Comunicación entre procesos

Sistemas Operativos (SO)

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya

Licencia Creative Commons

Licencia Creative Commons

Índice

Introducción

- Los procesos no comparten memoria
- Puede interesar que la información generada por un proceso sea utilizada por otro
  - ls > out.out; grep “prueba” < out.out
  - ls | grep “prueba”
- Es necesario que el sistema operativo nos ofrezca mecanismos para comunicar procesos entre sí
  - Envío/recepción de información
  - Sincronización

Introducción

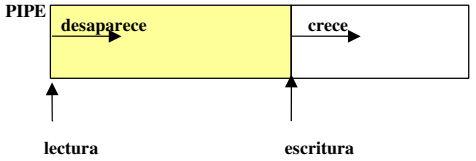
- Comunicación
  - Envio/recepción de información
    - Un proceso envía información a otro proceso
    - Ambos procesos conocen de qué tipo es la información
    - Servicios del sistema: ficheros, pipes, named pipes, sockets
  - Sincronización
    - Un proceso avisa de la finalización de una tarea necesaria para que otro proceso (o él mismo) puedan realizar otra tarea
    - No hay información relacionada, solamente es un aviso (evento)
    - Servicios del sistema: signals, pipes, sockets

Índice

- Introducción
- Pipes
  - Named pipes
  - Unnamed pipes
- Signals
- Sockets

Pipes

- Una pipe es una buffer circular de bytes sin tipo
- Sirve para comunicar 2 o más procesos entre sí



Los datos que se leen de la pipe desaparecen

- Dependiendo de la implementación pueden ser:
  - Unidireccionales: solo se puede leer o solo escribir
  - Bidireccionales: se puede leer y escribir

Pipes

- Dos tipos de pipes
  - Named pipes
  - Pipes o unnamed pipes o pipes ordinarias o pipes anónimas
- Se utilizan unas u otras dependiendo de la relación entre los procesos
  - Si están emparentados
    - Pipes
    - Named pipes
  - Si no están emparentados
    - Named pipes

Named pipes

- Utilizadas por dos procesos cualquiera
  - Emparentados o no
- Tienen una representación en el sistema de ficheros
  - Se tiene que crear un fichero especial antes de poder utilizarla
  - Solamente es necesario crear una vez este fichero
  - Una vez creado este fichero, se tiene que abrir como un fichero normal
  - Se trabaja con él como si fuese un fichero normal

UNIX: Named pipes

- Se crean mediante mknod:
  - int **mknod**(char *nombre_pipe, int mode,dev_t device)
  - Crea un dispositivo (genérico), lo usaremos sólo para pipes
    - Nombre_pipe, es el nombre que tendrá el fichero tipo pipe
    - Mode, indica que es una pipe y las protecciones del fichero.
    - Device, no es necesario ponerlo en el caso de una pipe
    - Devuelve 0 si OK y –1 si ERROR

  Ej: mknod(“mi_pipe”,S_IFIFO|0666)
- Después ya se puede abrir como un fichero normal:
  - Ej: open(“mi_pipe”, O_RDONLY)

W2K: Named pipes

- En W2K se crea y se abre la named pipe con una llamada:
  - HANDLE CreateNamedPipe(**LPCTSTR** *lpName*, **DWORD** *dwOpenMode*, **DWORD** *dwPipeMode*, **DWORD** *nMaxInstances*, **DWORD** *nOutBufferSize*, **DWORD** *nInBufferSize*, **DWORD** *nDefaultTimeOut*, **LPSECURITY_ATTRIBUTES** *lpSecurityAttributes* **);**
  - Se tiene que indicar si la pipe es de bytes o de mensajes
  - Ej:
    - CreateNamedPipe(\\\\.\\pipe\mi_pipe”, PIPE_ACCESS_DUPLEX, PIPE_TYPE_MESSAGE, PIPE_UNLIMITED_INSTANCES, 4096, 4096, NMP_WAIT_USE_DEFAULT_WAIT, NULL);

W2K: Named pipes

- Si queremos abrir una named pipe que existe:
  - CreateFile
  - CallNamedPipe
- Después se utilizan las llamadas de lectura/escritura normales:
  - ReadFile
  - WriteFile
- Se puede leer información de una named pipe sin eliminarla:
  - PeekNamedPipe

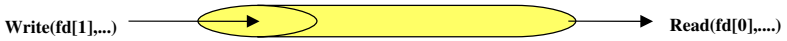
Pipes

- Entre procesos relacionados:
  - Normalmente padre/hijo
  - No tienen representación en el sistema de ficheros
    - En realidad su implementación consiste en un búffer en memoria del sistema
  - Se crea y se abre a la vez
  - Normalmente los canales/HANDLEs se heredan para poder compartir la pipe

UNIX: Pipes

- Usaremos una **ÚNICA** llamada a sistema que nos devolverá dos canales ya abiertos.
- int **pipe**(int fd[2])
- Abre dos canales de acceso a una pipe.
  - fd[0] canal de sólo lectura,
  - fd[1], canal de sólo escritura

Lo que se escribe en fd[1] se lee por fd[0]



W2K: pipes

- Semejante a Linux, una única llamada devuelve dos HANDLEs:
  - BOOL CreatePipe (PHANDLE hreadpipe, PHANDLE hwritepipe, LPSECURITY_ATTRIBUTES security, DWORD size)
  - Se indica el tamaño de la pipe

UNIX: Comportamiento de las llamadas al sistema

- Open: bloquea hasta que exista la pareja
  - Solo se usa open con named pipes
- Read:
  - Si la pipe está vacia:
  - **Si ningún proceso la tiene abierta para escritura** devuelve 0
  - **Si existe ESCRITOR** nos bloqueamos hasta que
    - Alguien escriba, o
    - Los ESCRITORES desaparecen.
    - En el caso 1 nos desbloqueamos y leemos lo escrito, en el caso 2 la llamada devuelve 0.

UNIX:Comportamiento de las llamadas al sistema

- Write
  - Si escribimos en una pipe que no tiene lectores devuelve –1 y errno=EPIPE y el proceso recibe un evento (signal) SIGPIPE
  - Si la pipe está llena el proceso se bloquea
- Lseek
  - No se aplica a las pipes

W2K: Comportamiento de las llamadas al sistema

- CreateNamedPipe
  - No bloqueante
  - Si se quiere esperar hasta que alguien se conecte:
    - ConnectNamedPipe
- ReadFile
  - Solamente acaba si:
    - Se han conseguido leer todos los datos solicitados
    - Se completa totalmente la última escritura
    - Hay error:
      - EBROKEN_PIPE (no hay escritores)

W2K: Comportamiento de las llamadas al sistema

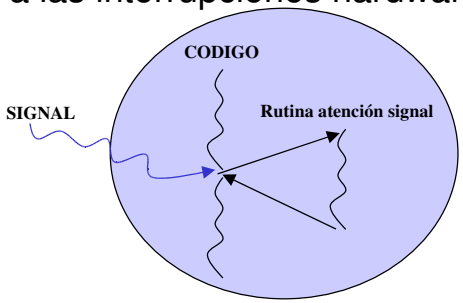
- WriteFile:
  - Solamente acaba si:
    - Se consiguen escribir todos los datos
    - Hay error:
      - EBROKEN_PIPE (no hay lectores)

Índice

- Introducción
- Pipes
  - Named pipes
  - Unnamed pipes
- Signals
- Sockets

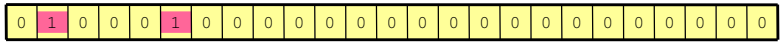
Introducción

- El Sistema ofrece un servicio de comunicación asíncrona de sucesos
  - Entre procesos (o el mismo proceso)
  - Del sistema operativo a un proceso
- Similares a las interrupciones hardware



Estructuras

- En el PCB hay un mapa de bits que contiene los signals pendientes de ser servidos



  - Si llegan varios signals de un mismo tipo, se pierden
- También contiene un vector donde se guarda la reprogramación de los signals
  - Puntero a la nueva reprogramación
  - Puntero a la antigua reprogramación

Signals

- Hay 32 signals (UNIX estándar)
  - En la mayoría el significado viene dado por el SO, pero algunos pueden ser definidos por el usuario
- Que hacer cuando recibimos un signal???
  - Tratamiento por defecto (SIG_DFL)
    - Exit: Acabar el proceso
    - Exit+Core: Acabar el proceso y volcar la memoria sobre un fichero
    - Stop: Parar el proceso hasta que reciba el signal SIG_CONT
    - Ignore: No hacer nada
  - No hacer nada (SIG_IGN)
  - Ejecutar una rutina de usuario (especificada por el usuario y programada por el)

Signals

- Tipos de signals
  - SIGINT, ^c durante la ejecución , Def=Exit
  - SIGSEGV, dirección ilegal, Def=exit+Core, No ignorar
  - SIGALARM, expiración timer, Def=Exit
  - SIGSTOP, parar proceso (^z), Def=Stop, No ignorar, No reprogramar
  - SIGCONT, continuar proceso parado, Def=continuar
  - SIGUSR1, definida por el usuario, Def=Exit
  - SIGUSR2, definida por el usuario, Def=Exit
  - SIGPIPE, escritura sobre pipe sin lector, Def=Exit
  - SIGCHLD, ha muerto un hijo, Def=ignorar
  - SIGKILL, killed, Def=Exit , No reprogramar, No ignorar
  - SIGTERM, terminated, Def=Exit

Envío de un signal

- Envío de signals:
  - Llamada a sistema: int kill(int pid, int sig)
  - Parámetros:
    - Pid: identificador de proceso al que se le envía el signal
    - Sig: tipo de signal que enviamos
  - Que devuelve:
    - 0 si enviado OK
    - -1 si error (no existe el proceso, etc)
- Desde el shell .....
  - Kill –sig pid

Programación de un signal

- Como se programa un signal?
  - Llamada a sistema: void (*signal (int sig, void (function)(int)))(int)
  - Parámetros:
    - Sig: signal que estamos programando
    - Function: puede ser (SIG_DFL/SIG_IGN/función de usuario)
  - Que devuelve?
    - (void *)-1 en caso de error
    - La dirección de la anterior rutina de tratamiento

Programación de un signal

- No todos los signals se pueden reprogramar
- La programación del signal sólo sirve para una vez, luego hay que reprogramarla
- La programación de signals se hereda al hacer un fork, pero se pierde al hacer un exec (se pierde el código anterior)
- La solicitud de aviso se hace al proceso
  - Si el proceso hace un fork el proceso hijo no lo recibe
  - Si el proceso hace un exec el aviso sigue pendiente

Tratamiento del signal

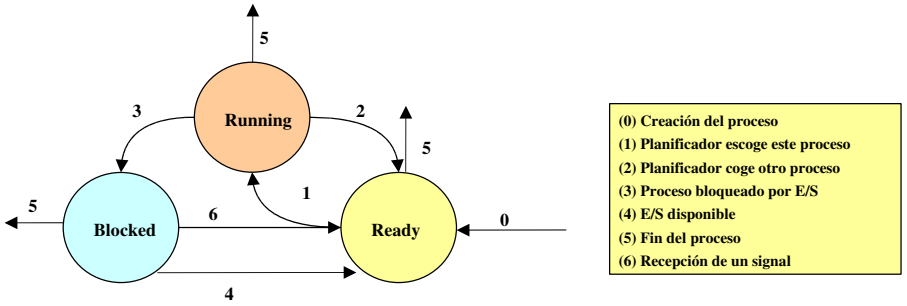
- Depende del signal recibido:
  - Tratamiento diferido
    - No fuerzan la finalización del proceso
  - Tratamiento inmediato
    - Fuerzan la finalización del proceso

Tratamiento del signal

- Tratamiento diferido:
  - El proceso que está en RUN ejecuta Kill
  - El SO lo apunta en el PCB del proceso que recibe el signal
    - Si el proceso estaba bloqueado, lo pasa a Ready, la llamada al sistema devuelve -1, errno=EINTR
  - Cuando el proceso pasa a RUN, se comprueba si tiene algún signal pendiente:
    - Si no tiene: ejecución normal
    - Si tiene: se ejecuta la rutina de atención al signal
  - Después de servir el signal, si el proceso no acaba, se desprograma el signal
    - Los signals se tienen que reprogramar una vez servidos

Ciclo de vida de un proceso

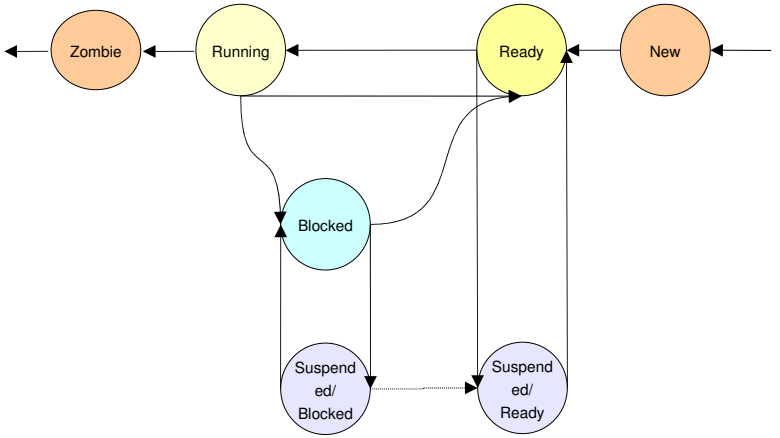
- Tratamiento inmediato:



(0) Creación del proceso
(1) Planificador escoge este proceso
(2) Planificador coge otro proceso
(3) Proceso bloqueado por E/S
(4) E/S disponible
(5) Fin del proceso
(6) Recepción de un signal

Procesos suspendidos

- SIGSTOP:



Prioridades

- Los signals pueden tener prioridades:
  - Más prioritarios:
    - Los que pueden forzar la finalización del proceso
    - SIGKILL, SIGTERM, SIGSEGV
  - Menos prioritarios:
    - Notificación
    - SIGALRM

Espera de signals

- Se puede forzar al proceso a que espere un signal:
  - Espera activa
    - Consume CPU
  - Espera pasiva
    - Pause
    - El proceso se bloquea hasta que llega un signal
    - Siempre devuelve -1
    - No consume CPU

Temporizadores

- Programar una alarma de tiempo
  - Llamada a sistema: unsigned int alarm(unsigned int sec)
    - Solicitud de aviso mediante signal SIGALARM al cabo de sec segundos de tiempo real
  - Qué devuelve???
    - Número de segundos restantes de la última petición
- SIGALRM es un evento y hay que programarlo específicamente (llamada a sistema signal)
- La llamada a sistema alarm no bloquea al proceso

Signals W2K

- Los signals en W2K son emulados:
  - Códigos portables
  - Misma llamada al sistema que en Linux
- En W2K un signal se convierte en un mensaje

Índice

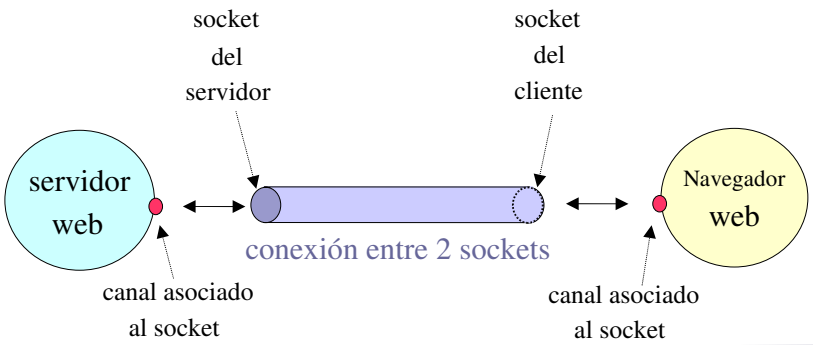
- Introducción
- Pipes
  - Named pipes
  - Unnamed pipes
- Signals
- Sockets

Introducción

- Mecanismo generalizado de comunicación entre procesos:
  - procesos en la misma máquina
  - procesos residentes en máquinas conectadas a través de una red
- La comunicación entre dos procesos se realiza a través de 2 sockets:
  - Cada socket es “extremo final de una comunicación”
  - Modelo similar al de las pipes pero sin las limitaciones de éstas:
    - Los procesos no necesitan tener una relación jerárquica entre sí
    - Los procesos pueden estar ejecutándose en distintas máquinas

Modelo cliente-servidor

- Son usados extensamente en el marco del modelo cliente-servidor
- Ejemplo (de comunicación orientada a conexión):



Tipos de comunicación

- Permiten trabajar con diversos tipos de comunicación
  - Orientado a conexión (stream)
    - Creación de un circuito virtual entre los dos extremos de la comunicación
    - Protocolo: TCP/IP
  - No orientado a conexión (datagram)
    - No se establece un circuito virtual entre el socket y otro socket
    - Cada packete debe especificar el destinatario del mensaje
    - Protocolo: UDP/IP

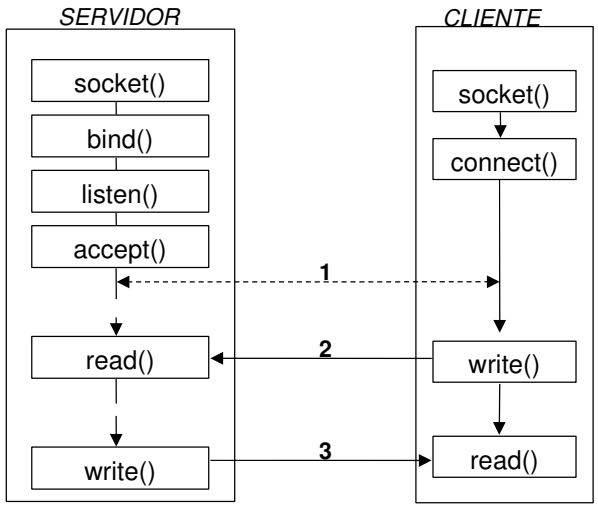
Fases de preparación y uso

- Un socket se crea de forma dinámica:
  - Llamada socket()
  - Local al proceso que lo crea
  - Se referencia mediante un file descriptor (canal)
  - Existe mientras está abierto el canal
- Para que otros procesos puedan conocer un socket se le ha de assignar un nombre público
  - La forma y estructura de este nombre dependen del dominio:
    - UNIX, Internet, etc.
- Se puede usar para enviar y recibir datos (full-duplex)
  - Stream: read(), write()
  - Datagram: sendto(), recvfrom()

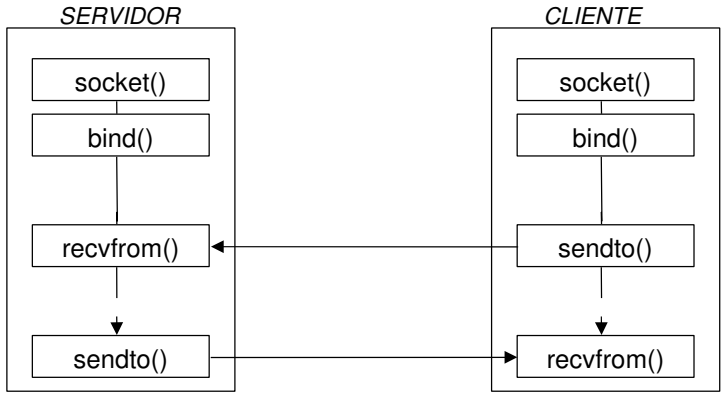
Capas inferiores no transparentes

- Corresponden a un nivel de abstracción alto en la jerarquia de protocolos de comunicaciones
- Pero no tienen aisladas totalmente las capas inferiores. Para usarlos el programador ha de decidir:
  - La familia (o dominio) de conexión
    - UNIX: entre procesos en una misma máquina
    - Internet: entre procesos en máquinas diferentes
    - etc.
  - Y el tipo de conexión:
    - stream (orientado a conexión)
    - datagrama (no orientado a conexión)

Esquema orientado a conexión



Esquema no orientado a conexión

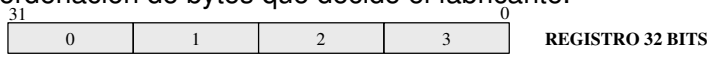


Llamadas principales

- Descripción breve:
  - *socket*( ): apertura del canal
  - *bind*( ): hace pública la dirección del socket
  - *listen*( ): disposición para acceptar connexiones
  - *accept*( ): acceptar una connexión. Bloquea al proceso hasta que recibe una petición de connexión
  - *connect*( ): petición de connexión
  - *rcvfrom*( ): acepta datos. Bloquea al proceso hasta que se recibe una petición de serveio
  - *sendto*(): envio de datos

Representación de los datos

- Classificación máquinas: BIG/LITTLE ENDIAN
  - Tipos de datos de >1 byte representados en la máquina con una ordenación de bytes que decide el fabricante.



Big endian

@3000	0
@3001	1
@3002	2
@3003	3

Little endian

@3000	3
@3001	2
@3002	1
@3003	0

Problemas: envio de datos en redes de computadores

- El receptor puede entender algo de forma diferente al emisor

Representación de los datos: solución

- Es necesario adaptar la representación de los datos que se envian entre máquinas
- Existen funciones que transforman datos del formato nativo de la máquina al de la red (también llamado formato estándard) y viceversa
  - htons (host to network short)
  - htonl (host to network long)
  - ntohs (network to host short)
  - ntohl (network to host long)
- Esto permite que un código se pueda portar a otras máquinas independiente de la ordenación de los bytes.