct sockaddr_in dir_servidor;
    unsigned short puerto_servidor = 4500u;
    char *direccion_servidor;
    struct protoent *protoent;
    struct hostent *hostent;
    in_addr_t in_addr;
```

Se definen constantes y variables para configurar los parámetros de la conexión, como el protocolo, puertos, direcciones IP, etc.

3.Validación de la dirección del servidor

```c
if (argc > 1)
    direccion_servidor = argv[1];
else {
    perror("Debe proporcionar un parámetro con la dirección del servidor");
    exit(-1);
}
```

Se valida que la dirección del servidor que se ingresó es correcta y válida

4.Establecimiento de conexiones

```
// Obtiene la configuración para el socket TCP
protoent = getprotobyname(protocolo);

// Crea el socket del dominio Internet, para comunicaciones orientadas a conexión y con el protocolo seleccionado
socket_cliente = socket(AF_INET, SOCK_STREAM, protoent->p_proto);

// Obtiene la dirección IP del router
hostent = gethostbyname(direccion_router);
in_addr = inet_addr(inet_ntoa(*(struct in_addr *)*(hostent->h_addr_list)));

// Configura la dirección en la que escucha el router
dir_router.sin_family = AF_INET;
dir_router.sin_addr.s_addr = in_addr;
dir_router.sin_port = htons(puerto_router);

// Realiza la conexión del cliente al router
if (connect(socket_cliente, (struct sockaddr *)&dir_router, sizeof(struct sockaddr_in)) == -1) {
    perror("Error en la conexión al router");
    exit(-1);
}
```

Se obtiene la configuración del socket TCP, se configura del socket del router al cliente, así como la configuración del socket del router a la del servidor, por último, se realiza la conexión del cliente al router.

5.lectura de la información

```
char buffer[200];
int nbytes = 0;

// Limpia el buffer
memset(buffer, 0, 200);

// Se lee una cadena desde el teclado
printf("Ingrese una cadena: ");
fgets(buffer, 200, stdin);

nbytes = strlen(buffer);

// Se envía la cadena al servidor final
write(socket_servidor, buffer, nbytes);

// Se lee la respuesta del servidor final
nbytes = read(socket_servidor, buffer, 200);

// Se imprime la cadena leída desde la conexión
printf("El servidor ha respondido: %s", buffer);
```

Se lee la información de la cadena del usuario, se envia al router, el router lo envia al servidor y llo devuelve al router, el cual lo devuelve al usuario que se imprime la cadena enviada con el mensaje “El servidor ha respondido:”.

6.Cierre de socket

```
    close(socket_servidor);

    return 0;
}
```

Se cierra el socket del servidor y finaliza el programa.