Aceasta tema mi s-a parut cea mai dificila dintre toate, avand nevoie de o intelegere aprofundata a conceptelor legate de operatii I/O avansate. Pentru setarea initiala a proiectului, am folosit fisierul epoll\_echo\_server.c, avand la baza un server echo-based de multiplexare. Am inceput prin setarea portului nostru la AWS\_LISTEN\_PORT. Structura din main a server-loop-ului este asemanatoare cu cea din laboratorul 11, unde prima oara ne ocupam de handle\_new\_connection. Aceasta foloseste functiile de wrapping pentru epoll. Acest sistem foloseste protocolul HTTP, cu clasicul model client-server, insa doar cu functionalitatea de transmitere fisiere clientilor.

Pentru parsarea HTTP, folosesc modelul prezent in resursele temei. In handle\_client\_request,extragem path-ul si facem un simple request cu html\_parser\_execute. Salvam pathname-ul si deschidem fisierul cu ajutorul unui file descriptor. Dupa aceea, in functie de file descriptor, trimitem mesajul HTTP 200 daca request path-ul este valid si reusim sa deschidem fisierul, 404 altfel.

Inainte de inchiderea conexiunii, trebuie sa tinem cont de primirea mesajului in receive\_message. Functia recv primeste mesaje de la conexiune si socket-ul primeste datele intr-un receive buffer. Trebuie sa avem grija ca acesta sa nu se umple printr-un flow control. O abordare similara este facuta in functia send\_message.

In send\_message, ne ocupam si de cazul in care ne aflam cu fisierul nostru. Daca avem un fisier in AWS\_DOCUMENT\_ROOT/static, documentele sunt transmite clientilor folosind API-ul zero-copying (sendfile). Am folosit fstat pentru a obtine dimensiunea fisierului si am transferat data intre file descriptorii sockfd si fd. Cu toate acestea, nu am primit punctaj pentru testele cu cmp.

Pentru abordarea dinamica, la fisierele din AWS\_DOCUMENT\_ROOT/dynamic, a trebuit sa implementez un API async unde avem o faza post-procesare. Am incercat sa iau un model de file sending din fisierul eventfd-aio-test.c si din laboatorul 11 de anul trecut, insa nu am reusit sa obtin punctele pe testare. Initial, am creat un event cu valoarea initiala 0. Am folosit eventfd. In teorie, de aici ar trebui sa extragem valoarea de operatii I/O incheiate si sa o folosim la io\_getevents. Dupa alocarea bufferelor si umplerea lui piocb, fiecare buffer citeste cu ajutorul lui io\_prep\_pread si dupa aceea se realizeaza o operatie async cu ajutorul lui io\_submit, fara a bloca procesul current. Am incercat trimiterea de evenimente pe rand, pana la terminare.

Pentru debugging, m-am asigurat sa am variabila DO\_CLEANUP=no setata si ca checker-ul sa nu imi stearga fisierele de output create. Am verificat ceea ce afisez in test.log. De asemenea, a fost nevoie sa folosesc si netstat.