TCP server example

```
void doaccept (int lfd) {
  sockaddr_in sin;
  bzero (&sin, sizeof (sin));
  socklen_t sinlen = sizeof (sin);
  int nfd = accept (lfd, (sockaddr *) &sin, &sinlen);
  if (nfd >= 0) { /* ... */ }
}
int main (int argc, char **argv) {
  // ...
  int lfd = inetsocket (SOCK_STREAM, your_port, INADDR_ANY);
  if (lfd < 0) fatal << "socket: " << strerror (errno) << "\n";
  if (listen (lfd, 5) < 0) fatal ("listen: %m\n");
  fdcb (lfd, selread, wrap (doaccept, lfd));
  amain ();
}
```

Simple list insert

```
struct element {
  int data;
  struct element *next;
};

void insert (element **list, int data) {
  element *e = New element;
  e->data = data;
  e->next = *list;
  *list = e;
}
```

First attempt at synchronization

```
int insert_lock;

void insert (element **list, int data) {
  while (insert_lock) /* Acquire lock */
    ;
  insert_lock = 1;

  element *e = New element;
  e->data = data;
  e->next = *list;
  *list = e;

  insert_lock = 0; /* Release lock */
}
```

Need atomic read-write operation!

- **Example:** `int test_and_set (int *lockp);`
  - Sets `*lockp = 1` and returns old value
- **Now can implement spinlocks:**

```
#define lock(lockp) while (test_and_set (*lockp))
#define unlock(lockp) *lockp = 0
```

Synchronization on i386

- `xchg` **instruction, exchanges reg with mem**

```
_test_and_set:
        movl    8(%esp), %edx
        movl    $1,% eax
        xchg    %eax,( %edx)
        ret
```

- **CPU locks memory system around read and write**
  - Prevents other uses of the bus (e.g., DMA)
- **Operates at memory bus speed, not CPU speed**
  - Must slower than cached read/buffered write

Synchronization on alpha

- `ldl_l` – **load locked**
  `stl_c` – **store conditional**

```
_test_and_set:
        ldq_l   v0, 0(a0)
        bne     v0, 1f
        addq    zero, 1, v0
        stq_c   v0, 0(a0)
        beq     v0, _test_and_set
        mb
        addq    zero, zero, v0
1:
        ret     zero, (ra), 1
```

New insert

```
int insert_lock;

void insert (element **list, int data) {
  lock (&insert_lock);  /* Acquire lock */

  element *e = New element;
  e->data = data;
  e->next = *list;
  *list = e;

  unlock (&insert_lock);  /* Release lock */
}
```

- **Are spinlocks enough?**

Need blocking mutexes, condition variables

- Can't do producer-consumer with just spinlocks
- Need to relinquish CPU until locks ready
- Must have cooperation of scheduler