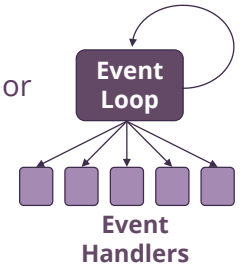
Event-Driven Asynchronous Network Programming Part 2

MCIT 595

Select() as an Event Loop

- Select loop often implemented as event loop
- Callback functions (by asynchronous I/O) are registered with event loop
- `Select()`
  - Wait on multiple file descriptors/sockets and timeout
  - Application does not consume CPU cycles while waiting
  - Return when file descriptors/sockets are ready to be read or written or they have an error, or timeout exceeded
- When file descriptor returns:
  - Examine the return parameters of the select call
  - Finds out which file descriptor has changed
  - Retrieve the handler and execute the appropriate code (callback)



I/O Multiplexing: Select ()

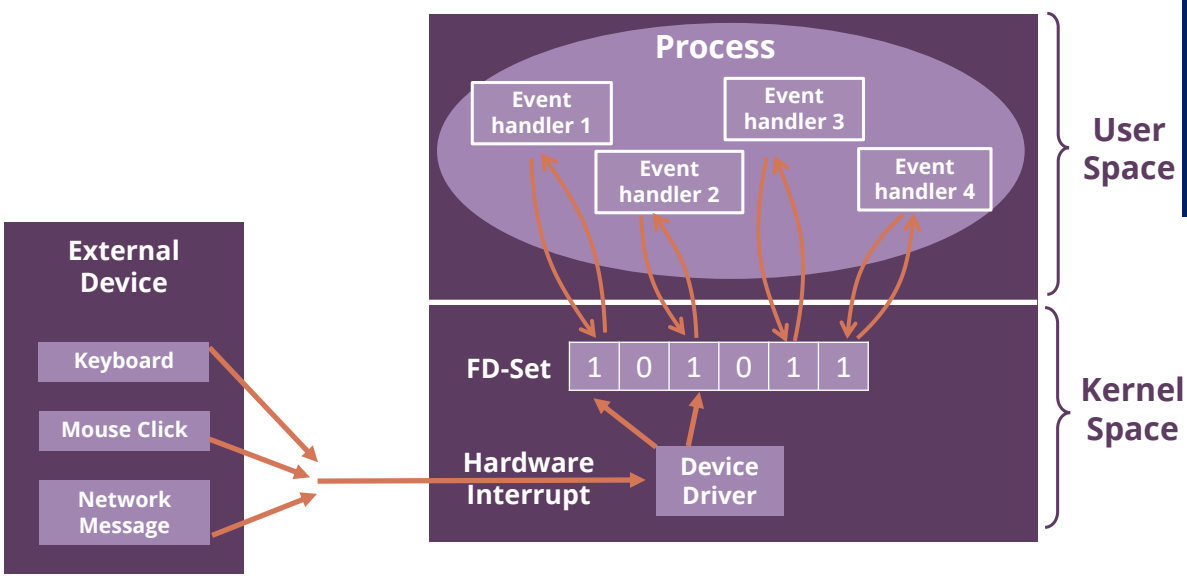
```
fd_set read_set;
struct timeval time_out;
while (1) {
  FD_ZERO (read_set);
  FD_SET (stdin, read_set); /* stdin is typically 0 */
  FD_SET (sock, read_set);
  time_out.tv_usec = 100000; time_out.tv_sec = 0;
  select_retval = select(MAX(stdin, sock) + 1, &read_set, NULL, NULL, &time_out);
  if (select_retval < 0) {
      perror ("select");
      abort ();
  }
  if (select_retval > 0) {
      if (FD_ISSET(sock, read_set)) {
          if (receive_packets(buffer, buffer_len, &bytes_read) != 0) {
              break;
          }
      if (FD_ISSET(stdin, read_set)) {
          if (read_user(user_buffer, user_buffer_len,
                        &user_bytes_read) != 0) {
              break;
          }
      }
  }
}
```

Set up select Parameters

Run select()

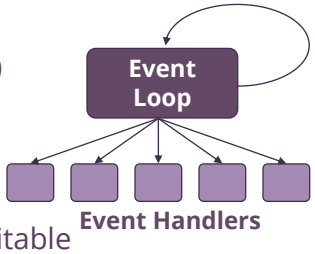
Interpret Result

Select Architecture



Shortcomings of Events

- **Long-running handlers** make application non-responsive
  - Fork off subprocesses for long-running things (e.g. multimedia), use events to find out when done
  - Break up handlers (e.g. event-driven I/O)
  - Periodically call event loop in handler (adds complexity)
- More work to maintain **local state** across events
- Threads provide true **CPU concurrency**:
  - Scalable performance on multiple CPUs
  - Harder to achieve CPU concurrency with events (not suitable for scientific apps)



Advantages and Common Usage

- Advantages:
  - Ease of programming (arguable)
  - Performance (no context switching, locking overhead)
  - Debugging is easier with events: Timing dependencies only related to events, not to internal scheduling
- Network protocols:
  - One handler for each source of input (socket, etc.)
  - As input changes, call appropriate event handler
- GUIs:
  - One handler for each event (press button, invoke menu entry, etc.)
  - Handler implements behavior (undo, delete file, etc.)