Problem 1

What is TLS? Please describe its usage.

TLS (Thread Local Storage) is a computer programming method that uses static or global memory local to a thread. (from wiki)

TLS is useful for data that are **logically global** to the thread. Its usage includes random number generator, strtok(), and errno handling.

Problem 2

Refer to the I/O multiplexing based concurrent echo server in Figure 12.8, indicate the pros and cons of this event-driven programming approach based on I/O multiplexing.

Pros:

1. Event-driven designs give programmers more control over the behavior of their programs than process-based designs.
2. Easy to share data between flows.
3. Event-driven designs are often significantly more efficient than process-based designs because they do not require a process context switch to schedule a new flow.

Cons:

1. Significantly more complex in code than process-based or thread-based designs.
2. Hard to provide fine-grained concurrency.
3. Not able to take advantage of multi-core.

Problem 3

Modify Tiny so that when it serves static content, it copies the requested file to the connected descriptor using calloc, rio_readn, rio_writen, instead of mmap and rio_writen.

```
srcfd = Open(filename,O_RDONLY, 0);
srcp = (char*)calloc(filesize);
Rio_readn(srcfd, srcp, filesize);
Rio_writen(fd, srcp, filesize);
close(srcfd);
free(srcp);
```

Problem 4

Consider the code in Figure 12.5:

```
1   #include "csapp.h"
2   void echo(int connfd);
3
4   void sigchld_handler(int sig) {
5       while(waitpid(-1, 0, WNOHANG) > 0);
6       return;
7   }
8
9   int main(int argc, char **argv) {
10      int listenfd, connfd, port;
11      socklen_t clientlen = sizeof(struct sockaddr_in);
12      struct sockaddr_in clientaddr;
13
14      if(argc != 2) {
15          fprintf(stderr, "usage: %s <port>\n", argv[0]);
16          exit(0);
17      }
18      port = atoi(argv[1]);
19
20      Signal(SIGCHLD, sigchld_handler);
21      listenfd = Open_listenfd(port);
22      while(1) {
23          connfd = Accept(listenfd, (SA *) &clientaddr, &clientlen);
24          if (Fork() == 0) {
25              Close(listenfd);
26              echo(connfd);
27              Close(connfd);
28              exit(0);
29          }
30          Close(connfd);
31      }
32  }
```

1. After the parent closes the connected descriptor in line 30 of the conncurrent server, the child is still able to communicate with the client using its copy of the descriptor. Why?
2. Assume that 12306.cn has used this design to provide services, what will happen when

Chunyun begins?

1. When the parent forks the child, it gets a copy of the connected descriptor and the **reference count** for the associated file table is incremented from 1 to 2. When the parent closes its copy of the descriptor, the reference count is decremented from 2 to 1. Since the kernel will not close a file until the reference counter in its file table goes to 0, the child's end of the connection stays open.
2. Since there are too many requests, the child processes will be spawned repeatedly and the system ends up with excessive processes.