Sistemi Operativi

Laurea in Ingegneria Informatica

Universita' di Roma Tor Vergata

Docente: Francesco Quaglia

Pipes, named pipes e scambio messaggi

1. Nozioni preliminari
2. Pipes e named pipes (FIFO) in sistemi UNIX e Windows
3. Message queues UNIX
4. Mail slots Windows

Concetti di base sulle PIPE

- permettono comunicare usando in I/O il modello stream
- il termine “pipe” significa tubo in Inglese, e la comunicazione avviene in modo monodirezionale
- una volta lette, le informazioni spariscono dalla PIPE e non possono più ripresentarsi, a meno che non vengano riscritte all'altra estremità del tubo
- a livello di sistema operativo, i PIPE non sono altro che buffer di dimensione più o meno grande (ad esempio 4096 byte), quindi è possibile che un processo venga bloccato se tenta di scrivere su una PIPE piena
- i processi che usano PIPE devono essere “relazionati”
- le named PIPE (FIFO) permettono la comunicazione anche tra processi non relazionati
- in sistemi UNIX l’uso della PIPE avviene attraverso la nozione di descrittore, in sistemi Windows attraverso la nozione di handle

PIPE in Sistemi UNIX

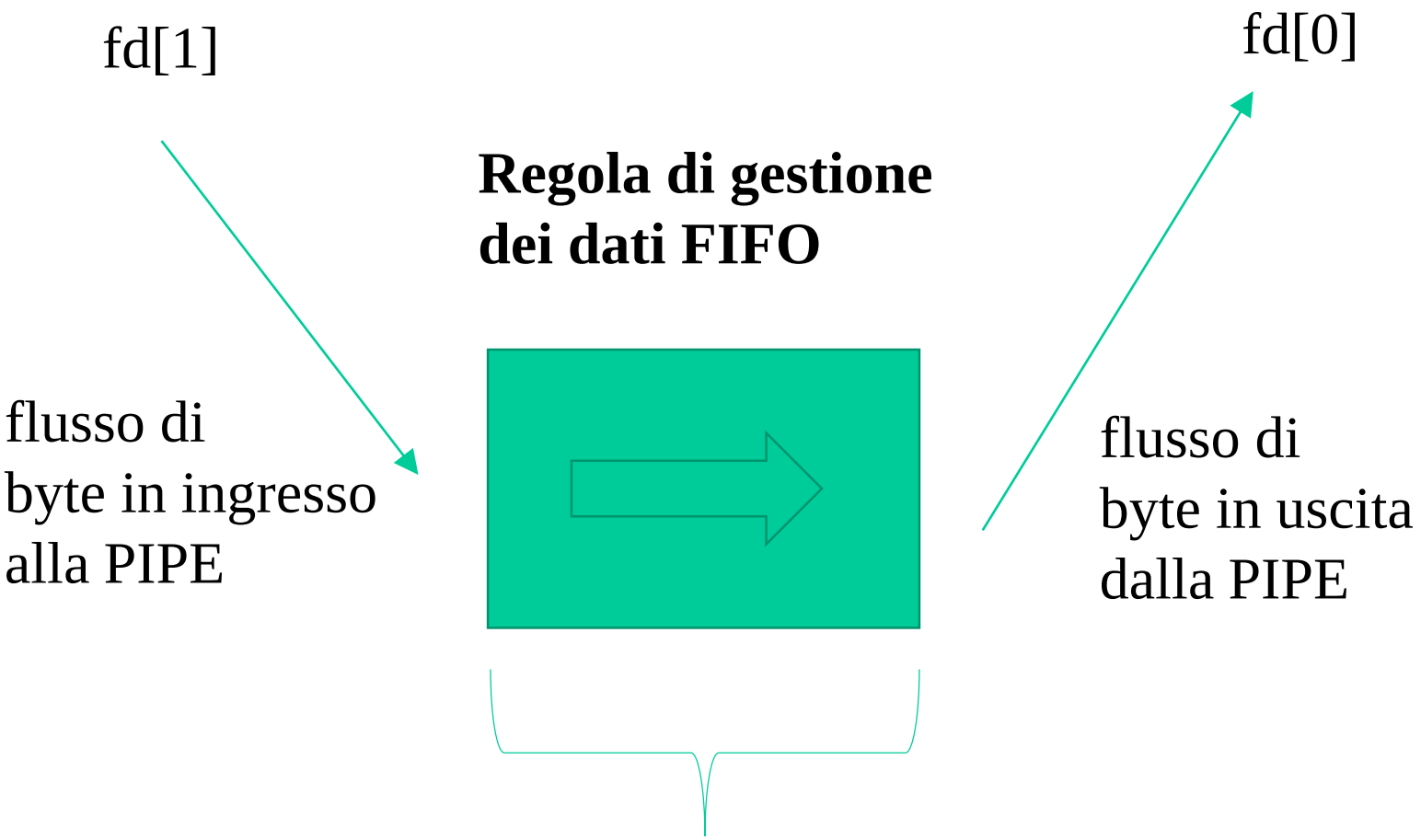
int pipe(int fd[2])	
Descrizione	invoca la creazione di una PIPE
Argomenti	fd: puntatore ad un buffer di due interi (in fd[0] viene restitutito il descrittore di lettura dalla PIPE, in fd[1] viene restituito il descrittore di scrittura sulla PIPE)
Restituzione	-1 in caso di fallimento

fd[0] è un canale aperto in lettura che consente di leggere dati da una PIPE

fd[1] è un canale aperto in scrittura che consente di immettere dati sulla PIPE

fd[0] e fd[1] possono essere usati come normali descrittori di file tramite `read()` e `write()`

Uno schema operativo



Buffer di livello kernel che implementa il dispositivo di I/O PIPE

Avvertenze

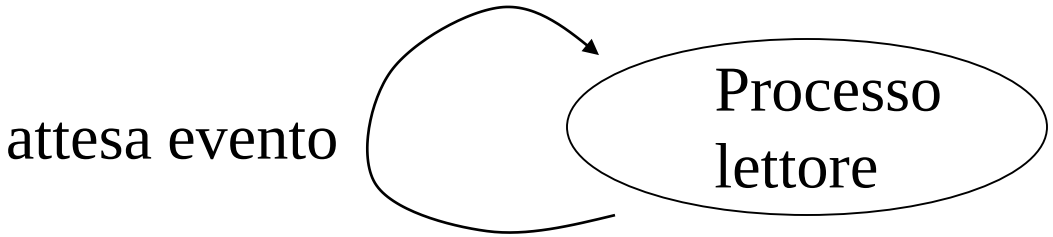
- le PIPE non sono dispositivi fisici, ma logici
- la fine di uno “stream” su una PIPE e’ una condizione logica sullo stato della PIPE
- per convenzione, la fine dello stream viene rilevata quando la PIPE non ha piu’ dati da consegnare e tutte le sessioni aperte in scrittura sono state chiuse
- tecnicamente, in questo caso la chiamata `read()` effettuata da un lettore restituisce zero
- allo stesso modo, se si tenta di scrivere sul descrittore fd[1] quando tutte le copie del descrittore fd[0] siano state chiuse (non ci sono lettori sulla PIPE), si riceve il “segnale SIGPIPE”, altrimenti detto Broken-pipe

PIPE e stalli (deadlock)

- Per fare in modo che tutto funzioni correttamente e non si verifichino situazioni di deadlock è necessario che tutti i processi chiudano i descrittori di PIPE che non gli servono, usando una normale `close()`

- Si noti che ogni processo lettore che erediti la coppia (fd[0],fd[1]) deve chiudere la propria copia di fd[1] prima di mettersi a leggere da fd[0] "dichiarando" così di non essere uno scrittore

- Se così non facesse, l'evento "tutti gli scrittori hanno terminato" non potrebbe mai avvenire se il lettore è impegnato a leggere, e si potrebbe avere un deadlock



Named PIPE (FIFO) in Sistemi UNIX

int mkfifo(char *name, int mode)

Descrizione invoca la creazione di una FIFO

Argomenti
1) *name: puntatore ad una stringa che identifica il nome della FIFO da creare
2) mode: intero che specifica modalità di creazione e permessi di accesso alla FIFO

Restituzione -1 in caso di fallimento

La rimozione di una FIFO dal file system avviene esattamente come per i file mediate la chiamata di sistema `unlink()`

Operare su una FIFO

- Per operare su una FIFO basta aprirla come se fosse un file regolare
- A differenza delle PIPE sulle FIFO si opera quindi con un unico descrittore
- Il driver che nel VFS implementa la logica di gestione della FIFO portera' ad operare sull'estremo di "ingresso" o su quello di uscita dipendendo da se l'operazione invocata e' una scrittura oppure una lettura

Avvertenze

- normalmente, l'apertura di una FIFO è bloccante, nel senso che il processo che tenti di aprirla in lettura (scrittura) viene bloccato fino a quando un altro processo non la apre in scrittura (lettura)

- se si vuole inibire questo comportamento è possibile aggiungere il flag `O_NONBLOCK` al valore del parametro mode passato alla system call `open()` su di una FIFO

- ogni FIFO deve avere sia un lettore che uno scrittore: se un processo tenta di scrivere su una FIFO che non ha un lettore esso riceve una notifica di errore tramite segnalazione (SIGPIPE) da parte del sistema operativo

PIPE in sistemi Windows

```
BOOL CreatePipe(PHANDLE hReadPipe,
                PHANDLE hWritePipe,
        LPSECURITY_ATTRIBUTES lpPipeAttributes,
                DWORD nSize)
```

Descrizione

• invoca la creazione di una PIPE

Argomenti

• hReadPipe: puntatore a una variabile in cui viene scritto l'handle all'estremità di lettura dalla pipe

• hWritePipe: puntatore a una variabile in cui viene scritto l'handle all'estremità di scrittura sulla pipe

• lpPipeAttributes: puntatore a una struttura SECURITY_ATTRIBUTES che specifica se gli handle ritornati sono ereditabili da processi figli

• nSize: dimensione suggerita della PIPE (0 attiva il default)

Restituzione

• 0 in caso di fallimento

Avvertenze

- hReadPipe è un canale aperto in lettura che consente ad un processo di leggere dati da una PIPE

- hWritePipe è un canale aperto in scrittura che consente ad un processo di immettere dati sulla PIPE

- hReadPipe e hWritePipe possono essere usati come se fossero normali handle di file tramite le chiamate `ReadFile()` e `WriteFile()`

- anche per Windows, la fine di un file su una PIPE è visibile, per convenzione, quando tutti i processi scrittori che condividevano l'handle hWritePipe lo hanno chiuso (e non ci sono piu' dati bufferizzati nella pipe)

- tecnicamente, in questo caso la chiamata `ReadFile()` effettuata da un lettore restituisce zero byte letti come notifica dell'evento che tutti gli scrittori hanno terminato il loro lavoro

Named PIPE in sistemi Windows

```
HANDLE CreateNamedPipe(LPCTSTR lpName,
                       DWORD dwOpenMode,
                       DWORD dwPipeMode,
                       DWORD nMaxInstances,
                       DWORD nOutBufferSize,
                       DWORD nInBufferSize,
                       DWORD nDefaultTimeOut,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes)
```

Descrizione

- invoca la creazione e/o apertura di un Named PIPE

Restituzione

- INVALID_HANDLE_VALUE in caso di fallimento; un handle alla named PIPE in caso di successo

Argomenti

- lpName: puntatore ad una stringa che identifica il nome della named PIPE da creare
- dwOpenMode: intero che specifica modalità di creazione e permessi di accesso alla named PIPE
- dwPipeMode: intero che specifica la tipologia dell'handle restituito
- nMaxInstances: specifica il massimo numero di istanze che possono essere create
- nOutBufferSize: dimensione massima del buffer di output (in bytes)
- nInBufferSize: dimensione massima del buffer di input (in bytes)
- nDefaultTimeOut: valore di timout (in millisecondi) da usare come default in particolari situazioni
- lpSecurityAttributes: puntatore a una struttura SECURITY_ATTRIBUTES che specifica se l'handle ritornato è ereditabile da processi figli

per dwOpenMode

Mode	Meaning
PIPE_ACCESS_DUPLEX 0x00000003	The pipe is bi-directional; both server and client processes can read from and write to the pipe. This mode gives the server the equivalent of GENERIC_READ | GENERIC_WRITE access to the pipe. The client can specify GENERIC_READ or GENERIC_WRITE, or both, when it connects to the pipe using the CreateFile function.
PIPE_ACCESS_INBOUND 0x00000001	The flow of data in the pipe goes from client to server only. This mode gives the server the equivalent of GENERIC_READ access to the pipe. The client must specify GENERIC_WRITE access when connecting to the pipe.
PIPE_ACCESS_OUTBOUND 0x00000002	The flow of data in the pipe goes from server to client only. This mode gives the server the equivalent of GENERIC_WRITE access to the pipe. The client must specify GENERIC_READ access when connecting to the pipe.

per dwPipeMode

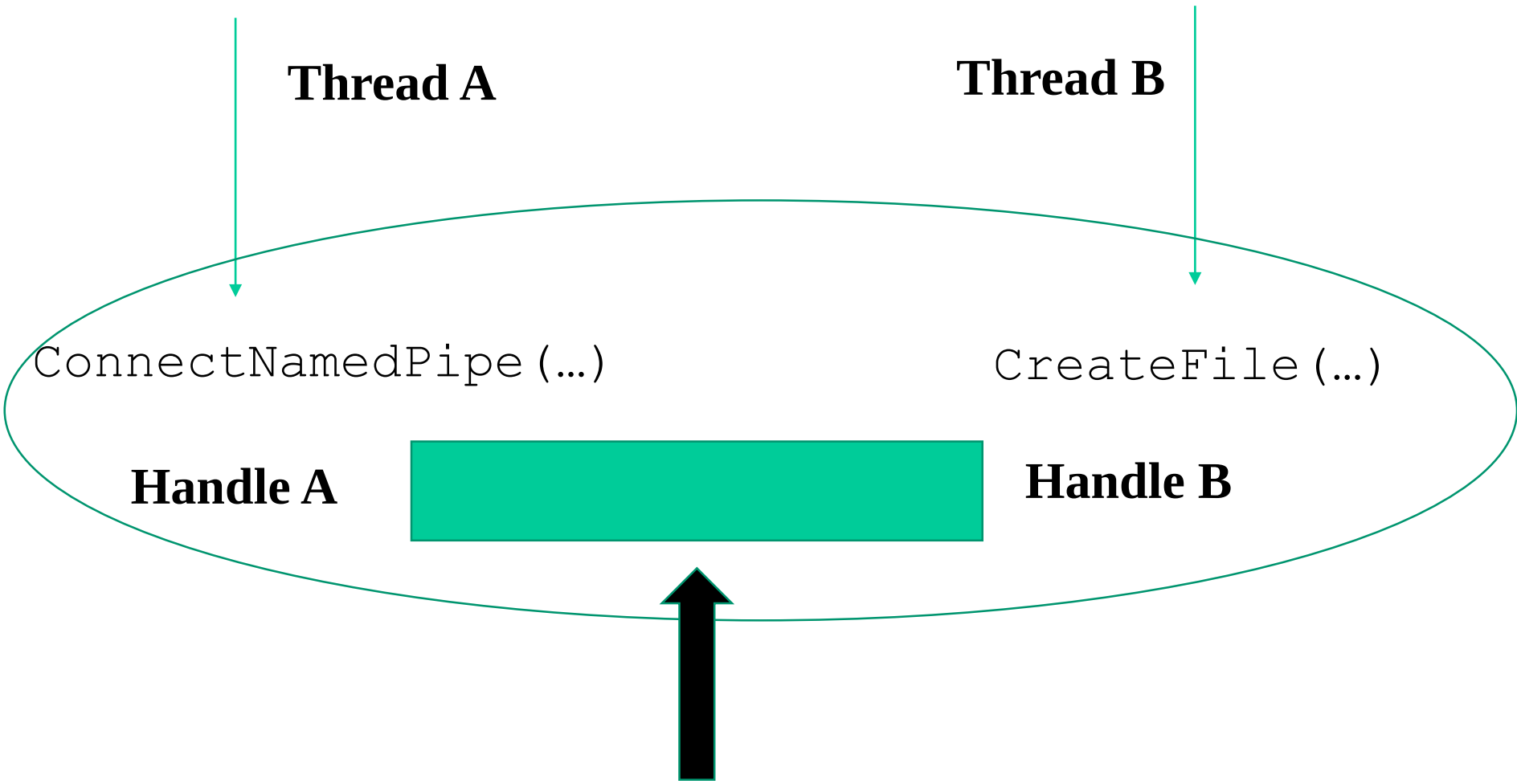
Mode	Meaning
PIPE_TYPE_BYTE 0x00000000	Data is written to the pipe as a stream of bytes. This mode cannot be used with PIPE_READMODE_MESSAGE.
PIPE_TYPE_MESSAGE 0x00000004	Data is written to the pipe as a stream of messages. This mode can be used with either PIPE_READMODE_MESSAGE or PIPE_READMODE_BYTE.

Mode	Meaning
PIPE_WAIT 0x00000000	Blocking mode is enabled. When the pipe handle is specified in the **ReadFile**, **WriteFile**, or ConnectNamedPipe function, the operations are not completed until there is data to read, all data is written, or a client is connected. Use of this mode can mean waiting indefinitely in some situations for a client process to perform an action.
PIPE_NOWAIT 0x00000001	Nonblocking mode is enabled. In this mode, **ReadFile**, **WriteFile**, and **ConnectNamedPipe** always return immediately. Note that nonblocking mode is supported for compatibility with Microsoft LAN Manager version 2.0 and should not be used to achieve asynchronous I/O with named pipes. For more information on asynchronous pipe I/O, see Synchronous and Overlapped Input and Output.

Avvertenze

- il nome della named PIPE (lpName) deve essere nel formato \\.\pipe\pipename
- “pipename” può essere qualunque stringa di lunghezza inferiore ai 256 caratteri e non includente i caratteri ``\'' e ``:''
- la rimozione di una named PIPE dal sistema avviene nel momento in cui viene chiuso l'ultimo handle aperto sulla named PIPE
- L’apertura avviene con una semplice `CreateFile()`

Schema di connessione



la named Pipe mette in “connessione” i due handle

Messaggi nei sistemi operativi

- sono unita' di dati che possono essere scambiate tra due o piu' processi tramite servizi di I/O
- il deposito (spedizione) di un messaggio avviene in modo "atomico"
- l'estrazione (ricezione) di un messaggio avviene anche essa in modo atomico
- due o piu' processi non possono quindi scambiarsi frazioni di un messaggio
- non necessariamente i messaggi hanno la stessa taglia

Primitive di spedizione

Send(destinazione, messaggio) ⟸ Modello generale

Modalita'

1. Sincrona: non ritorna il controllo fino a che il buffer contenente il messaggio puo' essere sovrascritto senza arrecare danni al messaggio appena spedito

2. Sincrona randez-vous: non ritorna il controllo fino a che il messaggio non viene ricevuto (uso di acknowledgment)

} bloccante e non

3. Asincrona: possibilita' di riutilizzo immediato del buffer contenente il messaggio (possibilita' di sovrascrittura del messaggio in corso di spedizione)

} non bloccante

Primitive di ricezione

Receive(sorgente, messaggio) ⟸ Modello generale

Modalita'

1. Bloccante: processo in wait fino a che non arriva almeno un messaggio dalla sorgente specificata } sincrona e non

3. Non-bloccante: il processo non va in wait neanche in caso un messaggio dalla sorgente specificata non sia disponibile } sincrona e non

---

nel caso di randez-vous la ricezione necessita di gestione di acknowledgment verso la sorgente del messaggio

Tecniche di indirizzamento

Diretta ⇨ Sorgenti e destinazioni coincidono con identificatori di processi (uso di wilde-cards in caso l'identificazione della sorgente non sia rilevante)

Esempio

Send(P,message)
Receive(Q,message)

Indiretta ⇨ Sorgenti e destinazioni coincidono con identificatori di mailbox (o code di messaggi) che fungono da deposito per i messaggi stessi

⇩

Le mailbox possono essere associate o non ad uno specifico processo (effetti sulla distruzione della mailbox alla terminazione del processo)

Esempio

Send(A,message)
Receive(A,message)

Possibilita' di relazione uno-a-molti e molti-a-molti

Buffering

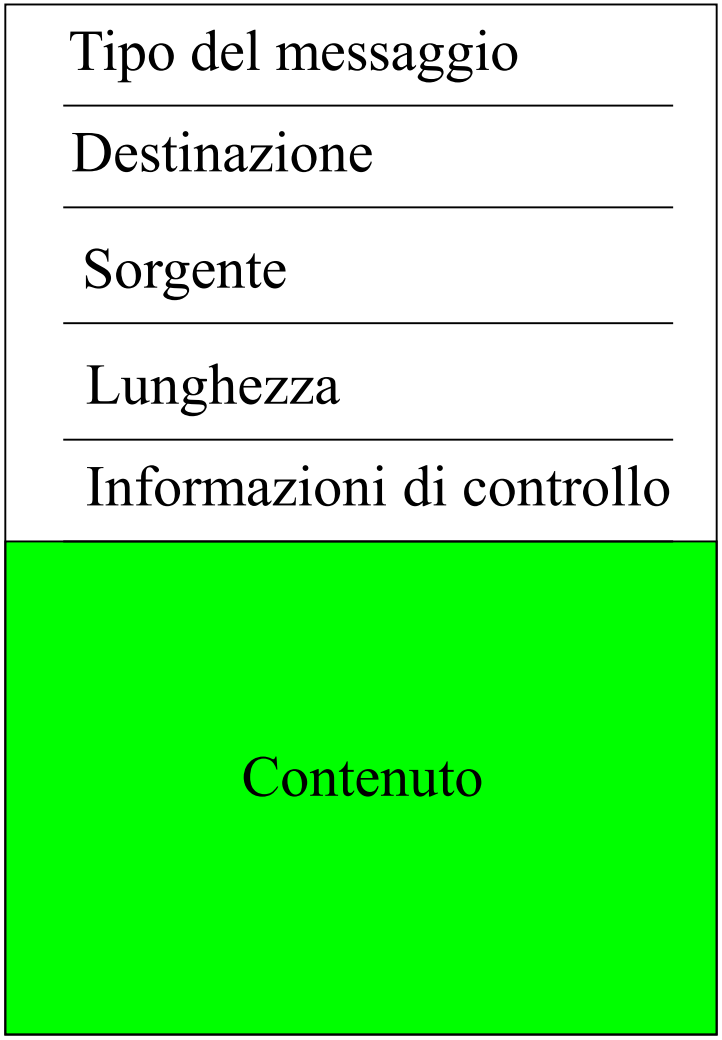
In memoria kernel

- non e' necessario che sia impostata una receive() all'atto dell'arrivo del messaggio
- la capacita' di bufferizzazione puo' essere nulla, limitata o illimitata
- in caso di capacita' nulla un solo messaggio alla volta puo' essere in transito (tipico del randez-vous) – adeguato in contesto di spedizione bloccante/sincrona
- e' possibile definire un timeout per la bufferizzazione di un messaggio, allo scadere del quale il messaggio viene scartato

In memoria utente

- e' necessario che sia impostata una receive() all'atto dell'arrivo del messaggio
- in caso la receive() non sia impostata, il messaggio in transito puo' venire perso (ad esempio per spedizioni non bloccanti sincrone)

Formato dei messaggi

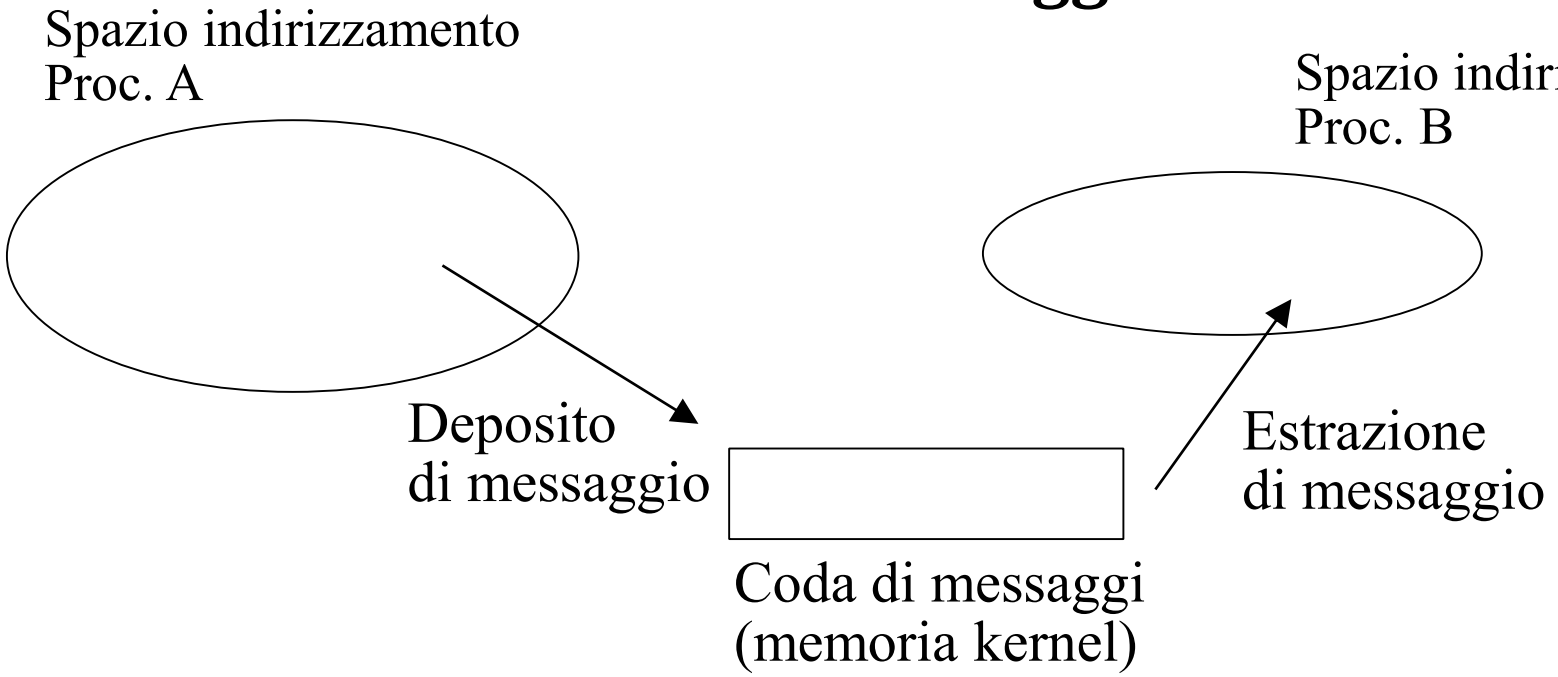


Campi tipici (ma non necessari) per applicazioni basate su scambio di messaggi

La definizione di questi campi come parte del contenuto del messaggio puo' essere a carico delle applicazioni e non del sistema operativo

Il tipo e le informazioni di controllo possono determinare l'ordine di consegna dei messaggi stessi

Code di messaggi UNIX



- send sincrona (bloccante o non)
- receive sincrona (bloccante e non bloccante)
- buffering in memoria kernel a capacita' limitata
- indirizzamento indiretto
- il sistema operativo gestisce il campo TIPO del messaggio (possibilita' di supporto al multiplexing)
- il contenuto di un messaggio ha taglia tipicamente variabile ma limitata superiormente

Creazione di una coda di messaggi

int msgget(key_t key, int flag)

Descrizione invoca la creazione una coda di messaggi

Parametri 1) key: chiave per identificare la coda di messaggi in maniera univoca nel sistema
2) flag: specifica della modalita' di creazione (IPC_CREAT, IPC_EXCL, definiti negli header file sys/ipc.h e sys/msg.h) e dei permessi di accesso

---

Descrizione identificatore numerico per l'accesso alla coda in caso di successo (descrittore di coda), -1 in caso di fallimento

NOTA

Il descrittore indicizza questa volta una struttura unica valida per qualsiasi processo

Controllo su una coda di messaggi

int msgctl(int ds_coda, int cmd, struct msquid_ds *buff)

Descrizione invoca l'esecuzione di un comando su una coda di messaggi

---

Parametri

1) ds_coda: descrittore della coda su cui si vuole operare
2) cmd: specifica del comando da eseguire (IPC_RMID, IPC_STAT, IPC_SET)
3) buff: puntatore al buffer con eventuali parametri per il comando

---

Descrizione -1 in caso di fallimento

`IPC_RMID` invoca la rimozione della coda dal sistema

Spedizione/ricezione di messaggi

int msgsnd(int ds_coda, const void *buff, size_t nbyte, int flag)

Descrizione invoca la spedizione di un messaggio su una coda

Parametri 1) ds_coda: descrittore della coda su cui si vuole operare
2) buff: puntatore al buffer che contiene il messaggio
3) nbyte: taglia del messaggio, in byte
4) flag: opzione di spedizione (IPC_NOWAIT e' non bloccante)

Descrizione -1 in caso di fallimento

int msgrcv(int ds_coda, void *buff, size_t nbyte, long type, int flag)

Descrizione invoca la ricezione di un messaggio da una coda

Parametri 1) ds_coda: descrittore della coda su cui si vuole operare
2) buff: puntatore al buffer che dovra' contiene il messaggio
3) nbyte: numero massimo di byte del messaggio da ricevere
4) type: tipo del messaggio da ricevere
5) flag: opzione di spedizione (IPC_NOWAIT e' non bloccante)

Descrizione -1 in caso di fallimento

Mailslot Windows

- send asincrona o sincrona (bloccante o non)
- receive bloccante e non bloccante (sincrona o non)
- buffering in memoria kernel ed a capacita' limitata
- indirizzamento indiretto
- nessuna gestione della tipologia di messaggi (no multiplexing)
- il contenuto di un messaggio ha taglia tipicamente variabile

---

La comunicazione avviene come scritture e letture simili a quelle su file

Ogni lettura ha effetto di estrarre tutto il messaggio scritto

Si ha errore se la lettura stessa chiede l'estrazione di un numero di byte inferiore rispetto all'intero contenuto del messaggio

Creazione di un Mailslot

```
HANDLE CreateMailslot(LPCTSTR lpName,
                      DWORD nMaxMessageSize,
                      DWORD lReadTimeout,
                      LPSECURITY_ATTRIBUTES
                          lpSecurityAttributes)
```

Descrizione

• invoca la creazione di un mailslot

Restituzione

• un handle al nuovo mailslot in caso di successo,

INVALID_HANDLE_VALUE in caso di fallimento

Parametri

- lpName: nome per identificare il mailslot in maniera univoca nel sistema

- nMaxMessageSize: massima dimensione del messaggio che puo' essere scritto nel mailslot

- lReadTimeout: tempo (in millisecondi) che una operazione di lettura su un Mailslot vuoto attende prima di ritornare (MAILSLOT_WAIT_FOREVER per aspettare indefinitamente)

- lpSecurityAttributes: puntatore a struttura SECURITY_ATTRIBUTES

Vincolo: il nome del mailslot deve avere la forma \\.\mailslot\[path]name

Apertura, lettura, scrittura e rimozione di un Mailslot

`CreateFile`	Permette di ottenere un handle per il mailslot preesistente
`ReadFile`	Permette di leggere messaggi dal mailslot
`WriteFile`	Permette di scrivere messaggi sul mailslot
`CloseHandle`	Permette rimuovere il mailslot in caso non vi siano altri handle validi per esso

Vincolo: la lettura deve specificare un numero di byte almeno pari a quelli scritti per un determinato messaggio