```
==========================================
// System Calls Used by Servers
// [event based-server (non-blocking)]

// backlog = max outstanding conn requests
rc = listen(server_sd, backlog);

n = select(max_fd, &rdset, &wrset, &exset,
           &timeout);

sd = accept(server_sd, &addr, &addr_len);

// Disable Nagle algorithm so we don't delay
// when sending small replies */
value = 1;
setsockopt(sd, SOL_TCP, TCP_NODELAY, &value,
           sizeof (value));

rc = read(sd, &buf, bytes);
rc = write(sd, &buf, bytes);
rc = close(sd);

------------------------------------------
```

```
-----------------------------------------------
Getting Events

rdset = rset; wrset = wset;

n = select(max_fd, &rdset, &wrset, &exset,
           &timeout);
-----------------------------------------------
```

```
-----------------------------------------------
// returns number of ready descriptors
n = poll(poll_array, poll_array_len, &timeout);

struct pollfd {
       int fd;           /* file descriptor */
       short events;     /* requested events */
       short revents;    /* returned events */
};

#defines
POLLIN      0x0001    /* There is data to read */
POLLPRI     0x0002    /* There is urgent data to read */
POLLOUT     0x0004    /* Writing now will not block */
POLLERR     0x0008    /* Error condition */
POLLHUP     0x0010    /* Hung up */
POLLNVAL    0x0020    /* Invalid request: fd not open */
-----------------------------------------------
```

```
-----------------------------------------------
// A Scalable and Explicit Event Delivery
// Mechanism for UNIX [Banga, Mogul, Druschel]

EVENT_READ   0x1
EVENT_WRITE  0x2
EVENT_EXCEPT 0x4

// tell system which events are of interest,
// if statemask != NULL returns current state
// (to avoid race condition - state change occurs
//  after accept but before declare_interest call)
int declare_interest(fd, interestmask, &statemask);

int get_next_event(array_max, evt_array, &timeout);

typedef struct {
  int fd;
  unsigned mask;
} event_descr_t;

-----------------------------------------------
```

```
-------------------------------------------
epfd = epoll_create(max_fds);

ep_evt.events =  (short) 0;
ep_evt.events |= (short) EPOLLIN;
ep_evt.events |= (short) EPOLLOUT;
ep_evt.fd = sd;

epoll_ctl(epfd, EPOLL_CTL_ADD, sd, &ep_evt);


n = epoll_wait(epfd, epoll_array, epoll_array_len,
               &timeout);

struct epoll_event {
        __uint32_t events; /* Epoll events */
        epoll_data_t data; /* User data variable */
};

events are similar to POLL e.g.,
        EPOLLIN  = 0x001,
        EPOLLPRI = 0x002,
        EPOLLOUT = 0x004,
-------------------------------------------
```

```
-----------------------------------------------
Writing Results

  rc = write(sd, &header, header_len);
  rc = write(sd, &result, result_len);

-----------------------------------------------
```

```
-----------------------------------------------
or
  struct iovec {
   __ptr_t iov_base; /* Starting address.  */
   size_t iov_len;   /* Length in bytes.  */
  };

  struct iov[2];
  iov[0].iov_base = &header;
  iov[0].iov_len  = header_len;
  iov[1].iov_base = &result;
  iov[1].iov_len  = result_len;

  rc = writev(sd, &iov, 2);

-----------------------------------------------
```

```
-----------------------------------------------
or
  // HP-UX, Solaris

  struct hdr_trlr[2];
  // header
  hdr_trlr[0].iov_base = &header;
  hdr_trlr[0].iov_len  = header_len;
  // trailer (in this case nothing to send)
  hdr_trlr[1].iov_base = 0;
  hdr_trlr[1].iov_len  = 0;

  // returns number of bytes written
  rc = sendfile(sd, fd, offset,
                SEND_REMAINDER, &hdr_trlr, 0);
-----------------------------------------------
```

```
-----------------------------------------------
or
  // Linux

  rc = write(sd, &header, header_len);
  rc = sendfile(sd, fd, offset, ~0UL);

  // Q: Problems?
-----------------------------------------------
```

```
-----------------------------------------------
or
  // Linux

  value = 1;
  rc = setsockopt(sd, SOL_TCP, TCP_CORK,
                  &value, sizeof(value));

  rc = write(sd, &header, header_len);

  rc = sendfile(sd, fd, offset, ~0UL);

  value = 0;
  rc = setsockopt(sd, SOL_TCP, TCP_CORK,
                  &value, sizeof(value));

===============================================
```