[Problem 11 :](#problem11)

/\*11. Write a program that receives strings of characters as command line arguments.For each string the program creates a thread which calculates the number of digits, the number of leters and the number of special characters (anything other than a letter or digit).The main program waits for the threads to terminate and prints the total results(total number of digits, letters and special characters across all the received command line arguments) and terminates.Use efficient synchronization. Do not use global variables\*/

[Problem 12 :](#problem12)

/\*12. Write a C program that receives integers as command line argument. The program will keep a frequency vector for all digits. The program will create a thread for each argument that counts the number of occurences of each digit and adds the result to the frequency vector. Use efficient synchronization.\*/

[Problem 13:](#problem13)

/\*13. Write a C program that reads a number N and creates 2 threads. One of the threads will generate an even number and will append it to an array that is passed as a parameter to the thread. The other thread will do the same, but using odd numbers. Implement a synchronization between the two threads so that they alternate in appending numbers to the array, until they reach the maximum length N.\*/

[Problem 14 :](#problem14)

/\*14. Create a C program that converts all lowecase letters from the command line arguments to uppercase letters and prints the result. Use a thread for each given argument.\*/

[Problem 16 :](#problem16)

/\* Write a C program that takes as command line arguments 2 numbers: N and M. The program will simulate a thread race that have to pass through M checkpoints. Through each checkpoint the threads must pass one at a time (no 2 threads can be inside the same checkpoint). Each thread that enters a checkpoint will wait between 100 and 200 milliseconds (usleep(100000) makes a thread or process wait for 100 milliseconds) and will print a message indicating the thread number and the checkpoint number, then it will exit the checkpoint.\*/

Problem 11 :

#include <pthread.h>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

typedef struct{

pthread\_mutex\_t\* mtxs;

int\* countD;

int\* countC;

int\* countS;

char\* str;

} data;

void\* f(void\* a) {

data d = \*((data\*)a);

int len = strlen(d.str);

int cc = 0, dc = 0, sc = 0;

int i;

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

/\*is digit\*/

if(d.str[i] >= '0' && d.str[i] <= '9') {

dc += 1;

}

/\*is character\*/

else if((d.str[i] >= 'a' && d.str[i] <= 'z') || (d.str[i] >= 'A' && d.str[i] <= 'Z'))

cc += 1;

/\*is special character\*/

else

sc += 1;

}

if(dc > 0) {

pthread\_mutex\_lock(&d.mtxs[0]);

\*(d.countD) += dc;

pthread\_mutex\_unlock(&d.mtxs[0]);

}

if(cc > 0) {

pthread\_mutex\_lock(&d.mtxs[1]);

\*(d.countC) += cc;

pthread\_mutex\_unlock(&d.mtxs[1]);

}

if(sc > 0) {

pthread\_mutex\_lock(&d.mtxs[2]);

\*(d.countS) += sc;

pthread\_mutex\_unlock(&d.mtxs[2]);

}

return NULL;

}

void initData(data\* d, pthread\_mutex\_t\* given\_mtxs, int\* given\_countD, int\* given\_countC, int\* given\_countS, char\* given\_str) {

d->mtxs = given\_mtxs;

d->countD = given\_countD;

d->countC = given\_countC;

d->countS = given\_countS;

d->str = given\_str;

}

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

if (argc < 2) {

printf("At least 1 arguments is to be provided");

exit(1);

}

int i;

int N = argc - 1;

pthread\_t T[N];

pthread\_mutex\_t\* mtxs = (pthread\_mutex\_t\*)malloc(sizeof(pthread\_mutex\_t) \* 3);

data\* arg = (data\*)malloc(sizeof(data) \* N);;

int dc = 0, cc = 0, sc = 0;

for(i = 0; i < 3; i++)

pthread\_mutex\_init(&mtxs[i], NULL);

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

initData(&arg[i], mtxs, &dc, &cc, &sc, argv[i + 1]);

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

pthread\_create(&T[i], NULL, f, (void\*)&arg[i]);

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

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

printf("Digit count: %d \nCharacters count: %d \nSpecial Characters count: %d \n", dc, cc, sc);

for(i = 0; i < 3; i++) {

pthread\_mutex\_destroy(&mtxs[i]);

}

free(arg);

free(mtxs);

return 0;

}

Problem 12 :

#include <pthread.h>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

typedef struct{

pthread\_mutex\_t\* mtx;

int\* freq;

int info;

} data;

void\* f(void\* a) {

data d = \*((data\*)a);

int digit;

int aux = d.info;

while(aux > 0) {

digit = aux % 10;

pthread\_mutex\_lock(d.mtx);

d.freq[digit] += 1;

pthread\_mutex\_unlock(d.mtx);

aux = aux / 10;

}

return NULL;

}

void initData(data\* d, pthread\_mutex\_t\* given\_mtx, int\* given\_freq, int given\_info) {

d->mtx = given\_mtx;

d->freq = given\_freq;

d->info = given\_info;

}

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

int N = argc - 1;

int i;

pthread\_t T[N];

pthread\_mutex\_t mtx;

int\* freq = (int\*)malloc(sizeof(int) \* 10);

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

freq[i] = 0;

data\* arg = (data\*)malloc(sizeof(data) \* N);

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

initData(&arg[i], &mtx, freq, atoi(argv[i+1]));

pthread\_mutex\_init(&mtx, NULL);

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

pthread\_create(&T[i], NULL, f, (void\*)&arg[i]);

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

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

printf("Frequency array\n");

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

printf("[%d]: %d\n", i, freq[i]);

pthread\_mutex\_destroy(&mtx);

free(freq);

free(arg);

return 0;

}

Problem 13 :

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <pthread.h>

#include <time.h>

typedef struct {

pthread\_mutex\_t\* mtx;

pthread\_cond\_t\* cond;

int\* array;

int\* index;

int n;

} data;

void initData(data\* d, pthread\_mutex\_t\* given\_mtx, pthread\_cond\_t\* given\_cond, int\* given\_array, int\* given\_index, int given\_n) {

d->mtx = given\_mtx;

d->cond = given\_cond;

d->array = given\_array;

d->index = given\_index;

d->n = given\_n;

}

/\* generate even numbers on even position\*/

void\* f1(void\* a) {

data d = \*((data\*)a);

int i;

pthread\_mutex\_lock(d.mtx);

if(\*(d.index) % 2 == 1)

pthread\_cond\_wait(d.cond, d.mtx);

while(\*(d.index) < d.n) {

int nr = (random() % 51) \* 2;

d.array[\*(d.index)] = nr;

\*(d.index) += 1;

printf("T1: ");

for(i = 0; i < \*(d.index); i++)

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

printf("\n");

pthread\_cond\_signal(d.cond);

if(\*(d.index) % 2 == 1)

pthread\_cond\_wait(d.cond, d.mtx);

}

pthread\_cond\_signal(d.cond);

pthread\_mutex\_unlock(d.mtx);

return NULL;

}

/\* generate odd numbers on odd position\*/

void\* f2(void\* a) {

data d = \*((data\*)a);

int i;

pthread\_mutex\_lock(d.mtx);

if(\*(d.index) % 2 == 0)

pthread\_cond\_wait(d.cond, d.mtx);

while(\*(d.index) < d.n) {

int nr = (random() % 51) \* 2 + 1;

d.array[\*(d.index)] = nr;

\*(d.index) += 1;

printf("T2: ");

for(i = 0; i < \*(d.index); i++)

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

printf("\n");

pthread\_cond\_signal(d.cond);

if(\*(d.index) % 2 == 0)

pthread\_cond\_wait(d.cond, d.mtx);

}

pthread\_cond\_signal(d.cond);

pthread\_mutex\_unlock(d.mtx);

return NULL;

}

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

if (argc != 2) {

printf("1 command line argument is to be provided!\n");

exit(1);

}

int N = atoi(argv[1]);

int\* array = (int\*)malloc(sizeof(int) \* N);

int index = 0;

pthread\_mutex\_t mtx;

pthread\_cond\_t cond;

pthread\_mutex\_init(&mtx, NULL);

pthread\_cond\_init(&cond, NULL);

pthread\_t T[2];

data arg[2];

initData(&arg[0], &mtx, &cond, array, &index, N);

initData(&arg[1], &mtx, &cond, array, &index, N);

pthread\_create(&T[0], NULL, f1, (void\*)&arg[0]);

pthread\_create(&T[1], NULL, f2, (void\*)&arg[1]);

pthread\_join(T[0], NULL);

pthread\_join(T[1], NULL);

pthread\_mutex\_destroy(&mtx);

pthread\_cond\_destroy(&cond);

free(array);

return 0;

}

Problem 14 :

#include <pthread.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

int N;

void\* f(void\* a) {

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

int len = strlen(text);

int i;

for(i = 0; i < len; i++)

if(text[i] >= 'a' && text[i] <= 'z')

text[i] += 'A' - 'a';

printf("Thread finished the word: %s\n", text);

return NULL;

}

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

int i;

N = argc;

pthread\_t t[N];

for(i = 1; i < N; i++)

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

for(i = 1; i < N; i++)

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

return 0;

}

Problem 16 :

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <pthread.h>

#include <time.h>

int M;

int N;

pthread\_mutex\_t\* mtxs;

pthread\_barrier\_t bar;

void\* race(void\* a) {

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

int i;

pthread\_barrier\_wait(&bar);

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

pthread\_mutex\_lock(&mtxs[i]);

printf("Thread %d at checkpoint %d\n", id, i);

/\*sleeping for a random amount of time between 100 and 200 milliseconds\*/

int n = (random() % 101 + 100) \* 1000;

usleep(n);

pthread\_mutex\_unlock(&mtxs[i]);

}

printf("Thread %d at finished\n", id);

return NULL;

}

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

if (argc != 3) {

printf("2 arguments are to be provided\n");

exit(1);

}

N = atoi(argv[1]);

M = atoi(argv[2]);

int ids[N];

pthread\_t racers[N];

mtxs = malloc(sizeof(pthread\_mutex\_t) \* M);

pthread\_barrier\_init(&bar, NULL, N);

int i;

/\*initializing the mutexes\*/

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

pthread\_mutex\_init(&mtxs[i], NULL);

}

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

ids[i] = i;

pthread\_create(&racers[i], NULL, race, &ids[i]);

}

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

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

}

/\*destroying the mutexes\*/

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

pthread\_mutex\_destroy(&mtxs[i]);

}

pthread\_barrier\_destroy(&bar);

free(mtxs);

return 0;

}