**แบบฝึกหัด บทที่ 4**

1. (5 คะแนน) เรื่อง User-Level Thread และ Kernel-Level Thread

• จงอธิบายความแตกต่างระหว่าง User-Level Thread และ Kernel-Level Thread

**ตอบ** User-Level threads จะเป็นแบบ Many - to - one หมายถึง ระบบปฏิบัติการจะวางเธรดทั้งหมดของ Multithread process ไว้ใน Execution context เพียงตัวเดียว ทำงานอยู่ภายใต้พื้นที่ของผู้ใช้งาน ไม่เกี่ยวกับพื้นที่ของเคอร์เนล ทำให้ไม่สามารถเข้าใช้งานเคอร์เนลได้

Kernel-level threads จะเป็นแบบ one - to - one หมายถึง ระบบปฎิบัติการจะวางเธรดแต่ละตัวไว้ใน Execution context ของตัวเอง ดังนั้น ในการวางเธรดแต่ละตัวจะต้องให้ระบปฎิบัติการช่วยในเรื่องการ Mapping ตัว User-level threads แต่ละตัวเข้ากับ Kernel-level threads

• สถานการณ์ใดที่เหมาะสมกับการใช้ User-Level Thread และ Kernel-Level Thread

**ตอบ** User-Level Thread เหมาะกับแบบ Many-to-One เป็นโมเดลที่ใช้ Kernel thread 1 หน่วย กับ User thread หลายหน่วย การจัดการ thread จะอยู่ในพื้นที่ของผู้ใช้ ซึ่งมีประสิทธิภาพ แต่ถ้า thread บล๊อก system call โปรเซสทั้งหมดจะถูกบล๊อกไปด้วย เนื่องจากจะมีเพียง thread เดียวเท่านั้นที่เอ็กเซส kernel ในเวลาหนึ่งๆ thread หลาย ๆ thread ไม่สามารถรันขนานกันในระบบมัลติโปรเซสเซอร์ได้

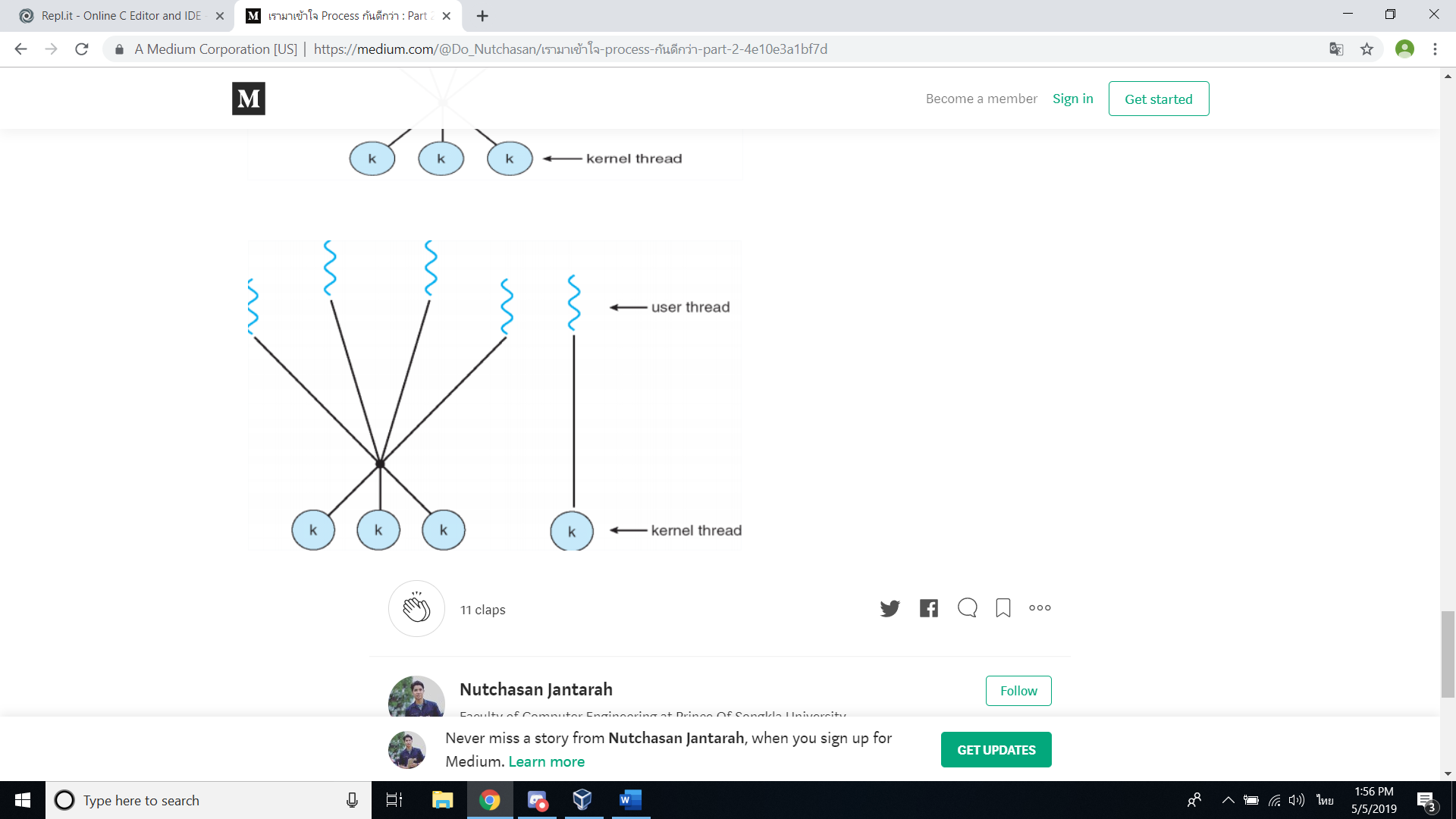
Kernel-Level Thread เหมาะกับแบบ One-to-One เป็นโมเดลที่แต่ละ User thread จะจับคู่กับ Kernel thread ในลักษณะ 1 ต่อ 1 ทำให้สามารถทำงานโดยยอมให้ thread อื่นรันได้เมื่อ thread บล๊อก system call นอกจากนี้โมเดลนี้ยังยอมให้รันหลาย ๆ thread แบบขนานกันในระบบมัลติโปรเซสเซอร์ โมเดลนี้นำมาใช้ในนระบบ เช่น Window NT , Window XP , Window 2000

• สถานการณ์ใดที่เหมาะสมกับการใช้ Kernel-Level Thread

Kernel-Level Thread เหมาะกับแบบ One-to-One เป็นโมเดลที่แต่ละ User thread จะจับคู่กับ Kernel thread ในลักษณะ 1 ต่อ 1 ทำให้สามารถทำงานโดยยอมให้ thread อื่นรันได้เมื่อ thread บล๊อก system call นอกจากนี้โมเดลนี้ยังยอมให้รันหลาย ๆ thread แบบขนานกันในระบบมัลติโปรเซสเซอร์ โมเดลนี้นำมาใช้ในนระบบ เช่น Window NT , Window XP , Window 2000

2. (5 คะแนน) สถานการณ์ใดที่ Multithreaded Solution ที่ใช้ Multiple Kernel Threads ทำ ให้ระบบมี ประสิทธิภาพดีกว่าการใช้ Single-Threaded Solution บนระบบที่มีโพรเซสเซอร์เดียว จงยกตัวอย่าง พร้อมอธิบาย

**ตอบ**

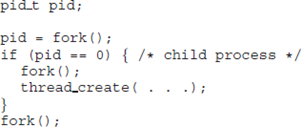


Multithread มีความสามารถที่ดีกว่า เพราะโครงสร้างของมัน เอื้อให้งานย่อยภายใน process ให้สามารถทำงานร่วมกัน ประสานจังหวะการทำงานและใช้ทรัพยากรของ process ร่วมกัน

3. (5 คะแนน) สิ่งใดต่อไปนี้ Register Values, Heap Memory, Global Variables, Stack Memory ที่ถูก แชร์ระหว่าง Multithreaded Process

**ตอบ** Multithreaded Process ได้แชร์ Heap Memory , Global Variables

4. (5 คะแนน) จากส่วนของโค้ดต่อไปนี้



• มีโปรเซสที่สร้างขึ้นกี่โปรเซส

**ตอบ** มีทั้งหมด 6 Process เพราะ Process ในเงื่อนไข fork ได้ 4 process อยู่นอก fork ได้ 2 process

• มี Thread ที่สร้างขึ้นกี่ Thread

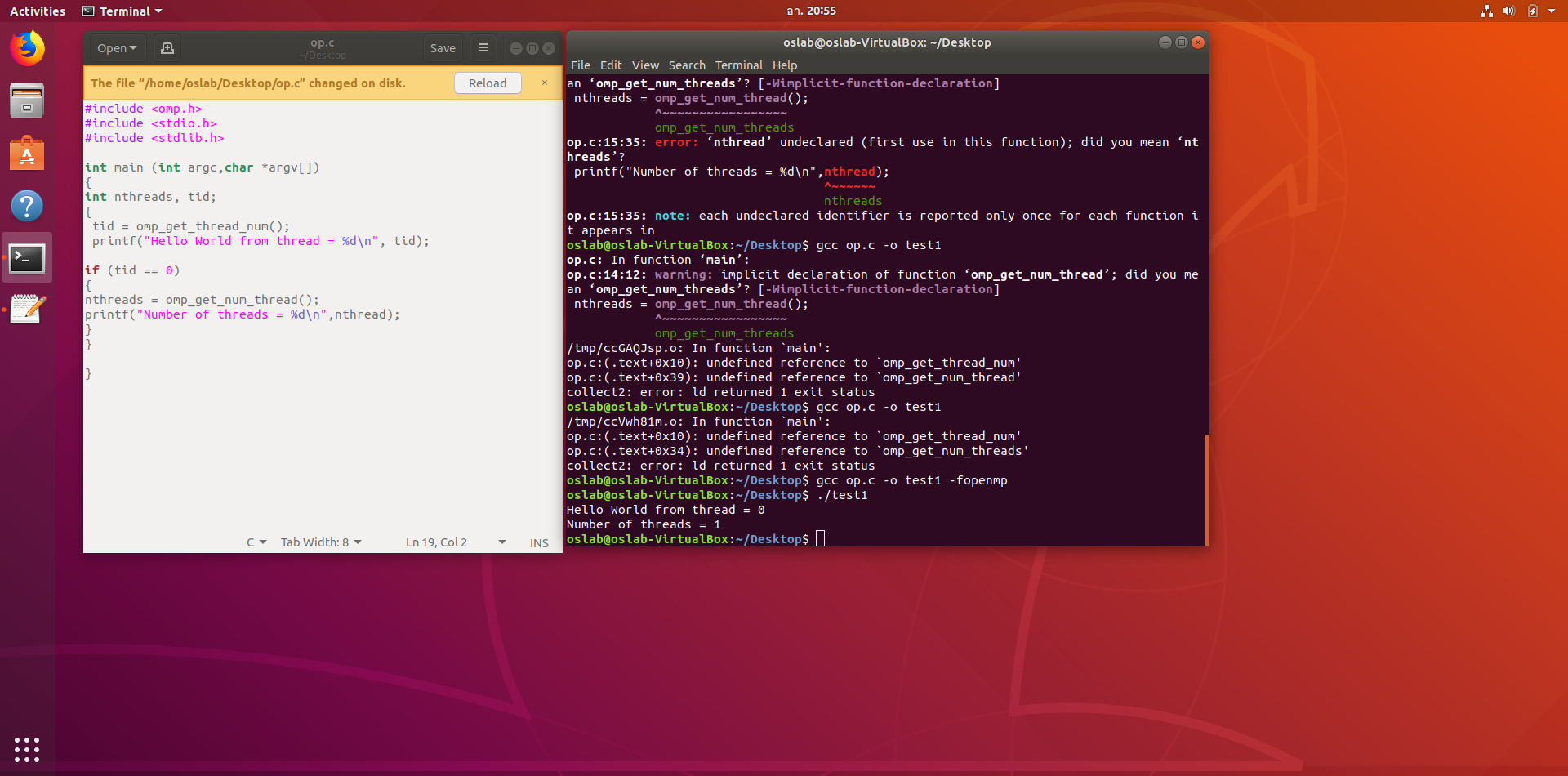
**ตอบ** มีทั้งหมด 8 thread

5. (5 คะแนน) Thread Pool คืออะไร จงอธิบาย

**ตอบ**  Thread Pool เป็น Class ที่ .NET Framework เตรียมเอาไว้ให้สำหรับการทำงานแบบ Multi-thread ที่ง่ายๆ ไม่ซับซ้อน รวมถึงการจัดการกับการทำงานของ Method ที่ Run แบบ Asynchronous และ Thread Pool นั้น Run แบบ Background Thread ดังนั้น Application สามารถหยุดการทำงานได้ทันทีถ้า Foreground Thread อื่นๆ สิ้นสุดการทำงานลง ThreadPool นั้นเหมาะสำหรับที่จะใช้จัดการ Thread ที่ทำงานเล็กๆ ไม่ยุ่งยากซับซ้อนและใช้เวลาในการประมวลผลไม่นานนัก รวมทั้งไม่คำนึงถึงลำดับความสำคัญของ Thread

6. (20 คะแนน) จงหาตัวอย่างของโปรแกรมที่เขียนด้วย Java Thread และ OpenMP แสดงโค้ดพร้อม ผลลัพธ์ของการรันที่ได้ และอธิบายพฤติกรรมของโปรแกรม

**OpenMP\_Code**



กำหนดจำนวน threads โดยใช้ตัวแปร OMP\_NUM\_THREADS หากต้องการเปลี่ยนจำนวนเธรด OpenMP ในเชลล์คำสั่งที่โปรแกรมกำลังรันให้ป้อน

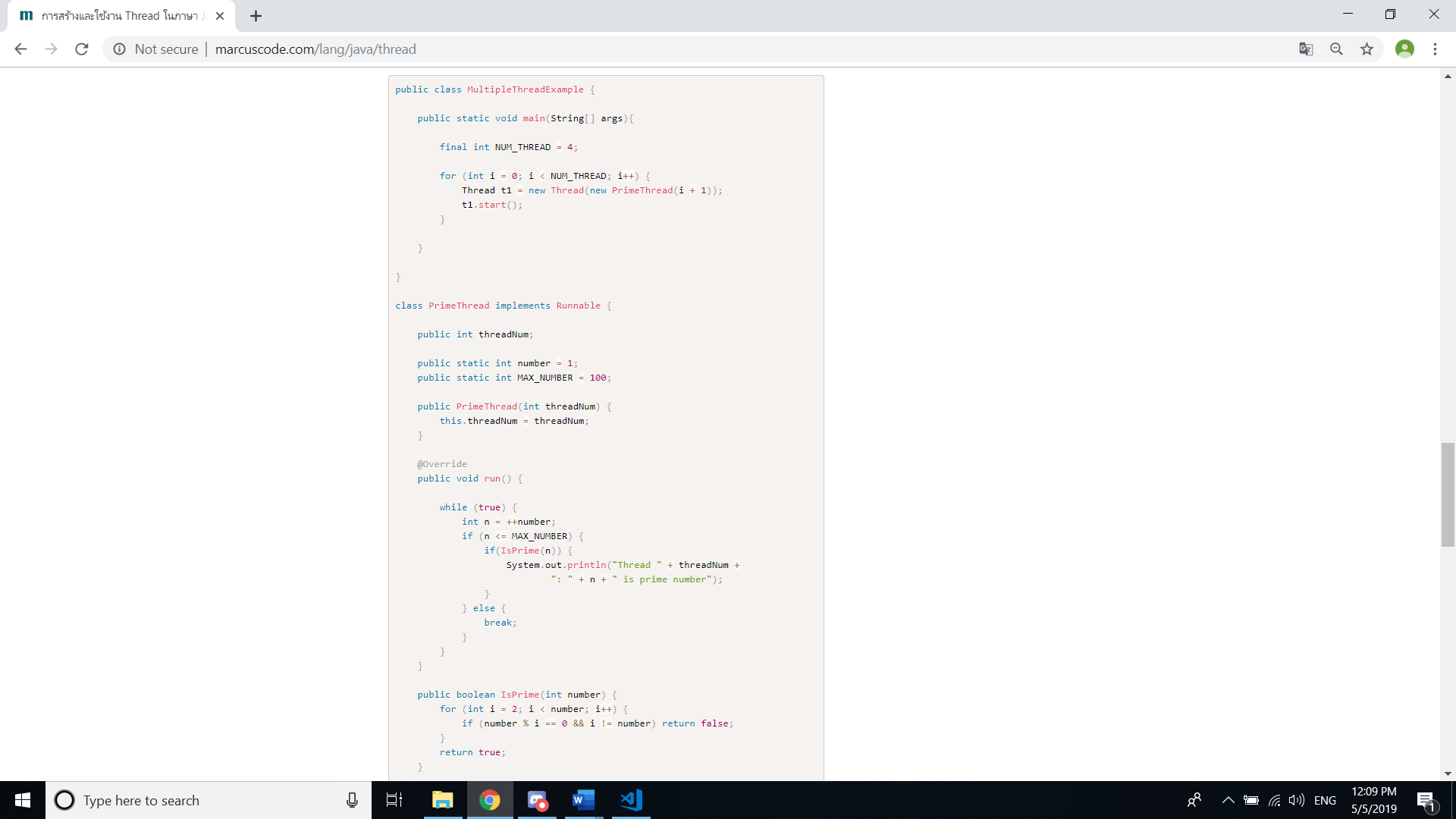
ตั้งค่า OMP\_NUM\_THREADS = <จำนวนเธรดที่ต้องการใช้>

เชลล์บางตัวต้องการตัวแปรและค่าที่จะส่งออก:

ส่งออก OMP\_NUM\_THREADS = <จำนวนกระทู้ที่จะใช้>

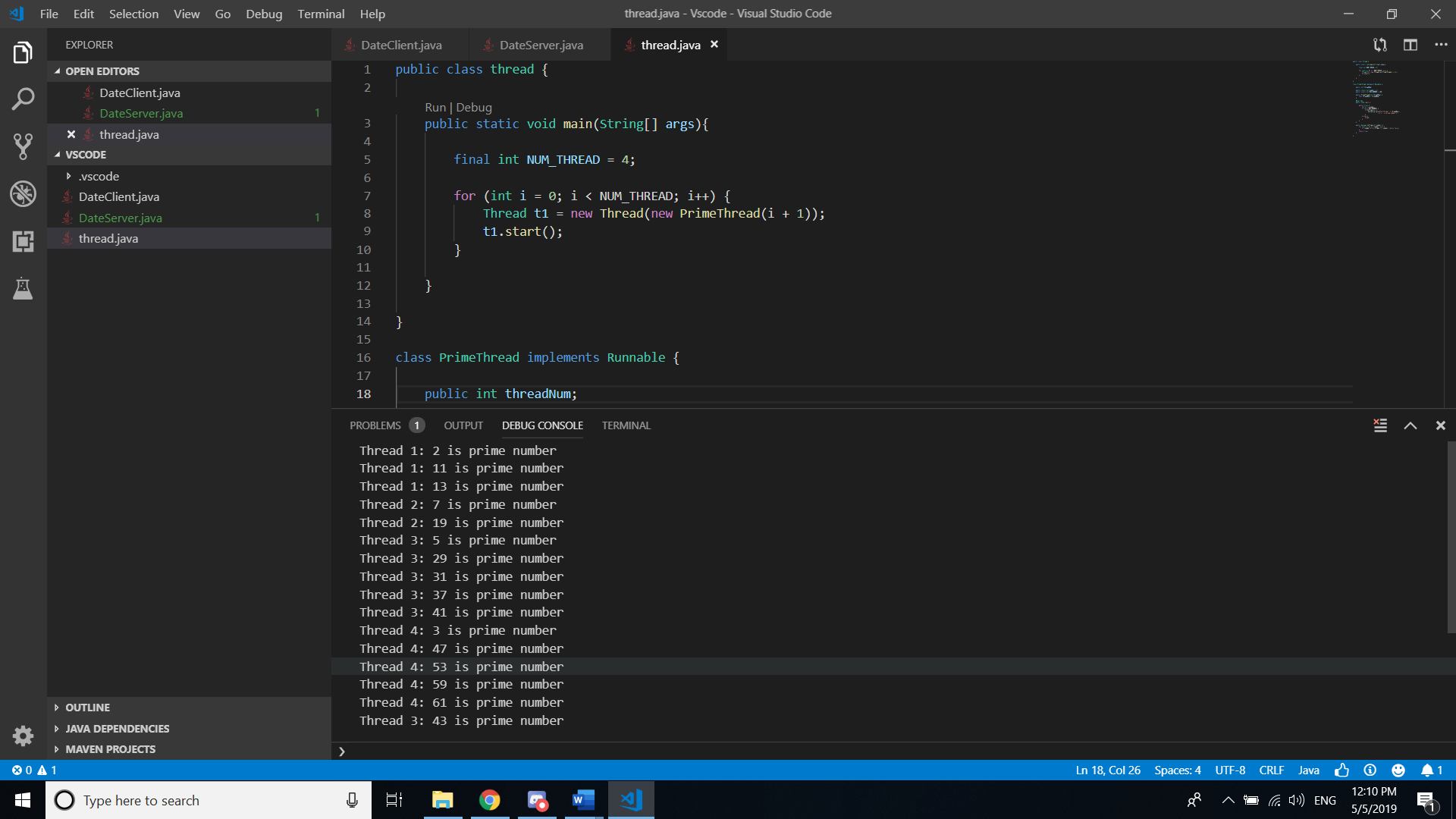
สามารถกำหนดค่าให้กับตัวแปรสภาพแวดล้อมโดยใช้แผงควบคุมระบบปฏิบัติการ Microsoft Windows

**Java Threads Code**



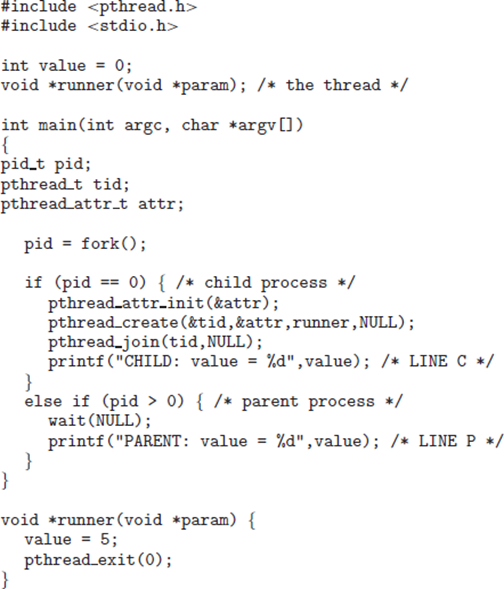
นี่เป็นโปรแกรมหลักในการสร้าง Thread และเพื่อให้โปรแกรมทำงาน ตัวแปร NUM\_THREAD เป็นการกำหนดจำนวนของ Thread ที่ต้องการสร้างสำหรับทำงาน ในตัวอย่างเราได้สร้าง 4 Thread นั้นหมายความว่าแต่ละ Thread นั้นจะมีการทำงานโดยเฉลี่ยคนละ 25 ตัวเลข ซึ่ง ถ้าหากการทำงานของการหาจำนวนเฉพาะของแต่ละตัวเลขนั้นเป็นอิสระต่อกัน การทำงานของโปรแกรมจะเร็วขึ้นเป็น 4 เท่าแต่ในการทำงานจริง เราจะเป็นต้องวัดประสิทธิภาพการทำงานของระบบหรือเรียกว่า System utilization เพื่อหาจำนวน Thread ที่ให้ประสิทธิภาพสูงสุดของการทำงานในโปรแกรมและในเมธอด run() เป็นการกำหนดส่วนการทำงานของโปรแกรม ซึ่งแต่ Thread อาจจะมีการหาตัวเลขจำนวนเฉพาะมากกว่า 1 ครั้งโดยการตรวจสอบด้วยตคำสั่ง If ในเงื่อนไข   
n <= MAX\_NUMBER และทุก Thread จะหยุดทำงานเมื่อเงื่อนไขนี้เป็นเท็จ หรือการหาของตัวเลขนั้นครบทั้ง 100

**ผลลัพธ์ของ Java thread**

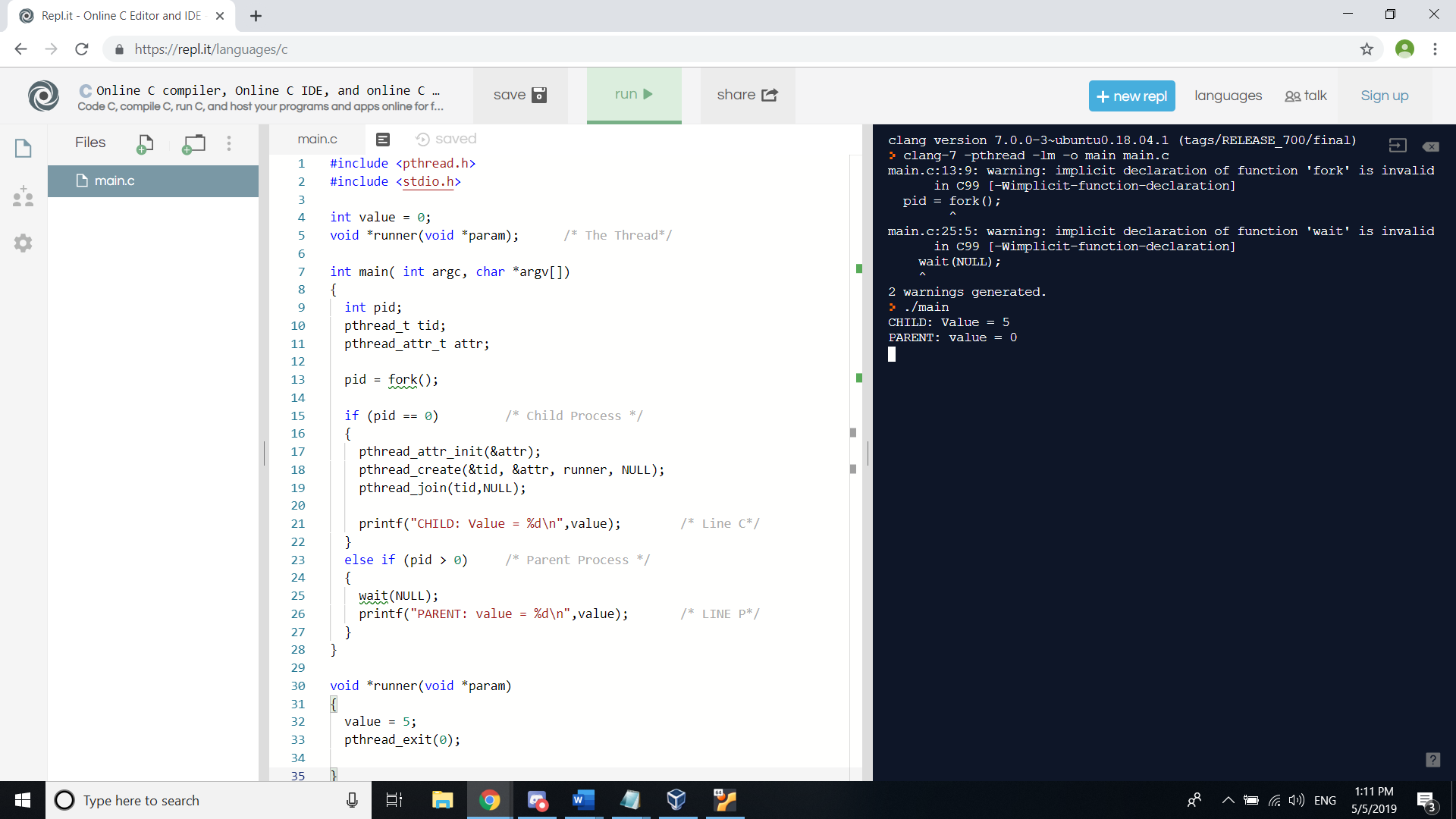


ผลลัพธ์การทำงานของโปรแกรมโดยการแสดงผลจำนวนประเฉพาะตั้งแต่ 1 - 100 คุณสามารถเปลี่ยนค่าในตัวแปร static number และ MAX\_NUMBER สำหรับแสดงผลของจำนวนเฉพาะในระยะอื่นที่ต้องการ

7. (10 คะแนน) จงทดลองโค้ดภาษาซีต่อไปนี้ ผลลัพธ์ที่ได้จาก Line C และ Line P เป็นอย่างไร



**ตอบ**



Child Process value =5 เพราะ child process จะรันก่อน Parent process และ parent process จะรันก็ต่อเมื่อ child process รันเสร็จแล้ว และใน child มีการสร้าง thread ในการเรียกใช้ void runner และ void runner ส่งค่ากลับไปยัง child process เป็นค่า pointer(\*) param โดยค่า value = 5 ดังนั้น child process จึงได้ผลลัพธ์ เท่ากับ 5

ส่วน Parent process มีค่าเท่ากับ 0 เพราะดึงค่าจากค่าเริ่มต้น int value = 0 เพราะ Parent process ไม่ได้ดึงค่าจากฟังก์ชัน void runner มาใช้ จึงเอาค่าเริ่มต้นมาใช้ค่าที่ออกมาเลย เท่ากับ 0

8. (30 คะแนน) จงเขียนโปรแกรมแบบ Multithread โดยมีการทำงานดังนี้

• Main Thread สร้างเลขจำนวนเต็มแบบสุ่ม 10 ค่า แล้วแสดงออกทางจอภาพ

• Main Thread เตรียมตัวแปร min, max, และ avg สำหรับเก็บค่าต่ำสุด, สูงสุด, และค่าเฉลี่ย ตามลำดับ

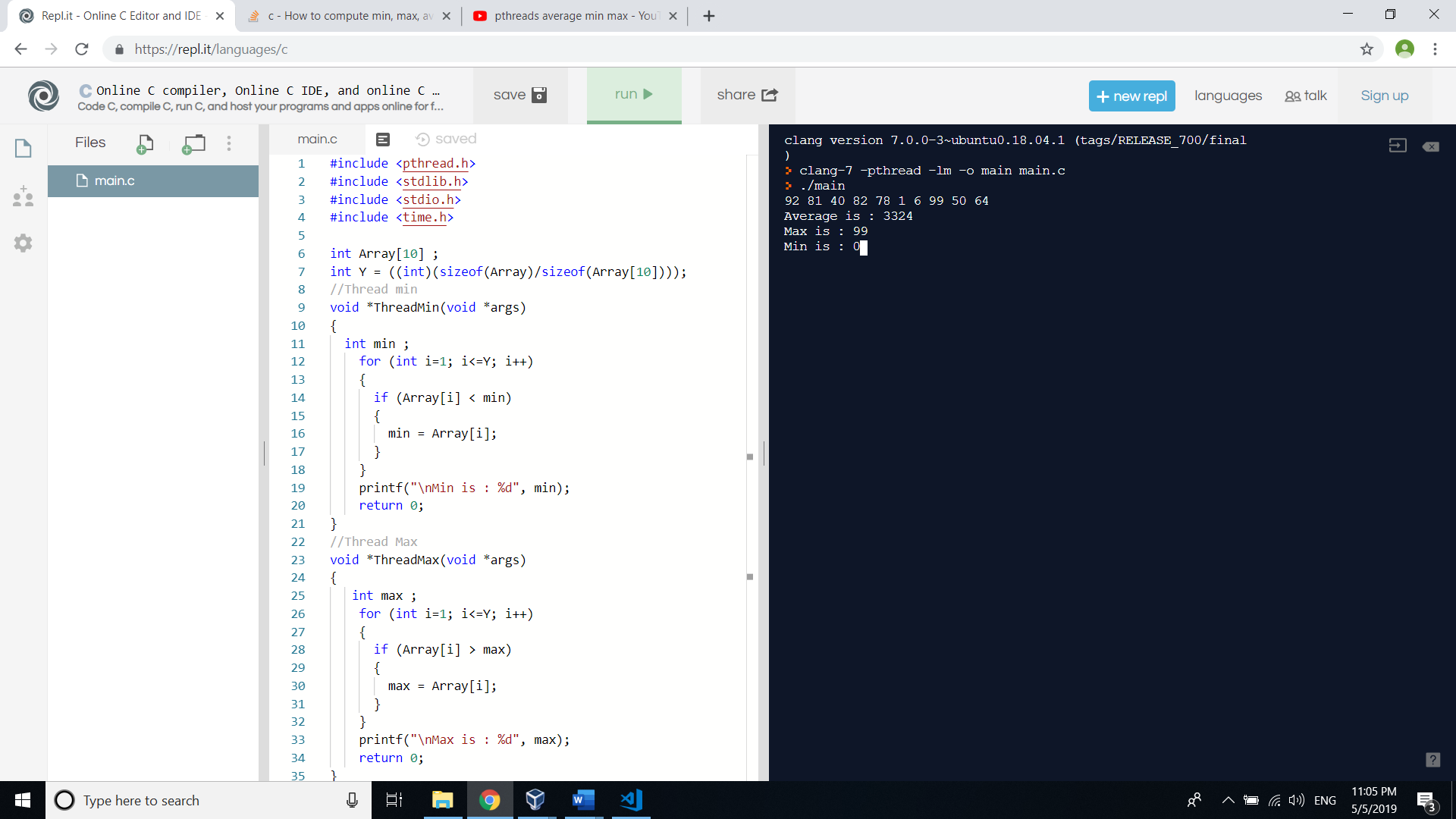
• สร้าง Thread ย่อย 3 Thread มีหน้าที่ดังนี้ จากเลขจำนวนเต็มที่สุ่มมาจาก Main Thread

i. Thread #1: ค้นหาค่าต่ าสุด เก็บลงใน min

ii. Thread #2: ค้นหาค่าสูงสุด เก็บลงใน max

iii. Thread #3: ค านวณค่าเฉลี่ย เก็บลงใน avg

• Main Thread แสดงผลลัพธ์ของ min, max, avg ออกทางจอภาพ



ผลลัพธ์

Average = 33.24

Max = 99

Min = 0

#include <pthread.h>

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

int Array[10] ;

int Y = ((int)(sizeof(Array)/sizeof(Array[10])));

//Thread min

void \*ThreadMin(void \*args)

{

int min ;

for (int i=1; i<=Y; i++)

{

if (Array[i] < min)

{

min = Array[i];

}

}

printf("\nMin is : %d", min);

return 0;

}

//Thread Max

void \*ThreadMax(void \*args)

{

int max ;

for (int i=1; i<=Y; i++)

{

if (Array[i] > max)

{

max = Array[i];

}

}

printf("\nMax is : %d", max);

return 0;

}

//Thread Average

void \*ThreadAverage(void \*args)

{

int aver ;

for (int i=1; i<=Y; i++)

{

aver += Array[i];

}

aver = (aver / Y) ;

printf("\nAverage is : %d ", aver);

return 0;

}

int main()

{

int min;

int max;

int average;

srand(time(NULL));

for (int i=1; i<= Y; i++)

{

Array[i] = rand( )%100;

printf("%d ", Array[i]);

}

//Creat thread id

pthread\_t th1,th2,th3;

pthread\_create(&th1, NULL,ThreadMin,NULL );

pthread\_create(&th2, NULL,ThreadMax,NULL );

pthread\_create(&th3, NULL,ThreadAverage,NULL );

pthread\_join(th1, NULL);

pthread\_join(th2, NULL);

pthread\_join(th3, NULL);

return 0;

}