Assignment 2 Guide

This document contains some ideas and pseudo codes which can be used to implement assignment 1. These are not the best and only solutions for meeting assignment 2 requirements. **You may use different ideas for completing the assignment requirements**.

Octa-Blocks and Octa-Legs:

One way to implement the size requirements is the following algorithm:

A. Send as many **Full Octa-Blocks** (8888 bytes) as you can:
   1. **Octa-Leg** = **Full Octa-Block** / 8
   2. send 8 Octa-Legs
   3. calculate the size of the remaining part of the file
   4. if the remaining part of the files is greater than **8888 bytes** → repeat from **step 1**

B. If the remaining part of the file is greater than **8 bytes** → Send at most one **Partial Octa-Block**:
   1. **Partial Octa-Bock** = remaining-part – (remaining part % 8)
   2. **Octa-Leg** = **Partial Octa-Block** / 8
   3. send 8 Octa-Legs
   4. calculate the size of the remaining part of the file

C. If the remaining part of the file is greater than 0 bytes → Send at most one **Tiny Octa-Block**:
   1. **Tiny Octa-Block** = 8
   2. **Octa-Leg** = **Tiny Octa-Block** / 8
   3. send 8 Octa-legs:
      i. if there are remaining bytes → send one
      ii. else if there is no remaining byte → send a space character (“ ”) as one byte padding

Sequence Numbers:

You can add 1 byte sequence number at the end of each Octa-Leg. For simplicity you can do this after you are done calculating length of each Octa-Leg.

Multi-Processing/Threading:

Multi-Processing/Threading is **needed** for implementing timeout and retransmission process. The basic requirement of the assignment is to send one ACK for each Octa-Block. This means right after you start sending an Octa-Block (by sending the first Octa-Leg of it) you should a timer and a ACK receiver working concurrently and Multi-Processing/Threading is used here. One solution is to use two different child process for each task. The first child is a timer and the second one is ACK receiver. These children can send their results to the parent process using **pipe()**. The following pseudo code explains the multi-processing part of this concept:

1. Create a **pipe1**
2. Call **fork()** to create child1 process
3. If the pid is for **child1**:
   I. Start the timer (sleep function can be used)
   II. Use **pipe1** to send the notification after the timer is done

4. Else if the pid is for **parent**:
    I. Create **pipe2**
    II. Call **fork()** to create **child2** process
    III. If the pid is for child2"
        a. Call **recevfrom()** function to receive the possible ACK from the client
        b. Use **pipe2** to send the notification to the **parent** process
    IV. Else if pid is for **parent**:
        a. Do whatever the server should do while the timer and receiver processes are working
        b. Read from **pipe2** to see if any ACK is received
        c. Read from **pipe1** to see if there is a time-out
        d. Do whatever the server should do based on the notifications you read from **pipe1** and **pipe2**

Using Multi-Processing/Threading for sending Octa-legs concurrently is not mandatory but feel free to do it if you are interested.

Other tips:

- We will cover these algorithms in code on the first tutorial of the next week and the second tutorial would be a general help session for the assignment.
- You have make various assumptions and decisions for implementing your protocol. Make sure to do it in a way to make your implementation simpler and easier for you. Also don't forget to justify your assumptions.
- For testing your program you can use your own laptop as the host for both server and client for now. For generating packet losses you can use random number generators to decide whether send the ACK or not to a received message and see if the server send the lost packet again or not. We have not yet decided on how to test your solutions on demo day. You will be informed as soon as it is decided. It worth mentioning that for testing you code on two computers you should consider using correct addresses of the client and the server.