**Excercise\_4**

**ข้อที่ 1**

เรื่อง User-Level Thread และ Kernel-Level Thread

• จงอธิบายความแตกต่างระหว่าง User-Level Thread และ Kernel-Level Thread

- User-Level Thread

คือ Threads ที่ นำเข้ามา โดย user โดยไม่ได้ยุ่งเกี่ยวกับ kernel ฝั่งแต่อย่างใด kernel จะมอง user threads เป็น single-threaded processes เสมอ

-Kernel-Level Thread

คือ threads ที่ถูกจัดการโดย ตัวระบบปฏิบัติการโดยตรง และจัดการโดยตัว kernel เอง เพราะฉะนั้นประสิทธิภาพจะดี แต่จะช้ากว่า User - Level Threads

• สถานการณ์ใดที่เหมาะสมกับการใช้ User-Level Thread และ Kernel-Level Thread

User-Level Thread เหมาะกับ Many-to-One ใช้ Kernel thread 1 หน่วย กับ User thread หลายหน่วย แต่

thread อื่นจะรันไม่ได้เลยเมื่อ system call ถูกบล็อค

แต่ Kernel-Level Thread เหมาะกับการใช้งานแบบ One-to-One

• สถานการณ์ใดที่เหมาะสมกับการใช้ Kernel-Level Thread

แบบ One-to-One  
เป็นโมเดลที่แต่ละ User thread จะจับคู่กับ Kernel thread ในลักษณะ 1 ต่อ 1 ทำให้สามารถทำงานโดยยอมให้ thread อื่นรันได้แม้ system call ถูกบล็อค

**ข้อที่ 2**

สถานการณ์ใดที่ Multithreaded Solution ที่ใช้ Multiple Kernel Threads ทำให้ระบบมีประสิทธิภาพดีกว่าการใช้ Single-Threaded Solution บนระบบที่มีโพรเซสเซอร์เดียว จงยกตัวอย่าง

พร้อมคำอธิบาย

การทำงานใด ๆ ที่ต้องการความผิดพลาดน้อยหรือต้องการความถูกต้องรวดเร็วสูง เพราะแม้ thread ใด thread หนึ่งจะเกิดอาการค้างหรือแฮง ก็สามารถประมวลผล thread อื่น ๆ ไปต่อได้

อาทิเช่นการทำงานของเว็บ เบาร์เซอร์ ตระกลู chome ที่ใช้งานการประมวลผลแบบแยกแต่ละ tab(ทุก tab แยกเป็น thread ของใครของมัน) ที่เปิดออกมาใหม่อย่างอิสระจาก tab อื่น ๆ ทำให้แม้จะมีการโหลด tab ใด ๆ ค้าง ก็สามารถโหลด tab อื่น ๆ ต่อไปได้ แต่ก็แลกมาด้วนการบริโภคทรัยากรเครื่องที่สูง

ส่วนเว็บเบาร์เซอร์ตระกลู firefox จะใช้การมาประผลแบบรวม tab (ทุก tab รวมเป็นหนึ่ง thread) ทำให้มักมีการค้างหรือแฮงบ่อย ๆ แต่ตัวเว็บเบาร์เซอร์จะบริโภคทรัยากรน้อยกว่าตระกลู chome

**ข้อที่ 3**

สิ่งใดต่อไปนี้ Register Values, Heap Memory, Global Variables, Stack Memory ที่ถูกแชร์ระหว่างMultithreaded Process

Global Variables และ Heap Memory

**ข้อที่ 4**

จากส่วนของโค้ดต่อไปนี้

• มีโปรเซสที่สร้างขึ้นกี่โปรเซส

ตอน fork() ครั้งแรกสองโปรเซส จากตัวหลัก แม่ ตัวรองลูก

ส่วน fork() ครั้งที่สองนั้น เป็นคำสั่งที่มีการตรวจสอบว่า โปรเซส ดังกล่าวเป็นลูกหรือไม่

fork() ครั้ง สาม ต่อท้ายหลังสร้าง 3 โปรเซส แรกไปแล้ว ทำให้มีเพิ่มมาอีก 3 โปรเซสที่แยกต่อจาก 3 โปรเซส ใน fork() ครั้งที่ สอง

ทำให้สุดท้ายนั้นมีทั้งสิ้น 6 โปรเซส

• มี Thread ที่สร้างขึ้นกี่ Thread

มีทั้งสิ้น 2 Thread จากการที่ คำสั่งในการสร้าง Thread นั้นอยู่ในส่วนที่มีการตรวจสอบว่า โปรเซส ดังกล่าวเป็นลูกหรือไม่ ทำให้มีเพียงลูก 2 โปรเซส ในการ fork() ครั้งที่ 2 ที่ทำการสร้าง Thread ขึ้นมาใหม่

**ข้อที่ 5**

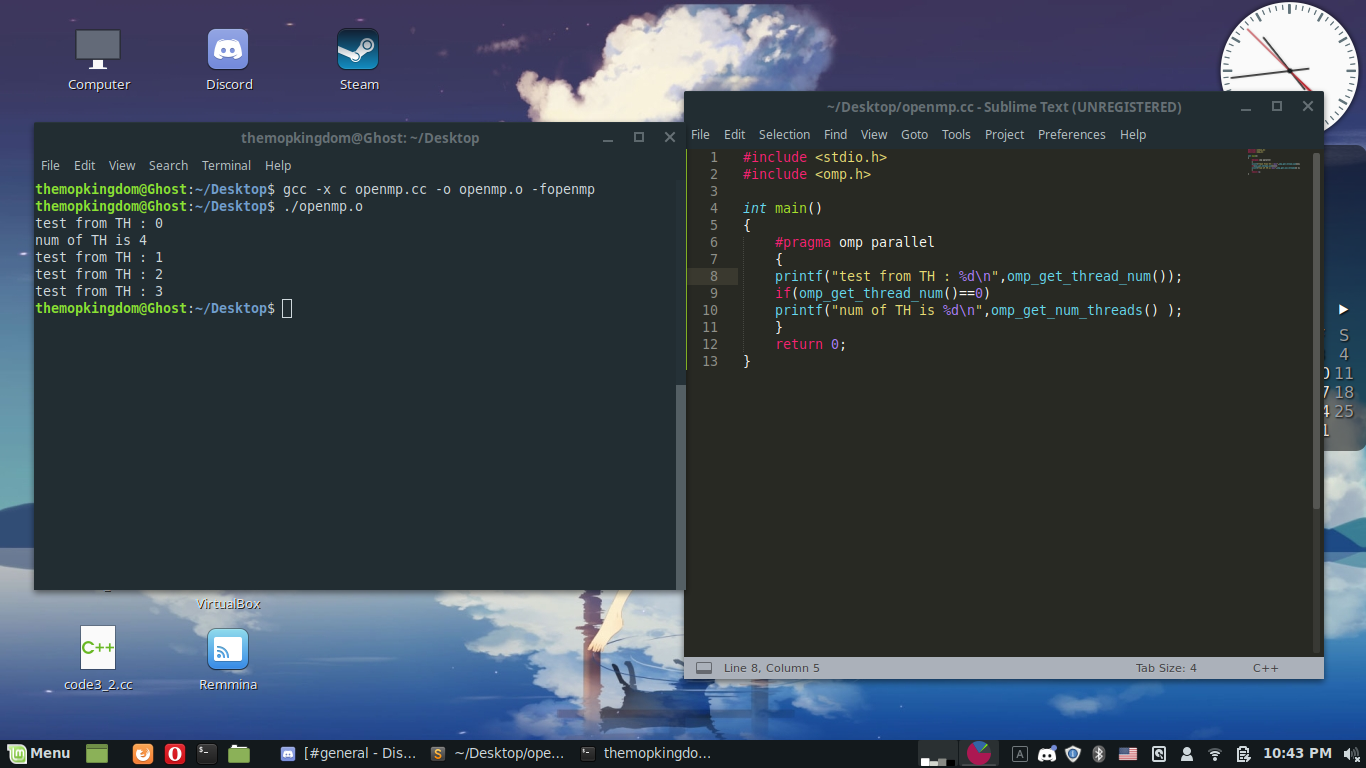
Thread Pool คืออะไร จงอธิบาย

เก็บ thread หลาย ๆ thread ที่รอนำมาประมวลผลและเป็น thread ที่สามารถนำมาประมวลผลพร้อม ๆ กันได้ซึ่งมีการเรียกใช้งานบ่อย ๆ และ ย้าย thread ออกหลังจากประมวลผลเสร็จ แต่จำไม่ทิ้ง thread ดังกล่าวทันทีแต่จะเก็บเอาไว้เผื่อมีการเรียกใช้งาน thread ดังกล่าวอีกครั้ง

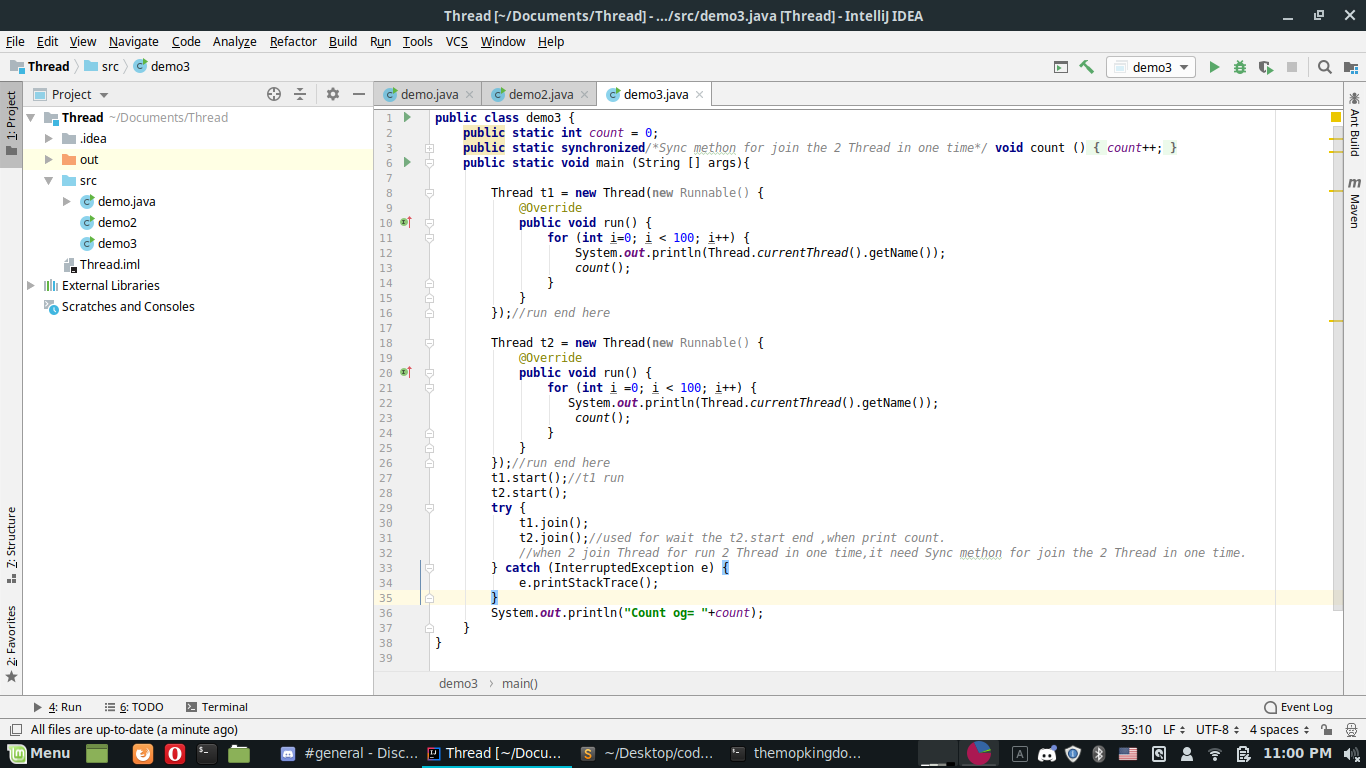
**ข้อที่ 6**

จงหาตัวอย่างของโปรแกรมที่เขียนด้วย Java Thread และ OpenMP แสดงโค้ดพร้อมผลลัพธ์ของการรันที่ได้ และอธิบายพฤติกรรมของโปรแกรม

- OpenMP

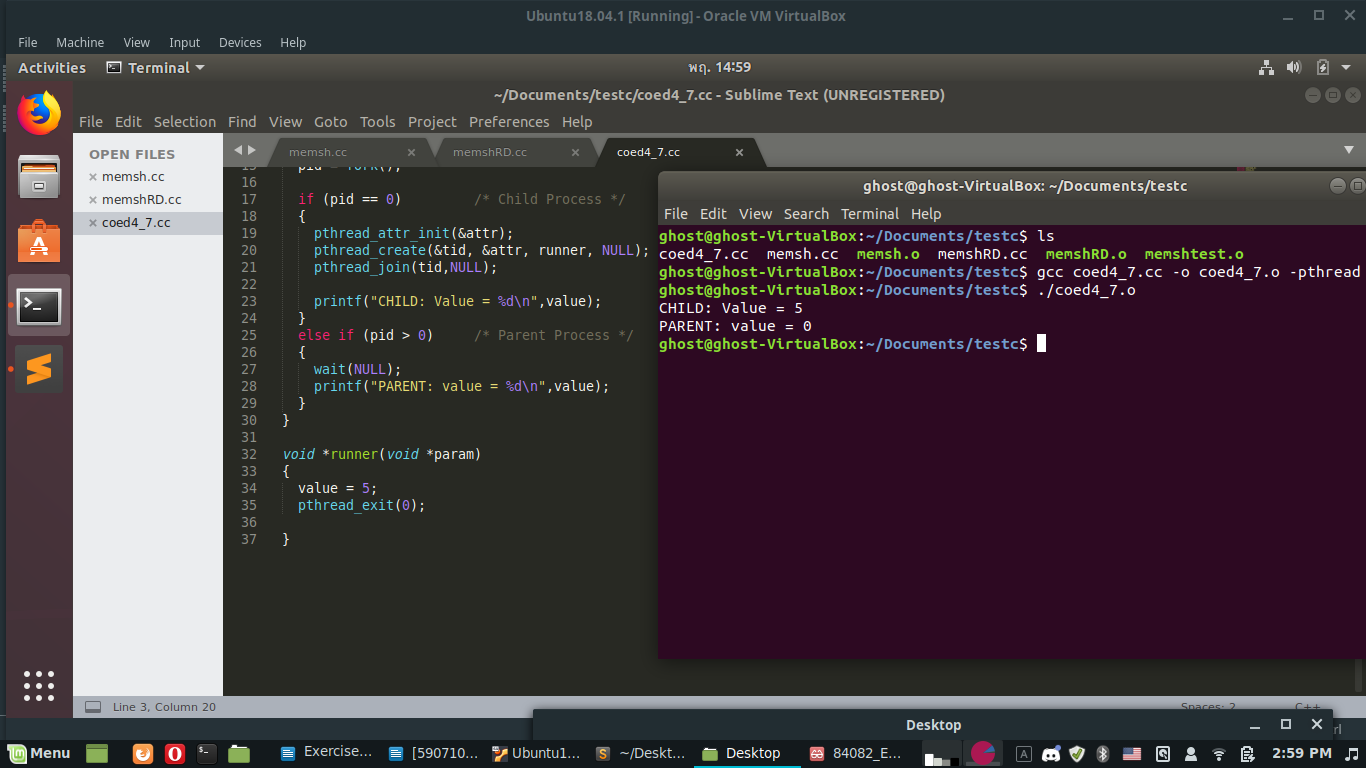
รันคำสั่ง print thread ที่ทำงานอยู่ออกมา และ print thread จำนวน thread ที่มีอยู่ทั้งหมด ซึ่งในที่นี้รันบน cpu 2 core 4 thread

Java Thread

ใช้ thread สอง thread มาช่วยกันนับจำนวน แบ่งเป็น thread ละร้อยและมีการ syn ทั้งสอง thread เข้าหากันไว้ ทำให้ผลลัพธ์ที่จะเป็น 200 เสมอ เพราะจำต้องรอให้นับเสร็จทั้งสอง thread จึงจะสามารถแสดงผลลัพธ์ได้

**ข้อที่ 7**

จงทดลองโค้ดภาษาซีต่อไปนี้ ผลลัพธ์ที่ได้จาก Line C และ Line P เป็นอย่างไร



**ข้อที่ 8**

จงเขียนโปรแกรมแบบ Multithread โดยมีการทำงานดังนี้

• Main Thread สร้างเลขจำนวนเต็มแบบสุ่ม 10 ค่า แล้วแสดงออกทางจอภาพ

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

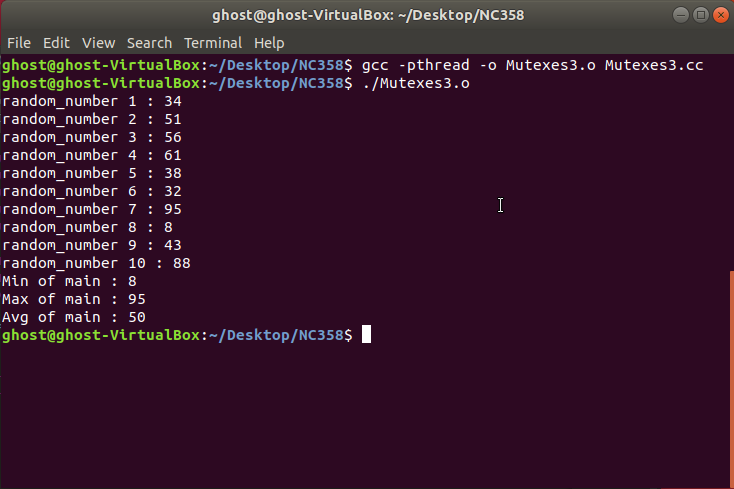
• สร้าง Thread ย่อย 3 Thread มีหน้าที่ดังนี้ จากเลขจำนวนเต็มที่สุ่มมาจาก Main Thread

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



**ไฟล์ code วางอยู่บน git ชื่อ Mutexes3.cc**