

```
=====
```

## POSIX Asynchronous I/O Standard

Part of the real-time standard

```
#include <aio.h>

int aio_read(struct aiocb *aiocbp);
int aio_write(struct aiocb *aiocbp);
```

---

---

## Asynchronous I/O Control Block

```
struct aiocb {  
    int             aio_fildes;      // descriptor  
    off_t           aio_offset;     // offset  
    volatile void * aio_buf;       // buf location  
    size_t          aio_nbytes;     // length  
    int             aio_reqprio;    // priority offset  
    struct sigevent aio_sigevent;  // Signum and value  
    int             aio_lio_opcode; // Operation  
}
```

Valid opcodes are:

LIO\_READ, LIO\_WRITE, and LIO\_NOP

---

```
-----  
// issue several calls in one call  
int lio_listio(int mode,  
               struct aiocb *restrict const list[restrict],  
               int nent, struct sigevent *restrict sig);  
  
mode : LIO_WAIT      - wait for all calls to complete  
       LIO_NOWAIT    - don't wait for completion,  
                      notify with sig  
                      (no notification if sig=NULL)  
  
restrict - used to help compiler with aliasing issues  
-----
```

```
-----  
// wait until at least one aio in list completes,  
// signal interrupts function,  
// or timeout expires (if != NULL)  
int aio_suspend(const struct aiocb * const list[],  
               int nent,  
               const struct timespec *timeout);  
  
[NOTE: can't tell which ones have completed or number ]  
[      must poll/check using aio_error on each ]  
[      aiocb in list ]  
  
// get errno from completed aio call,  
// EINPROGRESS indicates call hasn't completed  
int aio_error(const struct aiocb *aiocbp);  
  
// get return value from completed aio call  
ssize_t aio_return(struct aiocb *aiocbp);  
-----
```

```
-----  
// cancel one or more outstanding requests on fildes  
// if aiocbp == NULL cancel all requests on fildes  
int aio_cancel(int fildes, struct aiocb *aiocbp);  
  
// asynchronously force all outstanding requests on  
// aiocbp->fildes to the completed state  
// i.e., call fsync asynchronously  
int aio_fsync(int op, struct aiocb *aiocbp);  
-----
```

```
-----  
int rc;  
struct aiocb my_aiocbs[MAX_IOCBS];  
struct aiocb *my_list[MAX_IOCBS];  
struct aiocb *ip;  
char my_bufs[MAX_IOCBS][MAX_BUF_SIZE];  
  
for (i=0; i<MAX_IOCBS; i++) {  
    my_list[i] = &my_aiocbs[i];  
}  
}
```

```
ip = my_list[0];
ip->aio_lio_opcode = LIO_READ;
ip->aio_filedes = fd1;
ip->aio_buf = &my_bufs[0][0];
ip->aio_nbytes = bytes;
ip->aio_offset = 0;
ip->aio_sigevent.sigev_notify = SIGEV_NONE;

ip = my_list[1];
ip->aio_lio_opcode = LIO_WRITE;
ip->aio_filedes = fd2;
ip->aio_buf = &my_bufs[1][0];
ip->aio_nbytes = bytes;
ip->aio_offset = 0;
ip->aio_sigevent.sigev_notify = SIGEV_NONE;
```

```
num_calls = 2
lio_listio(LIO_NOWAIT, my_list, num_calls, NULL);

while (num_calls) {
    aio_suspend(my_list, num_calls, NULL);

    for (i=0; i<num_calls; i++) {
        err = aio_error(&my_aiocb[i]);
        if (err != EINPROGRESS) {
            rc = aio_return(&my_aiocb[i]);
            my_list[i] = 0;
            num_calls--;
        }
    }
}
-----
```

---

Pros:

- o it's a standard
- o supported e.g., on HP-UX, Solaris,  
in the future on Linux?

Cons:

- o weak support for obtaining completion info
  - o may really need to use signals in order to get  
completion events properly (e.g., sigwaitinfo)
  - o missing things that should be asynchronous
    - aio\_accept, aio\_connect, aio\_close
    - aio\_open, aio\_fstat
-