Informática

Unidad 3: Interacción con el hardware y el Sistema Operativo

Ingeniería en Mecatrónica

Facultad de Ingeniería
Universidad Nacional de Cuyo

Dr. Ing. Martín G. Marchetta
mmarchetta@fing.uncu.edu.ar

UNIVERSIDAD NACIONAL DE CUYO

FACULTAD DE INGENIERIA
en acción continua...

3C – Comunicación entre Procesos

(IPC, Inter-Process Communication)

3C – Comunicación entre Procesos

- Señales
  - La mayoría de los sistemas operativos modernos manejan 2 tipos de interrupciones:
    - **Interrupciones de hardware**: son causadas por un evento de un dispositivo externo a la CPU; el dispositivo **interrumpe** a la CPU mediante una línea del chip
    - **Interrupciones de software:** las genera el SO o algún proceso (a través del SO), y generalmente tienen como destino un proceso. En sistemas basados en Unix, y más recientemente en sistemas que siguen el estándar POSIX, estas interrupciones de software se llaman ***señales***.

3C – Comunicación entre Procesos

- Señales
  - Un proceso puede enviar una señal a otro proceso utilizando la llamada al sistema **kill** (signal.h)

```
int kill(__pid_t __pid, int __sig)
```

    - El primer argumento es el pid del proceso al que se le envía la señal (es decir, el pid del proceso al que se quiere interrumpir)
    - El segundo argumento, es la señal a enviar. En Linux, las señales disponibles tienen constantes asignadas en el header signum.h

  - Un proceso puede manejar estas señales de manera explícita o implícita. Para manejarlas de manera explícita, el proceso ejecuta la llamada al sistema **signal** (signal.h)

```
__sighandler_t signal(int __sig, __sighandler_t __handler)
```

    - El primer argumento es el número de señal al que se quiere responder
    - El segundo argumento es la función que manejará la señal

3C – Comunicación entre Procesos

- Señales
  - Ejemplo

```
#include <signal.h>
#include <signum.h>

void manejador_signals(int signal_type){
  if (signal_type == SIGTERM)
    printf("SIGTERM recibida. Terminando proceso.\n");

}

int main(void){

    ...

    //Informamos al kernel que esta funcion manejara signals SIGTERM
    signal(SIGTERM, manejador_signals);

    ...

    return 0;

}
```

Unidad 3A: Multithreading y Multiprocessing

- Tuberías (pipes)
  - Son una entidad similar a un archivo: se las puede crear, abrir, cerrar, escribir y leer
  - Son half-duplex: tienen un “extremo de lectura” y un “extremo de escritura”. Lo que un proceso escribe en el extremo de escritura, el otro proceso conectado al pipe puede leerlo en el extremo de lectura
  - Pueden tener un nombre, en cuyo caso se implementan como un archivo especial, pero el uso más simple es el de ***pipes sin nombre***
  - Para usar pipes sin nombre:
    - El proceso padre crea la pipe
    - Los procesos hijos “heredan” cada uno una copia del handler de la tubería
    - El padre y los hijos ***cierran*** el extremo de la tubería que no utilizarán
    - El hijo/padre puede escribir/leer en/de la tubería (sólo un tipo de operación, dependiendo de la opción elegida), tal y como si la tubería fuera un archivo

Unidad 3A: Multithreading y Multiprocessing

- Pipes: un ejemplo

```
#define ENTRADA_PIPE 0
#define SALIDA_PIPE 1

int main(void){
   int tuberia[2]; //Identificador de la tuberia
   pid_t pid_hijo;

   pipe(tuberia); //Creacion de la tuberia
   pid_hijo = fork(); //Creamos el hijo
   ...
   if(pid_hijo == 0){ //proceso hijo

      ...
      close(tuberia[ENTRADA_PIPE]);
      ...
   }
   else if(pid_hijo > 0){

      ...
      close(tuberia[SALIDA_PIPE]);
      ...
   }
}
```

Unidad 3A: Multithreading y Multiprocessing

- Sobre las pipes se puede leer y escribir, de manera similar a un archivo

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

  - fd: Identificador del extremo de la tuberia
  - *buf: apuntador al buffer en el que se pondrán los datos leidos, o desde donde se leerán los datos a escribir
  - count: cantidad de **bytes** a leer o escribir

- Nótese que para leer o escribir, se especifica el **extremo** de la pipe, no el arreglo que contiene los 2 extremos (lectura/escritura)

3C – Comunicación entre Procesos

- Sockets: generalidades
  - Los sockets (“enchufes”) son un tipo de comunicación entre procesos
  - A diferencia de las tuberías (pipes) y las señales (signals), los sockets permiten conectar procesos tanto en el host local, como en hosts remotos
  - Los sockets son una interfaz uniforme para la utilización de distintos protocolos
  - Los sockets cubren la comunicación a nivel de capas 3 y 4 del modelo OSI (capa de red y capa de transporte). Las capas inferiores quedan “ocultas” en funciones de biblioteca del sistema operativo, y las capas superiores deben ser implementadas por el usuario (programador)
  - Una de las familias de protocolos más utilizadas es TCP/IP (protocolo IP en capa de red, protocolo TCP en capa de transporte)

3C – Comunicación entre Procesos

- Sockets TCP/IP
  - Cuando se utilizan sockets, uno de los dos procesos cumple el rol de **servidor**, y el otro de **cliente**
  - Típicamente, un proceso servidor crea el socket, y queda bloqueado, en espera de conexiones de un cliente
  - Un cliente establece luego la conexión con el servidor, y una vez establecida la misma, cliente y servidor pueden intercambiar datos de manera **full-duplex**
  - Para que la conexión se pueda establecer, el servidor debe haber creado el socket y haberlo “activado” antes de que cliente intente conectarse

3C – Comunicación entre Procesos

- Sockets TCP/IP: procedimiento en el **servidor**
  - **Paso 1:** crear el socket (indicando la familia de protocolos)
  - **Paso 2:** crear una configuración de dirección para el socket; para la familia TCP/IP, se debe configurar la dirección IP y el puerto TCP/UDP *en el que "escuchará" el servidor*
  - **Paso 3:** enlazar el socket (paso 1) con la configuración (paso 2)
  - **Paso 4**: indicar que el socket "escuchará" conexiones (es decir, que el socket se comportará como un servidor)
  - **Paso 5:** aceptar conexiones; típicamente el servidor se bloquea hasta que llega la próxima conexión, aunque puede configurarse para que no se bloquee
  - **Paso 6:** enviar/recibir datos sobre la conexión establecida
  - **Paso 7:** cerrar el socket

- Sockets TCP/IP: procedimiento en el **cliente**
  - **Paso 1:** crear el socket (indicando la familia de protocolos)
  - **Paso 2:** crear una configuración de dirección para el socket; para la familia TCP/IP, se debe configurar la dirección IP y el puerto TCP/UDP *al que se conectará el cliente* (es decir, donde escucha el servidor)
  - **Paso 3:** establecer una conexión con el servidor, utilizando el socket (paso 1) y la configuración de conexión (paso 2)
  - **Paso 4:** enviar/recibir datos sobre la conexión establecida
  - **Paso 5:** cerrar el socket

- Sockets TCP/IP: Implementación
  - Encabezados necesarios

```
#include <sys/socket.h>
#include <arpa/inet.h>
```

  - Creación de un socket

```
int socket(int __domain, int __type, int __protocol)
```

    `__domain:` familia de protocolos
    `__type:` tipo de servicio
    `__protocol:` protoclo específico (opcional)

    Retorna: descriptor del socket

  - Ej: Socket TCP/IP con servicio orientado a la conexión (el único protocolo para esa combinación es TCP, por lo que se deja en cero)

```
int sd = socket(AF_INET, SOCK_STREAM, 0);
```

- Sockets TCP/IP: Implementación
  - Estructura de datos para la configuración de dirección

    `struct sockaddr_in`
  - Miembros a configurar
    - `sin_family`: familia de protocolos
    - `sin_addr`: dirección (donde escuchará el servidor o a donde se conectará el cliente)
    - `sin_port`: puerto (donde escuchará el servidor o a donde se conectará el cliente)
  - Dado que podemos estar conectando hosts con arquitecturas distintas, el orden de bytes para la configuración de la IP y del puerto debe ser convertido al formato “de la red”. Ej:

```
struct sockaddr_in direccionSocket;
inet_aton("127.0.0.1", &(direccionSocket.sin_addr));
direccionSocket.sin_port = htons(3333);
```

- Sockets TCP/IP: Implementación
  - Enlazado de un socket y una estructura de configuración (sólo se aplica al servidor)

```
int sd = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in direccion;
...
bind(sd, (struct sockaddr *)&direccion, sizeof(direccion));
```

  - Configuración del socket para “escuchar” en un puerto (sólo servidor)

```
int listen(int __fd, int __n)
```

    - `__fd:` descriptor del socket
    - `__n:` cantidad máxima de peticiones de conexión que pueden ponerse en cola

- Sockets TCP/IP: Implementación
  - Bloquearse esperando una conexión (sólo servidor)

    ```
    int accept(int __fd, struct sockaddr *__addr, int *__addr_len)
    ```

    - `__fd:` Descriptor del socket
    - `__addr:` Estructura de dirección del cliente
    - `__addr_len:` longitud (en bytes) de la estructura de dirección del ciente
    - Retorna un **nuevo socket**, que permite enviar/recibir datos (el socket original sigue "escuchando" conexiones).
  - Enviar datos sobre un socket que tiene asociado una conexión establecida

    ```
    ssize_t send(int __fd, __const void *__buf, size_t __n, int __flags);
    ```

    - `__fd:` Descriptor del socket
    - `__buf`: Buffer que apunta a los datos que se enviarán
    - `__n`: tamaño del buffer
    - `__flags`: banderas que permiten especificar opciones adicionales (puede dejarse en cero)

- Sockets TCP/IP: Implementación
  - Recibir datos sobre un socket que tiene asociado una conexión establecida

    ```
    ssize_t recv(int __fd, void *__buf, size_t __n, int __flags);
    ```

    - `__fd`: Descriptor del socket
    - `__buf`: Buffer en donde se colocarán los datos leídos
    - `__n`: cantidad de bytes a leer
    - `__flags`: banderas que permiten especificar opciones adicionales (puede dejarse en cero)

  - Cerrar un socket

    ```
    int close(int __fd);
    ```

- Sockets TCP/IP: Implementación
  - Conectarse con un servidor (sólo se aplica al cliente)

```
int connect(int __fd, __const struct sockaddr * __addr, socklen_t __len)
```

    - `__fd`: Descriptor del socket
    - `__addr`: Estructura de dirección que especifica la dirección IP + Puerto donde está escuchando el servidor al que el cliente se quiere conectar
    - `__len`: longitud en bytes de la estructura __addr