6CS005 Learning Journal - Semester 1 2018/19

Harpreet Jhita - 1607117

# Table of Contents

[Table of Contents 1](#_Toc527650969)

[1 POSIX Threads 2](#_Toc527650970)

[1.1 Password Cracking 2](#_Toc527650971)

[1.2 Image Processing 2](#_Toc527650972)

[1.3 Linear Regression 2](#_Toc527650973)

[2 CUDA 3](#_Toc527650974)

[2.1 Password Cracking 3](#_Toc527650975)

[2.2 Image Processing 3](#_Toc527650976)

[2.3 Linear Regression 3](#_Toc527650977)

[3 MPI 4](#_Toc527650978)

[3.1 Password Cracking 4](#_Toc527650979)

[3.2 Image Processing 4](#_Toc527650980)

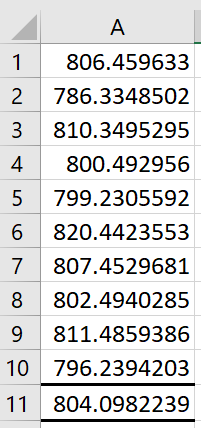
[3.3 Linear Regression 4](#_Toc527650981)

[4 Verbose Repository Log 5](#_Toc527650982)

# POSIX Threads

## Password Cracking

Insert a table of 10 running times and the mean running time.



Insert a paragraph that hypothesises how long it would take to run if the number of initials were to be increased to 3. Include your calculations.

The program will take longer. With 2 letters and 2 numbers, there is a total of 67,600 per run and if the programs runs 4 times to find all 4 passwords, that takes it to 270,400 combinations. While this is very inefficient, it is how the program runs by going from AA00 to ZZ99 to find one password at a time rather than all 4. Using the Google Cloud machine, I managed to get an average of 804 seconds (270,400/804). Using this data, we can say that the program finds approximately 336 combinations per second. Increasing the initial to 3 means that in 1 run from AAA00 to ZZZ99, there will be 1,757,600 combinations and if the program has to go through it 4 times before finishing, that means it will have to go through 7,030,400 combinations. At a speed of 336 combinations per second, it will take approximately 20,923 seconds to complete which equates to just under 350 minutes.

Paste your source code for a 3 initial password cracker here. The code should be neatly indented and lines should not wrap

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <crypt.h>

#include <math.h>

#include <time.h>

int n\_passwords = 4;

char \*encrypted\_passwords[] = {

"$6$KB$H8s0k9/1RQ783G9gF69Xkn.MI.Dq5Ox0va/dFlkknNjO7trgekVOjTv1BKCb.nm3vqxmtO2mOplhmFkwZXecz0",

"$6$KB$WksuNcTfYjZWjDC4Zt3ZAmQ38OrsWwHyDgf/grFJ2Sgg.qpOz56lMpBVfWYdQZa9Pksa2TJRVYVb3K.mbYx4Y1",

"$6$KB$0ZqvOLHpRgU9vLhzavKkL37MCDESwi2NDoTptnw4jyAjQGVtizjiKaluE60l1k7b.7YzDFU3biOo7Cr2SnvzT1",

"$6$KB$UwKD1iCsvhAryQWAH6o8C9B6dEtOUOhYCgBfwtvffD.Ycz83.8GZ/9dhfIyVodUtHRyUl8A8LRfCNSlx8Lb2O1"

};

void substr(char \*dest, char \*src, int start, int length){

memcpy(dest, src + start, length);

\*(dest + length) = '\0';

}

void crack(char \*salt\_and\_encrypted){

int x, y, z;

char salt[7];

char plain[7];

char \*enc;

int count = 0;

int third\_initial; // new loop counter for the third initial

substr(salt, salt\_and\_encrypted, 0, 6);

for(x='A'; x<='Z'; x++){

for(y='A'; y<='Z'; y++){

for(third\_initial = 'A'; third\_initial <= 'Z'; third\_initial++){ // new for loop for the third initial

for(z=0; z<=99; z++){

sprintf(plain, "%c%c%c%02d", x, y, third\_initial, z);

enc = (char \*) crypt(plain, salt);

count++;

if(strcmp(salt\_and\_encrypted, enc) == 0){

printf("#%-8d%s %s\n", count, plain, enc);

} else {

printf(" %-8d%s %s\n", count, plain, enc);

}

}

}

}

}

printf("%d solutions explored\n", count);

}

int time\_difference(struct timespec \*start, struct timespec \*finish, long long int \*difference) {

long long int ds = finish->tv\_sec - start->tv\_sec;

long long int dn = finish->tv\_nsec - start->tv\_nsec;

if (dn < 0) {

ds--;

dn += 1000000000;

}

\*difference = ds \* 1000000000 + dn;

return !(\*difference > 0);

}

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

struct timespec start, finish;

long long int time\_elapsed;

clock\_gettime(CLOCK\_MONOTONIC, &start);

int i;

for(i=0;i<n\_passwords;i<i++) {

crack(encrypted\_passwords[i]);

}

clock\_gettime(CLOCK\_MONOTONIC, &finish);

time\_difference(&start, &finish, &time\_elapsed);

printf("Time elasped was %lldns or %0.9lfs\n", time\_elapsed, (time\_elapsed / 1.0e9));

return 0;

}

Explain your results of running your 3 initial password cracker with relation to your earlier hypothesis.

Due to the length of the program, I ran the program with it searching for only 1 password. This time was 5201 seconds. If we multiply this by 4, we get 20,804 (346 minutes). While this isn’t the exact result, it is close to what the earlier hypothesis suggested.

Write a paragraph that compares the original results with those of your multithread password cracker.

After running the multi-threaded code once, the time came out to 5214 seconds. This is a drastic improvement and makes sense. Before it was a single core running all the program and now it is 2 cores, each running half. Given that the workload per core has halved and that there is an extra core, it makes sense that the difference is approximately 4 times better.

Paste your source code for your multithread password cracker here

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <crypt.h>

#include <math.h>

#include <time.h>

#include <pthread.h>

int n\_passwords = 4;

char \*encrypted\_passwords[] = {

"$6$KB$H8s0k9/1RQ783G9gF69Xkn.MI.Dq5Ox0va/dFlkknNjO7trgekVOjTv1BKCb.nm3vqxmtO2mOplhmFkwZXecz0",

"$6$KB$WksuNcTfYjZWjDC4Zt3ZAmQ38OrsWwHyDgf/grFJ2Sgg.qpOz56lMpBVfWYdQZa9Pksa2TJRVYVb3K.mbYx4Y1",

"$6$KB$0ZqvOLHpRgU9vLhzavKkL37MCDESwi2NDoTptnw4jyAjQGVtizjiKaluE60l1k7b.7YzDFU3biOo7Cr2SnvzT1",

"$6$KB$UwKD1iCsvhAryQWAH6o8C9B6dEtOUOhYCgBfwtvffD.Ycz83.8GZ/9dhfIyVodUtHRyUl8A8LRfCNSlx8Lb2O1"

};

void multiCore(){//method to create multiple cores

pthread\_t thread\_1, thread\_2;

void \*kernel\_function\_1();

void \*kernel\_function\_2();

pthread\_create(&thread\_1, NULL, kernel\_function\_1, NULL);

pthread\_create(&thread\_2, NULL, kernel\_function\_2, NULL);

pthread\_join(thread\_1, NULL);

pthread\_join(thread\_2, NULL);

}

void substr(char \*dest, char \*src, int start, int length){

memcpy(dest, src + start, length);

\*(dest + length) = '\0';

}

void \*kernel\_function\_1(){

int x, y, z;

char salt[7];

char plain[7];

char \*enc;

int count = 0;

char \*encrypted\_passwords[] = {

"$6$KB$H8s0k9/1RQ783G9gF69Xkn.MI.Dq5Ox0va/dFlkknNjO7trgekVOjTv1BKCb.nm3vqxmtO2mOplhmFkwZXecz0",

"$6$KB$WksuNcTfYjZWjDC4Zt3ZAmQ38OrsWwHyDgf/grFJ2Sgg.qpOz56lMpBVfWYdQZa9Pksa2TJRVYVb3K.mbYx4Y1",

"$6$KB$0ZqvOLHpRgU9vLhzavKkL37MCDESwi2NDoTptnw4jyAjQGVtizjiKaluE60l1k7b.7YzDFU3biOo7Cr2SnvzT1",

"$6$KB$UwKD1iCsvhAryQWAH6o8C9B6dEtOUOhYCgBfwtvffD.Ycz83.8GZ/9dhfIyVodUtHRyUl8A8LRfCNSlx8Lb2O1"

};

int third\_initial; // new loop counter for the third initial

substr(salt, encrypted\_passwords[0], 0, 6);

for(int i=0;i<n\_passwords;i<i++) {

for(x='A'; x<='M'; x++){

for(y='A'; y<='Z'; y++){

for(third\_initial = 'A'; third\_initial <= 'Z'; third\_initial++){ // new for loop for the third initial

for(z=0; z<=99; z++){

sprintf(plain, "%c%c%c%02d", x, y, third\_initial, z);

enc = (char \*) crypt(plain, salt);

count++;

if(strcmp(encrypted\_passwords[i], enc) == 0){

printf("#%-8d%s %s\n", count, plain, enc);

} else {

printf(" %-8d%s %s\n", count, plain, enc);

}

}

}

}

}

}

printf("%d solutions explored\n", count);

}

void \*kernel\_function\_2(){

int x, y, z;

char salt[7];

char plain[7];

char \*enc;

int count = 0;

char \*encrypted\_passwords[] = {

"$6$KB$H8s0k9/1RQ783G9gF69Xkn.MI.Dq5Ox0va/dFlkknNjO7trgekVOjTv1BKCb.nm3vqxmtO2mOplhmFkwZXecz0",

"$6$KB$WksuNcTfYjZWjDC4Zt3ZAmQ38OrsWwHyDgf/grFJ2Sgg.qpOz56lMpBVfWYdQZa9Pksa2TJRVYVb3K.mbYx4Y1",

"$6$KB$0ZqvOLHpRgU9vLhzavKkL37MCDESwi2NDoTptnw4jyAjQGVtizjiKaluE60l1k7b.7YzDFU3biOo7Cr2SnvzT1",

"$6$KB$UwKD1iCsvhAryQWAH6o8C9B6dEtOUOhYCgBfwtvffD.Ycz83.8GZ/9dhfIyVodUtHRyUl8A8LRfCNSlx8Lb2O1"

};

int third\_initial; // new loop counter for the third initial

substr(salt, encrypted\_passwords[0], 0, 6);

for(int i=0;i<n\_passwords;i<i++) {

for(x='N'; x<='Z'; x++){

for(y='A'; y<='Z'; y++){

for(third\_initial = 'A'; third\_initial <= 'Z'; third\_initial++){ // new for loop for the third initial

for(z=0; z<=99; z++){

sprintf(plain, "%c%c%c%02d", x, y, third\_initial, z);

enc = (char \*) crypt(plain, salt);

count++;

if(strcmp(encrypted\_passwords[i], enc) == 0){

printf("#%-8d%s %s\n", count, plain, enc);

} else {

printf(" %-8d%s %s\n", count, plain, enc);

}

}

}

}

}

}

printf("%d solutions explored\n", count);

}

int time\_difference(struct timespec \*start, struct timespec \*finish, long long int \*difference) {

long long int ds = finish->tv\_sec - start->tv\_sec;

long long int dn = finish->tv\_nsec - start->tv\_nsec;

if (dn < 0) {

ds--;

dn += 1000000000;

}

\*difference = ds \* 1000000000 + dn;

return !(\*difference > 0);

}

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

struct timespec start, finish;

long long int time\_elapsed;

clock\_gettime(CLOCK\_MONOTONIC, &start);

int i;

multiCore();

clock\_gettime(CLOCK\_MONOTONIC, &finish);

time\_difference(&start, &finish, &time\_elapsed);

printf("Time elasped was %lldns or %0.9lfs\n", time\_elapsed, (time\_elapsed / 1.0e9));

return 0;

}

## 6CS005 Image Progessing CoureworkImage Processing

Insert the image displayed by your program

Insert a table that has columns containing running times for the original program and your multithread version. Mean running times should be included at the bottom of the columns.

Insert the code for your multithreaded edge detector here

Insert an explanation of the results presented in the above table.

## Linear Regression

Insert a scatter plot of your data.

Have 3 guesses at the optimum values for m and c and present them in a graph that overlays your data.

Insert a graph that presents your data with the solution overlaid.

Insert a comment that compares your guesses with the solution found.

Paste your source code for your multithread linear regression program here.

Insert a table that shows running times for the original and multithread versions.

Write a short analysis of the results.

# CUDA

## Password Cracking

Paste your source code for your CUDA based password cracker here

Insert a table that shows running times for the original and CUDA versions.

Write a short analysis of the results

## Image Processing

Insert a table that shows running times for the original and CUDA versions.

Paste your source code for your CUDA based image processing.

Write a short analysis of the results

## Linear Regression

Paste your source code for your CUDA based linear regression

Insert a table that shows running times for the original and CUDA versions.

Write a short analysis of the results

# MPI

## Password Cracking

Insert a table that shows running times for the original and MPI versions.

Paste your source code for your MPI based password cracker here

Write a short analysis of the results

## Image Processing

Paste your source code for your MPI based image processor

Insert a table that shows running times for the original and MPI versions.

Write a short analysis of the results

## Linear Regression

Paste your source code for your MPI based linear regression

Insert a table that shows running times for the original and MPI versions.

Write a short analysis of the results

# Verbose Repository Log

Paste your verbose format repository log here. With subversion this can be achieved by the following:

svn update

svn –v log > log.txt

gedit log.txt

Then select, copy and paste the text here