Jordi Romero Suarez NIUB : 20081633
Jose Manuel Lopez Camuñas NIUB: 18079924

SISTEMES OPERATIUS II: Pràctica 1

1. A l'enunciat de la pràctica es comenta que la funció fgets és més segura que fscanf per llegir text pla. Podeu comentar per què? Quines avantatges aporta la funció fgets a nivell de seguretat?

La funció fgets es mes segura a la hora de llegir textos plans ja que llegeix línia per línia els caràcters. A la vegada també se li pot dir el nombre màxim de caràcters que vols que llegeixi. En comparació amb el fscanf que segueix un cert codi de format que tu demanes.

Per temes de seguretat l'avantatge que ens aporta el fgets es que emmagatzemant el text que llegeix per cada línia, en comparació amb el fscanf que no emmagatzema les dades que va llegint. També és molt senzill ja que en terminar de llegir totes les línies retorna un valor null (és fàcilment iterable amb un while).

2. Com s'ha adaptat el codi de la manipulació de l'arbre perquè la clau d'indexació sigui una cadena. Indiqueu de forma clara quines funcions s'han adaptat. Recordeu que, tal com s'indica a la secció 2.1, cada node ha d'emmagatzemar la paraula amb la mida mínima necessària (i.e. no es pot suposar que la paraula té una mida màxima de, per exemple, 100 caràcters). Podeu incloeu codi per facilitar la descripció de les modificacions realitzades.

Tant al mètode compare_key1_less_than_key2 com compare_key1_equal_to_key2 de la classe red-black-tree hem canviat els paràmetres de la funció. Inicialment aquests venien amb un "int" ja que comparava nombres. Nosaltres per poder dur a terme la funcionalitat de la practica vam haver de canviar-los per char a partir d'un punter. També hem canviat la funció internament, ja que abans es comparaven els dos números i nosaltres hem afegit una funció(strcasecmp) per que compari l'string sense sensibilitat a l'hora de ser minúscula o majúscula.

Jordi Romero Suarez NIUB : 20081633
Jose Manuel Lopez Camuñas NIUB: 18079924

Finalment per la part de la classe red-black-tree a la funció “find_node” hem canviat el paràmetre de key de tipus int a un char.

Per tal d’assegurar que les paraules ocupin el bytes que necessiten utilitzem la funció strdup que retorna el punter de la mida necessària donat un string.

3. Quin és el problema que té el codi d’extracció de paraules de la secció 2.2? Comenteu breument com ho heu arreglat.

El problema que tenim al codi d’extracció de paraules és que extreu les paraules independentment dels signes de puntuació que trobem a la paraula(ens referim als guions, apòstrofs). Es a dir, de la paraula taper-light, al trobar-se amb el guió la separa en dues paraules.

Aquest problema esdevé en a extraccio_paraules.c, ja que de la línia sencera agafarà una paraula, anirà comprovant caràcter a caràcter si es un caràcter alfanumèric, en el cas que no ho sigui, es a dir hi hagi un espai o sigui un signe de puntuació començarà a comptar una altra paraula.

Per la qual cosa tal com s’ha demanat a la practica per evitar que corroeixin aquests casos a l’hora de comprovar si la paraula ha arribat al final, creem una nova condició que comprovi si el caràcter que li toca imprimir es un apòstrof(‘) o un guio(-).

```
do {
    if(isalpha(line[i]) || isdigit(line[i]) || ispunct(line[i])){
        paraula[j] = line[i];
    }
    else{
        is_word = 0;
    }
    j++; i++;


    /* Check if we arrive to an end of word: space or punctuation character */
} while ((i < len_line) && (!isspace(line[i])) && (!ispunct(line[i]) || (line[i] == '\'') || (line[i] == '-')));
```

4. En cas que incloeu alguna cosa excepcional (diferent) de la demanada a l'enunciat, podeu incloure també l'explicació de la funcionalitat implementada.

Hem implementat el nostre programa de tal manera que necessitem 2 comandes addicionals per funcionar seguint l'estructura: ./main <paràmetre1> <paràmetre2>

- **Paràmetre1:** nom de la llista a executar (per exemple, llista_2.cfg).
- **Paràmetre2:** mode de funcionament on tenim "all" per mostrar per cada arxiu tota la comparació amb el diccionari(inclou paraules amb 0 ocurrències), "inserted" que alternativament només mostra les paraules insertades a cada arxiu i per últim "top10" que mostra <num_ocurrència> <paraula> per facilitar trobar els top.

En cas de no indicar el paràmetre 2 , funcionarà com "inserted" per defecte.

Per raons de modularitat hem separat certes funcions a un header i un .c per a importar com si fos una llibreria i no saturar l'arxiu main.c.

5. Quines són les 10 paraules que apareixen més cops als fitxers de texts proporcionats? Comenteu quantes vegades apareixen cadascuna d'aquestes 10 primeres paraules i indiqueu clarament com heu obtingut el resultat. Sou capaços d'obtenir el resultat ajudant-vos de les instruccions de la línia de comandes? És a dir, podeu obtenir el resultat sense haver
de programar-ho tot en C?

Entenem que se'ns demana trobar les paraules pel total de cada llista.

A la llista.cfg les paraules que més surten són:

144580 the

90910 OF

83260 to

77435 AND

64497 A

39625 IN

38733 HE

38446 her

35892 I

30617 was

A la llista_2.cfg les paraules que més surten són:

13266 AND

12551 the

6316 OF

5059 to

3642 A

3612 that

3541 HE

3531 I

3041 OF

2582 but

A la llista_10.cfg les paraules que més surten són:

13266 AND

12551 the

6947 OF

5059 to

4067 HE

3642 A

3612 that

3531 I

3064 was

2837 IN

Per fer-ho des de la línia de comandes, únicament ens permetria buscar una paraula en concret en un únic fitxer.

La instrucció que li has de passar a la línia de comandes es la següent:

$ grep -o -i XXXX YYYY.txt | wc -l

On les XXXX són la paraula a trobar al fitxer i les YYYY és el fitxer on vols revisar la paraula.

Alternativament tenim :

./main llista_2.cfg top10 > e.txt , on redireccionem el text que mostra el programa a un arxiu (e.txt).

sort -nr e.txt > top.txt , ordena numèricament e.txt i crea un fitxer top.txt amb el top disponible.