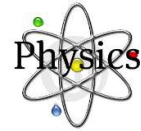





ΕΘΝΙΚΟ ΚΑΙ
ΚΑΠΟΔΙΣΤΡΙΑΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ



A.M: 201100079	Επώνυμο: Κορομηλάς	Όνομα: Χρήστος
-----------------------	---------------------------	-----------------------

POSIX THREADS



pthreads

➤ ΠΕΡΙΓΡΑΦΗ ΤΟΥ PROJECT

Στο συγκεκριμένο project φτιάχτηκε πίνακας με $A(1) = 1, A(2) = 2, \dots A(10^6) = 10^6$ και από αυτό τον πίνακα υπολογίστηκε ο πίνακας $B[i] = (A[i])^2$, με την χρήση 1 και 4 threads.

Για τη σύγκριση μεταξύ του 1 και 4 threads χρησιμοποιήθηκε **time counter** και για να βγει καλύτερη ακρίβεια υπολογίστηκε για 1000 φορές ο χρόνος εκτέλεσης.

Στη συνέχεια για να ελεγχθεί ο παραλληλισμός τυπώθηκαν χαρακτήρες για το κάθε thread για τα πρώτα 40 στοιχεία, αλλά προκειμένου να φανεί ο παραλληλισμός εισάχθηκαν και παραπάνω χαρακτήρες.

Επίσης ο κώδικας φτιάχτηκε για τη συγκεκριμένη εκφώνηση και μόνο. Ενώ θα μπορούσαμε να χρησιμοποιήσουμε περισσότερες συναρτήσεις με 1,4 threads η και παραπάνω και επίσης να ελέγχεται η εκτέλεση για μικρούς πίνακες. Καθώς πχ η διαίρεση για μέγεθος του πίνακα 10 με τα αντίστοιχα threads 4 μας δίνει 2 και άρα για integer με αποτέλεσμα να χάνουμε 2 στοιχεία.

➤ ΑΞΙΟΠΟΙΗΣΗ ΤΩΝ THREADS

Η εκτέλεση προγραμμάτων με threads δίνει τη δυνατότητα να εκτελεστούν υπολογισμοί και προγράμματα με ταχύτερο τρόπο. Όμως έχουν ένα τεράστιο πρόβλημα το λεγόμενο **race condition**. Οπότε πρέπει να γίνει ανάλογη και σωστή χρήση.

Έτσι έγινε χρήση των threads μοιράζοντας το διάστημα των υπολογισμών σε 4 ίσα μέρη για 4 threads και κανονικά όλο το εύρος για 1 thread.

Και για κάθε thread βάζουμε σαν input στη συνάρτηση που θα τρέξει την αρχή τους εύρους.

Πχ $a = 0, a = 250.000, a = 500.000$ και $a = 750.000$

Ωστε

ForLoop($k = a; i < a + (\text{array_size})/(\text{number_of_threads}); k++$)

$A[k] = k + 1;$

ForLoop($k = a; i < a + (\text{array_size})/(\text{number_of_threads}); k++$)

$B[k] = A[k] * A[k];$

Οπότε κάθε thread τρέχει σε ανεξάρτητο χώρο χωρίς να υπάρχει επικάλυψη στοιχείων στη καταχώρηση.

➤ ΑΠΟΤΕΛΕΣΜΑ ΕΚΤΕΛΕΣΗΣ

```

This program is going to compute the: B=(A[i])^2 of an array A with 10^6 elements.
First we are going to create the array A[i]=i.
Input the number of threads you are going to use.
Input 1 or 4: 1
So you will use: 1 threads.

Now we are going to compute A[i].
Thread computing is going to start with time counter.

__ Counter started __
__ Counter stopped __

Total Elapsed Time: 7.003600 ms.

Now we are going to compute B[i].
Thread computing is going to start with time counter

__ Counter started __
__ Counter stopped __

Total Elapsed Time: 7.997300 ms

Process returned 0 (0x0)   execution time : 1.324 s
Press any key to continue.
_

```

```

This program is going to compute the: B=(A[i])^2 of an array A with 10^6 element
s.
First we are going to create the array A[i]=i.
Input the number of threads you are going to use.
Input 1 or 4: 4
So you will use: 4 threads.

Now we are going to compute A[i].
Thread computing is going to start with time counter.

__ Counter started __
))))))))))))))))))))))))))))))))))))))))))+++++
*****
__ Counter stopped __

Total Elapsed Time: 9.000300 ms.

Now we are going to compute B[i].
Thread computing is going to start with time counter

__ Counter started __
))))))))))))))))))))))))))))))))))))))))),,,,,,+++++
+++++))))))*****
__ Counter stopped __

Total Elapsed Time: 9.999000 ms

Process returned 0 (0x0)   execution time : 0.880 s
Press any key to continue.

```

Προηγουμένως φαίνεται ξεκάθαρα ο παραλληλισμός με το τρόπο αυτό που φτιάχτηκε το πρόγραμμα αλλά τυπώνοντας αντί για 40, 1000 στοιχεία βλέπουμε το εξής:

[illegible]

Τώρα φαίνεται καλύτερα ο παραλληλισμός.

Επίσης Για μεγαλύτερη ακρίβεια των χρόνων χωρίς τις print των στοιχείων:

	Elements: 10 ⁶				Elements: 10 ⁷			
	A		B		A		B	
	1thread	4thread	1thread	4thread	1thread	4thread	1thread	4thread
Mean time (ms)	6,115698	2,159298	6,048686	2,0267	63,06699	17,954	63,53806	17,476
1thread/4thread	2,832262		2,984499		3,512699		3,635733	

Εδώ βλέπουμε το χρόνο εκτέλεσης και σύγκριση μεταξύ 1 και 4 threads. Ότι με 4 threads μειώνεται ο χρόνος σε σχέση με του 1 thread. Επίσης όσο αυξάνουμε τα στοιχεία τόσο αυξάνεται και ο χρόνος που κερδίζουμε.

```

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>

//Two 1-dimensional arrays of 10^6 numbers A(1)=1, A(2)=2, ... A(10^6)=10^6 .
//Compute the B(i)=square(A(i)) in: a) single thread b) four threads.
//Check the difference in the execution time.

#define ARRAY_SIZE 1000000          //the size of your array that you are going to use
#define BILLION 1000000000.0        // billion is using for time counter

int number_of_threads;              // global threads
double A[ARRAY_SIZE], B[ARRAY_SIZE]; // global arrays

void* creatingarray(void* arg)       //function that creates array A
{
    int index = *(int*)arg;
    for (int k=index; k<index+ARRAY_SIZE/number_of_threads; k++)
    {
        A[k]=(k+1);
        if (number_of_threads!=1 && k<index+40) //for printing the first 40 characters
            printf("%c",41+(index*number_of_threads/ARRAY_SIZE));
        //printf("%d\t",index);
        //printf("%13.0f\t",A[k]);
        //printf("%d\n",k+1);
    }
}

void* squaring(void* arg)            //function that computes array B[i]=(A[i])^2
{
    int index = *(int*)arg;
    for (int k=index; k<index+ARRAY_SIZE/number_of_threads; k++)
    {
        B[k]=(A[k])*A[k];
        if (number_of_threads!=1 && k<index+40) //for printing the first 40 characters
            printf("%c",41+(index*number_of_threads/ARRAY_SIZE));
        //printf("%d\t",index);
        //printf("%13.0f\t",B[k]);
        //printf("%d\n",k+1);
    }
}

```

```

int main(int argc, char* argv[])
{
    printf("This program is going to compute the: B=(A[i])^2 of an array A with 10^6 elements.\n");
    printf("First we are going to create the array A[i]=i.\n");
    printf("Input the number of threads you are going to use.");
    do
    {
        printf("\nInput 1 or 4: ");
        scanf("%d",&number_of_threads);

        } while(!((number_of_threads==1) || (number_of_threads==4)));
//Input number 1 for 1 thread or 4 for 4 threads
    int i;
    struct timespec start, start2, end,end2;          // for time counter
    printf("So you will use: %d threads.\n",number_of_threads);

    printf("\nNow we are going to compute A[i].\n");
    pthread_t th[number_of_threads];                //Creating an array of threads
    printf("Thread computing is going to start with time counter.\n");

    printf("\n___ Counter started ___\n");
    clock_gettime(CLOCK_REALTIME, &start);          //Counter here is starting
    for (i=0; i<number_of_threads; i++)
    {
        int* a=malloc(sizeof(int));                  // a helps to divide the work of computing into 4 equal
computational pieces
        *a = i*(ARRAY_SIZE/number_of_threads);
//checking if something is going wrong when pthread create
        if(pthread_create(th+i, NULL, &creatingarray, a) !=0)
        {
            printf("\n%d",i);
            perror("Failed to create thread!");
            return i+1;
        }
        }//else printf("\nStarted Succesfully!\n");
    }
for (i=0; i<number_of_threads; i++)
{
//checking if something is going wrong when pthread join
    if(pthread_join(th[i], NULL) !=0)
    {
        perror("Failed to join thread!");
        return i+1;
    }//else printf("\nFinished Succesfully!\n");
}
}

```



```
clock_gettime(CLOCK_REALTIME, &end);           //Counter here stops
printf("\n___ Counter stopped ___\n");
double time_spent = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec)/BILLION;
printf("\nTotal Elapsed Time: %.6f ms.\n", time_spent*1000.0);
        // next we are using the same logic for computing B[i]=(A[i])^2
        // we could use 4 threads for all computing more faster
        // with half code, but for educational use we did it with 2 times of using threads
printf("\nNow we are going to compute B[i].\n");
pthread_t th2[number_of_threads];
printf("Thread computing is going to start with time counter\n");
printf("\n___ Counter started ___\n");

clock_gettime(CLOCK_REALTIME, &start2);
for (i=0; i<number_of_threads; i++)
{
    int* a=malloc(sizeof(int));
    *a = i*(ARRAY_SIZE/number_of_threads);
    if(pthread_create(th2+i, NULL, &squaring, a) !=0)
    {
        perror("Failed to create thread!");
        return i+1;
    }//else printf("Started Successfully!\n");
}
for (i=0; i<number_of_threads; i++)
{
    if(pthread_join(th2[i], NULL) !=0)
    {
        perror("Failed to join thread!");
        return i+4;
    }//else printf("Finished Successfully!\n");
}
clock_gettime(CLOCK_REALTIME, &end2);
printf("\n___ Counter stopped ___\n");
time_spent = (end2.tv_sec - start2.tv_sec) + (end2.tv_nsec - start2.tv_nsec)/BILLION;
printf("\nTotal Elapsed Time: %.6f ms.\n", time_spent*1000.0);

return 0;
}
```