Threads

mutex Functions

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_unlock(worker->lock);
pthread_mutex_lock(worker->lock);
typedef struct {
  float *total;
  int n;
  pthread_mutex_t *lock;
  pthread_t thread;
} Worker;
```

pthread inits

```
int child = pthread_create(&worker->thread, NULL, run_summation, worker
    );
pthread_join(workers[i].thread, NULL);
```

Semephores

Channel defined

```
typedef struct {
  sem_t read;
  sem_t write;
  char *message; // Value stored in the channel
} Chan;
```

semephore functions

```
sem_wait(&channel->read);
sem_post(&channel->write);
/* Second var is to enable other processes to see or not */
sem_init(&channel->read, 0, 0); // sets read to wait imediately
sem_init(&channel->write, 0, 1); // sets write to start imediately
```

Signals

```
pid_t child_pid = 0; // init pid
signal(SIGCHLD, &waitChild); /* when signal is called, function is
    executed */

/* Example of a handler function to be called with a signal */
void waitChild() {
    int child;
    wait(&child);
```

```
}
```

Pipes

```
#define PIPE_IN 1
#define PIPE_OUT 0 // if getting confused it is useful to define

int parent_to_child[2]; // pipes have two ends, [0], [1]
visit ./lab3/pipe.c for more information
```

Sockets

Read and write

```
write(sockfd, line, strlen(line));

numbytes = read(sockfd, buffer, MAXDATASIZE - 1);
```

Client init

```
struct addrinfo their_addrinfo; // server address info
struct addrinfo *their_addr = NULL; // connector's address information


// init socket
sockfd = socket(AF_INET, SOCK_STREAM, 0);

// allocate memory to info
memset(&their_addrinfo, 0, sizeof(struct addrinfo));

// set info
their_addrinfo.ai_family = AF_INET;
their_addrinfo.ai_socktype = SOCK_STREAM;
getaddrinfo(hostname, port, &their_addrinfo, &their_addr);


// connect to the socket
int rc = connect(sockfd, their_addr->ai_addr, their_addr->ai_addrlen);
if (rc == -1) perror("connect"); exit(1);
```

Server init

```
int s = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in sa;
sa.sin_family = AF_INET;
sa.sin_addr.s_addr = INADDR_ANY;
```

```
6  sa.sin_port = htons(port);
7
8  int rc = bind(s, (struct sockaddr *)&sa, sizeof(sa));
9  rc = listen(s, 5); /* listen for connections on a socket */
10
11 // This can be done in a function
12 struct sockaddr_in caller;
13 int rc = accept(s, (struct sockaddr *)&caller, (socklen_t *)sizeof(
       caller));
14
15 // for error checking
16 if (rc == -1) {
17   perror("this doesnt work");
18 }
```