Universidade da Beira Interior

Departamento de Informática

Relatório do Exercicio 1

Elaborado por:

Joel Tapia a47275
Manuel Garcia a45500
Tiago Ribeiro a46346

11 de abril de 2023

Capítulo

1

Introdução

1.1 Motivação

Este projeto está enquadrado na **UC!** (**UC!**) de Sistemas operativos, do 2ª ano da Licenciatura de Engenharia Informática, da **UBI!** (**UBI!**).
O conceito base passava por implementar um programa que devolve uma sequência de instruções que podem ser utilizadas para encontrar uma string passada como argumento.

1.2 Objetivos

Os objetivos pretendidos para a parte B deste trabalho eram:

- Esconder uma string dada pelo utilizador num ficheiro binário dado pelo mesmo.
- Fornecer uma lista de instruções que permitam encontrar essa string.

Capítulo

2

Desenvolvimento e implementação

2.1 Descrição do código

A função que gera uma instrução aleatória tem 9 parâmetros importantes, os dois primeiros são feitos para o retorno da função que detém o "operador"e o "número de movimento". (*n_pass retorna alterando a variável principal), estas duas variáveis com "binaryl"fazem quase todos os casos. Assim, a primeira parte da função começa com o "switch"que seleciona aleatoriamente o operador, depois cada caso deve ter uma condição que verifique que após a realização da operação não excedemos o tamanho do ficheiro, pelo que todos os se devem ter o "binaril"em comparação com a posição atual "actualposition"então o programa guardará toda a informação se passar as condições e voltar ao principal.

Se o operador escolhido for "r", então realizará a comparação explicada acima, com uma comparação que verifica que o tamanho da string é maior do que o buffer. Depois, sempre que o tamanho da cadeia de caracteres e o tamanho do buffer não forem iguais, e também quando a posição atual estiver no fim do ficheiro, será criando um novo número aleatório que terá um intervalo entre um e o menor de (tamanho do ficheiro - posição atual) e (tamanho do fio - tamanho do buffer). -> (1 - o menor)

A última parte compara se a variável "sample"(que faz uma leitura de n elementos do ficheiro) e o "comparator"(que faz uma cópia de n elementos da palavra a pesquisar), são iguais, caso sejam, salvá-lo-á no buffer, caso contrário voltará ao início.

Toda a função tem um do-while necessário para repetir todo o algoritmo até encontrar um operador e um número que possa passar os condicionantes.

Na função main, no início temos comparadores que comprovam se o fi-

cheiro existe ou se foi possível abri-lo, após alocar as variáveis necessárias, o programa analisa se todos os caracteres da string dada pelo utilizador estão presentes no ficheiro binário, caso não tenha o programa fecha de imediato, caso esteja contínua e começamos um ciclo while que executara os operadores clamando a função generate_instructions e dependendo do resultado as condicionais fazerem as operações, o comando "read"tem operações especiais que só fazem o guardado no buffer dos caracteres lidos, utilizando primeiro o read numa string "apender"para depois copiar toda a informação no buffer.

2.2 Código fonte

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <time.h>
5
6  #define MAX_INSTR_LENGTH 6 // Tamanho ámximo de uma çãinstruo
7
8  char* generate_instruction(char *instruction, int *n_pass , int ↩
       tamDoBuffer, int tamDaStringARevelar ,FILE * position  ,int ↩
       binaryl, int actualposition, char * stringAProcurar , int *↩
       posicaoDaString) {
9      char *inst = malloc(MAX_INSTR_LENGTH * sizeof(char));
10     int n = 0, sum= 0, choice;
11
12     do{
13     switch(choice=rand() % 5) {
14         case 0:
15             strcpy(inst, "+");
16             break;
17         case 1:
18             strcpy(inst, "-");
19             break;
20         case 2:
21             strcpy(inst, "i");
22             break;
23         case 3:
24             strcpy(inst, "f");
25             break;
26         case 4:
27             strcpy(inst, "r");
28             break;
29     }
```

```
30
31      if (inst[0] == '+' && binaryl >= actualposition + n) {
32          if(binaryl - actualposition!=0){
33              n= rand() % (binaryl - actualposition) +1;
34          }
35          else{
36              continue;
37          }
38          sprintf(instruction, "%s %d", inst, n);
39          *n_pass = n;
40          if(binaryl >= actualposition + n)
41          sum = 1;
42      }
43
44      else if (inst[0] == '-' && (actualposition-n) >= 1)
45      {
46          n= rand() % actualposition + 1 ;
47          sprintf(instruction, "%s %d", inst, n);
48          *n_pass = n;
49          if((actualposition-n) >= 1)
50          sum = 1;
51      }
52
53      else if (inst[0] == 'i' && binaryl >= n) {
54          n= rand() % binaryl +1 ;
55          sprintf(instruction, "%s %d", inst, n);
56          *n_pass = n;
57          if(binaryl >= n)
58          sum = 1;
59      }
60
61      else if (inst[0] == 'f' && n <= binaryl)
62      {
63          n= rand()  % binaryl +1;
64          sprintf(instruction, "%s %d", inst, n);
65          *n_pass = n;
66          if(n <= binaryl)
67          sum = 1;
68      }
69      else if ((inst[0] == 'r') && (binaryl >= actualposition + n) && (↩
            tamDaStringARevelar > tamDoBuffer)) {
70          if((binaryl - actualposition !=0) && (tamDaStringARevelar -↩
                tamDoBuffer != 0)){
71              n= rand() % (((binaryl - actualposition)< (↩
                    tamDaStringARevelar -tamDoBuffer)) ? (binaryl -↩
                    actualposition) : (tamDaStringARevelar - tamDoBuffer))↩
                     +1;
72          }
73          else{
```

```
74              continue;
75          }
76          sprintf(instruction, "%s %d", inst, n);
77          *n_pass = n;
78
79          char *sample =malloc( sizeof(char) *  (n+1));
80          fread(sample,1,n,position);
81          sample[n] = '\0';
82          char *comparator = malloc(sizeof(char) * (n+1));
83          strncpy(comparator,stringAProcurar,n);
84          comparator[n] = '\0';
85          if(n +actualposition <= binaryl && (n<= (tamDaStringARevelar -↩
                tamDoBuffer)) && strcmp(comparator , sample) == 0){
86          sum = 1;
87          *posicaoDaString += n;
88
89          }
90          if(strcmp(comparator, sample) != 0){
91              fseek(position,-n,SEEK_CUR);
92          }
93
94          free(sample);
95          free(comparator);
96
97      }
98      } while (sum != 1 );
99      if(n!=0)
100         printf("%s\n",instruction);
101     return instruction;
102 }
103
104
105 int main(int argc, char* argv[]) {
106     if (argc != 3) {
107         printf("Binary file and message requiered\n");
108         return 1;
109     }
110
111     FILE* binary_file = fopen(argv[1], "rb");
112     if (binary_file == NULL) {
113         printf("Error opening file\n");
114         return 1;
115     }
116
117     char byte;
118     int found = 0;
119     char* string = argv[2];
120     strtok(string, "\n");
121     int string_len = strlen(string);
```

```
122     fseek(binary_file,0l,SEEK_END);
123     int binarylength = ftell(binary_file);
124     rewind(binary_file);
125 
126     while (fread(&byte, 1, 1, binary_file) == 1) {
127         if (byte == string[found]) {
128             found++;
129             if (found == string_len) {
130                 break;
131             }
132         }
133     }
134 
135     if (found != string_len) {
136         printf("String not found in binary file.\n");
137         return 1;
138     }
139 
140     srand(time(NULL)); // Inicia o gerador de únmeros óaleatrios
141 
142     char instruction[MAX_INSTR_LENGTH];
143     int param;
144     int maxBufferSize = strlen(argv[2]) +1 ;
145     char * buffer = malloc(maxBufferSize);
146     buffer[maxBufferSize] = '\0';
147     int buffer_pos = 0;
148     int posicaoNaString =0;
149 
150     while (strlen(buffer) < string_len) {
151 
152         generate_instruction(instruction, &param, strlen(buffer) , ↩
                strlen(string), binary_file, binarylength, ftell(↩
                binary_file), &(string[posicaoNaString]), &posicaoNaString↩
                 );
153 
154         if (instruction[0] == '+' && param != 0) {
155             fseek(binary_file, param, SEEK_CUR);
156         }
157         else if (instruction[0] == '-' && param != 0) {
158             fseek(binary_file, -param, SEEK_CUR);
159         }
160         else if (instruction[0] == 'i' && param != 0) {
161             fseek(binary_file, param, SEEK_SET);
162         }
163         else if (instruction[0] == 'f' && param != 0) {
164             fseek(binary_file, -param, SEEK_END);
165         }
166         else if (instruction[0] == 'r' && param != 0) {
167             char * apender = malloc(sizeof(char) *  (param +1));
```

```
168
169             fread (apender, 1, param ,binary_file );
170             apender[param] = '\0';
171
172             strcat(buffer,apender);
173
174             free(apender);
175         }
176         else{continue;}
177
178         if (buffer_pos >= maxBufferSize -1) {
179             printf("s 0\n");
180             memset(buffer, 0,maxBufferSize);
181             buffer_pos = 0;
182         }
183     }
184
185     free(buffer);
186     fclose(binary_file);
187     printf("s 0\n");
188     return 0;
189 }
```

Excerto de Código 2.1: Código fonte exercicio 1

Capítulo

3

Exemplos de execução

```
caesar@LeNovoDasCouves:~/SO/trabalho1ParteB$ gcc -std=c99 -Wall -o ProgA -lm main.c
caesar@LeNovoDasCouves:~/SO/trabalho1ParteB$ ./ProgA test.bin < input1.txt
password
caesar@LeNovoDasCouves:~/SO/trabalho1ParteB$ gcc -std=c99 -Wall -o ProgB -lm mainTiago.c
caesar@LeNovoDasCouves:~/SO/trabalho1ParteB$ ./ProgB test.bin password > inputA.txt
caesar@LeNovoDasCouves:~/SO/trabalho1ParteB$ ./ProgB test.bin password
```

```
- 2
i 10
- 4
f 5
i 8
- 5
+ 2
r 1
f 10
f 18
i 3
+ 5
- 4
i 4
f 17
f 4
+ 4
- 10
f 9
- 7
- 1
r 3
s 0
caesar@LeNovoDasCouves:~/SO/trabalho1ParteB$ ./ProgA test.bin < inputA.txt
password
caesar@LeNovoDasCouves:~/SO/trabalho1ParteB$
```