==========
Linux Systems calls to support AIO
From linux-2.6.0-test3

```
// Create aio_context that can receive up to nr_events.
// Returns a new ctx_id through *ctxp
// Can’t submit requests if not enough room
// for completion
asmlinkage long sys_io_setup(
    unsigned nr_events,
    aio_context_t *ctxp);

// Submit a list/array of requests
// Current support for READ/WRITE/FSYNC
// IOCB_CMD_NOOP may not work
asmlinkage long sys_io_submit(
    aio_context_t ctx_id,
    long nr, struct iocb **iocbpp);
```

```
-------------------------------------------------
struct iocb {
  /* these are internal to the kernel/libc. */
  /* data to be returned in event's data */
  __u64   aio_data;
  /* the kernel sets aio_key to the req # */
  __u32   PADDED(aio_key, aio_reserved1);

  /* common fields */
  __u16   aio_lio_opcode;
  __s16   aio_reqprio;
  __u32   aio_fildes;
  __u64   aio_buf;
  __u64   aio_nbytes;
  __s64   aio_offset;

  /* extra parameters */
  /* TODO: use this for a (struct sigevent *) */
  __u64   aio_reserved2;
  __u64   aio_reserved3;
}; /* 64 bytes */
-------------------------------------------------
```

```
-----------------------------------------------
// Try to get up to nr but at least min_nr
// from completion queue specified by ctx_id
// Array of events should be long enough to hold
// nr events
// If don’t get at least min_nr and timeout != NULL
// wait until we get at least min_nr or until timeout
asmlinkage long sys_io_getevents(
    aio_context_t ctx_id,
    long min_nr,
    long nr,
    struct io_event *events,
    struct timespec *timeout);

struct io_event {
  __u64 data; /* the data field from the iocb */
  __u64 obj;  /* what iocb this event came from */
  __s64 res;  /* result code for this event */
  __s64 res2; /* secondary result */
};
-----------------------------------------------
```

```
-------------------------------------------------
asmlinkage long sys_io_destroy(
   aio_context_t ctx);

asmlinkage long sys_io_cancel(
    aio_context_t ctx_id,
    struct iocb *iocb,
    struct io_event *result);
-------------------------------------------------
```

```
-------------------------------------------------
int rc;
aio_context_t ctx_id;
struct iocb my_iocbs[MAX_IOCBS];
struct iocb *my_iocb_ptrs[MAX_IOCBS];
struct iocb *ip;
char my_bufs[MAX_IOCBS][MAX_BUF_SIZE];
struct io_event my_events[MAX_IOCBS];

sys_io_setup(MAX_EVENTS, &ctx_id);

for (i=0; i<MAX_IOCBS; i++) {
  my_iocb_ptrs[i] = &my_iocbs[i];
}
```

```
ip = my_iocb_ptrs[0];
ip->lio_opcode = IOCB_CMD_PREAD;
ip->aio_filedes = fd1;
ip->aio_buf = &my_bufs[0][0];
ip->aio_nbytes = bytes;
ip->aio_offset = 0;
// identifier used with completion event
ip->aio_data = my_iocb_ptrs[0];

ip = my_iocb_ptrs[1];
ip->lio_opcode = IOCB_CMD_PWRITE;
ip->aio_filedes = fd2;
ip->aio_buf = &my_bufs[1][0];
ip->aio_nbytes = bytes;
ip->aio_offset = 0;
// identifier used with completion event
ip->aio_data = my_iocb_ptrs[1];
```

```
sys_io_submit(ctx_id, 2, my_iocb_ptrs);

/* wait forever for 2 completion events */
n = sys_io_getevents(ctx_id, 2, 2, my_events, 0);

for (i=0; i<n; i++) {
  // get the pointer to iocb used for request
  // this may be the same as my_events[i].obj
  ip = my_events[i].data;
  // return value from the call
  rc = my_events[i].res;
}
```