

► Problem:

- Imagine we want to communicate an n -byte buffer x from process P_i to process P_j .
- As described so far, the processes
 - are *totally* protected wrt. each other,
 - can be executed in *any* order,
 - can be suspended at *any* time,
 - can remain suspended for *any* period, and
 - can be resumed at *any* time.

► Solution: we need mechanisms to

1. identify the end-points,
2. synchronise the end-points, and
3. communicate the data,

i.e., **Inter-Process Communication (IPC)**.

► Note that:

- most UNIX-like kernels support *several* IPC interfaces, but we'll focus on POSIX only,
- there are *numerous* alternatives summarised by

Type	Standard	Mechanism	Identifier
Synchronisation	System V	semaphore	keyed
	POSIX	semaphore	named or unnamed
Notification	POSIX	signal	PID
Communication	System V	shared memory	keyed
	POSIX	shared memory	named
	Linux	shared mapping	named or unnamed
	System V	message queue	keyed
	POSIX	message queue	named
	POSIX	pipe	named or unnamed
	POSIX	domain socket	named or unnamed

but we'll focus on a few only.

Definition

A **critical region** (or **critical section**) is a portion of a (multi-threaded) program that may not be executed concurrently (i.e., not executed by more than one thread of execution at the same time). A typical example is access to some shared resource, which, if it *were* concurrent, would fail somehow.

IPC-related synchronisation: semaphore (1)

Definition (Dijkstra [18])

A **semaphore** s is a counter, equipped with two operations

$$\begin{aligned} V(s, x) &:= [s \leftarrow s + x] \\ P(s, x) &:= \textbf{forever do} [\textbf{if } s \geq x \textbf{ then } s \leftarrow s - x, \textbf{break}] \end{aligned}$$

to increment and decrement it by x (typically $x = 1$). The semaphore is used to control concurrent access to some resource: intuitively, s is the number of concurrent users allowed (resp. “units” of the resource available) and P waits until access is allowed.

Definition

A **mutex** can be described as a *binary* semaphore (cf. a counting semaphore, a generalisation from 2 to n values) that simply allows or disallows access.

IPC-related synchronisation: semaphore (2)

ARMv7-A synchronisation primitives

► ARMv7-A provides two forms of support in hardware, namely

1. atomic load and store [17, Section 1.2.1]:

$$\begin{aligned} \text{ldrex } r1, [r0] &\mapsto \begin{cases} \text{GPR}[1] \leftarrow \text{MEM}[\text{GPR}[0]] \\ \text{update monitor(s) wrt. address } x = \text{GPR}[0] \end{cases} \\ \\ \text{strex } r2, r1, [r0] &\mapsto \begin{cases} \text{if } \text{monitor(s) allow access to address } x = \text{GPR}[0] \text{ then} \\ \quad | \text{MEM}[\text{GPR}[0]] \leftarrow \text{GPR}[1] \\ \quad | \text{GPR}[2] \leftarrow 0 \\ \text{else} \\ \quad | \text{GPR}[2] \leftarrow 1 \\ \text{end} \end{cases} \end{aligned}$$

2. atomic swap [17, Appendix A]:

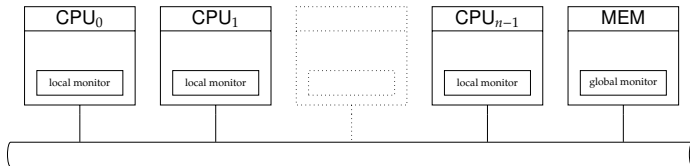
$$\text{swp } r2, r1, [r0] \mapsto \begin{cases} t \leftarrow \text{MEM}[\text{GPR}[0]] \\ \text{MEM}[\text{GPR}[0]] \leftarrow \text{GPR}[2] \\ \text{GPR}[1] \leftarrow t \end{cases}$$

the former of which is now (strongly) preferred.

IPC-related synchronisation: semaphore (3)

ARMv7-A synchronisation primitives

- **Idea:** hardware-based **exclusive access monitors**.



- two types, namely local and global, of monitor are operated; these are essentially state machines,
- for a given access to MEM[x],
 1. non-shared region \Rightarrow checked wrt. local *only*
 2. shared region \Rightarrow checked wrt. local *plus* global monitor
- ldrex from address x succeeds, but updates the monitor(s) by “tagging” (or remembering) x ,
- strex to address x succeeds iff. there was no more recent store wrt. x , otherwise need to retry.

IPC-related synchronisation: semaphore (6)

ARMv7-A synchronisation primitives

Listing ([17, Example 1-6])

```
1 sem_post: ldrex    r1, [ r0 ] ; s' = MEM[ &s ]
2          add     r1, r1, #1   ; s' = s' + 1
3          strex   r2, r1, [ r0 ] ; r <= MEM[ &s ] = s'
4          cmp     r2, #0       ; r == 0
5          bne     sem_post     ; if r != 0, retry
6          dmb                      ; memory barrier
7          bx      lr           ; return
```

Listing ([17, Example 1-6])

```
1 sem_wait: ldrex    r1, [ r0 ] ; s' = MEM[ &s ]
2          cmp     r1, #0       ; s' == 0
3          beq     sem_wait     ; if s' == 0, retry
4          sub     r1, r1, #1   ; s' = s' - 1
5          strex   r2, r1, [ r0 ] ; r <= MEM[ &s ] = s'
6          cmp     r2, #0       ; r == 0
7          bne     sem_wait     ; if r != 0, retry
8          dmb                      ; memory barrier
9          bx      lr           ; return
```

IPC-related synchronisation: semaphore (7)

System V

► System V keyed semaphore API:

- descriptor captured via type `int`,
- related operations performed via

- `semget` [15, Page 1836]:

- allocate n -entry semaphore set if necessary

`flg` \ni `IPC_CREAT` \Rightarrow allocate

`flg` \ni `IPC_EXCL` \Rightarrow allocate or fail

- return descriptor.

- `semctl` [15, Page 1833]:

- control i -th semaphore in set, e.g.,

`cmd` = `IPC_RMID` \Rightarrow deallocate

`cmd` = `IPC_STAT` \Rightarrow get configuration

`cmd` = `SETVAL` \Rightarrow set value

`cmd` = `GETVAL` \Rightarrow get value

- `semop` [15, Page 1839]:

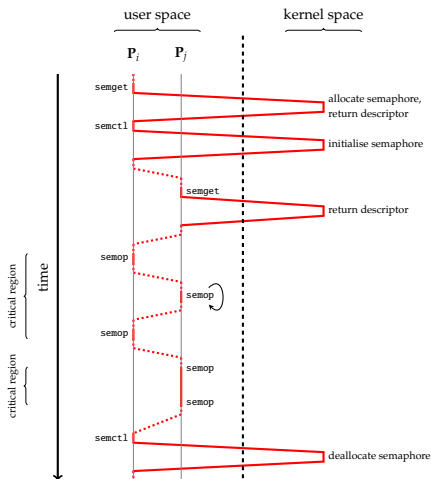
- update i -th semaphore in set, e.g.,

`op` < 0 \Rightarrow decrement

`op` = 0 \Rightarrow block

`op` > 0 \Rightarrow increment

- block if necessary.

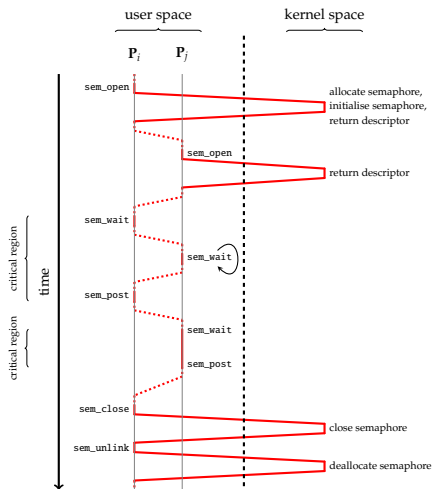


IPC-related synchronisation: semaphore (8)

POSIX

► POSIX named semaphore API:

- descriptor captured via type `sem_t`,
- related operations performed via
 - `sem_open` [15, Page 1820]:
 - allocate semaphore if necessary
 - $\text{flg} \ni \text{O_CREAT} \Rightarrow \text{allocate}$
 - $\text{flg} \ni \text{O_EXCL} \Rightarrow \text{allocate or fail}$
 - initialise semaphore value if necessary,
 - return descriptor.
 - `sem_wait` [15, Page 1832]:
 - decrement semaphore value,
 - block if necessary.
 - `sem_post` [15, Page 1823]:
 - increment semaphore value.
 - `sem_close` [15, Page 1812]:
 - close semaphore (i.e., stop using it).
 - `sem_unlink` [15, Page 1830]:
 - unlink semaphore (i.e., deallocate it).

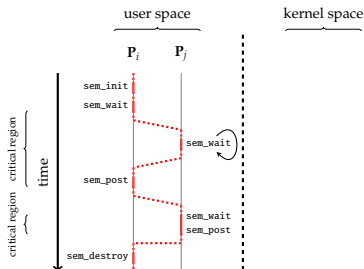


IPC-related synchronisation: semaphore (8)

POSIX

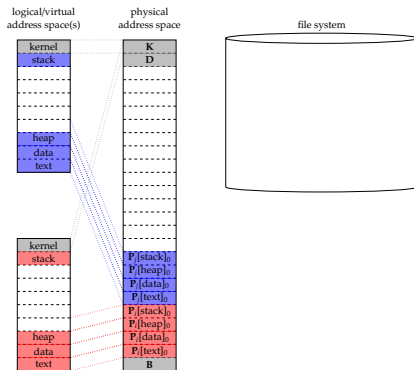
▶ POSIX unnamed semaphore API:

- ▶ descriptor captured via type `sem_t`,
- ▶ related operations performed via
 - ▶ `sem_init` [15, Page 1818]:
 - initialise semaphore value
 - $\text{shared} = 0 \Rightarrow$ shared between threads
 - $\text{shared} \neq 0 \Rightarrow$ shared between processes
 - ▶ `sem_wait` [15, Page 1832]:
 - decrement semaphore value,
 - block if necessary.
 - ▶ `sem_post` [15, Page 1823]:
 - increment semaphore value.
 - ▶ `sem_destroy` [15, Page 1814]:
 - destroy semaphore (i.e., stop using it).



IPC-related communication: shared memory/mapping (1)

► Idea:

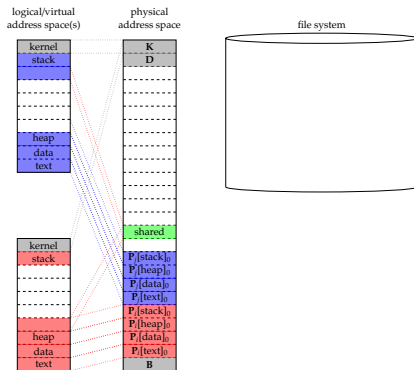


st.

- synchronisation needs to be explicit
- ± communication is unstructured (i.e., byte-oriented)
- + communication is (relatively) efficient
- + communication is bi-directional
- + supports n -to- m communication

IPC-related communication: shared memory/mapping (1)

► Idea:

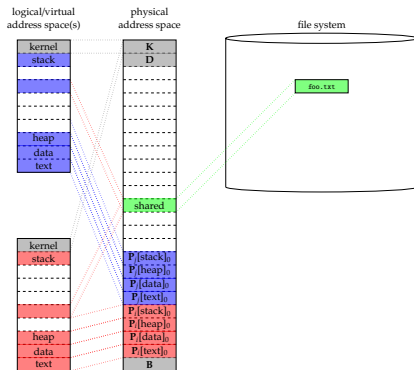


st.

- synchronisation needs to be explicit
- ± communication is unstructured (i.e., byte-oriented)
- + communication is (relatively) efficient
- + communication is bi-directional
- + supports n -to- m communication

IPC-related communication: shared memory/mapping (1)

► Idea:



st.

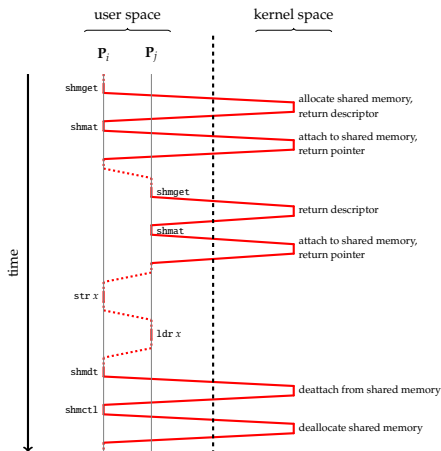
- synchronisation needs to be explicit
- ± communication is unstructured (i.e., byte-oriented)
- + communication is (relatively) efficient
- + communication is bi-directional
- + supports n -to- m communication

IPC-related communication: shared memory/mapping (2)

System V

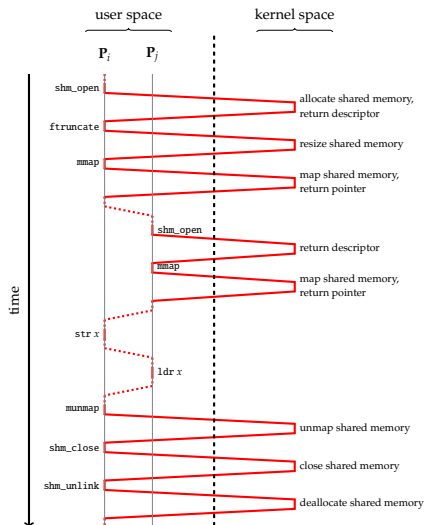
► System V keyed shared memory API:

- descriptor captured via type `int`,
- related operations performed via
 - `shmget` [15, Page 1911]:
 - allocate n -byte shared memory region if necessary
 - `flg ∋ IPC_CREAT` \Rightarrow allocate
 - `flg ∋ IPC_EXCL` \Rightarrow allocate or fail
 - return descriptor.
 - `shmat` [15, Page 1905]:
 - attach to shared memory region
 - `flg ∋ SHM_EXEC` \Rightarrow execute permission
 - `flg ∋ SHM_RDONLY` \Rightarrow read permission
 - return pointer.
 - `shmdt` [15, Page 1909]:
 - detach from shared memory region.
 - `shmctl` [15, Page 1907]:
 - control shared memory region
 - `cmd = IPC_RMID` \Rightarrow deallocate
 - `cmd = IPC_STAT` \Rightarrow get configuration



► POSIX named shared memory API:

- descriptor captured via type `int`,
- related operations performed via
 - `shm_open` [15, Page 1898]:
 - allocate n -byte shared memory region if necessary
 - `flag & O_CREAT` \Rightarrow allocate
 - `flag & O_EXCL` \Rightarrow allocate or fail
 - `flag & O_RDWR` \Rightarrow read/write permission
 - `flag & O_RDONLY` \Rightarrow read permission
 - return descriptor.
 - `mmap` [15, Page 1309]:
 - map shared memory region into virtual address space.
 - return pointer.
 - `munmap` [15, Page 1357]:
 - unmap shared memory region from virtual address space.
 - `shm_close`:
 - close shared memory region (i.e., stop using it).
 - `shm_unlink` [15, Page 1903]:
 - unlink shared memory region (i.e., deallocate it).



IPC-related communication: shared memory/mapping (4)

Linux

Linux shared mapping API:

▸ `mmap` [15, Page 1309]:

- map resource into virtual address space

`flg` \ni `MAP_SHARED` \Rightarrow shared mapping
`flg` \ni `MAP_PRIVATE` \Rightarrow unshared mapping
`flg` \ni `MAP_ANONYMOUS` \Rightarrow memory mapping

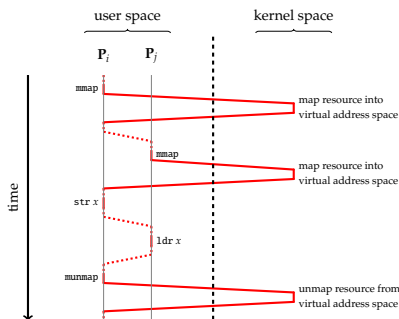
- initialise access permissions

`prot` \ni `PROT_EXEC` \Rightarrow execute permission
`prot` \ni `PROT_READ` \Rightarrow read permission
`prot` \ni `PROT_WRITE` \Rightarrow write permission

- return pointer.

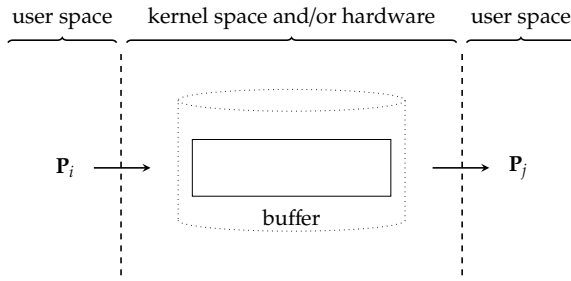
▸ `munmap` [15, Page 1357]:

- unmap resource from virtual address space.



IPC-related communication: pipe (1)

► Idea:



st.

- + synchronisation is implicit
- ± communication is unstructured (i.e., byte-oriented)
- communication is (relatively) inefficient (i.e., vs. shared memory)
- communication is uni-directional
- supports 1-to-1 communication

► POSIX named pipe API:

- descriptor captured via type `int`,
- related operations performed via

- `mkfifo` [15, Page 1295]:

- allocate pipe,
- initialise access permissions.

- `open` [15, Page 1379]:

- open pipe,
- return descriptor.

- `write` [15, Page 1263]:

- write data to pipe,
- block if necessary.

- `read` [15, Page 1737]:

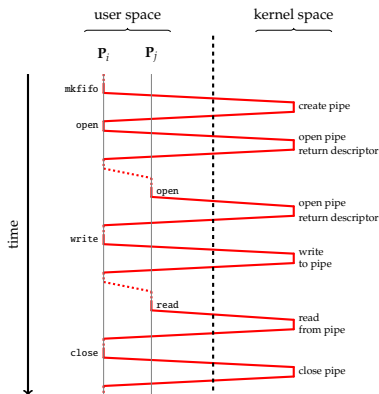
- read data from pipe,
- block if necessary.

- `close` [15, Page 676]:

- close pipe (i.e., stop using it).

- `unlink` [15, Page 2154]:

- unlink pipe (i.e., deallocate it).

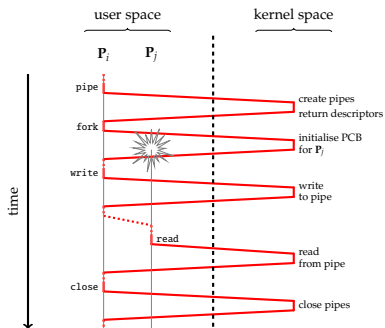


IPC-related communication: pipe (2)

POSIX

► POSIX unnamed pipe API:

- descriptor captured via type `int`,
- related operations performed via
 - `pipe` [15, Page 1400]:
 - allocate pipes,
 - return descriptors.
 - `write` [15, Page 2263]:
 - write data to pipe,
 - block if necessary.
 - `read` [15, Page 1737]:
 - read data from pipe,
 - block if necessary.
 - `close` [15, Page 676]:
 - close pipe (i.e., stop using it).



Conclusions

► Take away points:

- IPC is important, representing a core service delivered by kernel ...
- ... *good* IPC is tricky, wrt.

1. interface

- large design space,
- (ideally) needs to be flexible, uniform, etc.
- can unify other abstractions (e.g., files, sockets),
- should promote correctness,

2. implementation

- large design space,
- needs to be efficient: pure overhead wrt. concurrent computation,
- can share other mechanisms (e.g., files, sockets),
- should enforce correctness,

meaning *multiple*, complementary variants are the norm.

Conclusions

► Take away points:

- A study of existing standards highlights design philosophy, e.g.,

System V \mapsto $\begin{cases} \text{Xget} & \Rightarrow \text{allocate resource and descriptor} \\ \text{Xctl} & \Rightarrow \text{configure resource, plus deallocate descriptor and resource} \\ \text{Xop} & \Rightarrow \text{general-purpose operation} \end{cases}$

POSIX \mapsto $\begin{cases} \text{X_open} & \Rightarrow \text{allocate resource and descriptor, plus configure resource} \\ \text{X_Y} & \Rightarrow \text{special-purpose operation} \\ \text{X_close} & \Rightarrow \text{deallocate descriptor} \\ \text{X_unlink} & \Rightarrow \text{deallocate resource} \end{cases}$

Additional Reading

- ▶ A.B. Downey. *The Little Book of Semaphores*. URL: <http://greenteapress.com/wp/semaphores>.
- ▶ M. Kerrisk. "Chapter 47: System V semaphores". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010.
- ▶ M. Kerrisk. "Chapter 53: POSIX semaphores". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010.
- ▶ M. Kerrisk. "Chapter 20: Signals: fundamental concepts". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010.
- ▶ M. Kerrisk. "Chapter 21: Signals: signal handlers". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010.
- ▶ M. Kerrisk. "Chapter 49: Memory mappings". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010.
- ▶ M. Kerrisk. "Chapter 48: System V shared memory". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010.
- ▶ M. Kerrisk. "Chapter 54: POSIX shared memory". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010.
- ▶ M. Kerrisk. "Chapter 46: System V message queues". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010.
- ▶ M. Kerrisk. "Chapter 52: POSIX message queues". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010.
- ▶ M. Kerrisk. "Chapter 44: Pipes and FIFOs". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010.
- ▶ M. Kerrisk. "Chapter 57: Sockets: UNIX domain". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010.
- ▶ R. Love. "Chapter 4: Advanced file I/O". In: *Linux System Programming*. 2nd ed. O'Reilly, 2013.
- ▶ R. Love. "Chapter 10: Signals". In: *Linux System Programming*. 2nd ed. O'Reilly, 2013.

References

- [1] A.B. Downey. *The Little Book of Semaphores*. URL: <http://greenteapress.com/wp/semaphores> (see p. 22).
- [2] M. Kerrisk. "Chapter 20: Signals: fundamental concepts". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010 (see p. 22).
- [3] M. Kerrisk. "Chapter 21: Signals: signal handlers". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010 (see p. 22).
- [4] M. Kerrisk. "Chapter 44: Pipes and FIFOs". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010 (see p. 22).
- [5] M. Kerrisk. "Chapter 46: System V message queues". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010 (see p. 22).
- [6] M. Kerrisk. "Chapter 47: System V semaphores". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010 (see p. 22).
- [7] M. Kerrisk. "Chapter 48: System V shared memory". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010 (see p. 22).
- [8] M. Kerrisk. "Chapter 49: Memory mappings". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010 (see p. 22).
- [9] M. Kerrisk. "Chapter 52: POSIX message queues". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010 (see p. 22).
- [10] M. Kerrisk. "Chapter 53: POSIX semaphores". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010 (see p. 22).
- [11] M. Kerrisk. "Chapter 54: POSIX shared memory". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010 (see p. 22).
- [12] M. Kerrisk. "Chapter 57: Sockets: UNIX domain". In: *The Linux Programming Interface*. 6th ed. No Starch Press, 2010 (see p. 22).
- [13] R. Love. "Chapter 10: Signals". In: *Linux System Programming*. 2nd ed. O'Reilly, 2013 (see p. 22).
- [14] R. Love. "Chapter 4: Advanced file I/O". In: *Linux System Programming*. 2nd ed. O'Reilly, 2013 (see p. 22).
- [15] *Standard for Information Technology - Portable Operating System Interface (POSIX)*. Institute of Electrical and Electronics Engineers (IEEE) 1003.1-2008. 2008. URL: <http://standards.ieee.org> (see pp. 8–10, 14–16, 18, 19).

References

- [16] *ARM Architecture Reference Manual: ARMv7-A and ARMv7-R edition*. Tech. rep. DDI-0406C. ARM Ltd., 2014. URL: <http://infocenter.arm.com/help/topic/com.arm.doc.ddi0406c/index.html>.
- [17] *ARM Synchronization Primitives*. Tech. rep. DHT-0008A. ARM Ltd., 2009. URL: <http://infocenter.arm.com/help/topic/com.arm.doc.dht0008a/index.html> (see pp. 5, 7).
- [18] E.W. Dijkstra. *Over de sequentialiteit van procesbeschrijvingen*. Tech. rep. EWD-35. 1962 (approx.) URL: <http://www.cs.utexas.edu/users/EWD> (see pp. 3, 4).