



# Homework 1 – Ταυτόχρονος Προγραμματισμός

Γαρυφαλλιά Αναστασία Παπαδούλη | 3533

Δημήτριος Τσαλαπατάς | 3246

Νικόλαος Μπέτσος | 3267

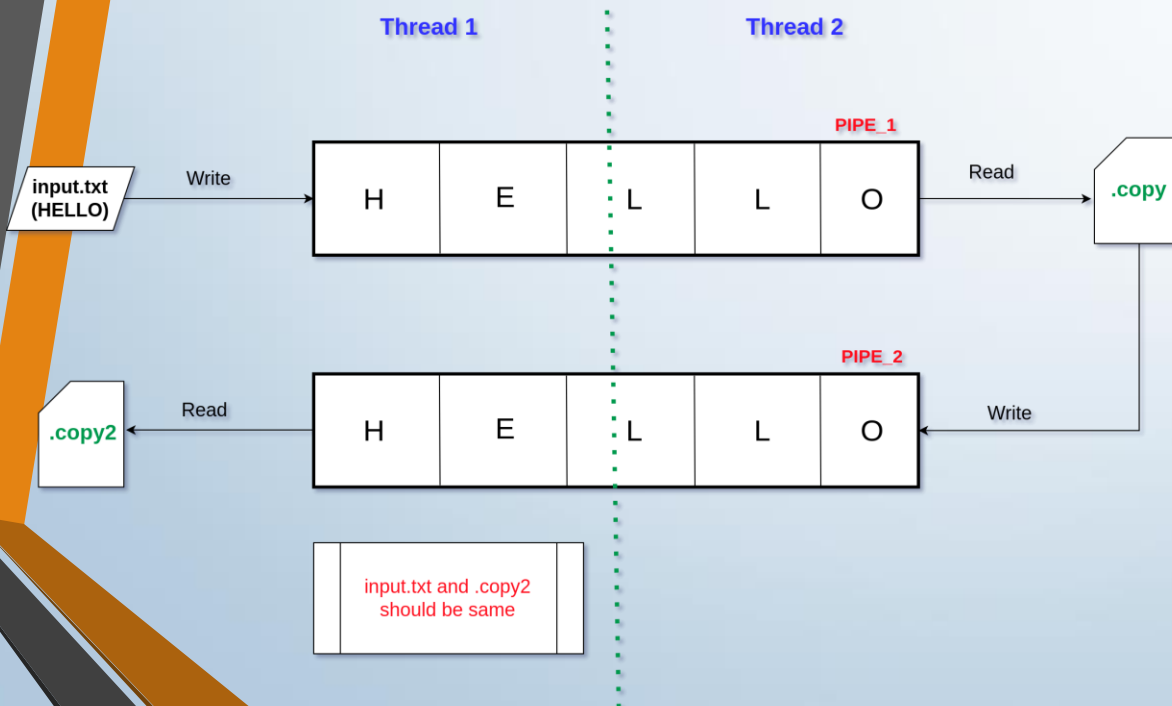
Ιωάννης Ρείνος | 3390

Ομάδα 4

# Άσκηση 1

- Hw1\_1.h
- Hw1\_1.c

- Hw1\_1\_lib.h
- Hw1\_1\_lib.c



```
typedef struct pipe_info {  
    int size;  
    int write;  
    int read;  
    char *pipe;  
} pipe_info;
```

Χρησιμοποιούμε έναν global πίνακα από το εξής struct:

**Size:** μέγεθος pipe

**Write:** Ο δείκτης από πού γράφει

**Read:** Ο δείκτης από πού διαβάζει

**Pipe:** ποιο pipe αφορούν τα παραπάνω

Στο lib.c κομμάτι του hw1 υλοποιούνται οι λειτουργίες που ζητούνται από την εκφώνηση για την δημιουργία και τον χειρισμό των pipes, καθώς και μία βοηθητική συνάρτηση που κάνει initialize το παραπάνω struct.

2 thread  
functions  
για τα δύο  
threads:

```
main:
    open_pipes;
    finish = 0; -> thread sync: notif that 2nd thread is done reading
    exit_main = 0; -> notification from threads that program is rdy to shut down
    read input;
    create thread(thread_func_1);
    create thread(thread_func_2);
    start_write = 1;

    while (!exit_main);

    return
```


```
thread_func_1:
    wait from notif from main to start writing (start_write)
    for (i: input size)
        check if there is available space
        pipe_write(0)
    write_done
    wait for thread2 to stop reading (finish)
    while (pipe1 open for reading)
        pipe_read(1)
    open & write .copy2
    exit_main = 1 -> notify main to finish
    exit
```

```
thread_func_2:
    while (pipe0 open for reading)
        pipe_read(0)
    notify that it stopped reading (finish)
    for (i: copy size)
        check if there is available space
        pipe_write(1)
    write_done
    exit
```


# Άσκηση 2

Πίνακας από struct του εξής τύπου:

```
struct worker
{
    int number;
    int status;
    int size;
    int *result[2];
};
```



Status	Value
Terminated	-2
To be terminated	-1
Busy	0
Available	1



Prime   not	Number
0	3
1	4
...	...

Η main αρχικοποιεί έναν πίνακα μεγέθους num\_workers. Όταν δημιουργούμε ένα thread, λαμβάνει ως όρισμα, pointer στο struct που του αναλογεί.

- Η main διαβάζει input και δίνει δουλειά όταν βρει ελεύθερο worker.
- Ο worker ελέγχει αν πρέπει να κλείσει και αν έχει δουλειά καλεί την find prime.

```
main
  initialize num_thread threads
do
  read input
  while(all workers busy);
  give work
  make worker busy (status = 0)
while (input > 0)

wait for work to be done
notify all to terminate
when all workers terminate:
  return
```

```
worker_thread:
  while(status != to be terminated);
  while(available);
  if (status = busy)
    find prime;
    append the array of results by one and save new num;
    status = available;
  status = terminated;
  exit
```

# Καταμέτρηση απόδοσης

- Ο επεξεργαστής που χρησιμοποιήθηκε έχει 8 πυρήνες και 16 νήματα (Ryzen 7 4800H)
- Δημιουργήσαμε ένα bash script, το οποίο τρέχει το πρόγραμμα δίνοντας από 1 μέχρι 30 threads
- Το script τρέχει 10 φορές το πρόγραμμα για κάθε thread-count για να βγει μέσος όρος από execution time

```
# Set the path to your hw1_2 program
HW1_2_PATH="./hw1_2"

# Input file
INPUT_FILE="primes.txt"

# Array of thread values to test
THREADS=(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30)

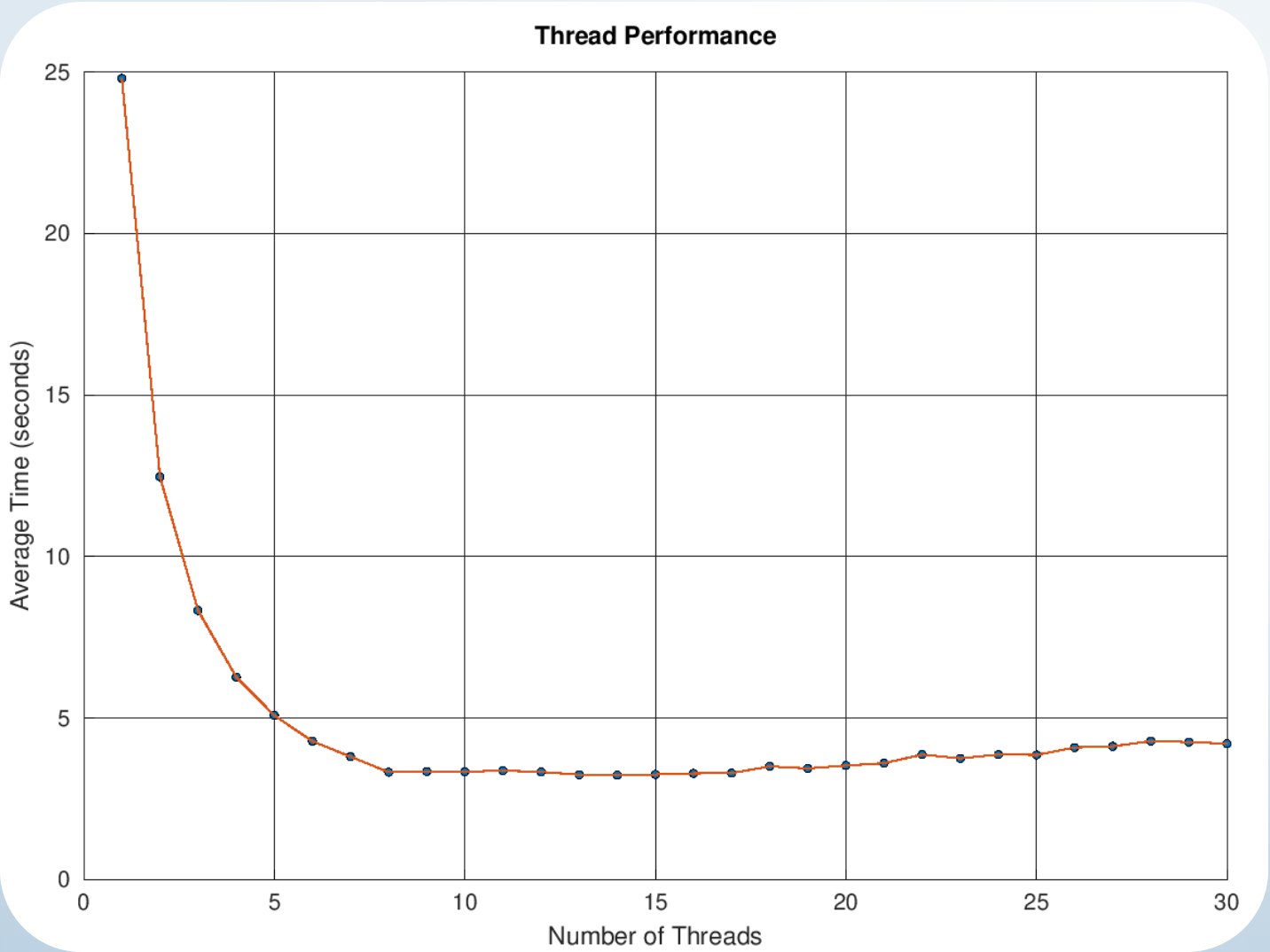
# Number of times to run the program for each thread count
RUNS=10

# Function to calculate average time
calculate_average_time() {
    local total_time=0
    for ((i = 1; i <= $RUNS; i++)); do
        local current_time=$(/usr/bin/time -p $HW1_2_PATH "$1" < "$INPUT_FILE" 2>&1 | grep real | awk '{print $2}')
        total_time=$(python3 -c "print('{:.3f}'.format($total_time + $current_time))")
    done
    local average_time=$(python3 -c "print('{:.3f}'.format($total_time / $RUNS))")
    echo $average_time
}

# Loop through each thread value and run the program
for thread_count in "${THREADS[@]}; do
    echo "Running with $thread_count threads:"
    average_time=$(calculate_average_time $thread_count)
    echo "Average time: $average_time seconds"
    echo "-----"
done
```

# Ανάλυση Απόδοσης

Threads_num	Seconds
1	24.800
2	12.471
3	8.332
4	6.269
5	5.082
6	4.284
7	3.805
8	3.326
9	3.348
10	3.340
11	3.372
12	3.344
13	3.246
14	3.236
15	3.253



Threads_num	Seconds
16	3.289
17	3.297
18	3.504
19	3.444
20	3.531
21	3.603
22	3.869
23	3.753
24	3.869
25	3.860
26	4.090
27	4.125
28	4.286
29	4.260
30	4.211

# Άσκηση 3

```
struct arr_positions
{
    int left, right, finish;
};
```

Left: αριστερό άκρο κομματιού πίνακα

Right: δεξί άκρο κομματιού πίνακα


Finish: 1 όταν επιστρέφει η αναδρομή (τέλος του sort)

```
thread_func:
    if (array size <= 64)
        selectionSort
    else
        allocate space for new array structs;
        divide array in half;
        create 2 new threads with half the array and thread_func;
        wait for threads to finish;
        merge the 2 sorted subarrays;
    finish = 1;
    exit

main:
    read input from file;
    create first thread;
    wait for thread to finish; (finish = 1)
    print final sorted array;
    return;
```

Η συνάρτηση του thread λαμβάνει έναν pointer σε struct και βλέπει το subarray που ορίζεται από αυτό. Εάν περιέχει ίσα ή λιγότερα από 64 στοιχεία, καλεί μία selection sort. Εάν όχι, τότε ξανά χωρίζει το sub array σε 2 ίσα κομμάτια και δημιουργεί 2 νέα threads με την ίδια συνάρτηση. Περιμένει ενεργά να τελειώσουν τα καινούρια threads και κάνει merge τα subarrays.





Ευχαριστούμε για τον  
χρόνο σας!