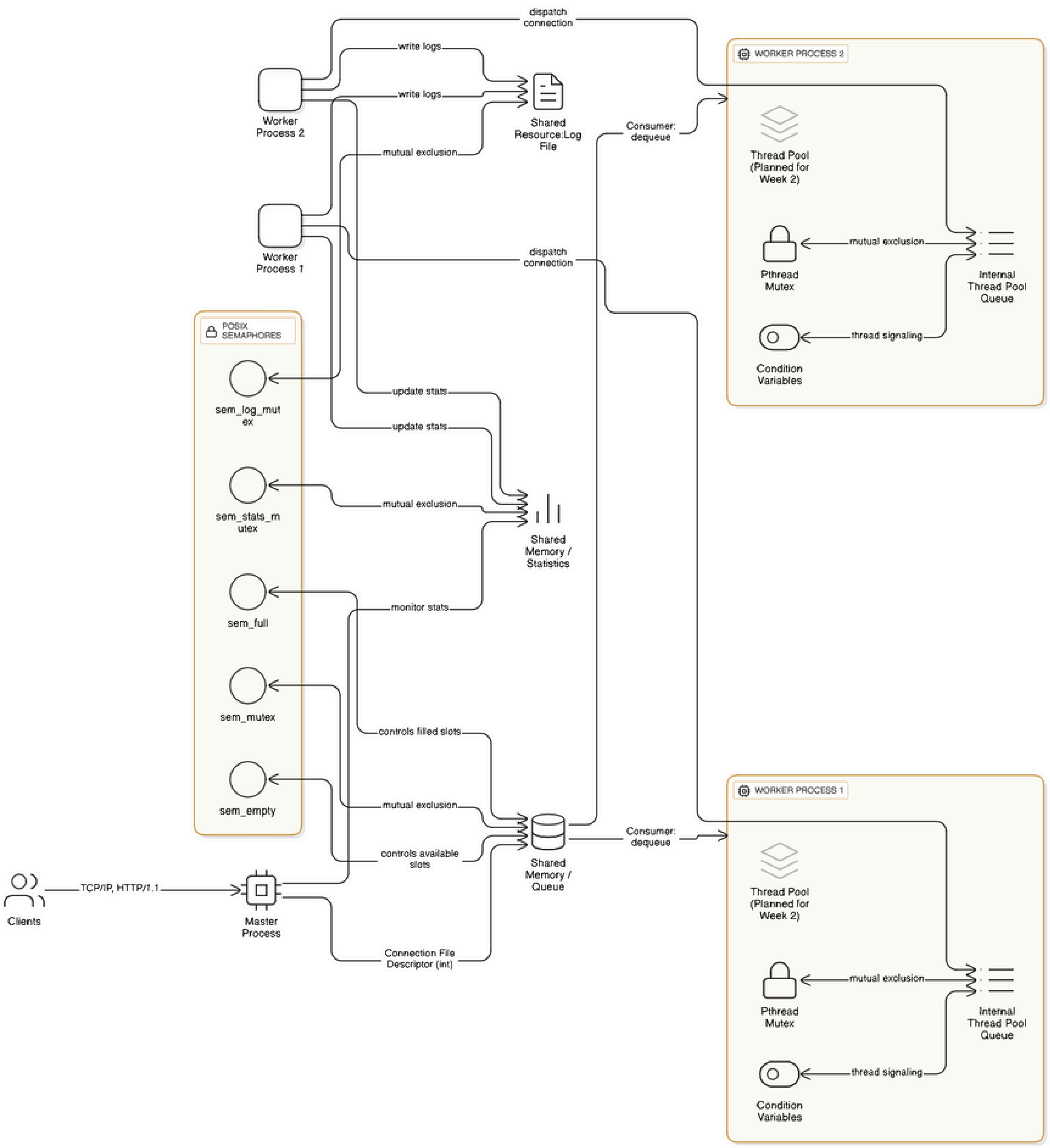
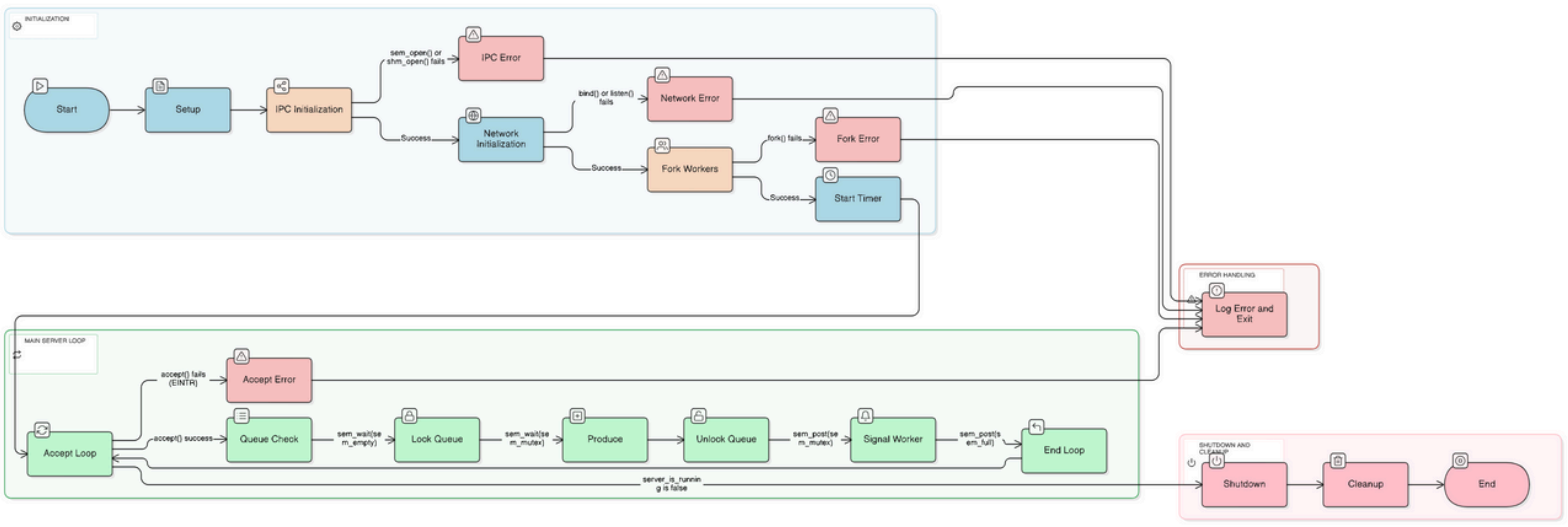
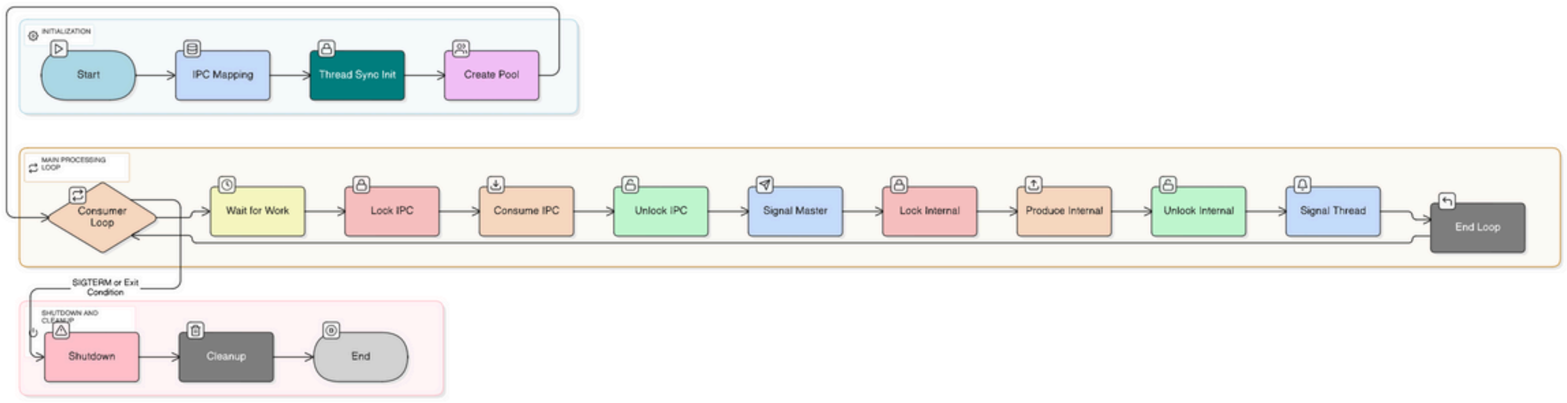
Architecture Diagram



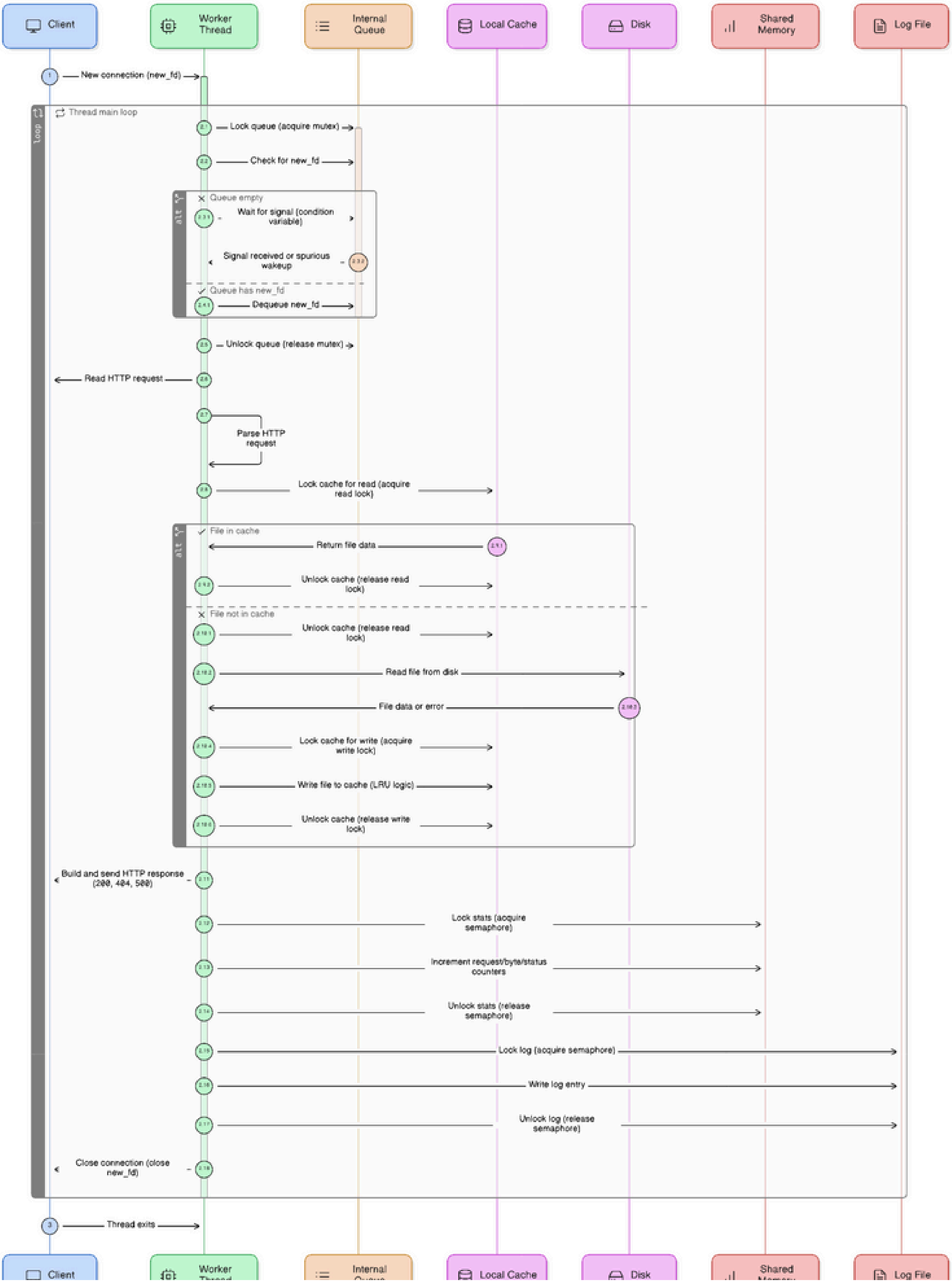
Master Process Flowchart



Worker Process Flowchart



Request-Handling Thread Flowchart



Client
Worker Thread
Internal Queue
Local Cache
Disk
Shared Memory
Log File
New connection (new_fd)
Loop
Thread main loop
Lock queue (acquire mutex)
Check for new_fd
alt
Queue empty
Wait for signal (condition variable)
Signal received or spurious wakeup
Queue has new_fd
Dequeue new_fd
Unlock queue (release mutex)
Read HTTP request
Parse HTTP request
Lock cache for read (acquire read lock)
alt
File in cache
Return file data
Unlock cache (release read lock)
File not in cache
Unlock cache (release read lock)
Read file from disk
File data or error
Lock cache for write (acquire write lock)
Write file to cache (LRU logic)
Unlock cache (release write lock)
Build and send HTTP response (200, 404, 500)
Lock stats (acquire semaphore)
Increment request/byte/status counters
Unlock stats (release semaphore)
Lock log (acquire semaphore)
Write log entry
Unlock log (release semaphore)
Close connection (close new_fd)
Thread exits
Client
Worker Thread
Internal Queue
Local Cache
Disk
Shared Memory
Log File

Skeleton Code

```
#define SHM_NAME   "/chs_shm_skel"

#define SEM_EMPTY  "/chs_sem_empty"

#define SEM_FULL   "/chs_sem_full"

#define SEM_MUTEX  "/chs_sem_mutex"

#define BUFFER_SIZE 5   // small for test


typedef struct {

    int buffer[BUFFER_SIZE];

    int head, tail, count;

} shared_data_t;


int main(void) {

    int shm_fd;

    shared_data_t *data;


    // --- Create shared memory ---

    shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);

    ftruncate(shm_fd, sizeof(shared_data_t));

    data = mmap(NULL, sizeof(shared_data_t),

                PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);

    data->head = data->tail = data->count = 0;
```

```
// --- Create semaphores ---
sem_t *sem_empty = sem_open(SEM_EMPTY, O_CREAT, 0666, BUFFER_SIZE);
sem_t *sem_full  = sem_open(SEM_FULL,  O_CREAT, 0666, 0);
sem_t *sem_mutex = sem_open(SEM_MUTEX, O_CREAT, 0666, 1);


pid_t pid = fork();


if (pid == 0) {
    // --- Child: Consumer ---
    for (int i = 0; i < 10; i++) {
        sem_wait(sem_full);
        sem_wait(sem_mutex);


        int item = data->buffer[data->head];
        data->head = (data->head + 1) % BUFFER_SIZE;
        data->count--;
        printf("[Consumer] got %d\n", item);


        sem_post(sem_mutex);
        sem_post(sem_empty);
        usleep(150000);
    }
    _exit(0);
} else {
    // --- Parent: Producer ---
    for (int i = 1; i <= 10; i++) {
            sem_wait(sem_empty);

            sem_wait(sem_mutex);


            data->buffer[data->tail] = i;

            data->tail = (data->tail + 1) % BUFFER_SIZE;

            data->count++;

            printf("[Producer] put %d\n", i);


            sem_post(sem_mutex);

            sem_post(sem_full);

            usleep(100000);

        }

        wait(NULL);

    }


    // --- Cleanup ---

    sem_close(sem_empty); sem_close(sem_full); sem_close(sem_mutex);

    sem_unlink(SEM_EMPTY); sem_unlink(SEM_FULL); sem_unlink(SEM_MUTEX);

    munmap(data, sizeof(shared_data_t));

    shm_unlink(SHM_NAME);

    return 0;

}
```