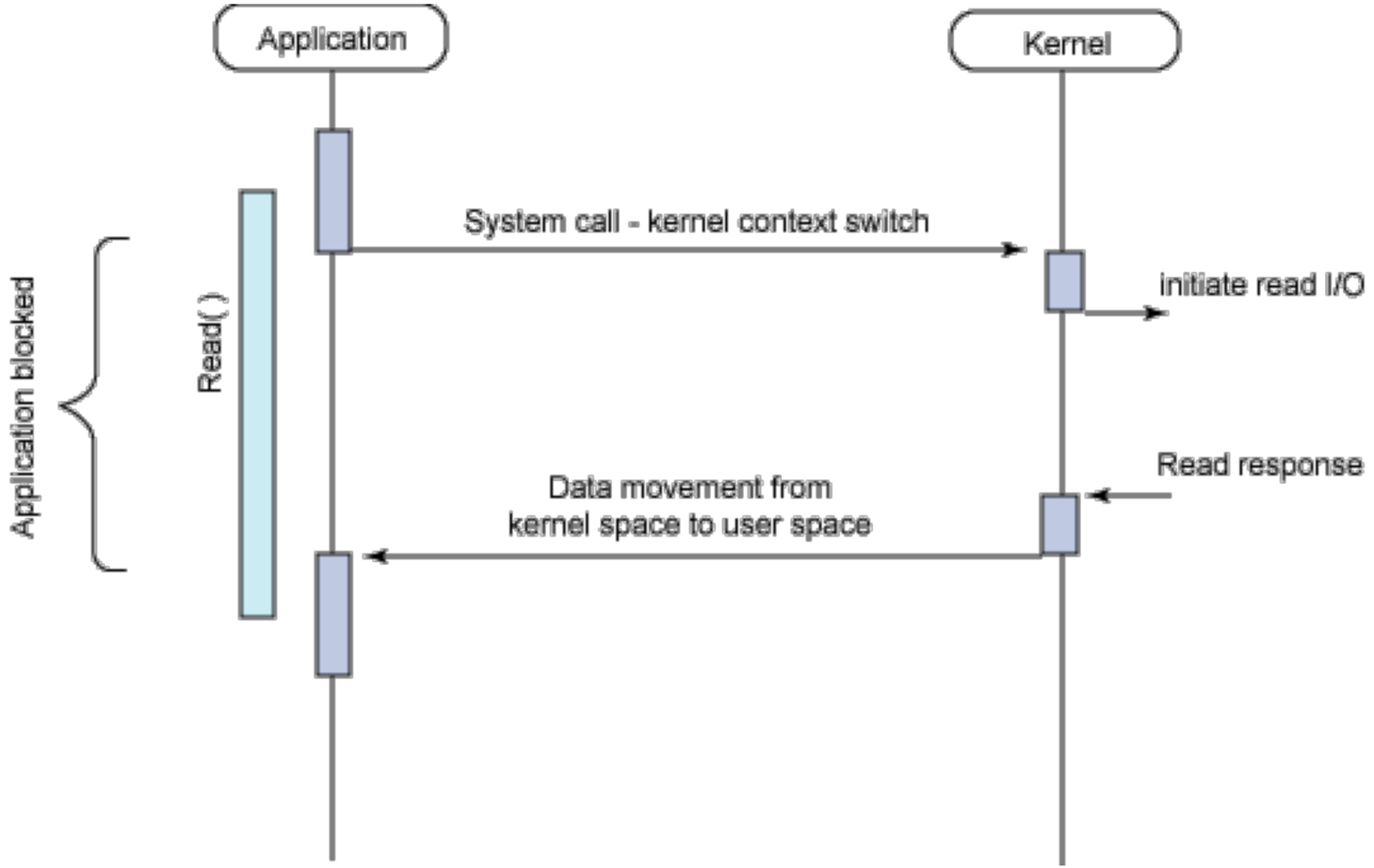
Advanced Programming in the UNIX Environment

Week 12, Segment 4:
Asynchronous and Memory Mapped I/O

Department of Computer Science
Stevens Institute of Technology

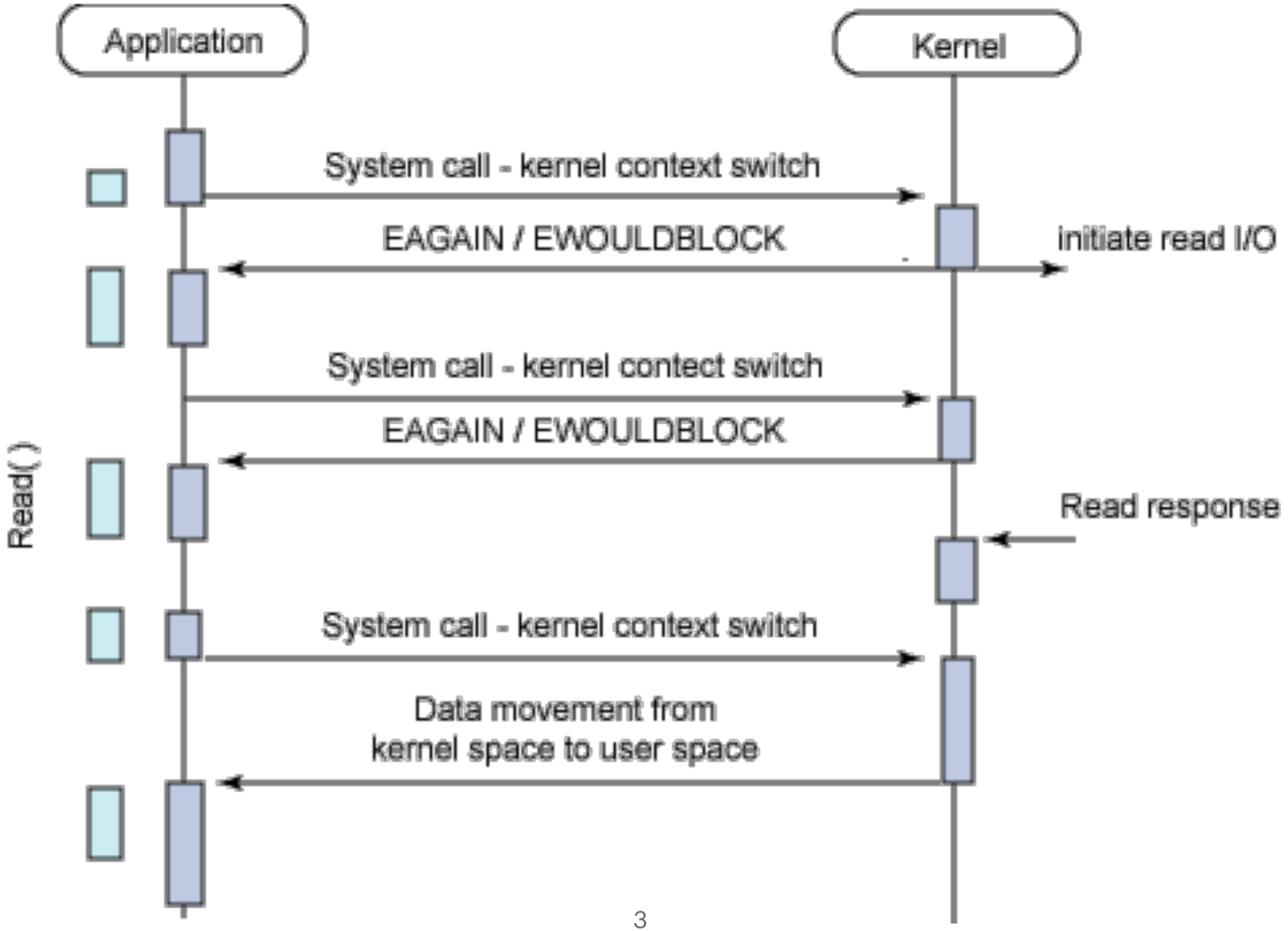
Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/631/

Synchronous, blocking I/O



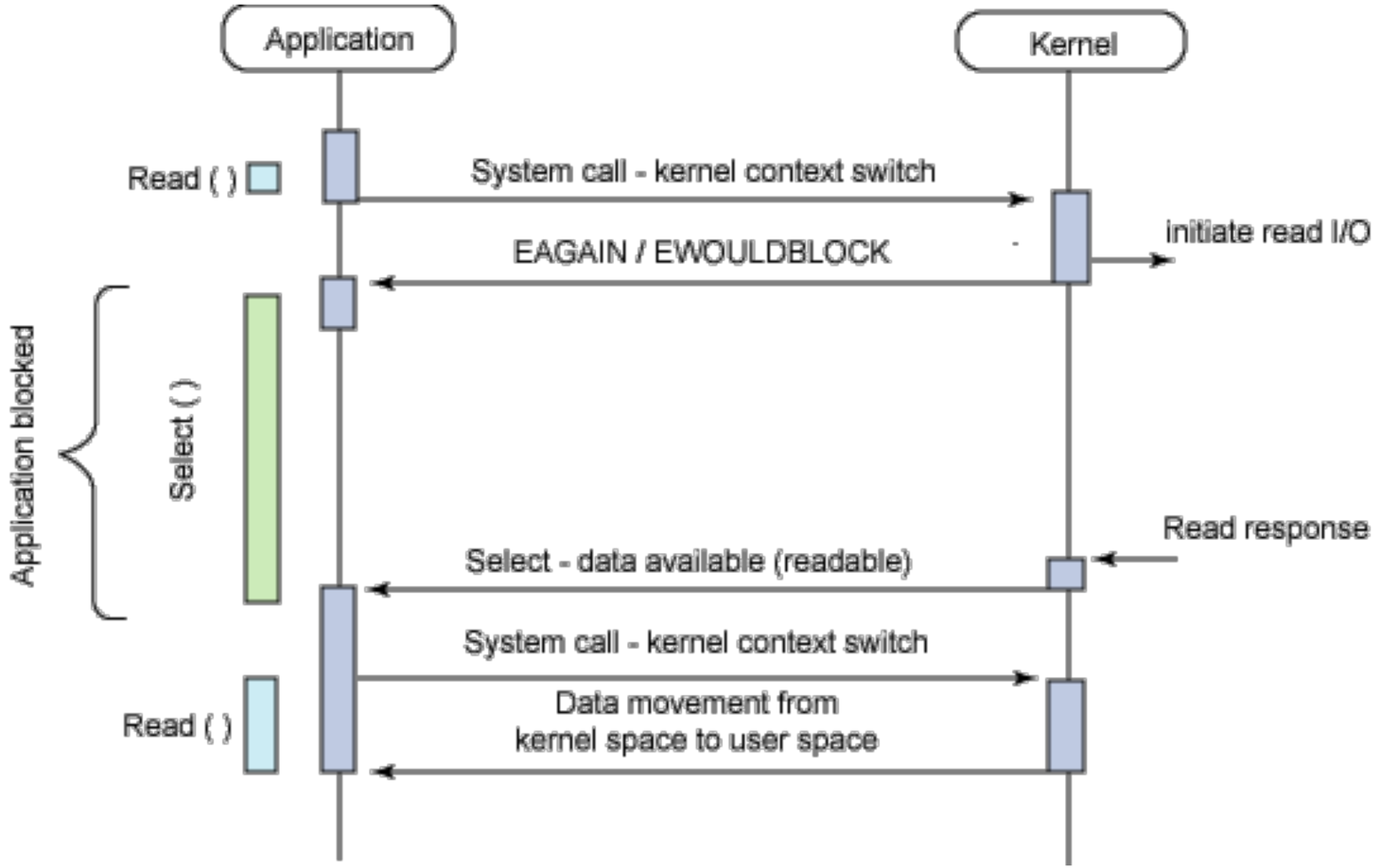
https://developer.ibm.com/technologies/linux/articles/l-async/

Asynchronous I/O



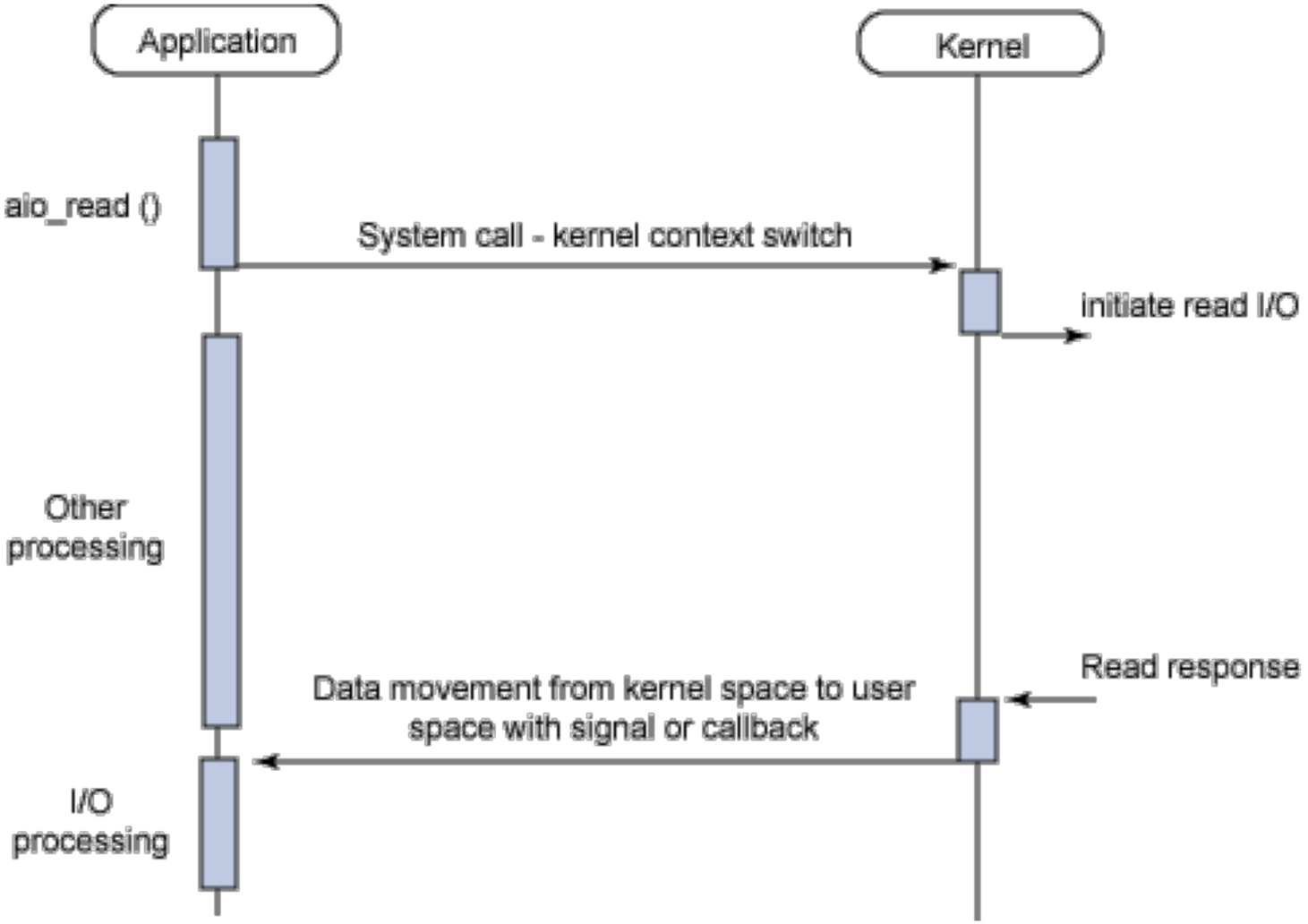
https://developer.ibm.com/technologies/linux/articles/l-async/

Asynchronous I/O



https://developer.ibm.com/technologies/linux/articles/l-async/

Asynchronous I/O



https://developer.ibm.com/technologies/linux/articles/l-async/

Asynchronous I/O

	Blocking	**Non-blocking**
Synchronous	`read(2)/write(2)`	`read(2)/write(2)` `O_NONBLOCK`
Asynchronous	I/O multiplexing (select(2)/poll(2))	AIO

https://developer.ibm.com/technologies/linux/articles/l-async/

Asynchronous I/O

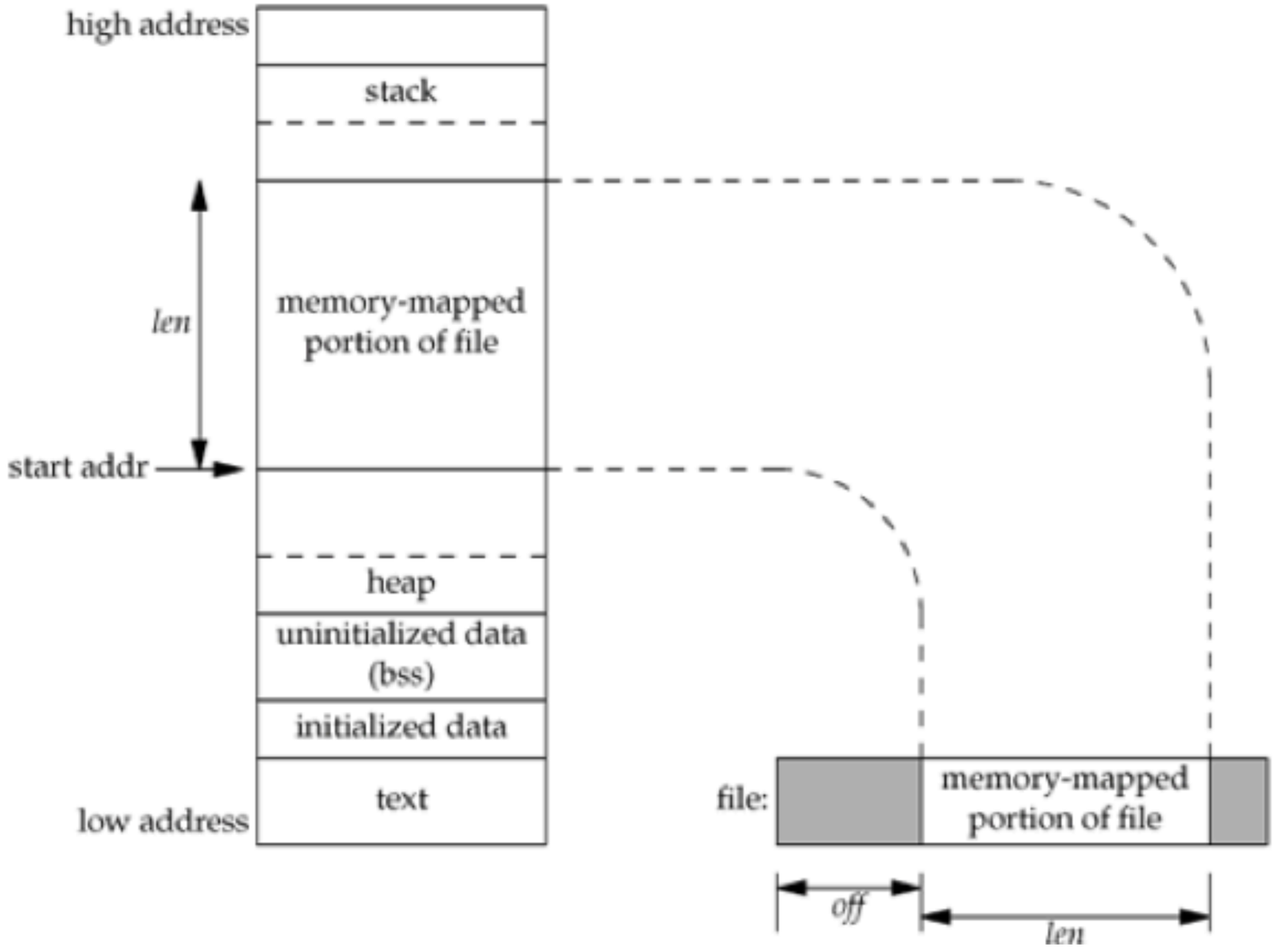
- semi-async I/O via select(2)/poll(2)
- System V derived async I/O
  - limited to STREAMS
  - enabled via ioctl(2)
  - uses SIGPOLL
- BSD derived async I/O
  - limited to terminals and networks
  - enabled via open(2)/fcntl(2) (O_ASYNC, F_SETOWN)
  - uses SIGIO and SIGURG

POSIX AIO

- see aoi(7) on NetBSD
- kernel process manages queued I/O requests
- notification of calling process via signal or sigevent callback function
- calling process can still choose to block/wait

- Linux has multiple implementations:
  - glibc aio(7) - https://is.gd/YZ5fuj
  - libaio - https://pagure.io/libaio

Memory Mapped I/O



mmap(2)

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

void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset);
```

Returns: pointer to mapped region on success, MAP_FAILED on error

- protection specified for a region:
  - PROT_READ – region can be read
  - PROT_WRITE – region can be written
  - PROT_EXEC – region can be executed
  - PROT_NONE – region can not be accessed
- flag needs to be one of MAP_SHARED or MAP_PRIVATE, which may be OR'd with other flags (see mmap(2) for details).

mmap(2)

Operation	Linux 2.4.22 (Intel x86)			Solaris 9 (SPARC)		
	User	System	Clock	User	System	Clock
`read/write`	0.04	1.02	39.76	0.18	9.70	41.66
`mmap/memcpy`	0.64	1.31	24.26	1.68	7.94	28.53

Exercises

- The Linux `aio(7)` manual page includes a code example - can you port this to NetBSD?
- Rewrite your HW1 cp to use `mmap(2)`/`memcpy(2)` instead of `read(2)`/`write(2)`.
- Benchmark your two implementations on different operating- and file systems.
- Review the NetBSD source code for `cp(1)` - why/when is `mmap(2)` used here? Why is it not used for all I/O?

  http://cvsweb.netbsd.org/bsdweb.cgi/src/bin/cp/utils.c?rev=HEAD