THREADS

**1. Write a program that receives strings as command line arguments and uses threads to capitalise each word. We will create for each argument a separate thread that will capitalise the initial letter**

1 #include <pthread.h>

2 #include <stdio.h>

3 #include <stdlib.h>

4 #include <string.h>

5 #include <ctype.h>

6 #define MAXLINIE 1000

7 pthread\_t tid[100];

8

9 void\* ucap(void \*numei) {

10 printf("Thread start: %ld ...> %s\n", pthread\_self(), (char\*)numei);

11 char numeo[100];

12 strcpy(numeo, (char\*)numei);

13 if ( numeo[0]>='a' && numeo[0]<='z' )

14 numeo[0]+='A'-'a';

15 printf("Thread finished: %ld ...> %s\n", pthread\_self(), (char\*)numeo);

16 return NULL;

17 }

18

19 int main(int argc, char\* argv[]) {

20 int i;

21 for(i=1; argv[i]; i++) {

22 pthread\_create(&tid[i], NULL, ucap, (void\*)argv[i]);

23 printf("Thread created: %ld ...> %s\n", tid[i], argv[i]);

24 }

25 for(i=1; argv[i]; i++) pthread\_join(tid[i], NULL);

26 printf("All threads finished\n");

27 return 0;

28 }

**2. Add numbers in parallel, two by two, using threads. Main rpogram receives these values and uses the partial sums to compute the whole sum. 1 2 6 3 6 7 5 8**

1 #include <pthread.h>

2 #include <stdio.h>

3 #include <stdlib.h>

4 #include <string.h>

5 #include <ctype.h>

6

7 pthread\_t tid[100];

8 //int a[]={1,2,3,4,5,6,7,8,9,10};

9 int array[1000];

10

11 typedef struct {

12 int ps;

13 } response;

14

15 void\* partial(void \*id) {

16 int nr=\*(int\*)id;

17 response\* r=malloc(sizeof(response));

18

19 //printf("Thread start: %ld ...> %d\n", pthread\_self(), nr);

20 r->ps=array[2\*nr]+array[2\*nr+1];

21 //printf("Thread finished: %ld ...> %d\n", pthread\_self(), r->ps);

22 pthread\_exit(r);

23 }

24

25 int main(int argc, char\* argv[]) {

26 int i=0;

27 int N=argc-1;

28 for(i=1;argv[i];i++){

29 //printf("%d %d \n", atoi(argv[i]), argc);

30 array[i-1]=atoi(argv[i]);

31 //printf("%d \n", array[i]);

32 }

33

34 // create threads

35 int tnr[N/2];

36 for (i=0; i<N/2; i++) {

37 tnr[i]=i;

38 pthread\_create(&tid[i], NULL, partial, (void\*)&tnr[i]);

39

40 }

41

42 // wait for result

43 int sum=0;

44 response \*r[N/2];

45 for (i=0; i<N/2; i++) {

46 pthread\_join(tid[i], (void\*\*)&r[i]);

47 }

48 printf("All threads finished\n");49

50 // compute the sum

51 for (i=0; i<N/2; i++) {

52 sum+=((response \*)r[i])->ps;

53 }

54 if (N/2%2==1)

55 sum+=array[N-1];

56 printf("Total sum is: %d \n", sum);

57

58 // free memory

59 for (i=0; i<N/2; i++) {

60 free(r[i]);

61 }

62 return 0;

63 }

**3. Write a program that has a global variable count and 1000threads. Each thread increments count 1000times.**

1 #include <pthread.h>

2 #include <stdio.h>

3

4 int count=0;

5 pthread\_t tid[1000];

6 pthread\_mutex\_t exclusive=PTHREAD\_MUTEX\_INITIALIZER;

7

8 void\* inc(void\* nume) {

9 int i;

10 for (i=0;i<1000;i++) {

11 pthread\_mutex\_lock(&exclusive);

12 int temp=count; temp++; count=temp; //critical point

13 pthread\_mutex\_unlock(&exclusive);

14 }

15 return NULL;

16 }

17

18 int main(int argc, char\* argv[]) {

19 int i;

20

21 for (i=0;i<1000;i++)

22 pthread\_create(&tid[i], NULL, inc, NULL);

23

24 for (i=0;i<1000;i++)

25 pthread\_join(tid[i], NULL);

26

27 printf("count=%d\n", count);

28

29 pthread\_mutex\_destroy(&exclusive);

30

31 return 0;

32

33 }

**4. Given n pairs of command line arguments which are integer numbers, compute how many pairs have (a) an even sum, (b) an odd sum, (c) ar least one argument is 0 or nonnumerical value.**

1 #include <pthread.h>

2 #include <stdio.h>

3 #include <stdlib.h>

4 #define MAXLINIE 1000

5

6 typedef struct {char\*n1; char\*n2;} PERECHE;

7

8 pthread\_t tid[100];

9 PERECHE pair[100];

10

11 pthread\_mutex\_t mut1=PTHREAD\_MUTEX\_INITIALIZER;

12 pthread\_mutex\_t mtxeven=PTHREAD\_MUTEX\_INITIALIZER;

13 pthread\_mutex\_t mtxodd=PTHREAD\_MUTEX\_INITIALIZER;

14

15 int pare=0,impare=0,nenum=0;

16

17 void\* computepairs(void\* pair) {

18 // get the 2 values and transform n int. If it's not int => atoi = 0

19 int n1=atoi(((PERECHE\*)pair)->n1);

20 int n2=atoi(((PERECHE\*)pair)->n2);

21

22 // start alg

23 if (n1==0||n2==0) {

24 pthread\_mutex\_lock(&mut1);

25 nenum++;

26 pthread\_mutex\_unlock(&mut1);

27 }

28 else if ((n1+n2)%2==0) {

29 pthread\_mutex\_lock(&mtxeven);

30 pare++;

31 pthread\_mutex\_unlock(&mtxeven);

32 }

33 else {

34 pthread\_mutex\_lock(&mtxodd);

35 impare++;

36 pthread\_mutex\_unlock(&mtxodd);

37 }

38 return NULL;

39 }

40

41 int main(int argc, char\* argv[]) {

42 int i,p,n=(argc-1)/2;

43

44 for(i=1,p=0;p<n;i+=2,p++) {

45 pair[p].n1=argv[i];

46 pair[p].n2=argv[i+1];

47 pthread\_create(&tid[p], NULL, computepairs, (void\*)&pair[p]);

48 }

49

50 for(i=0;i<n;i++)

51 pthread\_join(tid[i], NULL);

52

53 printf("pairs=%d even=%d odd=%d nonnumeric=%d \n",n,pare,impare,nenum);

54

55 pthread\_mutex\_destroy(&mut1);

56 pthread\_mutex\_destroy(&mtxodd);

57 pthread\_mutex\_destroy(&mtxeven);

58

59 return 0;

60 }

**5. Write a C program that receives multiple integer numbers between 0 and 999 as command line arguments. For each number the program creates a thread and passes that number to the thread as an argument. The thread will append the number to a global array if it is even and will increment a global counter otherwise. Once all the threads finish, the main process prints the contents of the aray and the global counter. Use efficient synchronization mechanisms.**

1 #include <pthread.h>

2 #include <stdlib.h>

3 #include <stdio.h>

4

5 pthread\_mutex\_t mtx\_even=PTHREAD\_MUTEX\_INITIALIZER;

6 pthread\_mutex\_t mtx\_odd=PTHREAD\_MUTEX\_INITIALIZER;

7 int\* array\_even;

8 int nr\_odd = 0;

9 int nr\_even = 0;

10

11 void\* f(void\* a) {

12 int nr = \*(int\*)a;

13 printf("%d\n", nr);

14 if(nr%2==0){

15 pthread\_mutex\_lock(&mtx\_even);

16 array\_even[nr\_even]=nr;

17 nr\_even++;

18 pthread\_mutex\_unlock(&mtx\_even);

19 }

20 else{

21 pthread\_mutex\_lock(&mtx\_odd);

22 nr\_odd++;

23 pthread\_mutex\_unlock(&mtx\_odd);

24 }

25 free(a);

26 return NULL;

27 }

28 int main(int argc, char\* argv[]) {

29 pthread\_t\* tid = (pthread\_t\*)malloc(sizeof(pthread\_t)\*(argc-1));

30 array\_even = (int\*)malloc(sizeof(int)\*(argc-1));

31 int i=0;

32 for(i=0;i<argc-1;i++){

33 int\* nr = (int\*)malloc(sizeof(int));

34 \*nr = atoi(argv[i+1]);

35 pthread\_create(&tid[i], NULL, f, (void\*)nr);

36 }

37 for(i=0;i<argc-1;i++){

38 pthread\_join(tid[i], NULL);

39 }

40

41 //printing

42 printf("Array with even elements:\n");

43 for(i=0;i<nr\_even;i++)

44 printf("%d\n", array\_even[i]);

45 printf("Number of odd elements:%d\n", nr\_odd);

46

47 pthread\_mutex\_destroy(&mtx\_even);

48 pthread\_mutex\_destroy(&mtx\_odd);

49 free(tid);

50 free(array\_even);

51 return 0;

52 }

**6. Write a program that reads from the keyboard N integers and store them into a vector (N is a constant you can hardcode in your program). After all N integers are read, there will be created N threads that will receive one of the integer read and it will check if it has less or more than 3 digits, increasing two global variables each time the number has more 3 digits, respectively less than 3 digits. The main program will print at the end these two counters and will finish execution.**

1 #include <pthread.h>

2 #include <stdio.h>

3 #include <stdlib.h>

4

5 #define N 10

6 int less, more;

7 pthread\_mutex\_t mtx\_less=PTHREAD\_MUTEX\_INITIALIZER;

8 pthread\_mutex\_t mtx\_more=PTHREAD\_MUTEX\_INITIALIZER;

9

10 void\* f(void\* a) {

11 int nr = \*(int\*)a;

12 if(nr<100){

13 pthread\_mutex\_lock(&mtx\_less);

14 less++;

15 pthread\_mutex\_unlock(&mtx\_less);

16 }

17 if(nr>=1000){

18 pthread\_mutex\_lock(&mtx\_more);

19 more++;

20 pthread\_mutex\_unlock(&mtx\_more);

21 }

22 return NULL;

23 }

24 int main(int argc, char\* argv[]) {

25 int array[N];

26 pthread\_t\* tid = (pthread\_t\*)malloc(sizeof(pthread\_t)\*N);

27 int i;

28 for(i=0;i<N;i++){

29 scanf("%d", &array[i]);

30 }

31 for(i=0;i<N;i++){

32 pthread\_create(&tid[i], NULL, f, &array[i]);

33 }

34 for(i=0;i<N;i++){

35 pthread\_join(tid[i], NULL);

36 }

37

38 //printing

39 printf("Numbers with <3digits:%d, numbers with >3digits:%d \n", less, more);

40 pthread\_mutex\_destroy(&mtx\_less);

41 pthread\_mutex\_destroy(&mtx\_more);

42 free(tid);

43 return 0;

44 }

**7. Se da un fisier binar “tmp/214-file” care contine numere aleatoare.Cititi primele 2000de numere intregi din fisierul dat. Folosind un numar potrivit de threaduri si un obiect de sincronizare, determinati cel mai mare numar pozitiv, care e divizibil cu 7. Afisati rezultatul obtinut in programul principal.**

1 #include<pthread.h>

2 #include<stdio.h>

3 #include<stdlib.h>

4 #define N 10

5

6 FILE \*ptr;

7 pthread\_mutex\_t mtx=PTHREAD\_MUTEX\_INITIALIZER;

8 int maxdiv7=-1;

9

10 void\* f(void \*a){

11 int i, maxim=-1;

12 for(i=0;i<2000/N;i++) {

13 int nr;

14 fread(&nr, sizeof(int),1,ptr);

15 if(nr>maxim && nr%7==0)

16 maxim=nr;

17 }

18 pthread\_mutex\_lock(&mtx);

19 if(maxdiv7<maxim)

20 maxdiv7=maxim;

21 pthread\_mutex\_unlock(&mtx);

22 return NULL;

23

24 }

25 int main(int argc, char\* argv[]){

26 ptr = fopen("/tmp/214-file", "rb");

27 pthread\_t\* tid = (pthread\_t\*)malloc(sizeof(pthread\_t)\*N);

28 int i;

29 for(i=0;i<N;i++){

30 pthread\_create(&tid[i], NULL, f, NULL);

31 }

32

33 for(i=0;i<N;i++){

34 pthread\_join(tid[i], NULL);

35 }

36

37 printf("Max positive number div by 7: %d", maxdiv7);

38 pthread\_mutex\_destroy(&mtx);

39 free(tid);

40 fclose(ptr);

41 return 0;

42 }

**8. Write a program that reads from keyboard N integers and store them into a vector (N is a constant you can hardcode in your program).After all N integers are read, there will be created N threads that will receive one of the integer read and will check if it end with 5 or is dividing with 2, increasing two variables each time the number ended with 5 respectively divided with 2. The main program will print at the end these two counters and will finish its execution.**

1 #include <pthread.h>

2 #include <stdio.h>

3 #include <stdlib.h>

4 #define N 10

5

6 int ends\_five=0, div\_two=0;

7 pthread\_mutex\_t mtx=PTHREAD\_MUTEX\_INITIALIZER;

8 pthread\_mutex\_t mtx2=PTHREAD\_MUTEX\_INITIALIZER;

9

10

11 void\* f(void\* a){

12 int nr = \*(int\*)a;

13 if (nr%10==5){

14 pthread\_mutex\_lock(&mtx);

15 ends\_five++;

16 pthread\_mutex\_unlock(&mtx);

17

18 }

19 else if (nr%2==0){

20 pthread\_mutex\_lock(&mtx2);

21 div\_two++;

22 pthread\_mutex\_unlock(&mtx2);

23 }

24

25 return NULL;

26 }

27

28 int main(int argc, char\* argv[]){

29 pthread\_t\* tid = (pthread\_t\*)malloc(sizeof(pthread\_t)\*N);

30

31 int i=0;

32 int array[N];

33 for(i=0;i<N;i++){

34 scanf("%d", &array[i]);

35 }

36

37 for(i=0;i<N;i++){

38 pthread\_create(&tid[i], NULL, f, (void\*)&array[i]);

39 }

40

41 for(i=0;i<N;i++){

42 pthread\_join(tid[i], NULL);

43 }

44

45 printf("Numbers that end with 5:%d, Numbers that are divided by 2:%d \n", ends\_five, div\_two);

46 pthread\_mutex\_destroy(&mtx);

47 pthread\_mutex\_destroy(&mtx2);

48 free(tid);

49 return 0;

50 }

**9. Write a program that reads from keyboard N integers and stores them into a vector (N is a constant you can hardcode in your program). After all N integers are read, there will be created N threads that will receive one of the integer read and it will check if it has less or more than 3 digits, increasing two global variables each time the number has more than 3 digits, respectively less than 3. The main pogram will print at the end these two counters and will finish its execution.**

1 #include<pthread.h>

2 #include<stdio.h>

3 #include<stdlib.h>

4

5 pthread\_t tid[10];

6 int twoDn=0,thirdDn=0;

7 pthread\_mutex\_t mtx=PTHREAD\_MUTEX\_INITIALIZER;

8

9 void \*f(void \*a){

10 int nr2=0,nr3=0;

11 while(nr2<5) {

12 int nr = rand()%101+50;

13 printf("%ld: %d \n", pthread\_self(), nr);

14 if(nr<100) {

15 nr2++;

16 }

17 else {

18 nr3++;

19 }

20 }

21 pthread\_mutex\_lock(&mtx);

22 twoDn+=nr2;

23 thirdDn+=nr3;

24 pthread\_mutex\_unlock(&mtx);

25 return NULL;

26 }

27

28 int main(int argc, char\* argv[]) {

29 int i;

30 for(i=0;i<10;i++) {

31 pthread\_create(&tid[i],NULL,f,NULL);

32 }

33 for(i=0;i<10;i++) {

34 pthread\_join(tid[i],NULL);

35 }

36 //printing

37 printf("2dn: %d, 3dn: %d\n", twoDn, thirdDn);

38 pthread\_mutex\_destroy(&mtx);

39 return 0;

40 }

**------------------VLAD--------------------**

**9. Write a C program that receives a number N as a command line argument. The program creates N threads. Each thread generates two random integers between 0 and 999. The thread will display its index number and the generated numbers. If both generated numbers have different parities (one odd, one even), the thread will add the number to a global array. Once all the threads finish, the main process prints the contents of the array. Use efficient sync.**

1 #include <stdio.h>

2 #include <pthread.h>

3 #include <stdlib.h>

4

5

6 int \*array, count;

7 pthread\_mutex\_t m;

8

9 void\* f(void\* a){

10 int x, y;

11 x=rand()%1000;

12 y=rand()%1000;

13

14 printf("%d: %d %d\n", (int)a, x, y);

15

16 if(x%2!=y%2){

17 pthread\_mutex\_lock(&m);

18 array[count] = x;

19 count++;

20 array[count]=y;

21 count++;

22 pthread\_mutex\_unlock(&m);

23 }

24

25 free(a);

26 return NULL;

27 }

28

29

30 int main(int argc, char\*\* argv){

31 pthread\_t\* t;

32 int i;

33 int n = atoi(argv[1]);

34

35 if(argc != 2){

36 perror("Incorrect number of arguments!\n");

37 return 1;

38 }

39 pthread\_mutex\_init(&m, NULL);

40 int \*a;

41 t = (pthread\_t\*)malloc(sizeof(pthread\_t)\*n);

42 array = (int\*)malloc(sizeof(int)\*n\*2);

43 for(i=0;i<n;i++){

44 a= (int\*)malloc(sizeof(int));

45 \*a=i;

46 pthread\_create(&t[i], NULL, f, (void\*)a);

47 }

48

49

50 for(i=0;i<n;i++)

51 {

52 pthread\_join(t[i], NULL);

53 }

54 printf("The array is: ");

55 for(i=0;i<count;i++){

56 printf(" %d ", array[i]);

57 }

58 printf("\n");

59 free(t);

60

61 free(array);

62 pthread\_mutex\_destroy(&m);

63 return 0;

64 }

65

**8. Write a program that reads from keyboard N integers and store them into a vector (N is a constant you can hardcode in your program).After all N integers are read, there will be created N threads that will receive one of the integer read and will check if it end with 5 or is dividing with 2, increasing two variables each time the number ended with 5 respectively divided with 2. The main program will print at the end these two counters and will finish its execution.**

1. #include <stdio.h>
2. #include <pthread.h>
3. #include <stdlib.h>
4. #define N 10

7. int c5, c2;
9. pthread\_mutex\_t m1, m2;
11. void\* f(void \* a){

14. int nr = \*(int\*)a;
16. if(nr%10 == 5){
17. pthread\_mutex\_lock(&m1);
19. c5+=1;
21. pthread\_mutex\_unlock(&m1);
22. }else{
23. if(nr%2 == 0){
25. pthread\_mutex\_lock(&m2);
27. c2+=1;
29. pthread\_mutex\_unlock(&m2);
30. }
31. }
32. return NULL;
33. }


37. int main(int argc, char\*\* argv){

40. int i, \*v;
41. v = (int\*)malloc(sizeof(int)\*N);
42. for(i=0;i<N;i++){
43. printf("%d: ", i);
44. scanf("%d", &v[i]);
45. }
47. pthread\_mutex\_init(&m1, NULL);
48. pthread\_mutex\_init(&m2, NULL);
49. pthread\_t\* t;
50. t = (pthread\_t\*)malloc(sizeof(pthread\_t)\*N);
51. for(i=0;i<N;i++){
52. pthread\_create(&t[i], NULL, f, (void\*)&v[i]);
53. }
55. for(i=0;i<N;i++){
56. pthread\_join(t[i], NULL);
57. }

60. printf("Numbers that end in 5: %d\nNumber that divide by 2: %d\n", c5, c2);
62. free(v);
63. free(t);
65. pthread\_mutex\_destroy(&m1);
67. pthread\_mutex\_destroy(&m2);
68. return 0;


72. }

**7. Se da un fisier binar “tmp/214-file” care contine numere aleatoare.Cititi primele 2000de numere intregi din fisierul dat. Folosind un numar potrivit de threaduri si un obiect de sincronizare, determinati cel mai mare numar pozitiv, care e divizibil cu 7. Afisati rezultatul obtinut in programul principal.**

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <pthread.h>
5. #define N 10

8. pthread\_mutex\_t m;
9. FILE \*ptr;
10. int gmax = -1;

13. void\* f(void\* a){

16. int\* buf = (int\*)malloc(sizeof(int));
17. //int buf = 0;
18. int i, lmax=-1;
19. for(i=0;i<2000/N;i++){
20. fread(buf, sizeof(int), 1, ptr);
21. if(\*buf % 7 == 0 && \*buf >=0 ){
22. if(lmax == -1)
23. lmax = \*buf;
24. else{
25. if(lmax < \*buf)
26. lmax = \*buf;
27. }
28. }
29. }
31. pthread\_mutex\_lock(&m);
32. if(gmax == -1){
33. gmax = lmax;
34. }
35. else{
36. if(gmax < lmax){
37. gmax = lmax;
38. }
39. }
40. pthread\_mutex\_unlock(&m);
42. return NULL;
43. }


47. int main(int argc, char \*\* argv){
49. pthread\_t \*t = (pthread\_t\*)malloc(sizeof(pthread\_t)\*N);
50. pthread\_mutex\_init(&m, NULL);
51. ptr=fopen("/tmp/214-file", "rb");
52. int i;
54. for(i=0;i<N;i++){
55. pthread\_create(&t[i], NULL, f, NULL);
56. }
58. for(i=0;i<N;i++){
59. pthread\_join(t[i], NULL);
60. }
61. if(gmax!=-1){
62. printf("The biggest positive number divisible by 7 is: %d\n", gmax);
63. }
64. else{
65. printf("There is no number positive and divisible by 7.\n");
66. }
67. free(t);
68. pthread\_mutex\_destroy(&m);
69. fclose(ptr);
70. return 0;
71. }

**10. Scrieti un program c care primeste ca argumente in linia de comanda oricate perechi alcatuite dintr-un fisier si un caracter. Pentru fiecare pereche, programul va crea catre un thread, fiecare thread va primi ca argmuente o pereche de argumente. Programul va retine intr-un vectorglobal frecventa fiecarei cifre din fisierele transmise thread-urilor, fiecare thread va adauga in vectorul de frecventa frecventele fiecarei cifre din fisierul primit(de thread), fiecare frecventa pe pozitia corespunzatoare cifrei. Fiecare thread va nota intr-un vector global(alocat dinamic) pozitia primei aparitii a literei primite ca argument in fisierul primit ca argument(de catre thread). Executia threadurilor trebuie sa se desfasoarein paralel. Acceasul la variabilele globale trb sync.**

1 #include <stdio.h>

2 #include <stdlib.h>

3 #include <pthread.h>

4 #include <string.h>

5

6 typedef struct {

7 int position;

8 char filename[100];

9 char c;

10 } PAIR;

11

12 int frec[10], \*pos;

13

14 pthread\_mutex\_t m, m2;

15

16 void\* f(void \*a){

17 PAIR\* p = (PAIR\*)a;

18 //printf("%s %c\n", p->filename, p->c);

19

20 int fre[10] = {};

21

22 FILE\*f = fopen(p->filename, "r");

23 int found=0;

24 char c1;

25 int nr, position = 0, i;

26 while(fscanf(f, "%c", &c1) == 1){

27 position +=1;

28 if(c1>='0' && c1<='9'){

29 nr = atoi(&c1);

30 fre[nr]++;

31

32 }

33 if(found == 0 && c1 == p->c){

34 found=1;

35 pthread\_mutex\_lock(&m2);

36 pos[p->position] = position;

37 pthread\_mutex\_unlock(&m2);

38 }

39 }

40

41 if(found==0){

42 pthread\_mutex\_lock(&m2);

43 pos[p->position] = -1;

44 pthread\_mutex\_unlock(&m2);

45 }

46

47

48 for( i =0;i<=9;i++){

49 pthread\_mutex\_lock(&m);

50 frec[i]+= fre[i];

51 pthread\_mutex\_unlock(&m);

52 }

53

54 free(a);

55 return NULL;

56 }

57

58

59 int main(int argc, char\*\* argv){

60

61 pthread\_t t = (pthread\_t)malloc(sizeof(pthread\_t)\*(argc-1)/2);

62 pos = (int\*)malloc(sizeof(int)\* (argc-1)/2);

63 int i, nrp, n;

64 PAIR \*p;

65

66 pthread\_mutex\_init(&m, NULL);

67 pthread\_mutex\_init(&m2, NULL);

68 n=(argc-1)/2;

69 for(i=1, nrp=0; nrp<n;i=i+2, nrp++){

70 // printf("%s %c\n", argv[i], argv[i+1][0]);

71 p = (PAIR\*)malloc(sizeof(PAIR));

72 strcpy(p->filename, argv[i]);

73 p->c = argv[i+1][0];

74 //printf("%s %c\n",p->filename,p->c);

75 p->position = nrp;

76 pthread\_create(&t[nrp], NULL,f, (void\*)p);

77

78

79 }

80

81

82

83 for(i=0;i<n;i++)

84 {

85 pthread\_join(t[i], NULL);

86 }

87

88 printf("Vector de frecventa: ");

89 for(i=0;i<=9;i++){

90 printf(" %d ", frec[i]);

91 }

92 printf("\n");

93

94 printf("Positions of the first character: ");

95 for(i=0;i<(argc-1)/2;i++){

96 printf(" %d ", pos[i]);

97 }

98 printf("\n");

99

100 free(t);

101 pthread\_mutex\_destroy(&m);

102 pthread\_mutex\_destroy(&m2);

103 return 0;

104 }