แบบฝึกหัดที่ 4

1. เรื่อง User-Level Thread และ Kernel-Level Thread

• จงอธิบายความแตกต่างระหว่าง User-Level Thread และ Kernel-Level Thread

User-Level thread จะมีชนิดการทำงานแบบ Many-to-One กล่าวคือ Thread ทั้งหมดจะอยู่ใน Execution context ตัวเดียวและผู้ใช้ไม่สามารถใช้งาน Kernel ได้ ซึ่งส่งผลให้หากมี Thread ตัวใดโดน block การทำงานของ I/O จะทำให้ Thread อื่นๆไม่สามารถทำงานได้แต่ก็ยังมีคุณสมบัติ Portable ที่สามารถย้าย Thread ต่างๆไประบบปฏิบัติการใดก็ได้และไม่ต้องเจาะจงว่า Thread จะต้องทำงานกับ API ของ OS เฉพาะใดๆ

Kernel-Level thread จะมีชนิดการทำงานแบบ One-to-One ซึ่งจะแตกต่างจาก User-level เพราะ Thread จะอยู่กับ Execution แบบหนึ่งต่อหนึ่ง จึงจะต้องให้ระบบปฏิบัติการช่วยเรื่องการ Mapping และมีข้อเสียตรงที่การเรียกใช้งาน Thread ผ่าน Kernel ทำให้เกิด Overhead แต่ก็สามารถประมวลผลแบบขนานได้ดีกว่าและยังสับเปลี่ยนหากเกิดการ Block I/O ของ Thread ใดๆ ก็จะไปใช้ Thread อื่นๆที่ทำงานได้แทน

• สถานการณ์ใดที่เหมาะสมกับการใช้ User-Level Thread และ Kernel-Level Thread

จะเป็นการที่ต้องทำงานแบบ Many-to-Many เพราะจะต้อง Mapping User-level-threads หลายๆตัวเข้ากับชุดของ Kernel-level threads

• สถานการณ์ใดที่เหมาะสมกับการใช้ Kernel-Level Thread

จะเป็น Threads ที่ถูกออกแบบมาสำหรับ ระบบปฏิบัติการนั้นๆเพราะจะสามารถทำงานได้ดีและมีความสามารถในการประมวลผลแบบขนานและยังเหมาะกับงานที่ไม่ได้ต้องการความรวดเร็วในการประมวลผล

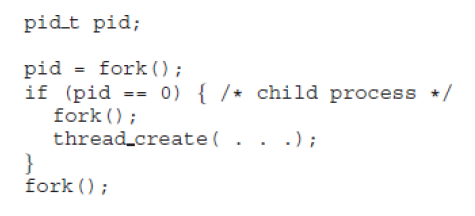
2. สถานการณ์ใดที่ Multithreaded Solution ที่ใช้ Multiple Kernel Threads ทำให้ระบบมีประสิทธิภาพดีกว่าการใช้ Single-Threaded Solution บนระบบที่มีโพรเซสเซอร์เดียว จงยกตัวอย่างพร้อมคำอธิบาย

เป็นข้อจำกัดของระบบ Single threads เพราะเป็นแบบ One-to-One ซึ่งจะมีปัญหาเมื่อถูก Block I/O เข้าที่ Thread ส่งผลให้ไม่สามารถใช้งาน Thread อื่นๆได้ แต่ Kernel นั้นจะแก้ปัญหาด้วยการเบือก Thread อื่นที่สามารถทำงานได้ใช้แทน

3. สิ่งใดต่อไปนี้ Register Values, Heap Memory, Global Variables, Stack Memory ที่ถูกแชร์ระหว่าง Multithreaded Process

Global Variables , Stack Memory

4. จากส่วนของโค้ดต่อไปนี้



• มีโปรเซสที่สร้างขึ้นกี่โปรเซส

6 process

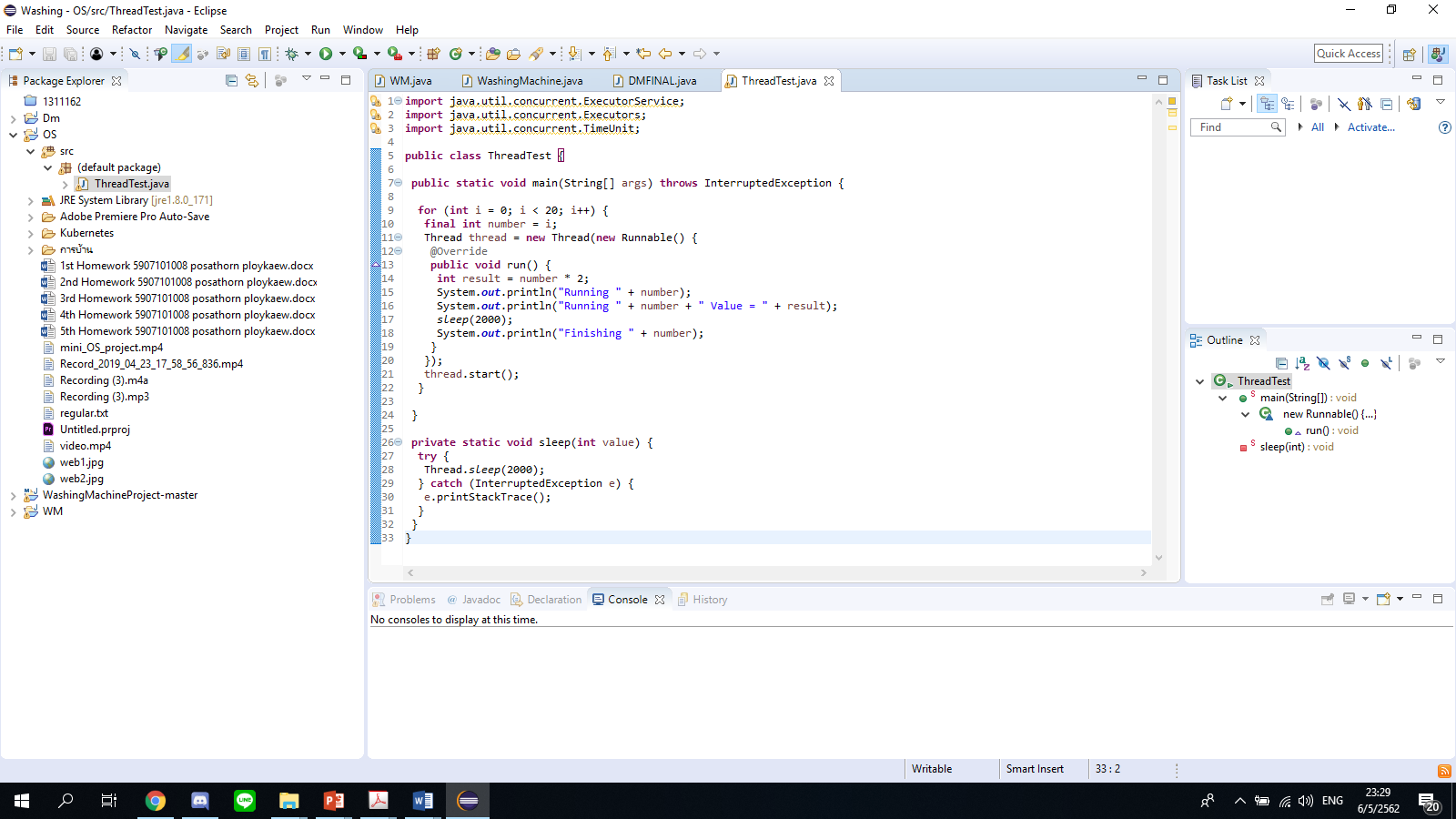
• มี Thread ที่สร้างขึ้นกี่ Thread

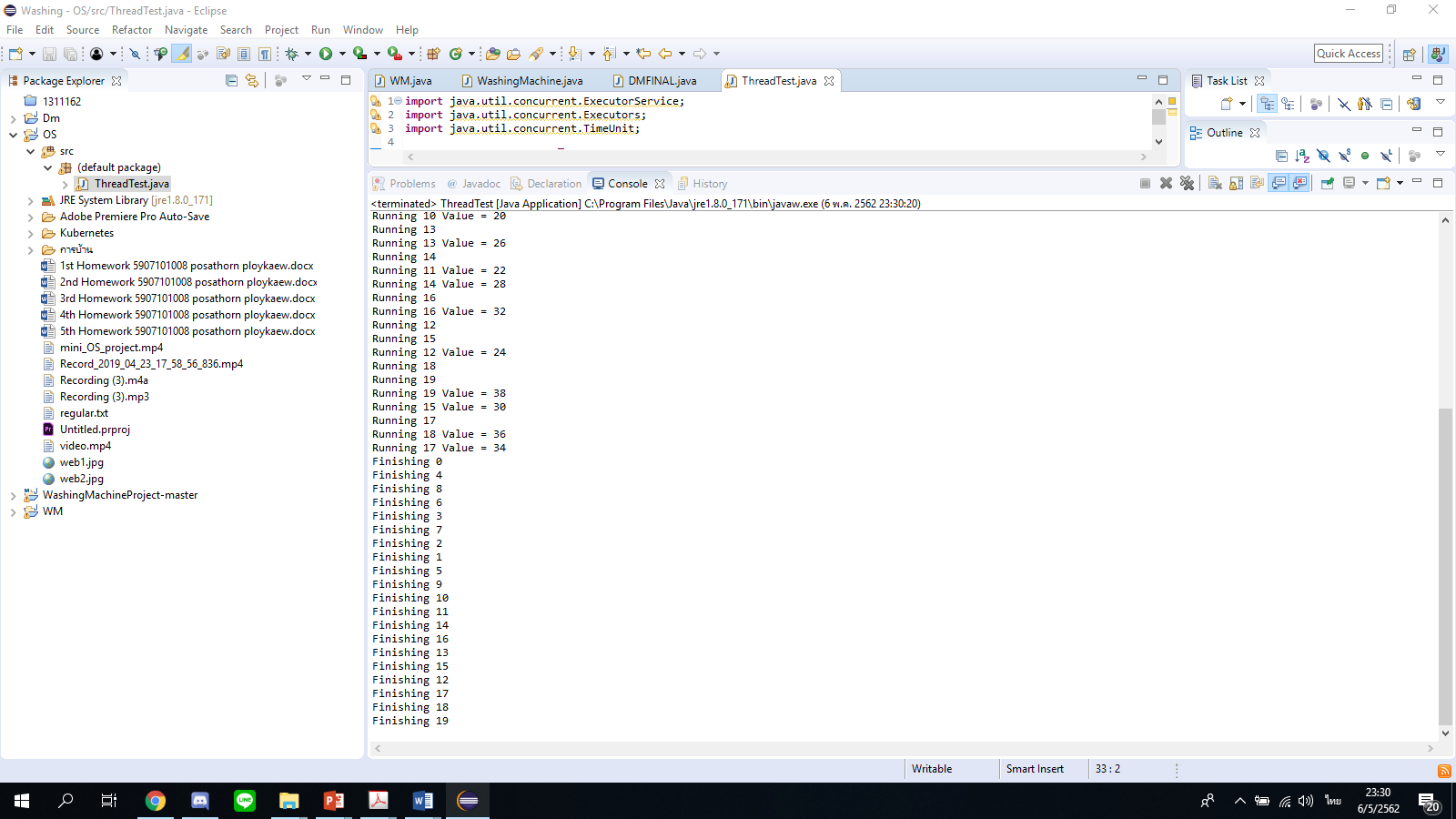
8 Thread

5. Thread Pool คืออะไร จงอธิบาย

การสร้าง Thread ขึ้นมาตามจำนวน Thread ที่รอการทำงาน โดยมีข้อดีคือ

จะทำงานตามคำขอของ Thread ที่มีอยู่แล้วได้เร็วกว่าการสร้าง Thread ใหม่และจำนวน Thread ของ App ต่างๆ จะขึ้นอยู่กับขนาดของ Pool

6. จงหาตัวอย่างของโปรแกรมที่เขียนด้วย Java Thread และ OpenMP แสดงโค้ดพร้อมผลลัพธ์ของการรันที่ได้ และอธิบายพฤติกรรมของโปรแกรม

(โปรแกรมที่เขียน JavaThread)

(ผลการรันโปรแกรม)

เป็นการสร้าง Thread ขึ้นมาจำนวน 20 Thread และจะให้ Thread ที่สร้างเสร็จ ทำการเก็บค่าโดยเอา จำนวน Thread ของตัวเอง \*2 แล้วมาประกาศจากนั้น จะให้ทำการพัก (Delay) แล้วจึงให้บอกว่า Thread ดังกล่าวสร้างเสร็จแล้ว

#include <omp.h>

#include <stdio.h>

int Openmp()

{

int i;

int numthreads = 3;

#pragma omp parallel for default(none) num\_threads(numthreads) private(i)

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

{

int tid = omp\_get\_thread\_num();

printf("Hello world from omp thread %d\n", tid);

}

return -1;

}

int main()

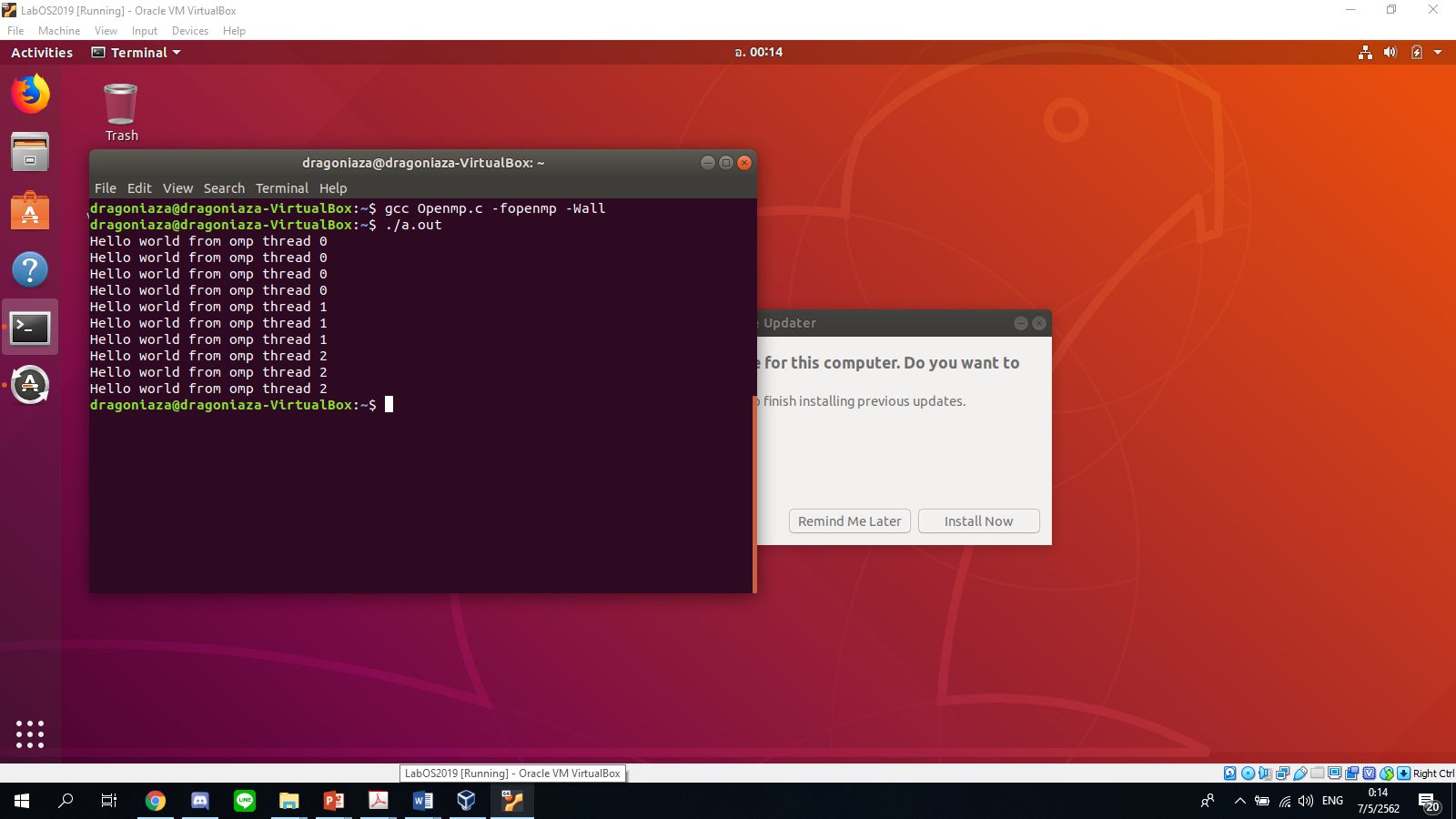
{

HelloFunc();

return 0;

}

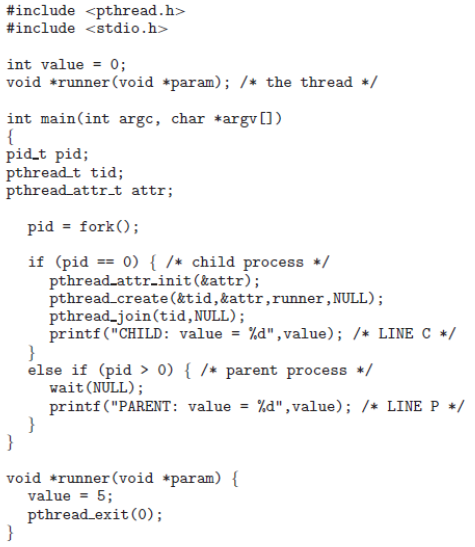
(โปรแกรม Thread จาก Openmp)

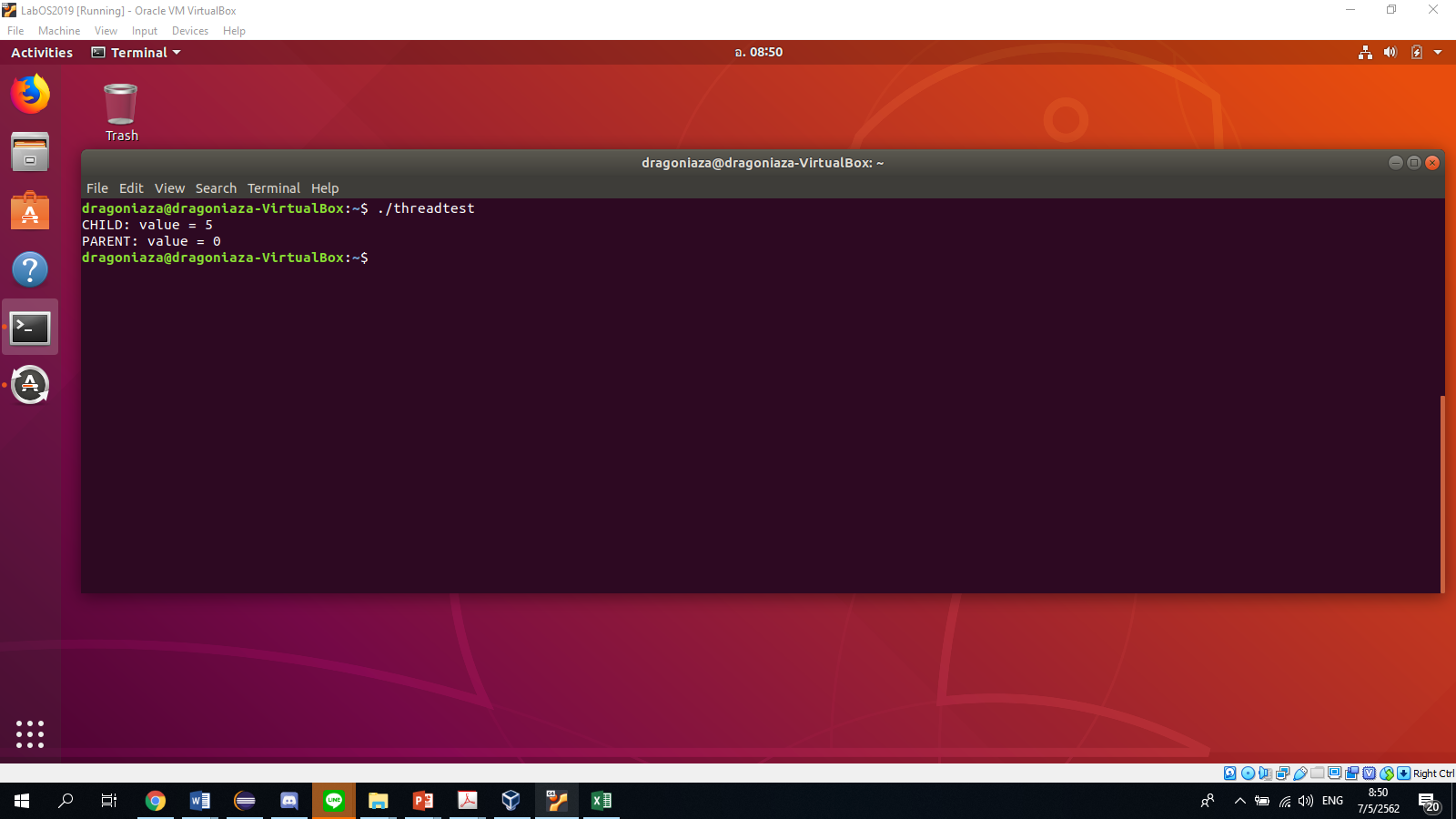


(ผลการรันโปรแกรม)

ผลการรัน : จะสร้าง Thread ขึ้นมา 3 Thread จากนั้นจะทำการประกาศออกมาว่ามีจำนวนกี่ Thread และให้ Thread นั้นเก็บตัวเลขไว้และแสดงผลออกมาตามจำนวนเลขที่ถูกเก็บ

7. จงทดลองโค้ดภาษาซีต่อไปนี้ ผลลัพธ์ที่ได้จาก Line C และ Line P เป็นอย่างไร





(ผลการรันโปรแกรม)

ในบรรทัด C ได้ค่า CHILD = 5 และในบรรทัด C ได้ค่า PARENT = 0 เนื่องจากโปรแกรมได้ทำการ Fork() thread เพิ่มโดยได้เอาตัว Parent ไปเชื่อมเข้ากับ thread attr แต่จะยังมี value =0เพราะเนื่องจากจะต้องรอให้การ fork() จนได้ Child ออกมาก่อนจึงจะกำหนดค่าให้ Child

8. จงเขียนโปรแกรมแบบ Multithread โดยมีการทำงานดังนี้

• Main Thread สร้างเลขจำนวนเต็มแบบสุ่ม 10 ค่า แล้วแสดงออกทางจอภาพ

• Main Thread เตรียมตัวแปร min, max, และ avg สำหรับเก็บค่าต่ำสุด, สูงสุด, และค่าเฉลี่ยตามลาดับ

• สร้าง Thread ย่อย 3 Thread มีหน้าที่ดังนี้ จากเลขจานวนเต็มที่สุ่มมาจาก Main Thread

i. Thread #1: ค้นหาค่าต่ำสุด เก็บลงใน min

ii. Thread #2: ค้นหาค่าสูงสุด เก็บลงใน max

iii. Thread #3: คำนวณค่าเฉลี่ย เก็บลงใน avg

• Main Thread แสดงผลลัพธ์ของ min, max, avg ออกทางจอภาพ

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<unistd.h>

#include<time.h>

int amain[10];

int sizea = (int)sizeof(amain)/sizeof(amain[10]);

void \*avg(void \*args){

int average;

for(int i=0; i<=sizea; i++)

{

average += amain[i];

}

average = average/sizea;

printf("average = %d\n",average);

return 0;

}

void \*min(void \*args){

int minimum;

for(int i=0; i<=sizea; i++)

{

if(amain[i]<minimum){

minimum=amain[i];

}

}

printf("min = %d\n",minimum);

return 0;

}

void \*max(void \*args){

int maximum;

for(int i=0; i<=sizea; i++)

{

if(amain[i]>maximum){

maximum=amain[i];

}

}

printf("max = %d\n",maximum);

return 0;

}

int main(void){

int i,maximum,minimum,average;

srand(NULL);

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

amain[i]=rand()%10;

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

}

pthread\_t t1,t2,t3;

pthread\_create(&t1,NULL,avg,NULL);

pthread\_create(&t2,NULL,max,NULL);

pthread\_create(&t3,NULL,min,NULL);

pthread\_join(t1,NULL);

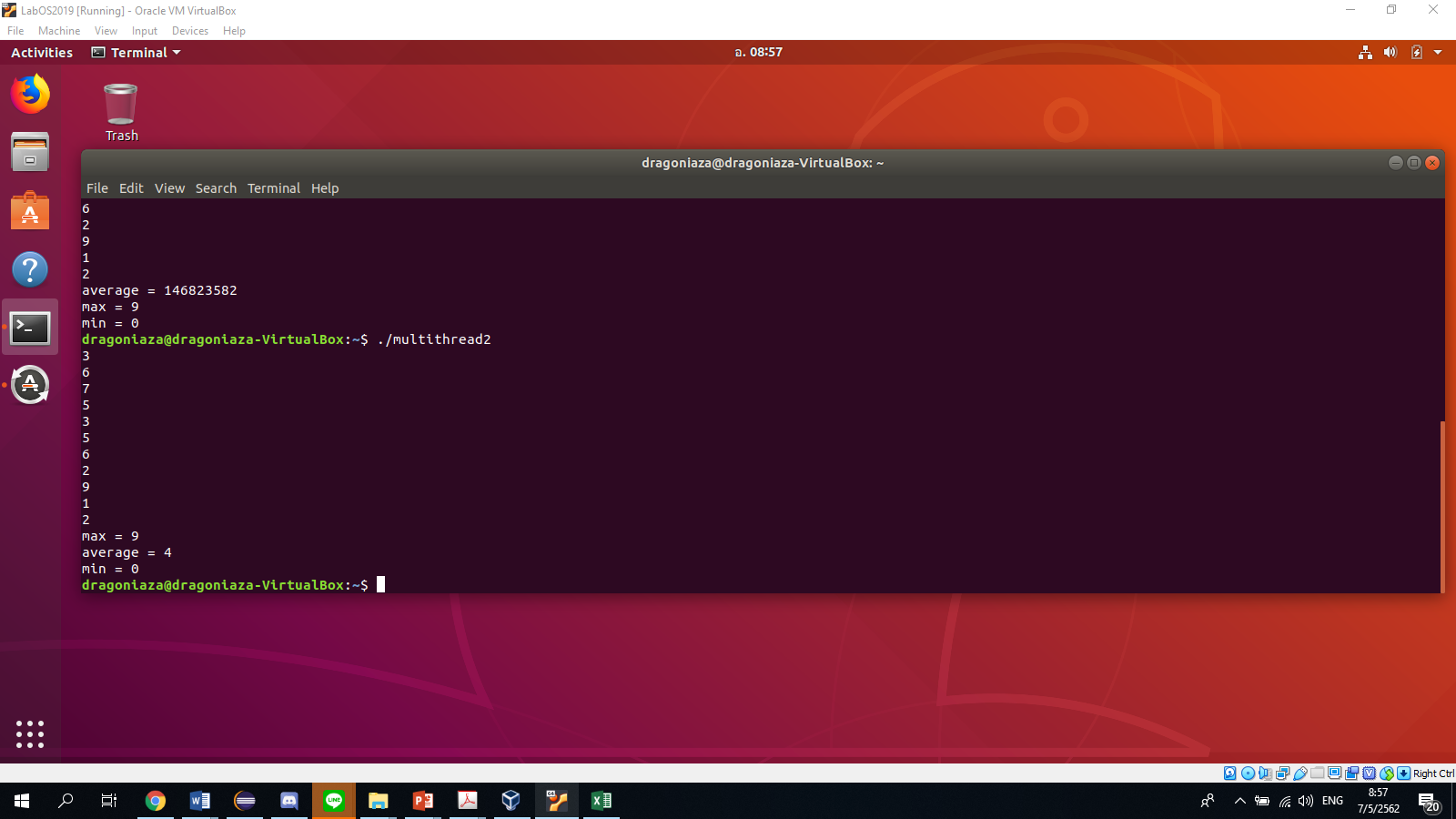
pthread\_join(t2,NULL);

pthread\_join(t3,NULL);

return(0);

}

(โปรแกรม Multithread)



(ผลการรันโปรแกรม)