รายงาน

เรื่อง Threads

จัดทำโดย

640911049 นายพิพัฒน์ แซ่อึ้ง

640911145 นายคมชาญ ตรีพงศ์พิพัฒน์

640911148 นายมูฮัมมัดซอลีฮีน ฮะแวบือซา

เสนอ

อาจารย์ศักดิ์ระพี ไพศาลนันทน์

รายงานนี้เป็นส่วนหนึ่งของวิชา 618344 วิศวกรรมระบบปฏิบัติการเบื้องต้น

สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์และระบบคอมพิวเตอร์

คณะวิศวกรรมศาสตร์และเทคโนโลยีอุตสาหกรรม

ภาคการศึกษาที่ 1 ปีการศึกษา 2566

มหาวิทยาลัยศิลปากร พระราชวังสนามจันทร์

Thread คืออะไร

Thread คือ หน่วยการทำงานย่อยที่อยู่ใน Process มีการแบ่งปันทรัพยากรต่างๆ ใน Process นั้นๆ โดยปกติ Process ที่มี 1 thread จะเรียกว่า Single thread หรือเรียกว่า Heavy Weight Process ซึ่งมักพบใน OS รุ่นเก่า แต่ถ้า 1 Process มีหลาย thread จะเรียกว่า Multithread หรือ Light Weight Process ซึ่งพบได้ใน OS รุ่นใหม่ที่ใช้กันในปัจจุบันทั่วไป และ Multithread ก็เป็นที่นิยมมากกว่า Single thread

องค์ประกอบภายในThreads (Threads) ประกอบด้วย

1. Threads ID หมายเลขเทรดที่อยู่ในโปรเซส
2. Counter ตัวนับเพื่อติดตามคำสั่งที่จะถูกดำเนินการเป็นลำดับถัดไป (Execute)
3. Register หน่วยความจำเก็บค่าตัวแปรที่ทำงานอยู่ปัจจุบัน
4. Stack เก็บประวัติการทำงาน (Execute)

ทำไมเราต้องใช้ Threads?

Threads ทำงานพร้อมกัน เพิ่มประสิทธิภาพของแอปพลิเคชัน แต่ละ Threads มีสถานะ CPU และสแต็กของตัวเอง แต่พวกเขาแชร์พื้นที่แอดเดรสของโปรเซสและสภาพแวดล้อม

Threads สามารถแบ่งปันข้อมูลทั่วไปทำให้ไม่จำเป็นต้องใช้การสื่อสารระหว่างโปรเซส คล้ายกับโปรเซส Threads ยังมีสถานะเช่นพร้อม, กำลังทำงาน, ถูกบล็อก ฯลฯ

ลำดับความสำคัญสามารถกำหนดให้Threadsเหมือนกับโปรเซส และThreadsที่มีความสำคัญสูงสุดจะถูกกำหนดตารางก่อน

แต่ละ Threads มีบล็อกควบคุม Threads (TCB) ของตัวเอง คล้ายกับโปรเซส การสลับบล็อกเกิดขึ้นสำหรับThreadsและเนื้อหาของทะเบียนถูกบันทึกไว้ใน (TCB) เนื่องจากThreadsแบ่งปันพื้นที่แอดเดรสและทรัพยากรเดียวกัน การประสานงานเพื่อกิจกรรมต่าง ๆ ของThreadsเป็นเรื่องจำเป็นด้วย

ทำไมถึงต้อง Multi-Threading?

Threads เป็นที่รู้จักกันอีกชื่อว่า Light Weight Process แนวคิดคือการทำให้เกิดการทำงานพร้อมกัน โดยการแบ่งกระบวนการออกเป็น Threads หลายๆ Threads เช่น เบราว์เซอร์สามารถมีหลายแท็บที่เป็น Threads ต่าง ๆ MS Word ใช้ Threads หลายตัว: Threads หนึ่งเพื่อจัดรูปแบบข้อความ, Threads อีกตัวเพื่อประมวลผลอินพุต ฯลฯ

Multithreading เป็นเทคนิคที่ใช้ในระบบปฏิบัติการเพื่อเพิ่มประสิทธิภาพและความตอบสนองของระบบคอมพิวเตอร์ การใช้ Multithreading ช่วยให้ Threads หลายตัว (เช่น Light Weight Process)สามารถแบ่งปันทรัพยากรเดียวกันของกระบวนการเดียวกัน เช่น CPU, หน่วยความจำ และอุปกรณ์ I/O ได้

ประโยชน์ของ Multi-Threads

1. การตอบสนอง (Response) ในเรื่องของการทำงานมีการตอบสนองที่ดีกับผู้ใช้ (user) ถ้าการทำงานของโปรแกรมประยุกต์ของผู้ใช้นั้นมีบางส่วนภายในโปรเซสถูกบล๊อกหรือใช้เวลามากเกินไป OS ก็ยังสามารถจัดสรรให้งานอื่นๆภายในโปรเซสนั้นประมวลผลต่อไปได้

2. การใช้ทรัพยากรร่วมกัน (Share Resource) สามารถใช้โค้ด (code) ใช้โปรแกรม (application) และใช้หน่วยความจำ (memory) ร่วมกันระหว่างโปรเซสเดียวกันได้

3. ประหยัด (Economic) ประหยัดการใช้หน่วยความจำในการทำงานของโปรเซสเนื่องจากแต่ละเทรด (Thread) มีการใช้หน่วยความจำของโปรเซสร่วมกัน

4. ด้านโครงสร้างของมัลติเทรด (Multithread Architecture) การเอื้อประโยชน์ด้านโครงสร้างระบบ ที่งานย่อยภายในโปรเซสให้สามารถทำงานร่วมกัน ประสานจังหวะการทำงานและใช้ทรัพยากรของโปรเซสร่วมกันได้

ความแตกต่างระหว่าง Process และ Threads

ความแตกต่างหลักคือ Threads ภายในกระบวนการเดียวกันทำงานในพื้นที่หน่วยความจำที่แบ่งปันกัน ในขณะที่ Process ทำงานในพื้นที่หน่วยความจำแยกกัน Threads ไม่เป็นอิสระจากกันเหมือน Process และเนื่องจากเช่นนั้น Threads แบ่งปันส่วนของรหัส (code section) ส่วนของข้อมูล (data section) และทรัพยากรของระบบปฏิบัติการ (เช่นไฟล์ที่เปิดและสัญญาณ) กับ Threads อื่นๆ แต่เหมือนกับ Process Threads มี program counter,register set และ stack space ของตัวเองด้วย

ประเภทของ Threads

Threads มีสองประเภท ดังนี้

User Level Thread

User-Level Thread คือประเภทของ Threads ที่ไม่ถูกสร้างขึ้นโดยใช้ system calls และ kernel ไม่มีหน้าที่ในการจัดการของ Threads ระดับผู้ใช้ ผู้ใช้สามารถสร้าง User-Level Thread ได้อย่างง่ายดาย ในกรณีที่ User-Level Thread เป็นกระบวนการที่มี Threads เดียว kernel-level thread จะจัดการกับมัน

Kernel Level Thread

Kernel Level Thread เป็นประเภทของ Threads ที่สามารถรู้จักระบบปฏิบัติการได้ง่าย มีตาราง Threads ของตัวเองที่ใช้ในการติดตามของระบบ และระบบปฏิบัติการช่วยในการจัดการ Threads context switching ของ Kernel Level Thread มักจะใช้เวลานานกว่า User-Level Thread เนื่องจากความซับซ้อนของการบริหารจัดการที่ระดับ Kernel

สรุป

Threads คือ การเรียกใช้หน่วยประมวลผลให้เกิดประโยชน์สูงสุด และทำให้การทำงานของโปรแกรมมีประสิทธิภาพมากขึ้นและมีประโยชน์ต่อระบบในปัจจุบันที่เป็น Multicore เพราะสามารถเรียกใช้ Threads หลายๆ Threads ได้พร้อม ๆ กัน ทั้งนี้ Threads ของโพรเซสเดียวกันสามารถจะทำงานแตกต่างกัน แต่ยังมีความเกี่ยวข้องกันบางอย่างและต้องทำงานอยู่ภายใต้สภาพแวดล้อมเดียวกัน Threads เป็นการควบคุมการทำงานภายในของโพรเซส Multithreaded ก็คือโพรเซสจะประกอบไปด้วย Threads ที่ทำหน้าที่ต่างกันแต่ทำงานอยู่บนตำแหน่งหน่วยความจำเดียวกัน ข้อดีของ Multithreaded ประกอบไปด้วย การเพิ่มการตอบสนองต่อผู้ใช้ทั้งแชร์ทรัพยากรในโพรเซส ความประหยัดและความสามารถของงาน ในสถาปัตยกรรมแบบมัลติโพรเซสเซอร์ ในระดับของThreads สำหรับผู้ใช้ คือสิ่งที่โปรแกรมเมอร์มองเห็นและระบบปฏิบัติการKernel จะสนับสนุนจัดการในระดับ Kernel โดยทั่วไปแล้วThreads สำหรับผู้ใช้จะเร็วกว่าในการสร้างและจัดการมากกว่า Threads สำหรับระบบปฏิบัติการและไม่ก้าวก่ายสิ่งที่ Kernel ต้องการ