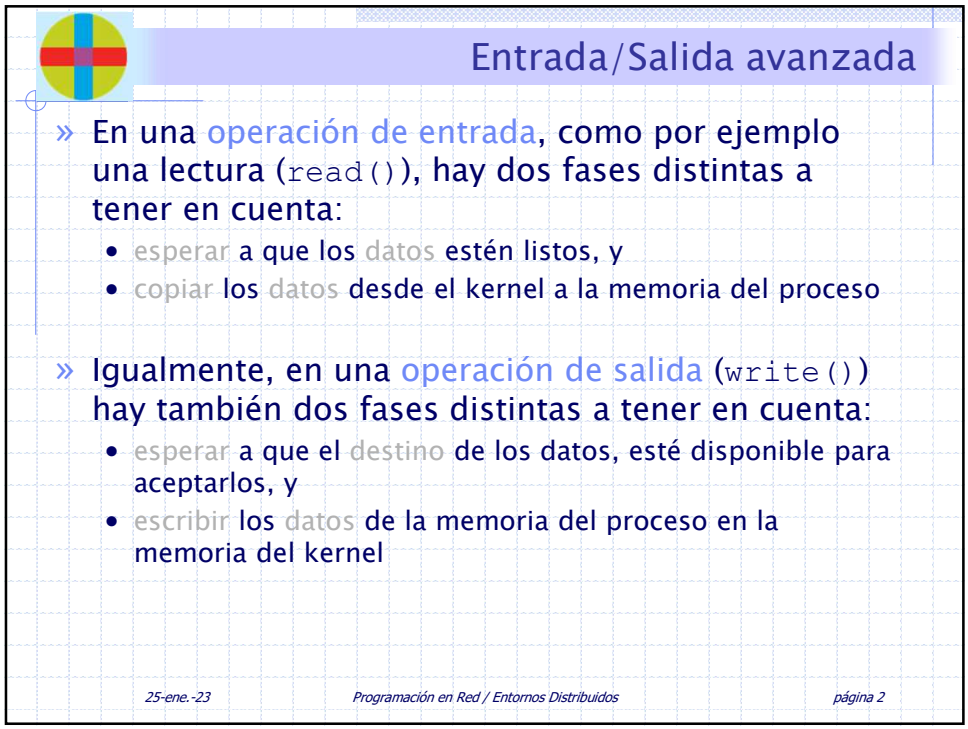

Técnicas avanzadas de Programación en Redes
Programación en Red / Entornos Distribuidos
Curso 2022/2023
Universidad San Pablo-CEU
Escuela Politécnica Superior
Campus de Montepríncipe

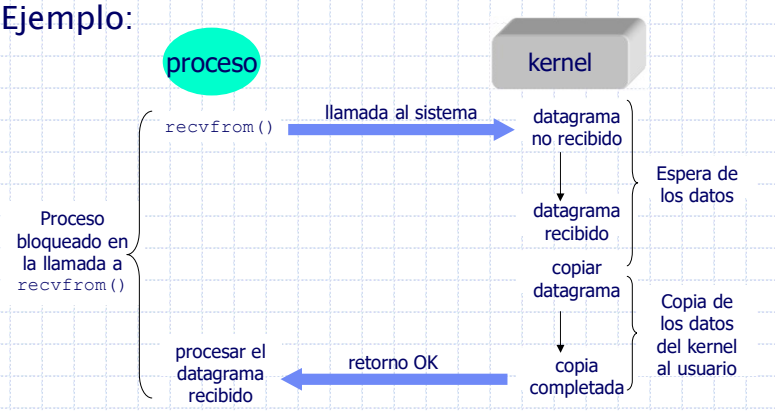
1


Entrada/Salida avanzada
» En una operación de entrada, como por ejemplo una lectura (read()), hay dos fases distintas a tener en cuenta:
• esperar a que los datos estén listos, y
• copiar los datos desde el kernel a la memoria del proceso
» Igualmente, en una operación de salida (write()) hay también dos fases distintas a tener en cuenta:
• esperar a que el destino de los datos, esté disponible para aceptarlos, y
• escribir los datos de la memoria del proceso en la memoria del kernel
25-ene.-23 Programación en Red / Entornos Distribuidos página 2

2

E/S bloqueante

- La forma más simple de las operaciones de E/S que hemos visto es la forma bloqueante:
  - el proceso espera (queda bloqueado) a que la lectura o escritura se pueda llevar a cabo
- Ejemplo:



3

E/S avanzada

- El modelo de E/S bloqueante tiene varios inconvenientes

- Los más importantes son:
  - ¿cómo puedo hacer lectura y escritura a la vez en el mismo descriptor (para soportar primitivas *full-duplex*)?
  - ¿no podría hacer otra cosa mientras espero a que termine la operación de E/S?
  - ¿cómo leo de más de una fuente de datos a la vez?

4

E/S no bloqueante

- El modelo de E/S no bloqueante sirve para:
  - cuando una operación de E/S tenga que bloquear al proceso, no realice el bloqueo
  - esta circunstancia se nos notifica como un error

5

Interrogación periódica (*polling*)

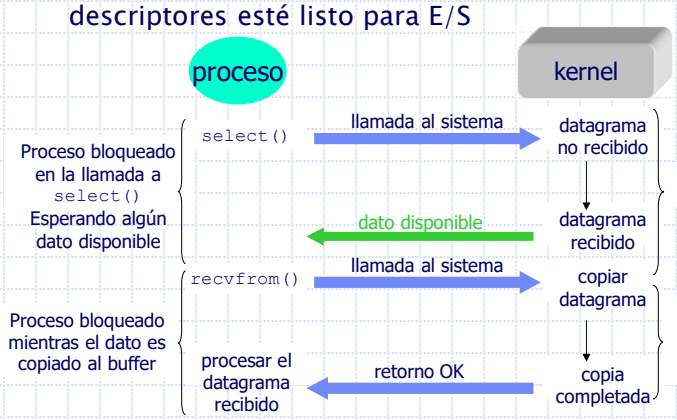
- En el ejemplo anterior se está haciendo *polling* o interrogación periódica, o muestreo.
- El período de interrogación debería ser
  - pequeño para evitar demoras al usuario, pero
  - grande para evitar saturar a la CPU
- El *polling* es un desperdicio de recursos de CPU y debería ser evitado en lo posible

6

E/S multiplexada

» El modelo de E/S multiplexada da la posibilidad de:

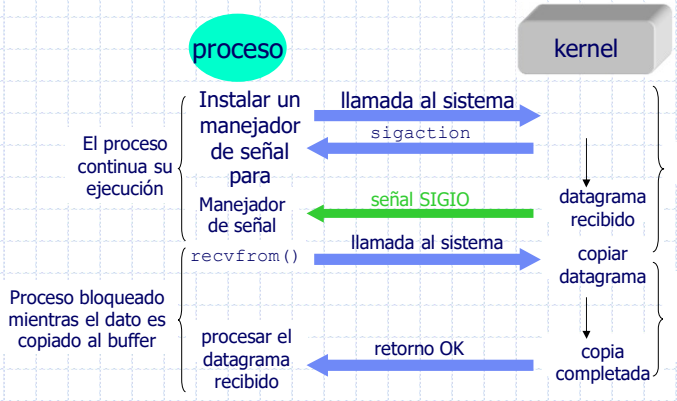
- construir una lista de descriptores en los que estamos interesados, y
- llamar a una función que no vuelve hasta que uno de los descriptores esté listo para E/S



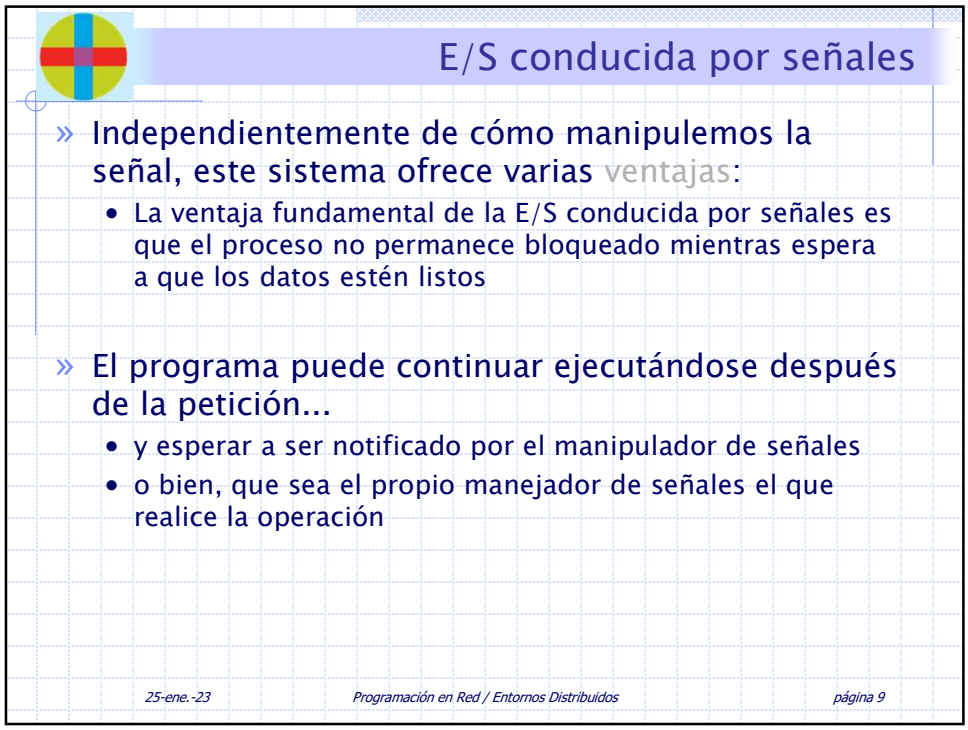
7

E/S conducida por señales

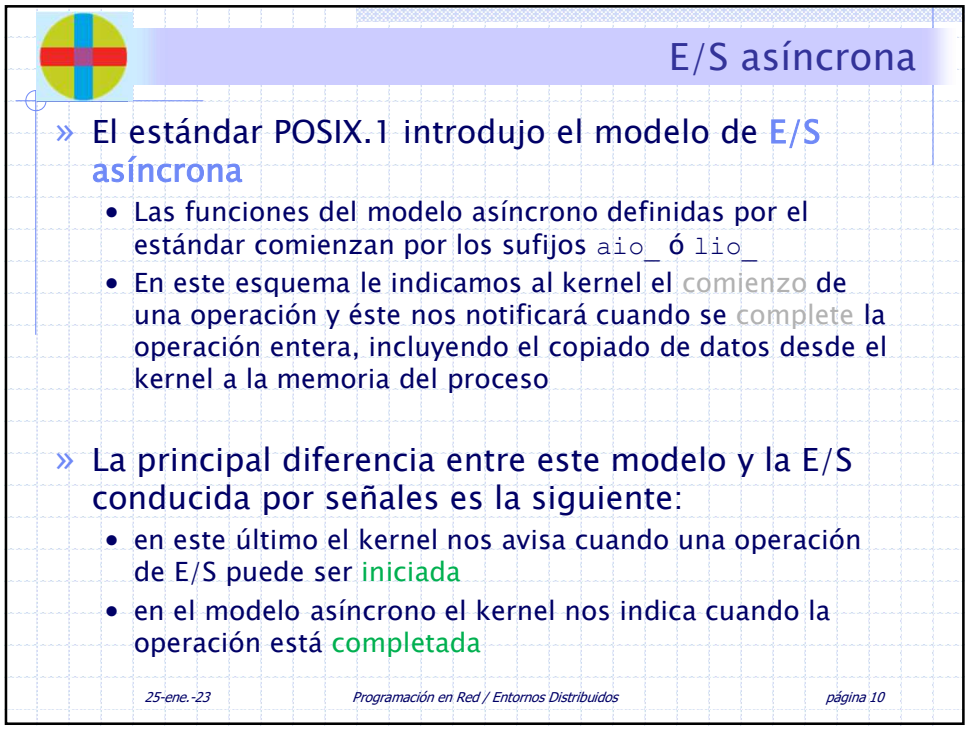
» En el modelo de E/S conducida por señales el kernel nos notifica con alguna señal (SIGIO o SIGPOLL) cuando una operación de E/S por la que un proceso esperaba está lista para ser realizada:



8


E/S conducida por señales
» Independientemente de cómo manipulemos la señal, este sistema ofrece varias ventajas:
• La ventaja fundamental de la E/S conducida por señales es que el proceso no permanece bloqueado mientras espera a que los datos estén listos
» El programa puede continuar ejecutándose después de la petición...
• y esperar a ser notificado por el manipulador de señales
• o bien, que sea el propio manejador de señales el que realice la operación
25-ene.-23 Programación en Red / Entornos Distribuidos página 9

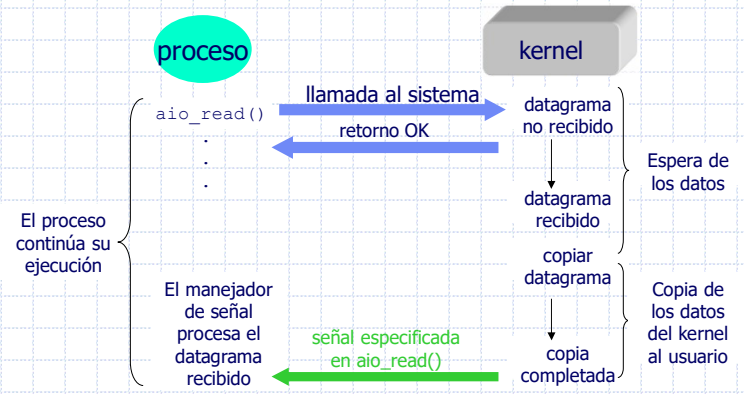
9


E/S asíncrona
» El estándar POSIX.1 introdujo el modelo de E/S asíncrona
• Las funciones del modelo asíncrono definidas por el estándar comienzan por los sufijos aio_ ó lio_
• En este esquema le indicamos al kernel el comienzo de una operación y éste nos notificará cuando se complete la operación entera, incluyendo el copiado de datos desde el kernel a la memoria del proceso
» La principal diferencia entre este modelo y la E/S conducida por señales es la siguiente:
• en este último el kernel nos avisa cuando una operación de E/S puede ser iniciada
• en el modelo asíncrono el kernel nos indica cuando la operación está completada
25-ene.-23 Programación en Red / Entornos Distribuidos página 10

10

E/S asíncrona

» El modelo de E/S asíncrona funciona de la siguiente forma:



11

E/S asíncrona

» La primitiva `aio_read()` le pasa al kernel:
- el descriptor de E/S
- el puntero al buffer intermedio
- el tamaño del buffer

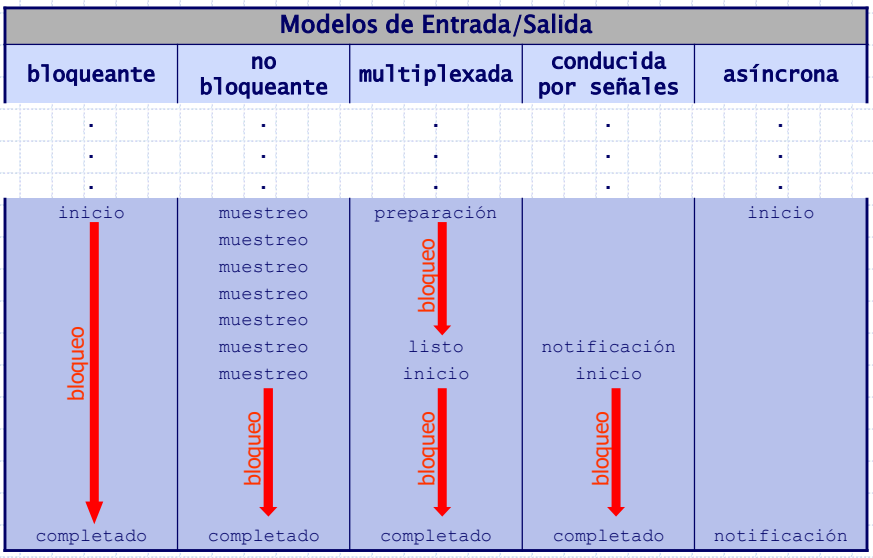
} mismos args. de `read()`

- un desplazamiento de fichero (*file offset*, similar al usado en la llamada `lseek()`)
- cómo nos notificará cuando la operación entera esté completa, seguramente, mediante algún tipo de señal

» Esta llamada al sistema retorna inmediatamente y nuestro proceso no es bloqueado por la espera de la operación de E/S.

12

Comparación de modelos de E/S



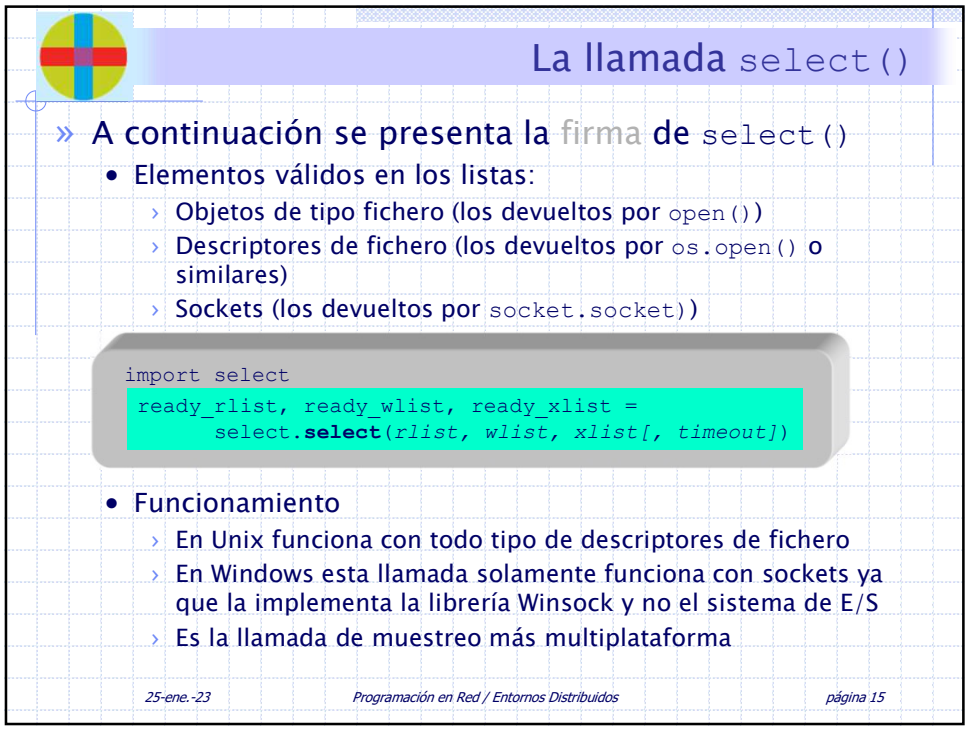
13

E/S multiplexada

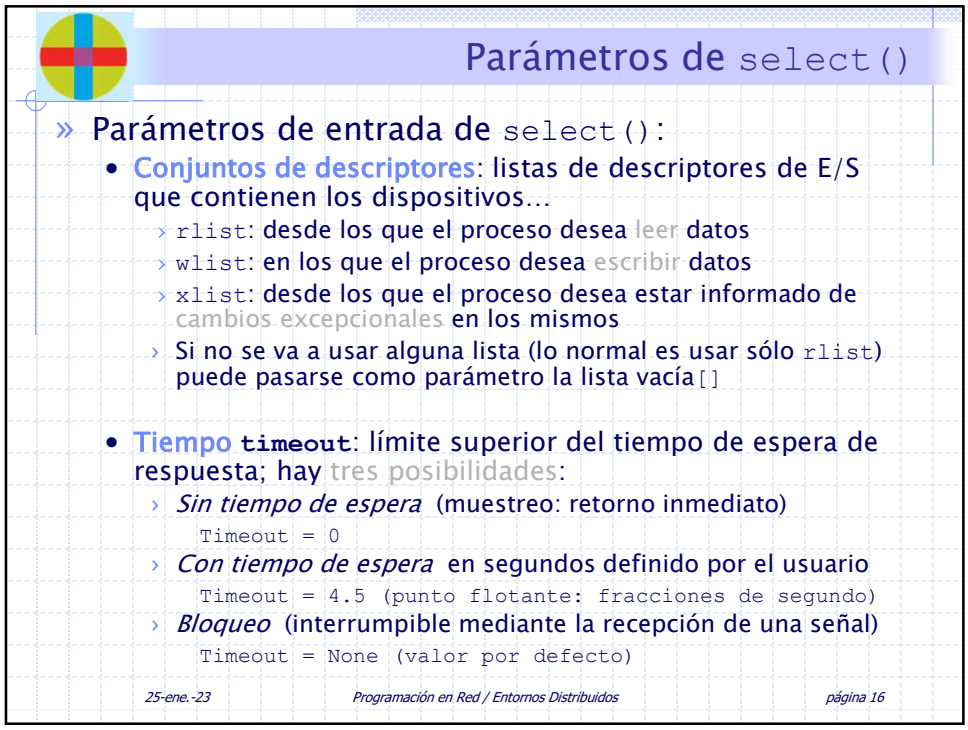
» La **E/S multiplexada** se realiza mediante la primitiva `select()`

- Se le pasa a `select()` una lista de descriptores de E/S
- La llamada no devolverá control hasta que alguno de los descriptores de los dispositivos de entrada/salida no esté listo para su uso
- La llamada nos devolverá los descriptores de los dispositivos que están listos para ser tratados
- Ello puede ocurrir por tres causas:
  1. *leer*
  2. *escribir*
  3. *obtener información de estado del dispositivo gestionado por el descriptor*

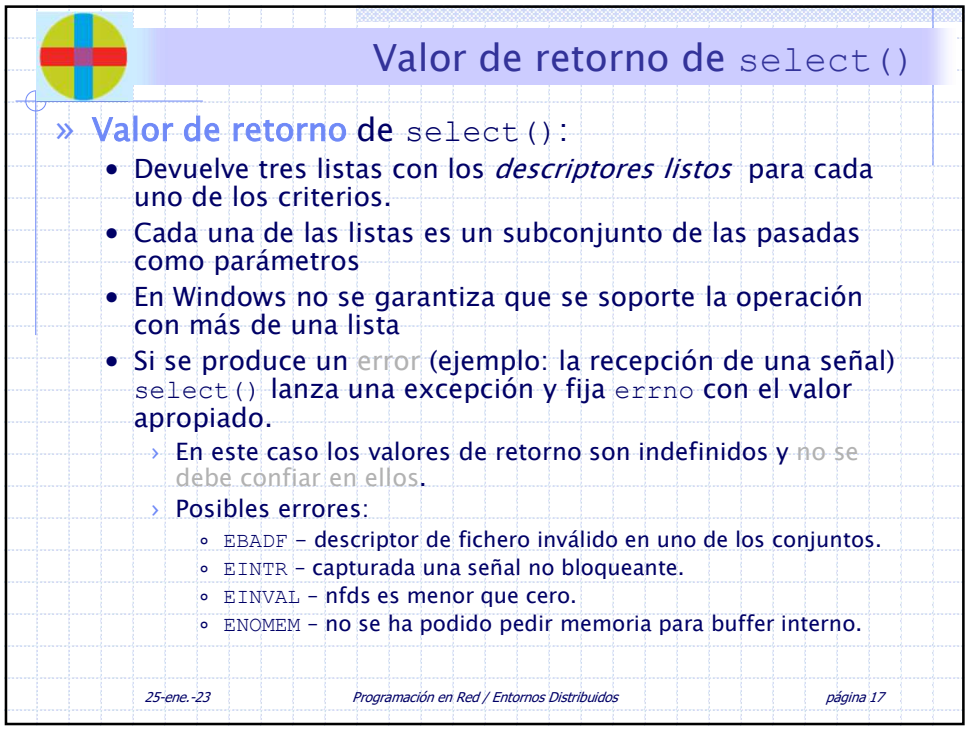
14


La llamada select()
» A continuación se presenta la firma de select()
• Elementos válidos en los listas:
› Objetos de tipo fichero (los devueltos por open())
› Descriptores de fichero (los devueltos por os.open() o similares)
› Sockets (los devueltos por socket.socket))
import select
ready_rlist, ready_wlist, ready_xlist =
select.select(rlist, wlist, xlist[, timeout])
• Funcionamiento
› En Unix funciona con todo tipo de descriptores de fichero
› En Windows esta llamada solamente funciona con sockets ya que la implementa la librería Winsock y no el sistema de E/S
› Es la llamada de muestreo más multiplataforma
25-ene.-23 Programación en Red / Entornos Distribuidos página 15

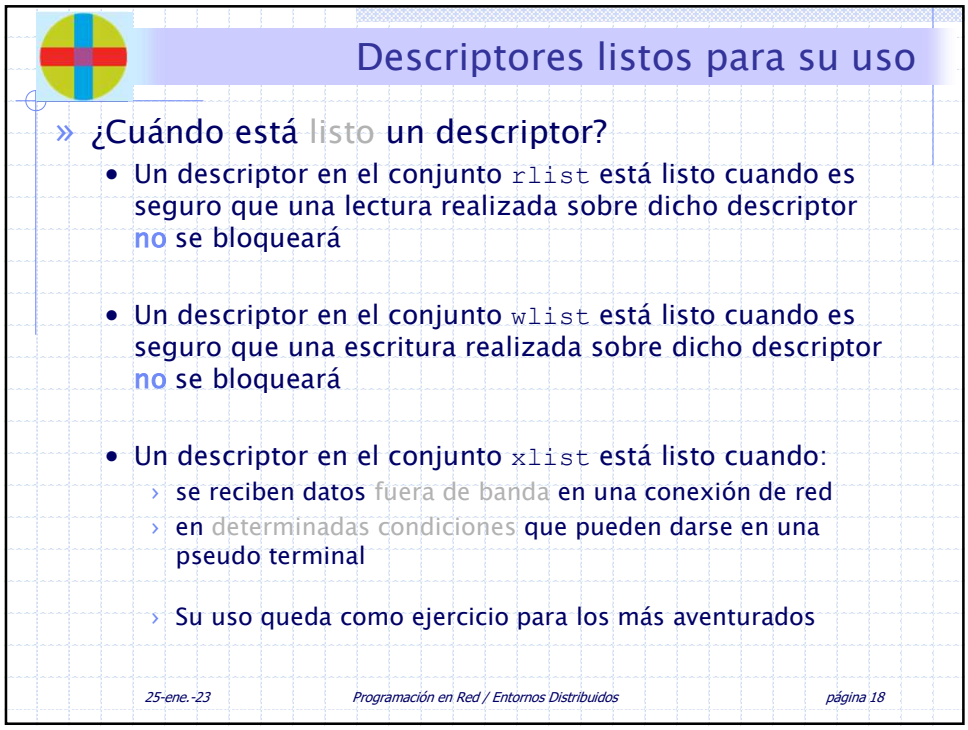
15


Parámetros de select()
» Parámetros de entrada de select():
• Conjuntos de descriptores: listas de descriptores de E/S que contienen los dispositivos…
› rlist: desde los que el proceso desea leer datos
› wlist: en los que el proceso desea escribir datos
› xlist: desde los que el proceso desea estar informado de cambios excepcionales en los mismos
› Si no se va a usar alguna lista (lo normal es usar sólo rlist) puede pasarse como parámetro la lista vacía []
• Tiempo timeout: límite superior del tiempo de espera de respuesta; hay tres posibilidades:
› Sin tiempo de espera (muestreo: retorno inmediato)
Timeout = 0
› Con tiempo de espera en segundos definido por el usuario
Timeout = 4.5 (punto flotante: fracciones de segundo)
› Bloqueo (interrumpible mediante la recepción de una señal)
Timeout = None (valor por defecto)
25-ene.-23 Programación en Red / Entornos Distribuidos página 16

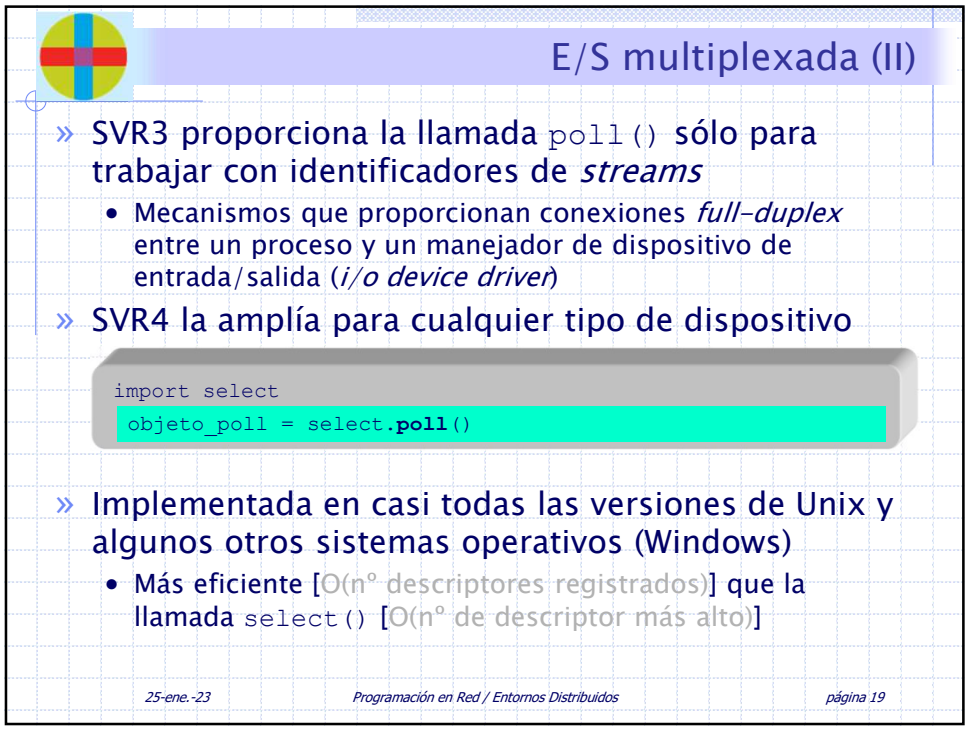
16


Valor de retorno de select()
» Valor de retorno de select():
• Devuelve tres listas con los descriptores listos para cada uno de los criterios.
• Cada una de las listas es un subconjunto de las pasadas como parámetros
• En Windows no se garantiza que se soporte la operación con más de una lista
• Si se produce un error (ejemplo: la recepción de una señal) select() lanza una excepción y fija errno con el valor apropiado.
› En este caso los valores de retorno son indefinidos y no se debe confiar en ellos.
› Posibles errores:
◦ EBADF – descriptor de fichero inválido en uno de los conjuntos.
◦ EINTR – capturada una señal no bloqueante.
◦ EINVAL – nfds es menor que cero.
◦ ENOMEM – no se ha podido pedir memoria para buffer interno.
25-ene.-23 Programación en Red / Entornos Distribuidos página 17

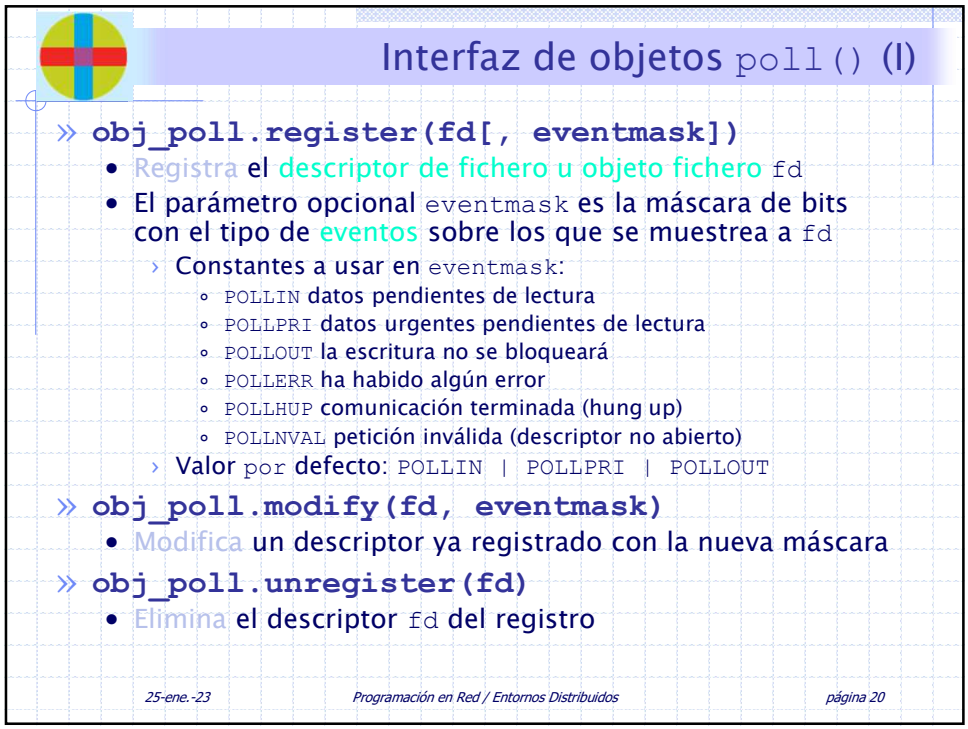
17


Descriptores listos para su uso
» ¿Cuándo está listo un descriptor?
• Un descriptor en el conjunto rlist está listo cuando es seguro que una lectura realizada sobre dicho descriptor no se bloqueará
• Un descriptor en el conjunto wlist está listo cuando es seguro que una escritura realizada sobre dicho descriptor no se bloqueará
• Un descriptor en el conjunto xlist está listo cuando:
› se reciben datos fuera de banda en una conexión de red
› en determinadas condiciones que pueden darse en una pseudo terminal
› Su uso queda como ejercicio para los más aventurados
25-ene.-23 Programación en Red / Entornos Distribuidos página 18

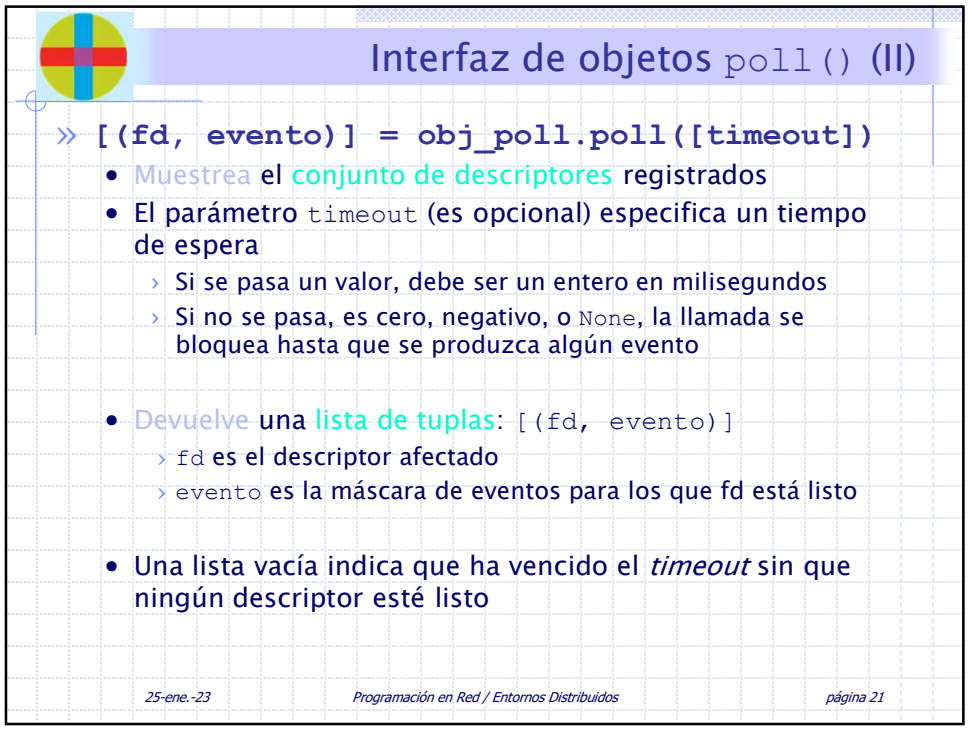
18


E/S multiplexada (II)
SVR3 proporciona la llamada poll() sólo para trabajar con identificadores de streams
Mecanismos que proporcionan conexiones full-duplex entre un proceso y un manejador de dispositivo de entrada/salida (i/o device driver)
SVR4 la amplía para cualquier tipo de dispositivo
import select
objeto_poll = select.poll()
Implementada en casi todas las versiones de Unix y algunos otros sistemas operativos (Windows)
Más eficiente [O(nº descriptores registrados)] que la llamada select() [O(nº de descriptor más alto)]
25-ene.-23
Programación en Red / Entornos Distribuidos
página 19

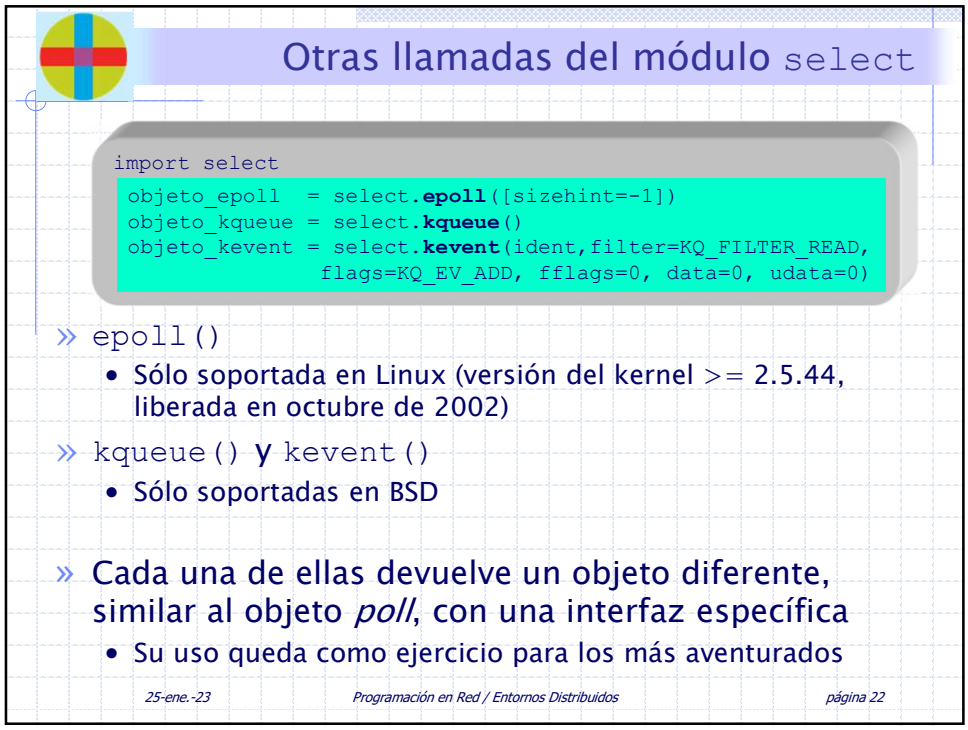
19


Interfaz de objetos poll() (I)
obj_poll.register(fd[, eventmask])
Registra el descriptor de fichero u objeto fichero fd
El parámetro opcional eventmask es la máscara de bits con el tipo de eventos sobre los que se muestrea a fd
Constantes a usar en eventmask:
POLLIN datos pendientes de lectura
POLLPRI datos urgentes pendientes de lectura
POLLOUT la escritura no se bloqueará
POLLERR ha habido algún error
POLLHUP comunicación terminada (hung up)
POLLNVAL petición inválida (descriptor no abierto)
Valor por defecto: POLLIN | POLLPRI | POLLOUT
obj_poll.modify(fd, eventmask)
Modifica un descriptor ya registrado con la nueva máscara
obj_poll.unregister(fd)
Elimina el descriptor fd del registro
25-ene.-23
Programación en Red / Entornos Distribuidos
página 20

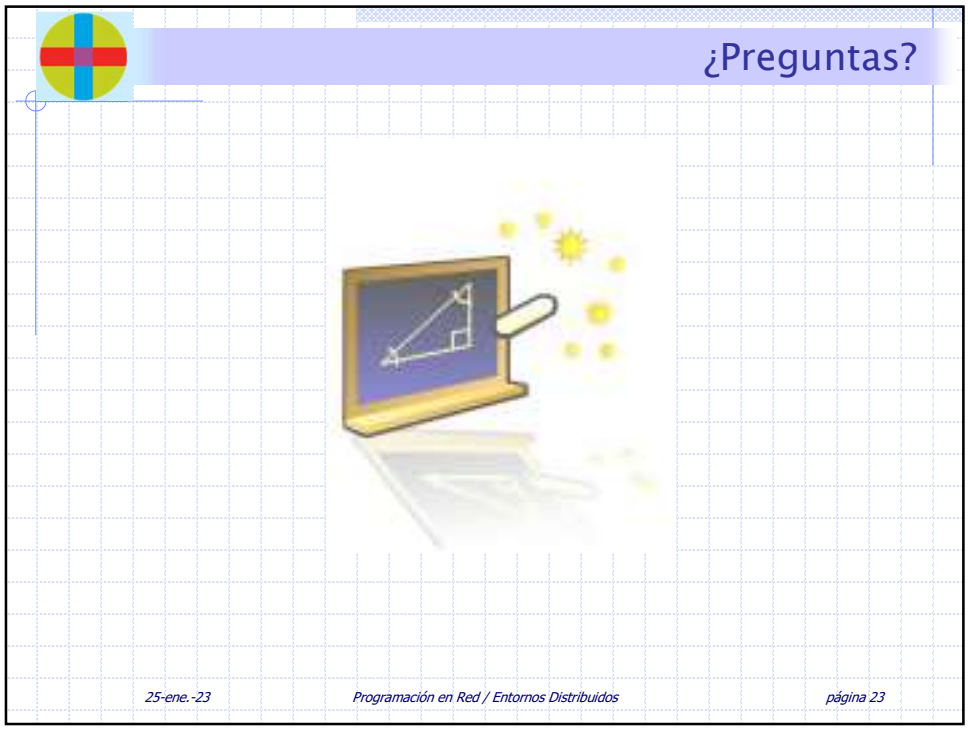
20


Interfaz de objetos poll() (II)
» [(fd, evento)] = obj_poll.poll([timeout])
• Muestrea el conjunto de descriptores registrados
• El parámetro timeout (es opcional) especifica un tiempo de espera
› Si se pasa un valor, debe ser un entero en milisegundos
› Si no se pasa, es cero, negativo, o None, la llamada se bloquea hasta que se produzca algún evento
• Devuelve una lista de tuplas: [(fd, evento)]
› fd es el descriptor afectado
› evento es la máscara de eventos para los que fd está listo
• Una lista vacía indica que ha vencido el timeout sin que ningún descriptor esté listo
25-ene.-23 Programación en Red / Entornos Distribuidos página 21

21


Otras llamadas del módulo select
import select
objeto_epoll = select.epoll([sizehint=-1])
objeto_kqueue = select.kqueue()
objeto_kevent = select.kevent(ident,filter=KQ_FILTER_READ,
flags=KQ_EV_ADD, fflags=0, data=0, udata=0)
» epoll()
• Sólo soportada en Linux (versión del kernel >= 2.5.44, liberada en octubre de 2002)
» kqueue() y kevent()
• Sólo soportadas en BSD
» Cada una de ellas devuelve un objeto diferente, similar al objeto poll, con una interfaz específica
• Su uso queda como ejercicio para los más aventurados
25-ene.-23 Programación en Red / Entornos Distribuidos página 22

22


¿Preguntas?
25-ene.-23
Programación en Red / Entornos Distribuidos
página 23