**x. Write a C program that creates N threads, where N is a command line argument.**

**- Each thread receives an index number as an argument;**

**- Each thread will generate a random integer A between 1 and 10 and will add it to a sum variable shared between all threads.**

**- After all threads have added their A number to the sum, each thread will generate another integer B, this time between 1 and 15**

**- Each thread will subtract their B number from the sum, only if the sum is currently greater or equal than B**

**- Each thread will print its index, both generated numbers and whether or not they were able to subtract B from the sum.**

**- After all threads terminate, the main program will print the value of the sum variable.**

1 #include <pthread.h>

2 #include <stdlib.h>

3 #include <stdio.h>

4

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

6 pthread\_barrier\_t barrier;

7

8 int sum;

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

10 int index = \*(int\*) a;

11 int A = rand()%10+1;

12 pthread\_mutex\_lock(&mtx);

13 sum+=A;

14 pthread\_mutex\_unlock(&mtx);

15 pthread\_barrier\_wait(&barrier);

16

17 int B = rand()%15+1;

18 if(sum>=B){

19 pthread\_mutex\_lock(&mtx);

20 sum-=B;

21 pthread\_mutex\_unlock(&mtx);

22 printf("Index:%d, A:%d, B:%d, you can subtract\n", i ndex, A, B);

23 }

24 else

25 printf("Index:%d, A:%d, B:%d, you cannot subtract\n" , index, A, B);

26 return NULL;

27 }

28

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

30 int N = atoi(argv[1]);

31 pthread\_t t[N];

32 pthread\_barrier\_init(&barrier, NULL, N);

33 int i;

34 int tin[N];

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

36 tin[i]=i;

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

38 }

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

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

41 }

42 printf("The total of the sum: %d", sum);

43 pthread\_mutex\_destroy(&mtx);

44 return 0;

45 }

46

**X. Write a C program that creates a child process. The parent process generates a random number N between 10 and 30 and creates a string of length N.The string will be initialized with the character ‘a’ on each position. The parent then sends the string to the child process which, for every character in the string, generates a random number between 0 and 24 and adds it to that character’s ASCII code. The child will send the modified string back to the parent. The parent should display the generated number and the string received from the child.**

1 #include <stdlib.h>

2 #include <stdio.h>

3 #include <unistd.h>

4 #include <sys/types.h>

5 #include <sys/wait.h>

6

7

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

9

10 int p2c[2], c2p[2];

11 pipe(p2c); pipe(c2p);

12 int p=fork();

13

14 if(p<0){perror("Fork not possible!\n");return 1;}

15 if(p==0){//child

16 close(p2c[1]); close(c2p[0]);

17 int n=0;

18 read(p2c[0], &n, sizeof(int));

19 char string[n];

20 read(p2c[0], string, sizeof(char)\*(n+1));

21

22 int i;

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

24 int nr = rand()%25;

25 string[i]+=nr;

26 }

27

28 write(c2p[1], string, sizeof(char)\*(n+1));

29

30 close(p2c[0]); close(c2p[1]);

31 exit(0);

32 } //parent

33 close(p2c[0]); close(c2p[1]);

34

35 int N = rand()%21+10;

36 char string[N+1];

37 int i;

38 for(i=0;i<N;i++)

39 string[i]='a';

40 string[N]='\0';

41

42 write(p2c[1], &N, sizeof(int));

43 write(p2c[1], string, sizeof(char)\*(N+1));

44

45 read(c2p[0], string, sizeof(char)\*(N+1));

46 printf("The final string:%s \n", string);

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

48 int nr = string[i]-'a';

49 printf("For character %c: %d\n", string[i], nr);

50 }

51

52

53 close(p2c[1]); close(c2p[0]);

54 wait(0);

55 return 0;

56 }

**X. Write a C program that creates N threads and one child process (N given as a command line argument). Each thread will receive a unique id from the parent. Each thread will generate two random numbers between 1 and 100 and will print them together with its own id. The threads will send their generated numbers to the child process via pipe or FIFO. The child process will calculate the average of each pair of numbers received from a thread and will print it alongside the thread id. Use efficient synchronization.**

#include <pthread.h>

2 #include <stdio.h>

3 #include <stdlib.h>

4

5 int p2c[2], c2p[2];

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

7

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

9 int id = \*(int\*) a;

10 int nr1 = rand()%100+1;

11 int nr2 = rand()%100+1;

12

13 pthread\_mutex\_lock(&mtx);

14 if(0>write(p2c[1], &id, sizeof(int))){

15 perror("Error writing\n");

16 }

17 if(0>write(p2c[1], &nr1, sizeof(int))){

18 perror("Error writing\n");

19 }

20 if(0>write(p2c[1], &nr2, sizeof(int))){

21 perror("Error writing\n");

22 }

23 pthread\_mutex\_unlock(&mtx);

24

25 return NULL;

26 }

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

28 pipe(p2c);pipe(c2p);

29 int p=fork();

30 int N = atoi(argv[1]);

31

32 if(p<0){perror("Fork not possible;\n"); return 1;}

33 if(p==0){ // child

34 close(p2c[1]); close(c2p[0]);

35 int i;

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

37 int id, nr1, nr2;

38 if(0>read(p2c[0],&id,sizeof(int))){

39 perror("Error reading\n");

40 }

41 if(0>read(p2c[0],&nr1,sizeof(int))){

42 perror("Error reading\n");

43 }

44 if(0>read(p2c[0],&nr2,sizeof(int))){

45 perror("Error reading\n");

46 }

47 printf("Id:%d Average:%f \n", id, (double)(nr1+nr2)/ 2);

48 }

49 close(p2c[0]); close(c2p[1]);

50 exit(0);

51 }// child

52

53 close(p2c[0]); close(c2p[1]);

54 pthread\_t t[N];

55 int i;

56 int tin[N];

57

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

59 tin[i]=i;

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

61 }

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

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

64 }

65 close(p2c[1]); close(c2p[0]);

66 pthread\_mutex\_destroy(&mtx);

67 wait(0);

68 return 0;

69 }

**X. Write a C program that receives any number of strings as command line arguments. The program creates two child processes, which inherit the parent's command line arguments (ie. no need to send the arguments via pipe/fifo to the children for this problem). Each child process creates a thread for each of the command line arguments. Each thread created by the first child will extract the vowels from its argument and will append them to a string shared among the threads. Each thread created by the second child process will extract the digits from its argument and will add them to a sum shared among the threads. Both child processes wait for their respective threads to finish and send the result to the parent via pipe. The parent displays the results.**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

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

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

char vocaleConcatenare[500];

int lenVocale=0;

int sum;

void\* f(void\* a){

char\* word = (char\*) a;

char vocale[]="aeiouAEIOU";

int i;

int N = strlen(word);

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

if(strchr(vocale, word[i])){

pthread\_mutex\_lock(&mtx);

vocaleConcatenare[lenVocale]=word[i];

lenVocale++;

pthread\_mutex\_unlock(&mtx);

}

}

return NULL;

}

void\* f2(void\* a){

char\* word = (char\*) a;

char digits[]="123456789";

int i;

int N = strlen(word);

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

if(strchr(digits, word[i])){

pthread\_mutex\_lock(&mtx2);

sum+=word[i]-'0';

pthread\_mutex\_unlock(&mtx2);

}

}

return NULL;

}

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

int p1, p2, i;

int N=argc-1;

int p2a[2], a2p[2], p2b[2], b2p[2];

pipe(p2a); pipe(a2p);

p1=fork();

if(p1<0){perror("Fork1 impossible\n");return 1;}

if(p1==0){ //child

close(a2p[0]); close(p2a[1]);

pthread\_t t[N];

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

pthread\_create(&t[i], NULL, f, (void\*)argv[i+1]);

}

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

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

}

write(a2p[1], &lenVocale, sizeof(int));

write(a2p[1], vocaleConcatenare, sizeof(char)\*lenVocale);

close(a2p[1]); close(p2a[0]);

exit(0);

} //parent

close(a2p[1]); close(p2a[0]);

pthread\_mutex\_destroy(&mtx);

wait(0); //finished first child

char result[500];

int len=0;

read(a2p[0], &len, sizeof(int));

read(a2p[0], result, sizeof(char)\*len);

close(a2p[0]); close(p2a[1]);

// start second child

pipe(p2b); pipe(b2p);

p2=fork();

if(p2<0){perror("Fork2 impossible\n");return 1;}

if(p2==0){ //child

close(b2p[0]); close(p2b[1]);

pthread\_t t[N];

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

pthread\_create(&t[i], NULL, f2, (void\*)argv[i+1]);

}

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

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

}

write(a2p[1], &sum, sizeof(int));

close(b2p[1]); close(p2b[0]);

exit(0);

} //parent

close(b2p[1]); close(p2b[0]);

wait(0); //finished second childe

int sum=0;

read(b2p[0], &sum, sizeof(int));

pthread\_mutex\_destroy(&mtx2);

close(b2p[0]); close(p2b[1]);

printf("Sum of digits:%d \n", sum);

printf("String of vowels:%s \n", result);

return 0;

}