Syscalls: Adv. I/O 1: Overview

1. **Overview**
   - **basic vs. advanced I/O**
   - **advanced I/O syscalls**
2. Multiplexed I/O
3. Additional Uses for I/O Muliplexing
4. Nonblocking I/O
5. Signal-Driven I/O
6. Memory-Mapped I/O
7. Zero-Copy I/O
8. Scatter-Gather I/O
9. Asynchronous I/O (AIO)

Basic vs. Advanced I/O

By *"basic I/O"* we mean the standard I/O techniques shown in the **Syscalls: Files** lectures:

- files/sockets use defaults:
  - `open()` called with *no* `flags` options (e.g., `O_NONBLOCK`)
  - `fcntl()` *not* used to change file descriptor **status flags**
  - I/O operations are default **blocking mode**
- I/O done via basic `read()`/`write()` syscalls
- selection among multiple file descriptors handled by program code, not syscalls
- file **caching** handled by kernel, using default behavior

Basic vs. Advanced I/O

By *"advanced I/O"* we mean use of non-default settings and/or a variety of special I/O related syscalls:

- **multiplexed I/O** (`poll()` and `select()`)
- `epoll` API
- **nonblocking mode I/O** (`O_NONBLOCK`)
- kernel file buffer advice (for regular files only)
- **signal-driven I/O** (`O_ASYNC`)
- signals as file descriptors
- **memory mapping** files
- **zero-copy I/O**
- **scatter/gather I/O**
- **AIO** (**asynchronous I/O**): POSIX, kernel, io_uring

Basic vs. Advanced I/O (contd.)

In general, these "advanced I/O" techniques have one or more of the following goals:

- simplifying code required to handle multiple I/O connections
- simplifying code required to handle asynchronous and unpredictable I/O connections
- efficiently handling large numbers of I/O connections
- safely handling malicious/errorful I/O connections
- reducing data copying for syscalls

By *"I/O connections"* we mean open regular files, pipes/FIFOs, sockets, and devices.

Basic vs. Advanced I/O (contd.)

Due to the often implementation-specific nature of efficiency, many of the relevant system calls are *Linux specific*.

Other UNIX OS's often have comparable syscalls, but not always (and not necessarily same name).

Advanced I/O Syscalls

System calls relevant to advanced I/O:

- multiplexed I/O:
  - `int poll(struct pollfd *fds, nfds_t nfds, int timeout)`
  - `int ppoll(struct pollfd *fds, nfds_t nfds, const struct timespec *timeout_ts, const sigset_t *sigmask)`
  - `int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)`
  - `int pselect(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, const struct timespec *timeout, const sigset_t *sigmask)`
- `epoll` API:
  - `int epoll_create(int size)`
  - `int epoll_create1(int flags)`
  - `int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event)`
  - `int epoll_wait(int epfd, struct epoll_event *events, int maxevents, int timeout)`
  - `int epoll_pwait(int epfd, struct epoll_event *events, int maxevents, int timeout, const sigset_t *sigmask)`

Advanced I/O Syscalls (contd.)

System calls relevant to advanced I/O: (contd.)

- nonblocking I/O:
  - `int open(const char *pathname, int flags)`:
    as `open(pathname, flags|O_NONBLOCK)`
  - `int fcntl(int fd, int cmd, ... /* arg */ )`:
    as `fcntl(fd, F_SETFL, flags|O_NONBLOCK)`
- kernel file buffer advice:
  - `int posix_fadvise(int fd, off_t offset, off_t len, int advice)`
  - `ssize_t readahead(int fd, off64_t offset, size_t count)`

Advanced I/O Syscalls (contd.)

System calls relevant to advanced I/O: (contd.)

- signal-drven I/O:
  - `int open(const char *pathname, int flags)`:
    as `open(pathname, flags|O_ASYNC)`
  - `int fcntl(int fd, int cmd, ... /* arg */ )`:
    as `fcntl(fd, F_SETFL, flags|O_ASYNC)`
- signals as FDs:
  - `int signalfd(int fd, const sigset_t *mask, int flags)`
  - `int timerfd_create(int clockid, int flags)`
  - `int timerfd_settime(int fd, int flags, const struct itimerspec *new_value, struct itimerspec *old_value)`
  - `int timerfd_gettime(int fd, struct itimerspec *curr_value)`
  - `int eventfd(unsigned int initval, int flags)`

Advanced I/O Syscalls (contd.)

System calls relevant to advanced I/O: (contd.)

- memory mapping files:
  - `void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset)`
  - `int munmap(void *addr, size_t length)`
  - `int madvise(void *addr, size_t length, int advice)`
  - `int posix_madvise(void *addr, size_t len, int advice)`
- zero-copy I/O:
  - `ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count)`
  - `ssize_t splice(int fd_in, loff_t *off_in, int fd_out, loff_t *off_out, size_t len, unsigned int flags)`
  - `ssize_t tee(int fd_in, int fd_out, size_t len, unsigned int flags)`
  - `ssize_t vmsplice(int fd, const struct iovec *iov, unsigned long nr_segs, unsigned int flags)`
- scatter-gather I/O:
  - `ssize_t readv(int fd, const struct iovec *iov, int iovcnt)`
  - `ssize_t writev(int fd, const struct iovec *iov, int iovcnt)`
  - `ssize_t preadv(int fd, const struct iovec *iov, int iovcnt, off_t offset)`
  - `ssize_t pwritev(int fd, const struct iovec *iov, int iovcnt, off_t offset)`

Advanced I/O Syscalls (contd.)

System calls relevant to advanced I/O: (contd.)

- POSIX AIO:
  - `int aio_read(struct aiocb *aiocbp)`
  - `int aio_write(struct aiocb *aiocbp)`
  - `int lio_listio(int mode, struct aiocb *const aiocb_list[], int nitems, struct sigevent *sevp)`
  - `...aio_cancel(3), aio_error(3), aio_fsync(3), aio_return(3), aio_suspend(3)`
- Linux kernel AIO:
  - `int io_submit(aio_context_t ctx_id, long nr, struct iocb **iocbpp)`
  - `...io_cancel(2), io_destroy(2), io_getevents(2), io_setup(2)`
- io_uring AIO (2019):
  - `int io_uring_setup(int entries, struct io_uring_params *params)`
  - `int io_uring_enter(unsigned int fd, u32 to_submit, u32 min_complete, u32 flags)`
  - `int io_uring_register(unsigned int fd, unsigned int opcode, void *arg)`