Interprocess Communications - II

Franco Maria Nardini

XSI IPC

- Tre tipi di IPC introdotti da System V:
  - semafori
  - shared memory
  - code di messaggi
- Comunicazioni tra processi su stesso host
  - Tutte consentono comunicazione *asincrona*

XSI IPC

- Ogni struttura IPC è identificata mediante un *identificatore*
  - intero non negativo che il kernel usa per riferire la risorsa
  - diversamente dai *file descriptors* (**?**), non sono piccoli
  - crescono e tornano a zero, quando raggiunto il MAX_INT
- Serie di comandi e system calls per interagire
  - `msgget(2)`, `semop(2)`
  - `ipcs(1)`

XSI IPC: Semafori

- Un semaforo è:
  - un contatore per consentire il corretto accesso ad una risorsa condivisa da parte di processi multipli
- Per ottenere la risorsa, un processo deve:
  - testare il semaforo che controlla la risorsa
  - se valore > 0, decremento la risorsa e la uso. al termine, incremento di nuovo il valore
  - se valore == 0, aspetto finché valore > 0
- I semafori sono ottenuti con **`semget(2)`**, controllati con **`semctl(2)`**. Le operazioni sono fatte con **`semop(2)`**.

Esempio

```
$ wget http://hpc.isti.cnr.it/~nardini/siselab/12/
semdemo.c

$ cc -Wall ./semdemo.c

$ ./a.out (prima shell)

$ ./a.out (seconda shell)

$ ipcs -s

$ ipcrm -s semID (rimuove il semaforo)
```

XSI IPC: Shared Memory

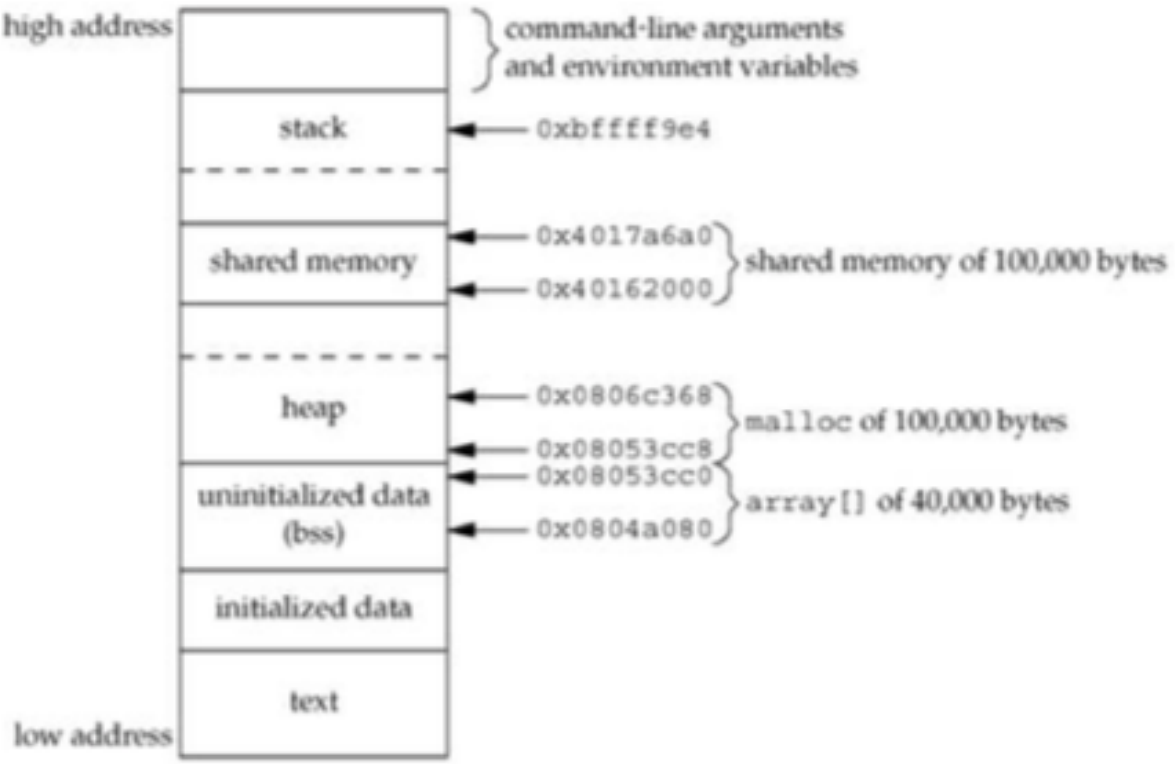
- Metodo più veloce di IPC
- Consiste nell'accedere ad una memoria condivisa a volte controllata con semafori
- ottenimento di un identificatore di memoria condivisa con **`shmget(2)`**
- collegamento del segmento condiviso allo spazio degli indirizzi di un processo con **`shmat(2)`**
- scollegamento con **`shmdt(2)`**

Esempio

```
$ wget http://hpc.isti.cnr.it/~nardini/siselab/12/
shmdemo.c

$ cc -Wall ./shmdemo.c

$ ./a.out ViBoccioTutti:)!

$ ipcs
```

Esempio

```
$ wget http://hpc.isti.cnr.it/~nardini/siselab/12/memory-layout.c

$ cc -Wall ./memory-layout.c

$ ./a.out
```



XSI IPC: Message Queues

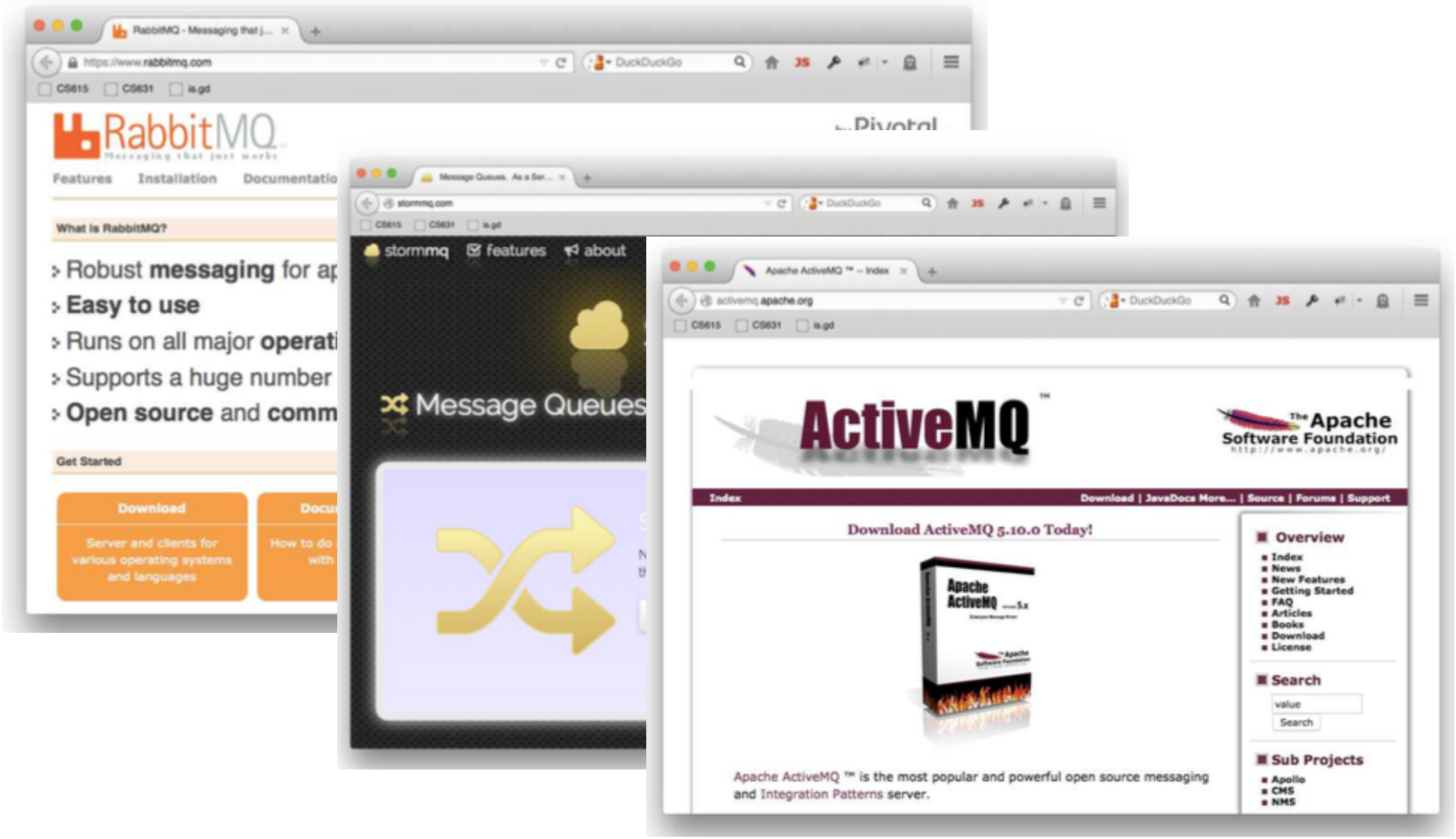
- Liste linkate (**?**) di messaggi memorizzate dal kernel
- Creare e aprire una coda con **`msgget(2)`**
- Aggiungere un messaggio a fine coda con **`msgsnd(2)`**
- Controllo della coda con msgctl(2)
- Ricezione messaggi dalla coda msgrcv(2)

XSI IPC: Message Queues

- Il messaggio ha una struttura definita dall'utente

```
struct mymsg {
    long mtype;       /* message type */
    char mtext[512]; /* body of message */
};
```

Altre Code


RabbitMQ - Messaging that j...
https://www.rabbitmq.com
RabbitMQ
Messaging that just works
Features
Installation
What is RabbitMQ?
Robust messaging for ap
Easy to use
Runs on all major operati
Supports a huge number
Open source and comm
Get Started
Download
Server and clients for various operating systems and languages
stormmq
features
about
Message Queues
Apache ActiveMQ ™ – Index
activemq.apache.org
ActiveMQ
The Apache Software Foundation
http://www.apache.org/
Index
Download | JavaDocs More... | Source | Forums | Support
Download ActiveMQ 5.10.0 Today!
Overview
Index
News
New Features
Getting Started
FAQ
Articles
Books
Download
License
Search
value
Search
Sub Projects
Apollo
CMS
NMS
Apache ActiveMQ ™ is the most popular and powerful open source messaging and Integration Patterns server.

Sockets

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

int socket(int domain, int type, int protocol);
```

Domain	Description
PF_LOCAL	local (previously UNIX) domain protocols
PF_INET	ARPA Internet protocols
PF_INET6	ARPA IPv6 (Internet Protocol version 6) protocols
PF_ARP	RFC 826 Ethernet Address Resolution Protocol
...	...

Type	Description
SOCK_STREAM	sequenced, reliable, two-way connection based byte streams
SOCK_DGRAM	connectionless, unreliable messages of a fixed (typically small) maximum length
SOCK_RAW	access to internal network protocols and interfaces
...	...

UNIX/LOCAL domain

- si crea un socket usando **`socket(2)`**
- ci si “attacca” al socket usando **`bind(2)`**
- **`bind`**are un nome in dominio UNIX, corrisponde a creare un file sul file system
- entrambe i processi devono conoscere e concordare il nome del socket
- i file servono per il rendezvous, non per il message delivery quando la connessione è stabilita
- il socket deve esser rimosso con **`unlink(2)`**

Esempio

```
$ wget http://hpc.isti.cnr.it/~nardini/siselab/12/
udgramsend.c

$ wget http://hpc.isti.cnr.it/~nardini/siselab/12/
udgramread.c

$ cc -Wall ./udgramsend.c -o send

$ cc -Wall ./udgramread.c -o read
```

Esempio

```
1$ ./read

2$ ls -l

2$ ./send socket

1$ (?)
```

Internet Domain

- utilizzano Internet! :)
- diversamente dai socket visti prima, i nomi di socket Internet non passano dal file system
  - non c'e' bisogno di unlink dopo la chiusura
- indirizzo può essere un qualsiasi indirizzo di rete valido
- porte riservate a root: 1-1023
- richiesta di una specifica porta con **`bind(2)`**
- determinare la porta usata con **`getsocketname()`**

Esempio

```
$ wget http://hpc.isti.cnr.it/~nardini/siselab/12/
dgramsend.c

$ wget http://hpc.isti.cnr.it/~nardini/siselab/12/
dgramread.c

$ cc -Wall ./dgramsend.c -o send

$ cc -Wall ./dgramread.c -o read
```

Esempio

```
1$ ./read

2$ ./send localhost #porta

1$ (?)
```

Stream Sockets

- connessioni asimmetriche: un processo richiede connessione, l'altro processo accetta la richiesta
- un socket creato per ogni richiesta accettata
- socket "sotto ascolto" con **`listen(2)`**
- richieste in attesa accettate con **`accept(2)`**
  - **`accept(2)`** blocca se nessuna connessione disponibile
- **`select(2)`** per controllare la presenza di connessioni in attesa

Esempio

```
$ wget http://hpc.isti.cnr.it/~nardini/siselab/12/
streamread.c

$ wget http://hpc.isti.cnr.it/~nardini/siselab/12/
streamwrite.c

$ cc -Wall ./streamread.c -o read

$ cc -Wall ./streamwrite.c -o write
```

Esempio

```
1$ ./read

2$ ./write localhost #porta

1$ (?)
```

Manipolare il socket

- I/O su socket è fatto su descrittori, come normale I/O visto finora
  - `read(2)` e `write(2)` possono esser usate.
- Alcuni usi avanzati hanno bisogno di funzioni ad-hoc:
  - `send(2)`, `sendto(2)`, `sendmsg(2)`
  - `recv(2)`, `recvfrom(2)`, `recvmsg(2)`

Manipolare il socket

- Per gestire le proprietà del socket: `setsockopt(2)`

Option	Description
SO_DEBUG	enables recording of debugging information
SO_REUSEADDR	enables local address reuse
SO_REUSEPORT	enables duplicate address and port bindings
SO_KEEPALIVE	enables keep connections alive
SO_DONTROUTE	enables routing bypass for outgoing messages
SO_LINGER	linger on close if data present
SO_BROADCAST	enables permission to transmit broadcast messages
SO_OOBINLINE	enables reception of out-of-band data in band
SO_SNDBUF	set buffer size for output
SO_RCVBUF	set buffer size for input
SO_SNDLOWAT	set minimum count for output
SO_RCVLOWAT	set minimum count for input
SO_SNDTIMEO	set timeout value for output
SO_RCVTIMEO	set timeout value for input
SO_TIMESTAMP	enables reception of a timestamp with datagrams
SO_TYPE	get the type of the socket (get only)
SO_ERROR	get and clear error on the socket (get only)

Homework

- Esercizio 8:
  - scrivere una chat bidirezionale. Il processo client e server si scambiano stringhe che vengono stampate a video E salvate su un file di testo (una stringa per riga).