



6. Inter Process Communication (1)

5
6. IPC

6.1 Introduction

/home/esser/Daten/Dozent/Folien/bs-esser-17-english.odp

**Addendum to chapter 5:
Synchronization with Python**

Sep 19 14:27:41 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[29278]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 20 01:00:01 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[30103]: (root) CMD ('/sbin/evlogmgr -c "age > 30d"')
Sep 20 12:45:44 amd64 sshd[15161]: Accepted rra for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:45:44 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 20 12:48:41 amd64 sshd[6699]: Accepted rra for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6949]: Accepted rra for esser from ::ffff:87.234.201.207 port 62422
Sep 20 13:00:01 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 20 15:27:35 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 20 16:37:11 amd64 sshhd[10102]: Accepted rra for esser from ::ffff:87.234.201.207 port 6337
Sep 20 16:47:11 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 21 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 21 01:00:01 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[17878]: (root) CMD ('/sbin/evlogmgr -c "age > 30d"')
Sep 21 02:00:01 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 21 17:43:26 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 21 17:53:39 amd64 sshd[31261]: Accepted rra for esser from ::ffff:87.234.201.207 port 6439
Sep 21 18:43:26 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[14741]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 22 01:00:01 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[4499]: (root) CMD ('/sbin/evlogmgr -c "age > 30d"')
Sep 22 02:00:01 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 22 03:00:01 amd64 /usr/sbin/cron[14741]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 23 01:00:01 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[3855]: (root) CMD ('/sbin/evlogmgr -c "age > 30d"')
Sep 23 18:04:05 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 23 18:04:05 amd64 sshd[6554]: Accepted rra for esser from ::ffff:87.234.201.207 port 62093
Sep 24 01:00:01 amd64 /usr/sbin/cron[12431]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 24 01:00:01 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[7647]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 24 02:00:01 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 24 11:15:48 amd64 sshd[20981]: Accepted rra for esser from ::ffff:87.234.201.207 port 6445
Sep 24 13:49:08 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 24 13:49:08 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 24 15:42:07 amd64 kernel: [nd_mdev_nid] event: unmapported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 24 20:25:33 amd64 sshd[23939]: Accepted rra for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:33 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[6621]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 25 01:00:02 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 25 02:00:02 amd64 /usr/sbin/cron[6621]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 25 02:00:02 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rra for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 25 11:59:25 amd64 sshd[11160]: Accepted rra for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:59:25 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 25 11:59:25 amd64 sshd[11160]: Accepted rra for esser from ::ffff:87.234.201.207 port 66209
Sep 25 14:05:37 amd64 sshd[11545]: Accepted rra for esser from ::ffff:87.234.201.207 port 6282
Sep 25 14:05:37 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rra for esser from ::ffff:87.234.201.207 port 6295
Sep 25 14:07:17 amd64 syslog-ng[7651]: STATSS: dropped 0
Sep 25 14:08:33 amd64 sshd[11603]: Accepted rra for esser from ::ffff:87.234.201.207 port 6370
Sep 25 15:25:33 amd64 sshd[12901]: Accepted rra for esser from ::ffff:87.234.201.207 port 6777

Synchronization with Python (1)

- 100 Threads manipulate variable
 - Result of calculation should be 0
 - Problem: critical section when accessing the global variable

```
ite a from threading import Thread

class testthread(Thread):
    def __init__(self):
        Thread.__init__(self)
    def run(self):
        global globalcount
        # start of critical section
        for j in range(0,99999):
            globalcount += 100
            globalcount -= 100
        # end of critical section

globalcount=0          # glob. variable
threads = []
for i in range(0,100):  # start threads
    t = testthread()
    threads.append(t)
    t.start()
for t in threads: t.join()  # clean-up

print "Result:",globalcount
```

Hans-Georg Eßer, FH Münch

Operating Systems I, WS 2006/07 – 2006-12-

6. IPC (1) – Slide 3

Synchronization with Python (2)

1st solution: Loc|

- acquire: lock
 - release: unlock

```
$ ./lock.py
Result: 0
$ ./lock.py
Result: 0
$ ./lock.py
Result: 0
$ ./lock.py
Result: 0
```



```
$ time ./test.py
real    0m5.560s
$ time ./lock.py
real    0m5.560s
```

(no speed difference)

```

from threading import Thread, Lock

class testthread(Thread):
    def __init__(self):
        Thread.__init__(self)
    def run(self):
        global globalcount
        # start critical section
        mylock.acquire()
        for j in range(0,99999):
            globalcount += 100
            globalcount -= 100
        mylock.release()
        # end of critical section

globalcount=0                      # glob. variable
threads = []
mylock = Lock()                     # global lock
for i in range(0,100):               # start threads
    t = testthread()
    threads.append(t)
    t.start()
for t in threads: t.join()          # clean-up

print "Result:",globalcount

```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-

6. IPC (1) – Slide 4

Synchronization with Python (3)

- **Lock**

- acquire can also be used in a non-blocking mode:
`mylock.acquire(0)`
 - when the lock is available, `acquire()` returns **True** (the calling thread now holds the lock)
 - when another process already holds the lock, `acquire(0)` will not wait, but immediately return **False**.

- **RLock (Reentrant Lock)**

- like **Lock**, but can be used recursively
- lock must be `release()`d as often as it was `acquire()`d

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 5

Synchronization with Python (4)

2nd solution:
BoundedSemaphore

- acquire: count down
- release: count up

```
$ ./sem.py  
Result: 0  
$ ./sem.py  
Result: 0  
$ ./sem.py  
Result: 0  
$ ./sem.py  
Result: 0
```

```
from threading import Thread, BoundedSemaphore  
  
class testthread(Thread):  
    def __init__(self):  
        Thread.__init__(self)  
    def run(self):  
        global globalcount  
        # start critical section  
        mysem.acquire()  
        for j in range(0,99999):  
            globalcount += 100  
            globalcount -= 100  
        mysem.release()  
        # end of critical section  
  
        globalcount=0          # glob. variable  
        threads = []  
        mysem = BoundedSemaphore(1) # init: 1  
        for i in range(0,100):    # start threads  
            t = testthread()  
            threads.append(t)  
            t.start()  
        for t in threads: t.join() # clean-up  
  
        print "Result:",globalcount
```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 6

Synchronization with Python (5)

- **BoundedSemaphore**

- for granting access to several (identical) resources: initialize with bigger counter value
- **acquire** = **Wait** operation
 - with argument 0: non-blocking
- **release** = **Signal** operation

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 7

Synchronization with Python (6)

- **Condition: condition variables**

- cf. Java / Monitor
- condition variable is always protected by a lock
- functions
 - cv.acquire ()** acquire corresponding lock (or block)
 - cv.release ()** release corresponding lock
 - cv.wait ()** release lock and block until signal arrives
 - cv.notify ()** wake up one of the threads waiting for **cv**
 - cv.notifyAll ()** wake up all threads waiting for **cv**

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 8

Synchronization with Python (7)

Condition: Producer Consumer Problem

```
from threading import Thread, Condition
cv = Condition()

# Consume one item
cv.acquire()
while not an_item_is_available():
    cv.wait()
get_an_available_item()
cv.release()

# Produce one item
cv.acquire()
make_an_item_available()
cv.notify()
cv.release()
```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 9

6.1 IPC: Introduction

```
Sep 19 14:28:18 amd64 sshd[20494]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61557
Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[29278]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[101031]: (root) CMD (/sbin/evlogmgr -c "age > 30d")
Sep 20 02:00:01 amd64 sshd[6516]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 sshd[6516]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:54:44 amd64 sshd[6594]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 sshd[9077]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[101021]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:37:11 amd64 sshd[101401]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[170555]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[170555]: (root) CMD (/sbin/evlogmgr -c "age > 30d")
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[11088]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:39 amd64 sshd[312691]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 17:43:39 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[46741]: (root) CMD (/sbin/evlogmgr -- "severity=DEBUG")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[14898]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 20:23:21 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[247391]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[132531]: (root) CMD (/sbin/evlogmgr -c "age > 30d")
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[6554]: Accepted publickey for esser from ::ffff:87.234.201.207 port 61330
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sshd[6606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62993
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[132531]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[231971]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel_and_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 kernel_and_seq_neo: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sshd[293991]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[4621]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[14841]: (root) CMD (/sbin/evlogmgr -c "age > 30d")
Sep 25 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[88991]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[93211]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[93721]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[115541]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog_ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[115861]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[116081]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:07:17 amd64 sshd[116301]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[129301]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778
```

Synchronization with Python (8)

Event: corresponds to „manual events“ (Windows)

```
from threading import Thread, Event
ev = Event()

ev.set()                      # set event
ev.clear()                     # reset/clear event
ev.isSet()                     # query event status
ev.wait()                      # block until status changes to set
```

longer description in the Python documentation:
<http://docs.python.org/lib/module-threading.html>

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 10

Inter Process Communication

- message exchange between several processes or threads
- connection through communication system
- sender and receiver
- necessary when there is no shared memory
- yet another synchronization method

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 12

IPC Characterization (1)

- communication model:
 - point-to-point communication
 - publish-subscribe communication
 - broadcast communication
- transmission direction:
 - simplex / uni-directional
 - duplex / bi-directional
- Synchronicity
 - synchronous / blocking
 - asynchronous / non-blocking

Communication Model

- point-to-point
 - exactly one sender and one receiver
- broadcast
 - one sender and several receivers
- publish & subscribe
 - peer-to-peer communication
 - publisher: threads which change some system state
 - subscriber: threads which are notified when such changes occur

IPC Characterization (2)

- message type
 - messages
 - stream
- platform (in-)dependence
- portability
- locality
 - system-bound (local) or
 - allows network communication (across machine borders)

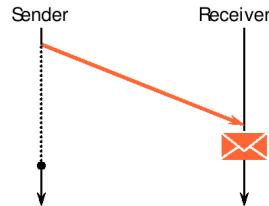
Transmission Direction

- simplex message from sender to receiver
 - beginning: sender posts message,
 - end: delivery at the receiver
 - no reply expected
- duplex message
 - beginning: sender posts message,
 - end: acknowledgement / receipt is delivered at the sender
 - in between: request is processed by the receiver
 - in the simplest case: two messages (there and back)
 - detect missing receipts via timeouts (negative receipts)

Synchronicity

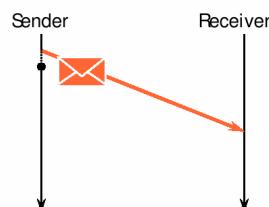
Synchronous / blocking communication

- sender blocks until message arrives (at receiver)
- needs almost no buffer capacity
- restricted parallelism



Asynchronous / non-blocking communication

- sender only blocks until message has been copied into the buffer
- needs larger buffer capacity
- sender can send several messages in succession



Picture: (c)Peter Sturm, Uni Trier, „Episodes on Operating Systems“
http://strathisla.uni-trier.de/ectblog/wp-content/Syssoft1_13_Communication.pdf

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 17

Four Elementary Communication Types

combining two transmission directions and the choice of synchronous vs. asynchronous leads to

	asynchronous	synchronous
simplex: message	-1- asynchronous message	-2- synchronous message
duplex: request	-3- asynchronous request	-4- synchronous request

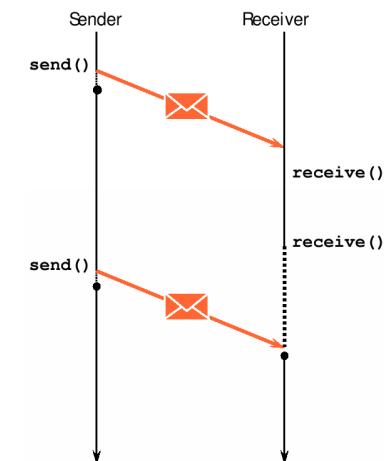
Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 18

-1- asynchronous message

- sender and receiver are uncoupled -> parallelism
- **UDP**: User Datagram Protocol (IP-based)
- signals: Linux software interrupts



graphics on the following four slides:
(c) Peter Sturm, Uni Trier, „Episodes on Operating Systems“,
http://strathisla.uni-trier.de/lectblog/wp-content/Syssoft1_13_Communication.pdf

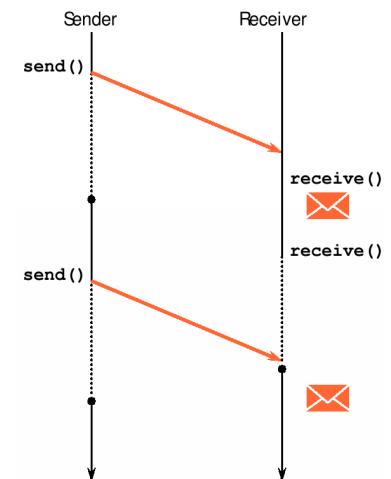
Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 19

-2- synchronous message

- limited parallelism: sender must wait for a receipt from the receiver
- no buffers needed
- sender knows that message arrived at destination



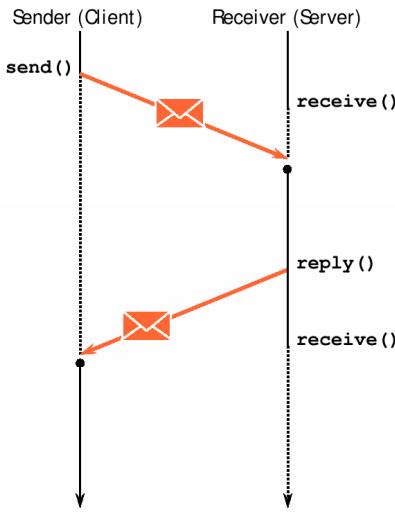
Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 20

-4- synchronous request

- RPC: Remote Procedure Call
- sender waits until he has received a reply to his request [not for a receipt]
- e.g. databases



Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 21

Sockets (1)

- sockets are a network communication tool that offers a communication channel with the following properties:
 - they allow IPC between processes
 - either local or over the network
 - can work with several (low-level) protocols
 - platform-independent
- bi-directional communication

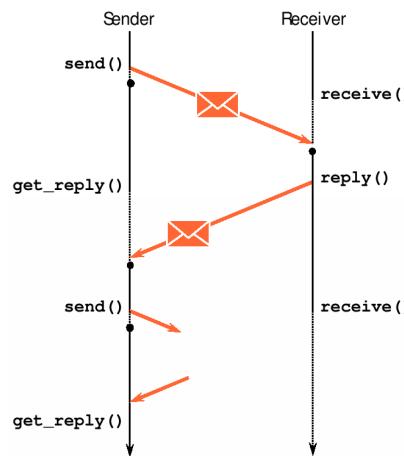
Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 23

-3- asynchronous request

- client sends request, but does not wait for the result
- at a later point in time he accepts the server's reply
- thus: fewer blocking times, more parallelism



Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 22

Sockets (2)

- addressing, e.g.
 - local (AF_UNIX): path name
 - network address (AF_INET): host + port, (local via AF_INET: host = localhost, 127.0.0.1)
- reliability:
 - reliable, connection oriented (e.g. TCP): error-free, no packet loss, no duplicates, message order is preserved
 - unreliable, connection-less (e.g. UDP / datagram)

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

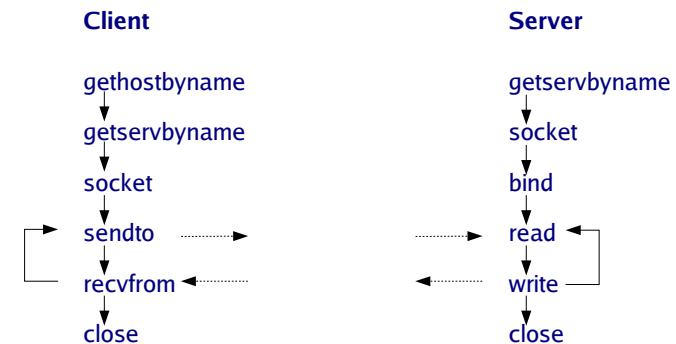
6. IPC (1) – Slide 24

Sockets (3)

- communication types:
 - message-oriented: **recvmsg()** or **sendmsg()**
 - stream-oriented: **sendto()**, **recfrom()**, **read()**, **write()**
 - typically with an 8 KByte buffer
 - sending of larger data will block until the other side has read the data
 - optionally choose non-blocking behavior
 - Unix/Linux maps sockets to file descriptors (accessing them is similar as with regular files)

Connectionless Sockets

Connectionless communication via datagrams / UDP

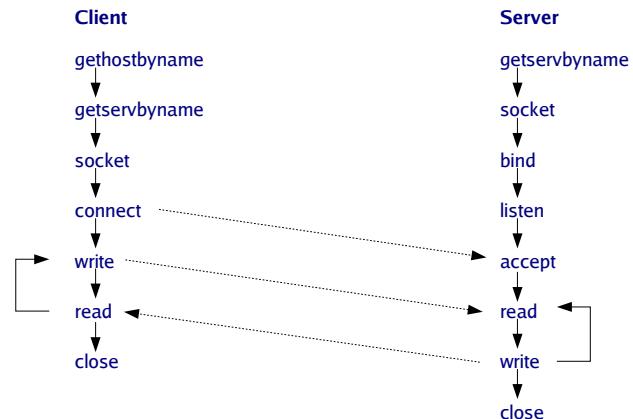


Sockets (4)

- Application
 - possible problems:
 - port busy --> **lsof** („list open files“ command)
 - port is still busy --> using „linger“ option may help
 - message boundaries might not remain intact:
e.g. `send(170 bytes) + send(230 bytes)`
→ `receive(400 bytes)`
 - API functions allow optional choice of non-blocking behavior
 - for experiments: C examples for Client/Server

Connection-oriented Sockets

Connection-oriented communication via streams / TCP



Datagram Server

```
#include <unistd.h>           // read(), close()
#include <arpa/inet.h>         // sockaddr_in, INADDR_ANY
#include <sys/socket.h>         // SOCK_DGRAM, socket(), bind()

const short port = 5242;
int n, sockfd;
char buf[256];
struct sockaddr_in serv_addr;

int main() { // UDP_Socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) perror("opening datagram");

    // Create name with wildcards
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(port);

    if (bind(sockfd, (sockaddr *)&serv_addr, sizeof(serv_addr)) != 0)
        perror("binding to address");

    n = read(sockfd, buf, sizeof(buf)); printf("Received: %s\n", buf);
    close(sockfd); return 0;
}
```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 29

Connection-oriented Server

```
#include <unistd.h>           // read(), close()
#include <arpa/inet.h>         // sockaddr_in, INADDR_ANY
#include <sys/socket.h>         // SOCK_DGRAM, socket(), bind()

const short port = 5242, waitqueuelen = 1;
int n, sockfd, con;
char buf[256];
struct sockaddr_in serv_addr;

int main() { // TCP_Socket
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) perror("opening stream");
    serv_addr...;

    if (bind(sockfd, (sockaddr *)&serv_addr, sizeof(serv_addr)) != 0)
        perror("binding to address");

    if (listen(sockfd, waitqueuelen) != 0) perror("listening to address");

    if ((con = accept(sockfd, (sockaddr *)&peer_addr, sizeof(serv_addr))) < 0)
        perror("accepting client");

    n = read(con, buf, sizeof(buf)); printf("Received: %s\n", buf);
    write(con, buf, n);
    close(con); close(sockfd); return 0;
}
```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 31

Datagram Client

```
#include <unistd.h>           // read(), close()
#include <arpa/inet.h>         // sockaddr_in, AF_INET
#include <sys/socket.h>         // SOCK_DGRAM, socket(), bind()
#include <sys/param.h>          // MAXHOSTNAMELEN
#include <netdb.h>              // gethostbyname()

const short port = 5242;
char hostname[MAXHOSTNAMELEN+1] = "server";
int sockfd;
struct sockaddr_in peer_addr;

int main() { // UDP_Socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) perror("opening datagram");

    struct hostent * hp = gethostbyname(hostname);
    bcopy(hp->h_addr, (char *)&peer_addr.sin_addr, hp->h_length);
    peer_addr.sin_family = AF_INET;
    peer_addr.sin_port = htons(port);

    if (sendto(sockfd, "Hello World", 11, 0, (sockaddr *)&peer_addr,
               sizeof(peer_addr)) < 0) perror("sending data");

    close(sockfd); return 0;
}
```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 30

Connection-oriented Client

```
#include <unistd.h>           // read(), close()
#include <arpa/inet.h>         // sockaddr_in, AF_INET
#include <sys/socket.h>         // SOCK_DGRAM, socket(), bind()
#include <sys/param.h>          // MAXHOSTNAMELEN
#include <netdb.h>              // gethostbyname()

const short port = 5242;
char hostname[MAXHOSTNAMELEN+1] = "server";
int sockfd;
struct sockaddr_in peer_addr;

int main() { // TCP_Socket
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) perror("opening stream");
    peer_addr...;

    if (connect(sockfd, (sockaddr *)&peer_addr, sizeof(peer)) != 0)
        perror("connecting server");

    write(sockfd, "Hello World", 11);
    read(sockfd, buf, sizeof(buf)); printf("Received: %s\n", buf);

    close(sockfd); return 0;
}
```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-13

6. IPC (1) – Slide 32