Συνοδευτικό αρχείο τεκμηρίωσης .

Στο αρχείο filereader υλοποιώ 2 συναρτήσεις.
Στη πρώτη συνάρτηση είναι ο κώδικας ανάγνωσης του αρχείο , όπου η συνάρτηση "load segment" παίρνει ως ορίσματα απο ποιο αρχείο , το segment , και ποσες γραμμές και πηγένει να το διαβάσει . Κάνει rewind για να ξεκινήσει το αρχείο από την αρχή και προσπερνάει τις ανάλογες γραμμές ώστε να φτάσουμε στο segment που θέλουμε (skip_lines) .
Η συνάρτηση δεσμεύει δυναμικά μνήμη για να αποθηκεύσει τις γραμμές που διαβάστηκαν από το αρχείο και στη συνέχεια τις ενώνει .

Στο filereader υπάρχει ακομα μια συναρτηση bool , η segment_already_loaded που παίρνει ως όρισμα ενα segment και αν αυτο το segment έχει φορτωθεί επιστρέφει true αλλιώς false.

```c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#include "limits.h"
#include "FileReader.h"

static int segment_loaded = -1;

char * load_segment(FILE * fp, int segment, int segment_lines) {
    char buffer[LINE_LENGTH] = { 0 };
    int skip_lines = segment*segment_lines;

    rewind(fp);

    while (fgets(buffer, sizeof (buffer), fp) && skip_lines > 0) {
        skip_lines--;
    }


    char ** buffers = (char **) malloc(sizeof(char*)*segment_lines);

    for (int i=0;i<segment_lines;i++) {
        buffers[i] = malloc(sizeof(char)*LINE_LENGTH);
    }

    for (int i=0;i<segment_lines;i++) {
        fgets(buffers[i], sizeof (buffer), fp);
    }

    int M = sizeof(char)*segment_lines*(LINE_LENGTH +1);
    char * merged = malloc(M);

    strcpy(merged, "");

    for (int i=0;i<segment_lines;i++) {
        strcat(merged, buffers[i]);
    }

    for (int i=0;i<segment_lines;i++) {
        free(buffers[i]);
    }

    free(buffers);

    segment_loaded = segment;

    printf("################ Segment loaded: %d \n", segment_loaded);

    return merged;
}

bool segment_already_loaded(int segment) {
    if (segment_loaded == segment) {
        return true;
    } else {
```

Για να κρατάω το segment σε περίπτωση που ζητηθεί και να μη πηγένει στη μνήμη να το ξαναφέρει το πρόγραμα εχει υλοποιηθεί ως εξής:

Όταν έρχεται ένα request , έχω μια μεταβλητη που μπορεί να ειναι ειτε request είτε notification (ορισμένα και τα 2 στο limits.h) με τιμές 0 και 1.
Άν ειναι request τοτε αυξάνω τον μετρητη requests_proccessed και εκτυπώνω ενα μήνυμα με τον αριθμο του request.
Έχω 2 περιπτώσεις. Η πρώτη είναι να είναι η πρώτη αίτηση , και η δεύτερη το segment να ειναι ηδη loaded και να έχω hit (το segment ειναι ήδη φορωμένο) .
Άν είναι η πρώτη αίτηση το φορτώνω , αλλιώς αν έχω hit δεν φορτώνω κατι και ξυπνάω το παιδί.
Τώρα έχω miss , τότε αν δεν υπάρχει κάποιο παιδί που να διαβάζει φορά φορτώνω το καινούργιο segment και αυξάνω τους readers . Ενώ αν υπάρχουν readers τότε τους καταγράφω στο hold id με το αντίστοιχο id που είναι κάθε παιδί ποιό segment θέλει και σε ποιο timestamp το ήθελε.
Αν rt==notification τοτε αυξάνω το votifications_proccessed , έπειτα μειώνω τους readers και κοιτάω αν κανένα παιδί δεν διαβάζει εκείνη την ώρα. Αν δεν διαβάζει κανένα τότε βρίσκω μέσω του on_hold_ids ποιο παιδί περιμένει περισσότερο και φορτώνω το segment που έχει ζητηθεί για αντίστοιχο παιδί και στη συνέχεια το ξυπνάω .
Για κάθε παιδί αν και αυτό περιμένει και το segment που θα ήθελε ειναι το ίδιο με αυτο που ειναι στο min_pos τοτε σε αυτό το παιδί βάζω not waiting γιατι δεν περιμένει πια , αυξάνω τους readers και ξυπνάω το παιδι.

```
upEmpty(semId);

if (rt == REQUEST) {
    requests_processed++;

    printf(">> Request: %d \n", requests_processed);

    if (requests_processed == 1 || segment_already_loaded(s) == true) { // first request or cache hit
        if (requests_processed == 1) {
            char * merged = load_segment(fp, s, segment_lines);
            int M = sizeof (char)*segment_lines * (LINE_LENGTH + 1);
            memcpy(segment_pointer, merged, M);
            free(merged);
        }

        readers++;
        upChild(semid, id);
    } else { // cache miss
        if (readers == 0) {
            char * merged = load_segment(fp, s, segment_lines);
            int M = sizeof (char)*segment_lines * (LINE_LENGTH + 1);
            memcpy(segment_pointer, merged, M);
            free(merged);

            readers++;
            upChild(semid, id);
        } else {
            // do not wake, child, since it cannot be serviced
            int timestamp = requests_processed;
            on_hold_ids[id][0] = s;
            on_hold_ids[id][1] = timestamp;
        }
    }
} else if (rt == NOTIFICATION) {
    notifications_processed++;
    readers--;

    if (readers == 0) {
        int min_pos = -1;
        int timestamp = INT_MAX;

        for (int i = 1; i <= children; i++) {
            if (on_hold_ids[i][0] != NOT_WAITING && on_hold_ids[i][1] < timestamp) {
                min_pos = i;
                timestamp = on_hold_ids[i][1];
            }
        }

        if (min_pos != -1) {
            char * merged = load_segment(fp, on_hold_ids[min_pos][0], segment_lines);
            int M = sizeof (char)*segment_lines * (LINE_LENGTH + 1);
            memcpy(segment_pointer, merged, M);
            free(merged);

            readers++;
```

-Στο αρχείο limits υπάρχουν ορισμένες με defined συγκεκριμένα οι σταθερές που έχουν ζητηθεί για την εργασία απο το e class , καθως άλλες σταθερές για διευκόλυνση.

-Στο semun απλά έχει δηλωθεί η συνάρτηση semun

-Για τον κώδικα για τους σημαφόρους , έχουν χρησιμοποιηθεί ως πηγές τα αρχεία του μαθήματος στο e class.