**Exercise 4**

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

* จงอธิบายความแตกต่างระหว่าง User-Level Thread และ Kernel-Level Thread

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

การพัฒนาเธรดในพื้นที่ของผู้ใช้งานคือ User-level threads ไม่จำเป็นต้องให้ระบบปฏิบัติการมาสนับสนุนการทำงานของเธรด และเธรดที่ถูกสร้างขึ้นก็ไม่ได้เจาะจงว่าจะต้องรันกับ API(Application progarmming interface)ของระบบปฏิบัติการใดระบบปฏิบัติการหนึ่งโดยเฉพาะ จึงทำให้สะดวกในการย้าย User-level threads ไปยังระบบปฏิบัติการใดก็ได้ เรียกว่าคุณสมบัติ Portable

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

* 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 เหมาะสำหรับ One-to-One ที่ Thread แต่ลพ Thread จับคู่กันแบบ 1 ต่อ 1 เพื่อประสิทธิภาพในการทำงานที่มีความสำคัญสูง แต่ประมวลผลช้า

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

* Single thread มีข้อจำกัดเพราะเป็นแบบ 1 ต่อ 1 ส่วนMultithread มีความสามารถที่ดีกว่า เพราะโครงสร้างของมัน เอื้อให้งานย่อยภายใน process ให้สามารถทำงานร่วมกัน ประสานจังหวะการทำงานและใช้ทรัพยากรของ process ร่วมกัน

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

Multithreaded Process ได้แชร์ Heap Memory , Global Variables

**4. จากส่วนของโค้ดต่อไปนี้**

* **มีโปรเซสที่สร้างขึ้นกี่โปรเซส**

มี 6 โปรเซส

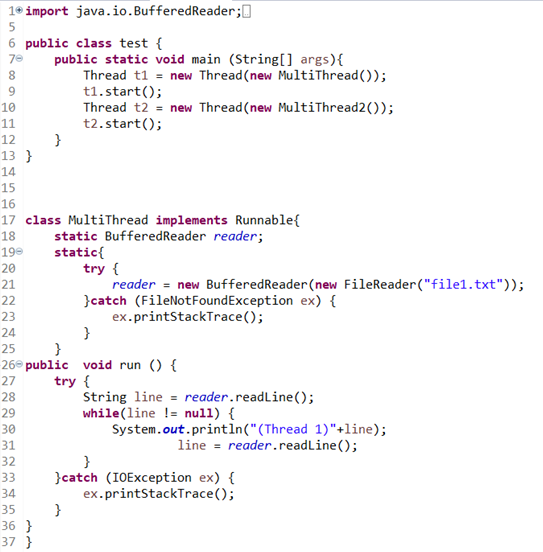
* **มี Thread ที่สร้างขึ้นกี่ Thread**

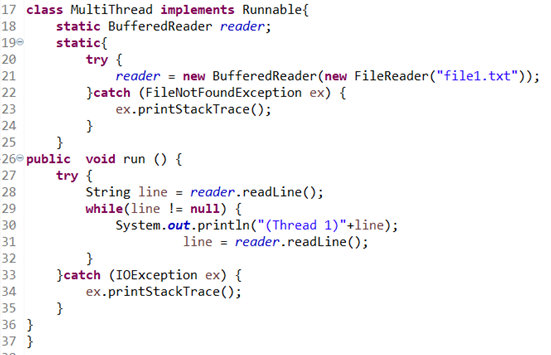
มี 8 Thread

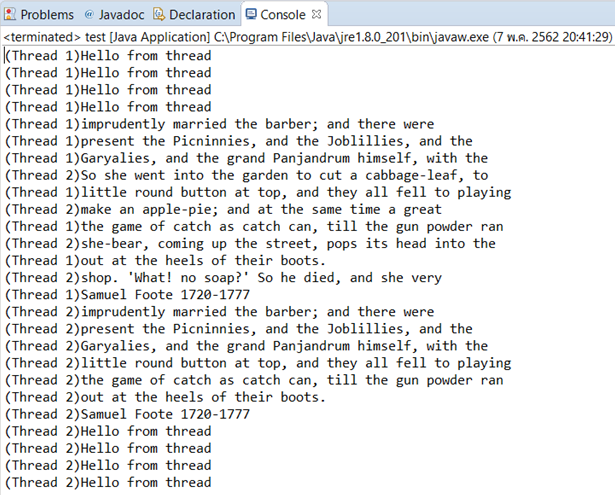
**5. Thread Pool คืออะไร จงอธิบาย**

* เปรียบเหมือนสระว่ายน้ำของ Thread ที่สามารถกักเก็บ Thread ซึ่งจะถูกใช้งานอยู่บ่อยๆแต่เล็กน้อย เป็นการทำงานช่วงสั้นๆ แล้วพอใช้งานเสร็จก็จะถูกนำมาเก็บไว้เช่นเดิม ซึ่งช่วยในการลดระยะเวลาในการทำงานไม่ต้องสร้าง Thread แต่นำ tHread ที่มีอยู่ใน Thread Pool มาใช้ได้ทันที

**6. จงหาตัวอย่างของโปรแกรมที่เขียนด้วย Java Thread และ OpenMP แสดงโค้ดพร้อมผลลัพธ์ของการรันที่ได้ และอธิบายพฤติกรรมของโปรแกรม**

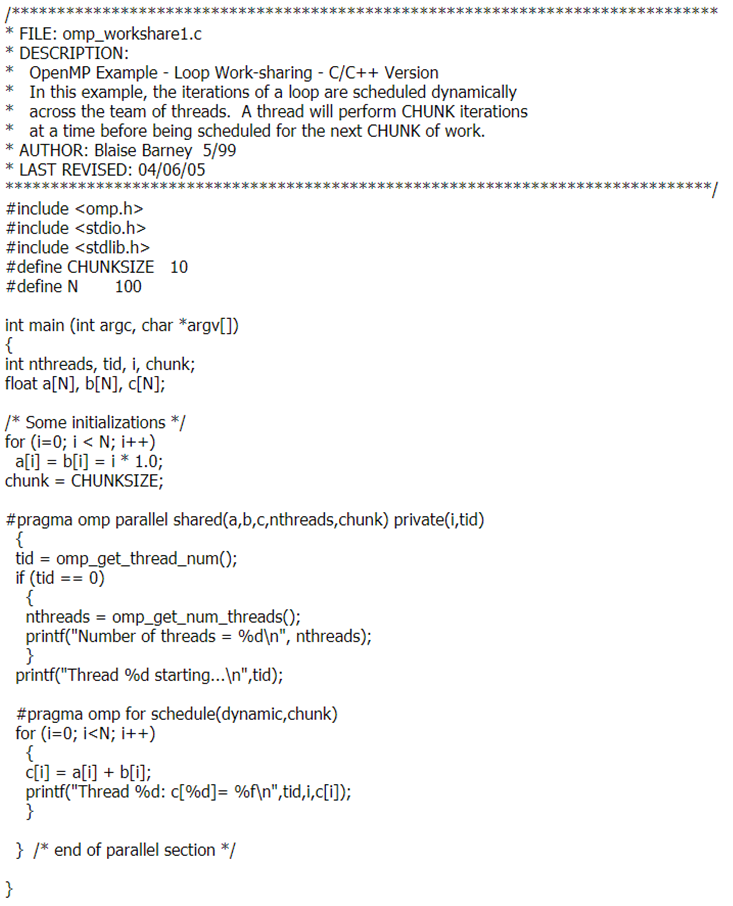
**Java Thread Sample code**

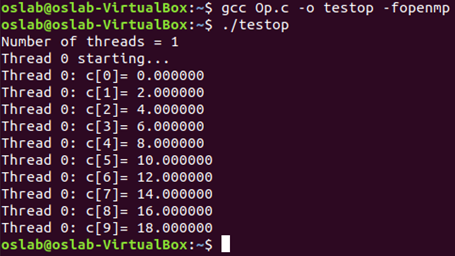




โปรแกรมทำการสร้าง thread ขึ้นมา 2 Thread และเริ่มการทำงาน Thread ทั้ง 2 จากคลาส MultiThread1 และ MultiThread2 โดยมีการทำงานคืออ่านค่าจากไฟล์ text ที่ต่างกันในเวลาเดียวกันและแสดงค่าที่อ่านได้ออกมาเป็นบรรทัด Thread ไหนทำเสร็จก่อนก็จะแสดงออกมาก่อน

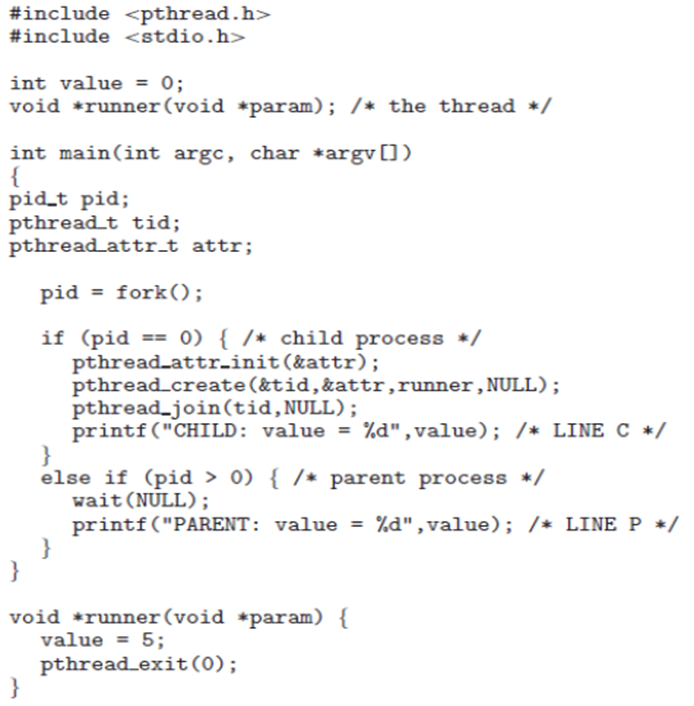
**OpenMP Code Sample code**

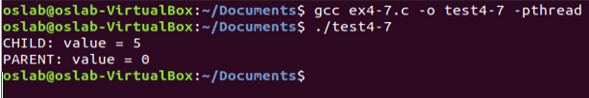
****



จะสร้าง Thread ขึ้นมา 10 Thread จากนั้นจะทำการประกาศออกมาว่ามีจำนวนกี่ Thread และให้ Thread นั้นเก็บตัวเลขไว้และแสดงผลออกมาตามจำนวนเลขที่ถูกเก็บ

**7. จงทดลองโค้ดภาษาซีต่อไปนี้ ผลลัพธ์ที่ได้จาก Line C และ Line P เป็นอย่างไร**

****



child process เกิดการ fork จากนั้น parent process รอ child process ทำงานจนเสร็จ จะมี Thread ถูกสร้างขึ้นสำหรับ child process และเรียกใช้ฟังก์ชัน runner() ซึ่งมีการกำหนดค่า value = 5 เมื่อจบการทำงานของ child process ที่ parent process จะมีค่า value = 0

**8. จงเขียนโปรแกรมแบบ Multithread โดยมีการทำงานดังนี้**

* Main Thread สร้างเลขจำนวนเต็มแบบสุ่ม 10 ค่า แล้วแสดงออกทางจอภาพ
* Main Thread เตรียมตัวแปร min, max, และ avg สำหรับเก็บค่าต่ำสุด, สูงสุด, และค่าเฉลี่ย
* 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);

}

