แผนบริหารการสอนประจำบทที่ 2

ระบบปฏิบัติการเบื้องต้น

(Introduction to Operating System)

วัตถุประสงค์

1. เพื่อเข้าใจเป้าหมายการพัฒนา และหน้าที่ของระบบปฏิบัตการ
2. เพื่อทราบถึงวิวัฒนาการระบบปฏิบัติการ
3. เพื่อสามารถอธิบายการส่วนบริการในระบบปฏิบัติการ
4. เพื่อสามารถอธิบายส่วนต่อประสานระหว่างผู้ใช้และระบบปฏิบัติการ
5. เพื่อสามารถอธิบายการเรียกใช้ระบบ (System Calls)
6. เพื่อสามารถอธิบายส่วนต่อประสานโปรแกรมเรียกใช้ระบบ (APIs)
7. เพื่อสามารถอธิบายความสัมพันธ์ระหว่าง System Calls และ APIs
8. เพื่อสามารถอธิบายประเภทต่าง ๆ ของ System Calls
9. เพื่อสามารถอธิบายโครงสร้างระบบปฏิบัติการ
10. เพื่อสามารถอธิบายกระบวนการเริ่มต้นการทำงานระบบปฏิบัติการ

เนื้อหา

1. เป้าหมาย และหน้าที่ของระบบปฏิบัติการ (Desired Goals and Roles of Operating Systems)
2. วิวัฒนาการระบบปฏิบัติการ (Evolution of Operating Systems)
3. ส่วนบริการในระบบปฏิบัติการ (Operating System Services)
4. ส่วนต่อประสานระหว่างผู้ใช้และระบบปฏิบัติการ (User Opearting-System Interfaces)
5. การเรียกใช้ระบบ (System Calls)
6. ส่วนต่อประสานโปรแกรมประยุกต์ (Application Programming Interfaces : APIs)
7. ความสัมพันธ์ระหว่างการเรียกใช้ระบบและส่วนต่อประสานโปรแกรม (Relation between System calls and APIs)
8. ประเภทของการเรียกใช้ระบบ (Types of System calls)

9. โครงสร้างระบบปฏิบัติการ (Operating-System Structure)
10. กระบวนการเริ่มต้นการทำงานระบบปฏิบัติการ (System Boot)

กิจกรรมการเรียนการสอน

1. บรรยาย
2. แสดงตัวอย่างฮาร์ดแวร์
3. วิดีโออนิเมชั่นการทำงานคอมพิวเตอร์
4. ค้นคว้าด้วยตัวเอง

สื่อการเรียนการสอน

1. เอกสารประกอบการสอน
2. สไลด์การสอน
3. โปรแกรม google class room
4. โปรแกรม facebook

การวัดผลและการประเมินผล

1. แบบฝึกหัดท้ายบท
2. การถามตอบในชั้นเรียน
3. รายงานวิวัฒนาการระบบปฏิบัติการชนิดต่าง ๆ

บทที่ 2
ระบบปฏิบัติการเบื้องต้น
(Introduction to Operating System)

จากหน้าที่ของระบบปฏิบัติการ เป็นเสมือนตัวกลางคอยบริการผู้ใช้เพื่อเข้าถึงอุปกรณ์ต่าง ๆ ของคอมพิวเตอร์ด้วยการจัดสรรทรัพยากรฮาร์ดแวร์อย่างเหมาะสม การบริการผู้ใช้โดยผ่านโปรแกรมประยุกต์ (application program) หรือส่วนต่อประสานผู้ใช้ (user interface) ในการจัดการฮาร์ดแวร์

2.1 เป้าหมาย และหน้าที่ของระบบปฏิบัติการ (Desired Goals and Roles of Operating Systems)

ระบบปฏิบัติการ ถูกพัฒนาขึ้นภายใต้วัตถุประสงค์ดังนี้

1) ความสะดวกสบาย (Convenience): เพื่อให้คอมพิวเตอร์ใช้งานง่ายและสะดวก

2) ประสิทธิภาพ (Efficiency): เพื่อให้การเข้าถึงและใช้งานทรัพยากรต่าง ๆ ในคอมพิวเตอร์ได้อย่างมีประสิทธิภาพ

3) ความสามารถในการพัฒนา (Ability to evolve): เพื่อให้การพัฒนา ทดสอบ และนำระบบปฏิบัติการ ที่ได้รับการปรับปรุงสู่การใช้งานได้สะดวก

โดยเราสามารถจำแนกหน้าที่ของระบบปฏิบัติการ ได้ดังต่อไปนี้

1) การพัฒนาโปรแกรม (Program development): ระบบปฏิบัติการจำเป็นต้องมีเครื่องมือ และบริการต่าง ๆ เพื่อช่วยเหลือในการเขียนโปรแกรมให้แก่นักพัฒนาโปรแกรม เช่น บรรณาธิกรณ์ (editor) หรือส่วนตรวจสอบจุดบกพร่องของโปรแกรม (debugger) สำหรับนักพัฒนาโปรแกรม

2) การประมวลผลโปรแกรม (Program execution): ระบบปฏิบัติการมีหน้าที่สำคัญในการจัดการงาน (task) ต่าง ๆ ให้กับผู้ใช้ โดยการจัดสรรทรัพยากร จัดเรียงลำดับชุดคำสั่งของโปรแกรมอย่างเหมาะสม และประมวลผลชุดคำสั่งและชุดข้อมูลต่าง ๆ

3) การเข้าถึงอุปกรณ์อินพุทเอาท์พุท (Access to I/O devices): ระบบปฏิบัติการมีหน้าที่เป็นตัวกลางสื่อสารกับอุปกรณ์อินพุทเอาท์พุทประเภทต่าง ๆ ที่มีรูปแบบแตกต่างกันผ่านโปรแกรมขับ (program driver) เฉพาะอุปกรณ์นั้น ๆ เพื่อให้นักพัฒนาโปรแกรมเข้าถึงและใช้งานอุปกรณ์อินพุทเอาท์พุทได้ง่าย

4) การควบคุมการใช้งานไฟล์ข้อมูล (Controlled access to files): ระบบปฏิบัติการมีหน้าที่ควบคุมเข้าถึงไฟล์ข้อมูลที่ถูกจัดเก็บอย่างมีระบบ และมีกลไกป้องกันไฟล์ข้อมูลไม่ให้มีการแก้ไขเปลี่ยนแปลงจากผู้ที่ไม่สิทธิ์

5) การเข้าถึงระบบ (System access): ในกรณีของระบบที่มีการแชร์ หรือระบบที่สามารถใช้งานได้จากที่อื่นในระยะไกล ระบบปฏิบัติการมีหน้าที่ควบคุมการเข้าถึงระบบของผู้ใช้ รักษาความปลอดภัย และควบคุมการใช้งานทรัพยากรของระบบอย่างเหมาะสม

6) การตรวจสอบความผิดพลาด และการตอบสนอง (Error detection and response): โดยทั่วไปข้อผิดพลาด (error) สามารถเกิดขึ้นได้จากความผิดพลาดของฮาร์ดแวร์ การคำนวนทางคณิตศาสตร์ และการเข้าถึงตำแหน่งหน่วยความจำที่ไม่ได้รับอนุญาต ดังนั้นระบบปฏิบัติการ มีหน้าที่ในการตรวจตราการเกิดข้อผิดพลาด และมีมาตรการสนองตอบอย่างเหมาะสม เช่น การพยายามหยุดการทำงานของโปรแกรมที่เป็นสาเหตุของความผิดพลาด พยายามประมวลผลโปรแกรมอื่น หรือเพียงรายงานให้ผู้ใช้งานได้รับทราบ เป็นต้น

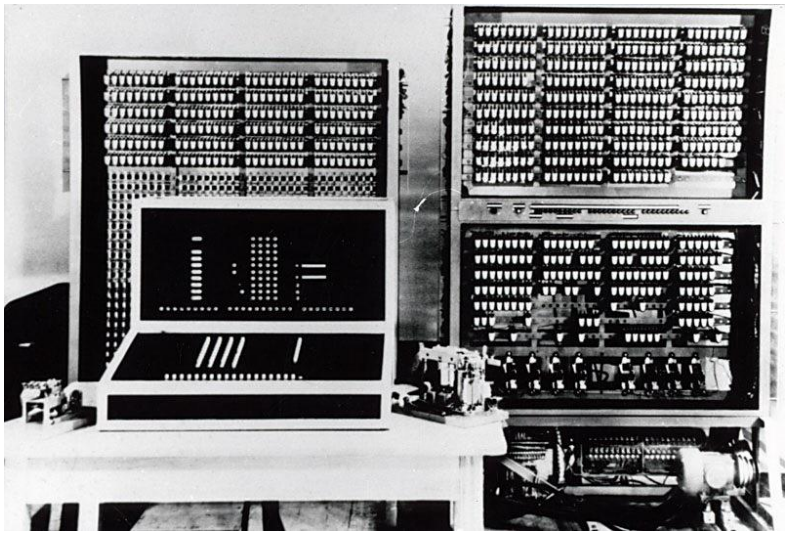
7) การคำนวนบัญชี (Accounting): ระบบปฏิบัติการมีหน้าที่ให้การเก็บรวบรวมสถิติการใช้งานทรัพยากร และจัดทำรายงาน เพื่อแสดงปริมาณการใช้งานทรัพยากรระบบคอมพิวเตอร์

2.2 วิวัฒนาการระบบปฏิบัติการ (Evolution of Operating Systems)

ระบบปฏิบัติการมีวิวัฒนาการควบคู่กับวิวัฒนาการทางด้านคอมพิวเตอร์ฮาร์ดแวร์ตั้งแต่ในอดีตจนกระทั่งในปัจจุบัน โดยสามารถแบ่งเป็นยุคได้ดังนี้

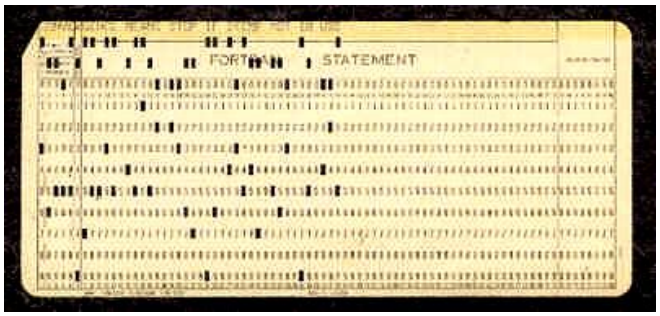
2.2.1 เริ่มจากยุคก่อนระบบปฏิบัติการ (Early Generation)

ซึ่งอยู่ในช่วงกลางคริสต์ทศวรรษ 1940s – 1950s (พ.ศ.2483 - 2493) เรียกว่ายุค การประมวลผลแบบเรียงลำดับ (Serial Processing) เป็นช่วงต้นที่เครื่องคอมพิวเตอร์ทำงานโดยไม่มีระบบปฏิบัติการมาควบคุมดังรูปที่ 2-1 การทำงานของคอมพิวเตอร์ต้องใช้คนทุกในขั้นตอน เช่นการนำบัตรเจาะรูเข้าสู่เครื่อง กดปุ่มให้อ่านบัตร ซึ่งบัตรเจาะรูดังรูปที่ 2-2 จะบรรจุชุดคำสั่งต่าง ๆ ให้คอมพิวเตอร์ทำงานที่ละขั้นตอน และการแสดงผลการทำงานเป็นเพียงหลอดไฟติดหรือดับเพื่อบ่งบอกสถานะ หรือพิมพ์ออกเครื่องพิมพ์ แทนการใช้เครื่องพิมพ์ดีด การทำงานที่ละขั้น ๆ เช่นนี้จึงเป็นที่มาชื่อเรียก serial processing



รูปที่ 2-1: เครื่องคอมพิวเตอร์ Zuze Z3

ที่มา: **(Timeline of Computer History, 2558)**



รูปที่ 2-2: บัตรเจาะรูบรรจุคำสั่งและข้อมูลในการประมวลผล

ที่มา: **(Operating System Evolution, 2557)**

2.2.2 ยุคแรกของระบบปฏิบัติการ (1st Generation)

ยุคแรกของระบบปฏิบัติการอยู่ในช่วงระหว่างกลางคริสต์ทศวรรษ 1950s – ต้นคริสต์ทศวรรษ 1960s (พ.ศ. 2493 - 2503) เรียกว่ายุค Simple Batch Systems ซึ่งเป็นชุดคำสั่งแบบอย่างง่ายทำงานบนคอมพิวเตอร์ในยุคเดียวกันเช่น IBM 701 ซึ่งเป็นเครื่องคอมพิวเตอร์เครื่องแรก ๆ ที่มีระบบปฏิบัติการ ในลักษณะเป็นกลุ่มคำสั่ง (batch system) สั่งงานระบบคอมพิวเตอร์ โดยผู้ปฏิบัติการ (operator) รวบรวมบัตรงานจากผู้ใช้ (users) หลาย ๆ งานเข้าด้วยกันเป็นกลุ่ม และงานที่รวบรวมจะถูกจัดเรียง (batch of jobs) และนำเข้างานทั้งหมดเข้าสู่เครื่องคอมพิวเตอร์เพื่อประมวลผล ดังนั้น ระบบปฏิบัติการ ในยุคนี้ถูกเรียกว่า “monitor” ซึ่งทำหน้าที่อ่านกลุ่มคำสั่งและข้อมูลจาก batch มาประมวลผล

รูปที่ 2-3: IBM 701

ที่มา: **(Timeline of Computer History, 2558)**

ปัญหาหลักของระบบคอมพิวเตอร์ในยุคนี้คือ ประสิทธิภาพของการทำงานซีพียู สาเหตุเกิดจากการประมวลผลโปรแกรมมีการทำงานตามลำดับซีพียู จึงมีช่วงเวลาว่างเยอะจากการรอการทำงานส่วนอื่น ๆ ให้ทำงานแล้วเสร็จจึงกลับมาให้ ซีพียู ทำงานต่อไปได้

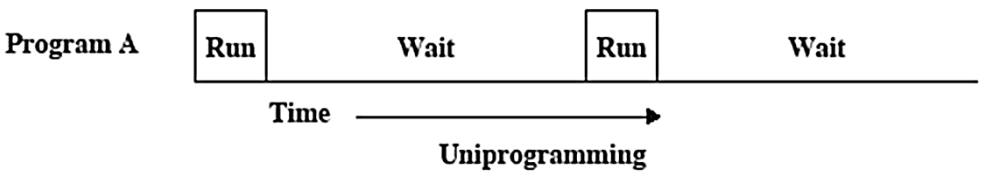
พิจารณาการประมวลผลตามภาระกิจดังนี้

Read one record from file	ใช้เวลา	$15\ \mu s$
Execute 100 instructions	ใช้เวลา	$1\ \mu s$
Write one record to file	ใช้เวลา	$\underline{15\ \mu s}$
	รวมใช้เวลา	$31\ \mu s$

ดังนั้นประสิทธิภาพการทำงานของ ซีพียู สามารถคำนวนได้จาก

$$\text{Percentage of CPU utilisation} = \frac{1}{31} = 0.032 = 32\%$$

จะเห็นได้ว่า ซีพียู มีการทำงานได้อย่างรวดเร็ว โดยประมวลผล 100 คำสั่ง ใช้เวลาเพียง 1 ไมโครวินาที (μs) แต่เวลาที่ใช้ไปสำหรับการอ่านและเขียนของอุปกรณ์นำเข้าและส่งออกข้อมูล (IO device) ใช้เวลานานกว่ามาก ซีพียู จึงจำต้องรอการทำงานของอุปกรณ์เหล่านั้นเสร็จสิ้นเสียก่อนจึงจะดำเนินการต่อได้ และเป็นสาเหตุให้ประสิทธิภาพการทำงานต่ำ ดังรูปที่ 2-4



รูปที่ 2-4: ช่วงเวลาการทำงานของซีพียู

ที่มา: **(Introduction to Operating System, 2558)**

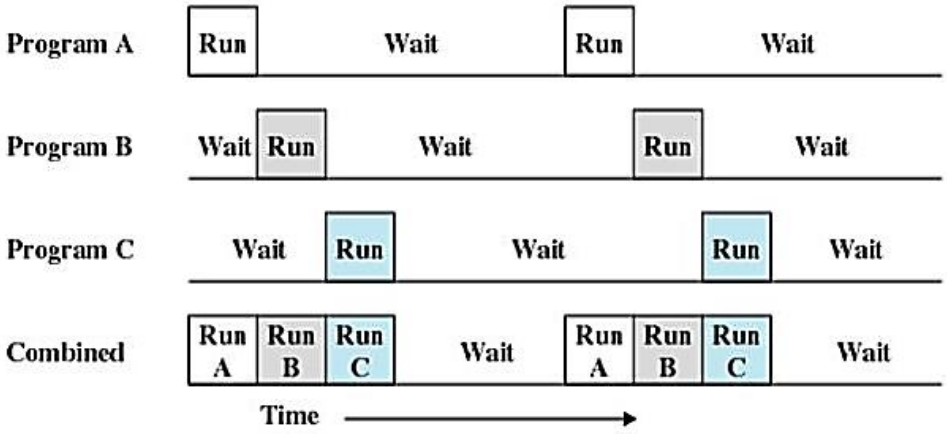
2.2.3 ยุคที่สอง (2nd Generation)

เรียกว่ายุค Multiprogrammed Batch Systems อยู่ในช่วงต้นปีคริสต์ทศวรรษ 1960s – กลางคริสต์ทศวรรษ 1960s (พ.ศ. 2503 - 2508) มีการปรับปรุงเพื่อเพิ่มประสิทธิภาพการใช้งานของ ซีพียู จึงมีการออกแบบระบบปฏิบัติการ ให้มีการทำงานพร้อมกันหลายโปรแกรมสลับกันไป โดยโหลดหลาย ๆ โปรแกรมเข้าสู่หน่วยความจำ และแบ่งการใช้ทรัพยากรร่วมกัน การทำงานลักษณะนี้เป็นที่รู้จักกันในชื่อ multitasking ซึ่งเป็นต้นแบบของการทำงานระบบปฏิบัติการ ในยุคปัจจุบัน ระบบคอมพิวเตอร์ในยุคที่รองรับการทำงานลักษณะนี้เช่น IBM 1401 ดังรูปที่ 2-5

การทำงานในรูปแบบ multiprogrammed batch system นั้นเจ้าหน้าที่ปฏิบัติการรวบรวมงานในรูปของโปรแกรมชุดคำสั่ง (batch) จากผู้ใช้ และทำการบรรจุงานเหล่านี้ให้แก่ระบบคอมพิวเตอร์ในคราวเดียวกันเพื่อให้ ซีพียู ประมวลผล ดังในรูปที่ 2-6 มีโปรแกรมจำนวน 3 โปรแกรมในระบบคอมพิวเตอร์ ทำงานสลับกัน เมื่อ ซีพียู ประมวลคำสั่งของชุดโปรแกรม A และเข้าสู่ภาวะรอการทำงานส่วนอื่น ชุดคำสั่งโปรแกรม B จะมอบให้ ซีพียู ประมวลผลต่อไป ทำนองเดียวกัน เมื่อซีพียูเข้าสู่ภาวะคอยการทำงานจากส่วนอื่น โปรแกรม C จะถูกบรรจุให้ ซีพียู ประมวลผล สลับการทำงานของโปรแกรมเช่นนี้จนกระทั่งเสร็จงาน ดังนั้นประสิทธิภาพการใช้งาน ซีพียู จึงเพิ่มขึ้น

รูปที่ 2-5: IBM 1401

ที่มา: **(Operating System Evolution, 2557)**



รูปที่ 2-6: การทำงานแบบ Multiprogramming

ที่มา: **(Introduction to Operating System, 2558)**

2.2.4 ยุคที่สาม (3rd Generation)

เข้าสู่ยุค Time-Sharing System ซึ่งอยู่ในช่วงกลางคริสต์ทศวรรษ 1960s – ปลายคริสต์ทศวรรษ 1960s (พ.ศ. 2508 - 2513) เป็นยุคที่มีการออกแบบระบบปฏิบัติการให้ตอบสนองต่อการทำงานของผู้ใช้หลาย ๆ คน ผู้ใช้จะสั่งการให้คอมพิวเตอร์ทำงานผ่านเทอร์มินอล (terminal) ซึ่งมีเพียงคีย์บอร์ดและจอภาพดังรูปที่ 2-7 โดยจะต้องส่งข้อมูลไปประมวลผล ณ ระบบคอมพิวเตอร์ซึ่งมีหน่วยประมวลผลกลาง ดังรูปที่ 2-8 ระบบปฏิบัติการจะทำการตอบสนองต่อผู้ใช้หลาย ๆ คนผ่านเมอร์มินอลเสมือนทำงานได้พร้อม ๆ กันด้วยการแบ่งเวลากันใช้

รูปที่ 2-7: เทอร์มินอล

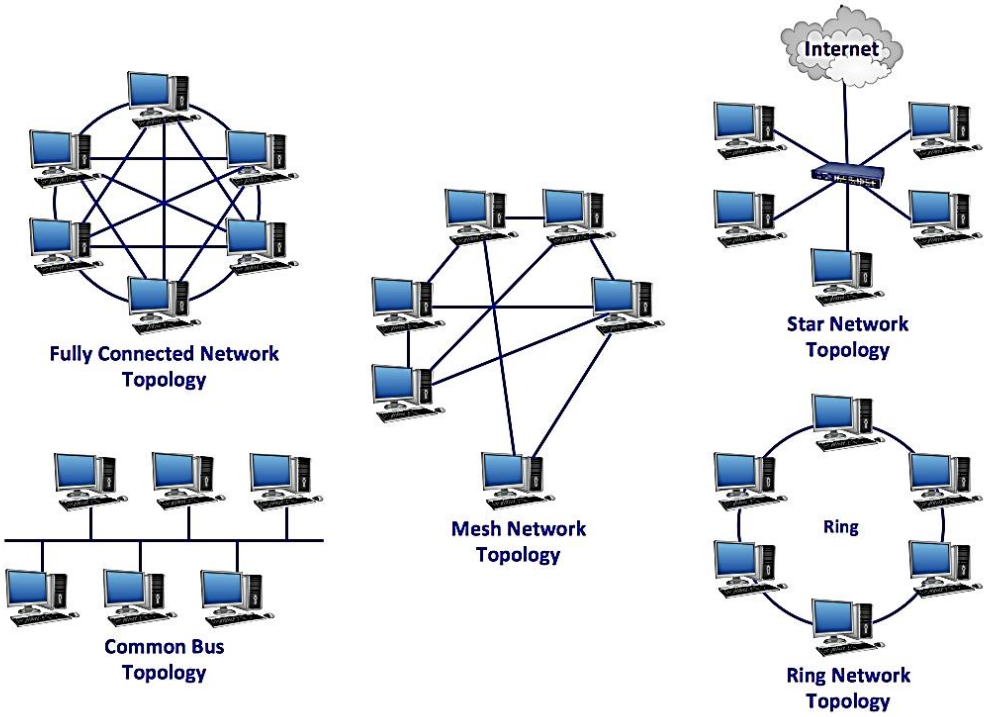
ที่มา: **(Degnan, 2554)**

รูปที่ 2-8: IBM 360

ที่มา: **(Operating System Evolution, 2557)**

2.2.5 ยุคที่สี่ (4th Generation)

ยุคนี้เริ่มต้นเมื่อช่วงปลายคริสต์ทศวรรษ 1960s (พ.ศ. 2513) กระทั่งถึงปัจจุบัน เรียกว่ายุคนี้ว่ายุคแห่งเครือข่ายคอมพิวเตอร์ Computer Networking ดังรูปที่ 2-9 ระบบปฏิบัติการสามารถรองรับการต่อเชื่อมคอมพิวเตอร์โยงใยเป็นเครือข่ายไม่มีที่สิ้นสุดดังใน สามารถแบ่งปันทรัพยากรระหว่างคอมพิวเตอร์จำนวนมาก

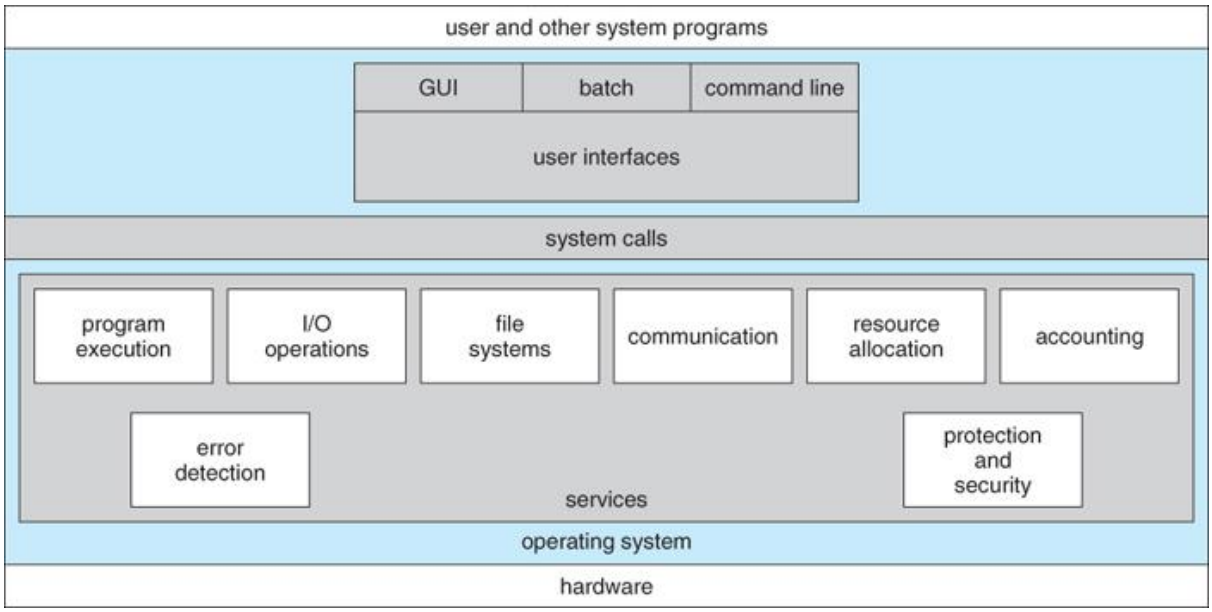


รูปที่ 2-9: เครือข่ายคอมพิวเตอร์

ที่มา: (Network Topology, 2559)

2.3 ส่วนบริการในระบบปฏิบัติการ (Operating-System Services)

ระบบปฏิบัติการมีหน้าที่ที่สำคัญในการจัดสภาพแวดล้อมต่าง ๆ ที่จำเป็นบริการให้แก่ผู้ใช้ทั้งในระดับผู้พัฒนาโปรแกรม (programmers) และผู้ใช้งานโปรแกรม (users) ในการเข้าใช้ทรัพยากรของระบบคอมพิวเตอร์ และยังบริการแก่โปรแกรมให้สามารถประมวลผลได้อย่างมีประสิทธิภาพ ในระบบปฏิบัติการที่มีใช้งานอยู่ในปัจจุบันมีความแตกต่างในการให้บริการ อย่างไรก็ตามบริการหลักที่จำเป็นสำหรับทุก ๆ ระบบปฏิบัติการสามารถแสดงได้ในรูปที่ 2-10 ซึ่งสามารถแบ่งออกเป็น 2 กลุ่มคือบริการสำหรับผู้ใช้ และกลุ่มบริการสำหรับระบบปฏิบัติการเอง



รูปที่ 2-10: บริการหลักในระบบปฏิบัติการ

ที่มา: (Silberschatz, Galvin, & Gagne, 2010, p. 50)

2.3.1 บริการสำหรับผู้ใช้ (User Services)

บริการในกลุ่มแรกที่ระบบปฏิบัติการจัดเตรียมฟังก์ชันใช้งานไว้ให้เป็นประโยชน์สำหรับผู้ใช้มีรายละเอียดดังนี้

1) User interface (UI): คือส่วนเชื่อมประสานกับผู้ใช้ ระบบปฏิบัติการทุกประเภทมีบริการนี้แก่ผู้ใช้ในรูปแบบที่แตกต่างกันเช่น Graphical User Interface (GUI), Command-Line Interface (CLI)

หรือ Batch Interface ซึ่ง UI เป็นช่องทางติดต่อเพื่อให้ผู้ใช้ได้เข้าถึงอุปกรณ์ฮาร์ดแวร์ หรือโปรแกรมต่าง ๆ ทั้งโปรแกรมประยุกต์ และโปรแกรมระบบในคอมพิวเตอร์ได้

2) Program Execution: ส่วนบริการประมวลผล โดยระบบปฏิบัติการทำหน้าที่บรรจุโปรแกรมที่ต้องการทำงาน (load) ด้วยการดึงชุดคำสั่งจากแหล่งจัดเก็บข้อมูล (storage) เข้าสู่หน่วยความจำหลัก (ram) และทำการประมวลผล นอกจากนี้ในส่วนบริการจะต้องสามารถหยุดการทำงานของโปรแกรมที่กำลังประมวลผล ไม่ว่าจะเป็นการหยุดเนื่องจากเสร็จงานหรือหยุดเนื่องจากเกิดข้อผิดพลาดก็ตาม

3) I/O operation: เป็นส่วนที่ให้บริการเข้าถึงอุปกรณ์รับส่งข้อมูล (I/O devices) เช่น CD/DVD โดยระบบปฏิบัติการทำหน้าที่เป็นตัวกลางให้แก่ผู้ใช้สั่งงานอุปกรณ์ I/O ซึ่งอุปกรณ์ต่าง ๆ มีวิธีการและใช้สัญญาณควบคุมที่แตกต่างกัน

4) File-system manipulation: เป็นส่วนที่ให้บริการเข้าถึงไฟล์ต่าง ๆ ในระบบคอมพิวเตอร์ ระบบปฏิบัติการทำหน้าที่จัดการไฟล์ให้อยู่ในโครงสร้างที่เข้าถึงได้รวดเร็ว สามารถเขียน อ่าน หรือลบ ทำหน้าที่จัดแบ่งไดเร็กทอรี่ให้แก่ไฟล์ นอกจากนี้ทำหน้าที่รักษาความปลอดภัยให้แก่ข้อมูลในไฟล์จากการเข้าถึงของผู้ใช้ที่ไม่มีสิทธิ

5) Communications: เป็นส่วนที่ระบบปฏิบัติการให้บริการการสื่อสาร แลกเปลี่ยนข้อมูลซึ่งอาจเกิดขึ้นระหว่าง processes ในเครื่องคอมพิวเตอร์เดียวกัน หรือเป็นการสื่อสารเพื่อแลกเปลี่ยนข้อมูลระหว่างระบบคอมพิวเตอร์ การสื่อสารอาจกระทำผ่านการแชร์ หรือการส่งข้อมูลในรูปของข้อความไปให้ยังปลายทางก็ได้

6) Error detection: ระบบปฏิบัติการมีบริการตรวจสอบเฝ้าระวังการเกิดข้อผิดพลาดใด ๆ ที่อาจเกิดขึ้น ซึ่งความผิดพลาดเกิดขึ้นจากหลายสาเหตุเช่นจากอุปกรณ์ I/O หรือ จากหน่วยความจำ ทั้งนี้ระบบปฏิบัติการ มีการตอบสนองที่เหมาะสมระบบปฏิบัติการ ยังมีส่วนช่วยให้ผู้ใช้สามารถแก้ไขข้อผิดพลาดต่าง ๆ ได้อย่างมีประสิทธิภาพเช่น สนับสนุนเครื่องมือการตรวจสอบหน่วยความจำว่ามีจุดเสียหรือไม่

2.3.2 บริการสำหรับระบบ (System Services)

บริการในกลุ่มที่สองเป็นบริการสำหรับระบบปฏิบัติการเอง โดยมีเป้าหมายเพื่อให้ระบบปฏิบัติการสามารรถรองรับการใช้งานของผู้ใช้หลาย ๆ ผู้ใช้ และการใช้ทรัพยากรในระบบคอมพิวเตอร์ได้อย่างมีประสิทธิภาพ บริการดังกล่าวมีดังต่อไปนี้

1) Resource allocation: เป็นบริการในส่วนการจัดสรรทรัพยากรให้แก่ผู้ใช้ซึ่งเข้าใช้งานระบบคอมพิวเตอร์พร้อมกันหลาย ๆ คน หรือเป็นการจัดสรรทรัพยากรให้แก่การทำงานหลาย ๆ งานพร้อมกัน ส่วนนี้ยังรวมถึงการจัดตารางการทำงานของ ซีพียู ให้มีประสิทธิภาพสูงสุดด้วย

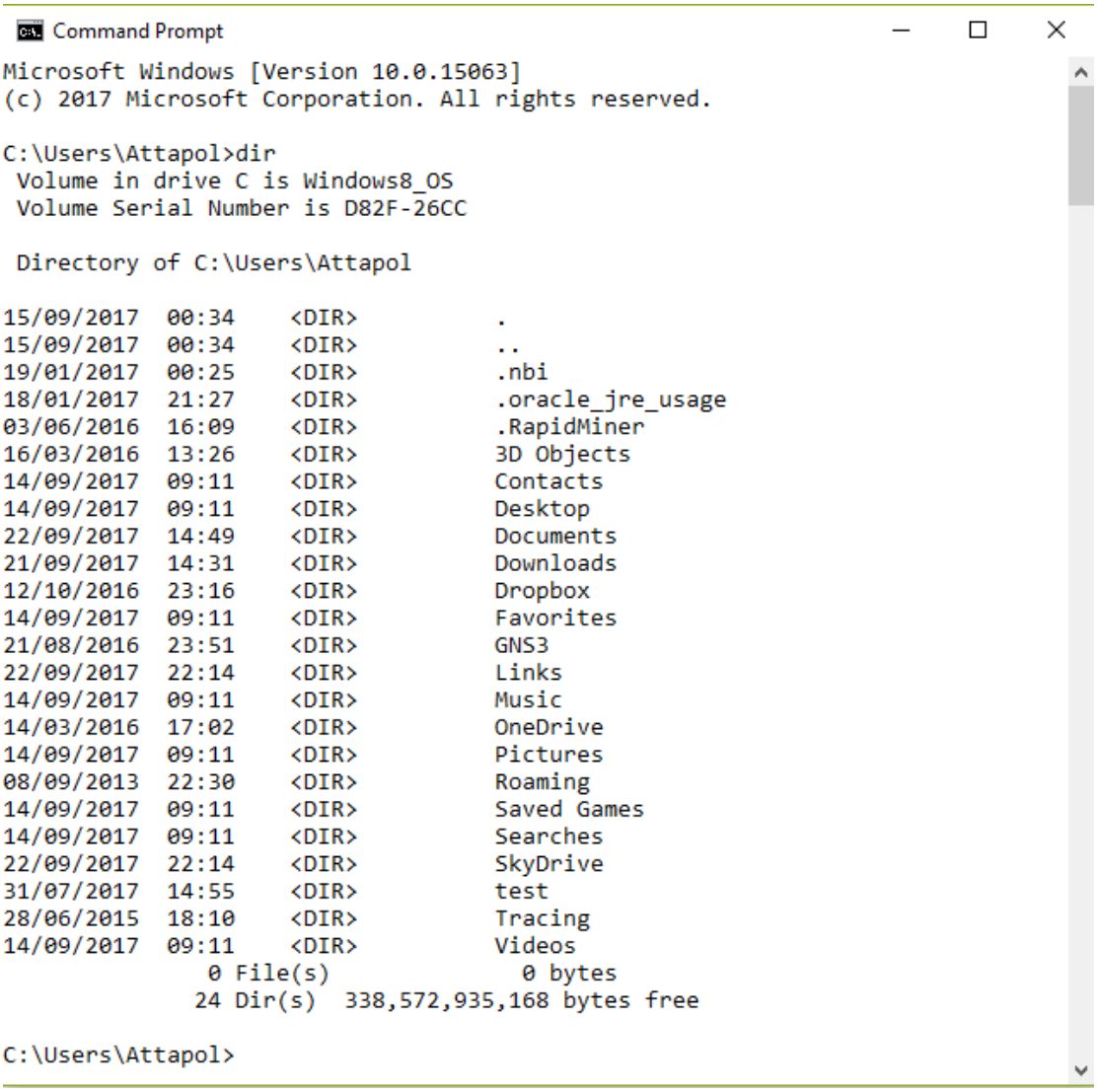
2) Accounting: เป็นการบริการให้แก่ผู้ดูแลระบบ สามารถบันทึกปริมาณการใช้ทรัพยากรของผู้ใช้ เช่นการใช้งานอินเทอร์เน็ต หรือการใช้เนื้อที่เก็บข้อมูล ทั้งนี้เพื่อวัตถุประสงค์ในการคิดค่าใช้จ่าย นอกจากนี้ยังมีประโยชน์ในการจัดทำรายงานสถิติ เพื่อวางแผนพัฒนาระบบในอนาคต

3) Protection and security: เป็นการบริการด้านความปลอดภัย โดยเฉพาะความปลอดภัยของข้อมูล เนื่องจากการใช้งานในปัจจุบัน ผู้ใช้หลาย ๆ คนอาจเข้ามาร่วมใช้ทรัพยากรของระบบคอมพิวเตอร์ มีการเก็บบันทึกข้อมูลซึ่งผู้อื่นไม่มีสิทธิเข้าถึงได้ ดังนั้นการป้องกันและควบคุมสิทธิ์การใช้งานทรัพยากรของผู้ใช้ให้มีขอบเขตอย่างเหมาะสมจึงเป็นหน้าทีและบริการที่สำคัญของระบบปฏิบัติการในยุคปัจจุบัน

2.4 ส่วนต่อประสานระหว่างผู้ใช้และระบบปฏิบัติการ (User Operating-System Interface)

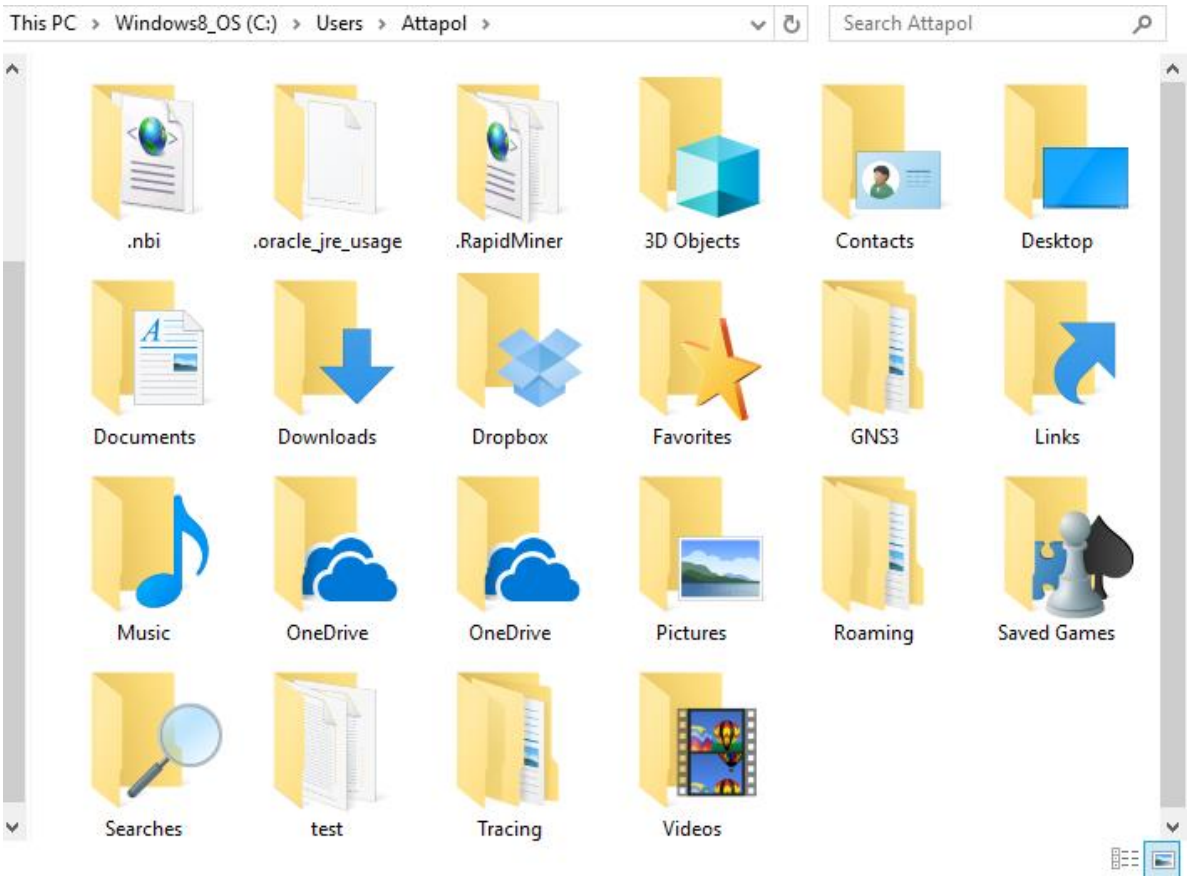
จากรูปที่ 2-10 ส่วนสำคัญของระบบปฏิบัติการคือส่วนต่อประสานระหว่างผู้ใช้และระบบปฏิบัติการ ซึ่งเป็นช่องทางในการสั่งงานและรับทราบผลการประมวลผล มีรายละเอียดดังนี้

1) Command-Line Interface (CLI): ส่วนต่อประสานแบบคำสั่งทีละบรรทัด โดยผู้ใช้สามารถสั่งการคอมพิวเตอร์ผ่านการพิมพ์คำสั่งผ่านส่วนประสานเรียกว่า shell เช่น MS-Dos หรือ C shell ชุดคำสั่งอาจอยู่ในลักษณะรหัส หรือคำสั่งในรูปแบบเฉพาะ ตัวอย่างเช่นแสดงในรูปที่ 2-11 ผู้ใช้สั่งงานด้วยคำสั่ง dir ผ่าน CLI บนระบบปฏิบัติการ Windows เพื่อต้องการแสดงสิ่งที่อยู่ภายในไดเร็กทอรี่ปัจจุบัน ระบบปฏิบัติการจะทำการประมวลคำสั่งทีละคำสั่งตามที่ผู้ใช้งานสั่งการและแสดงผลสำหรับคำสั่งนั้น ๆ



รูปที่ 2-11: Command Line Interface ของระบบปฏิบัติการ Windows ในแบบ MS-Dos

2) Graphical User Interface (GUI): ส่วนต่อประสานแบบกราฟิก ซึ่งเป็นส่วนประสานที่อำนวยความสะดวกให้แก่ผู้ใช้โดยไม่ต้องจดจำคำสั่งต่าง ๆ อย่างเช่น CLI การสั่งการผ่านเมนู หรือรูปภาพในลักณะไอคอน โดยมีการจัดหมวดหมู่ไว้อย่างมีระบบ GUI แรกเป็นของ Xerox พัฒนาขึ้นเมื่อปี ค.ศ.1970s จากนั้น Apple พัฒนาในปี ค.ศ.1980s และ Window version 1.0 พัฒนาขึ้นมาใช้ควบคู่กับ MS-DOS ที่เป็น CLI ในรูปที่ 2-12 แสดงตัวอย่าง GUI ในระบบปฏิบัติการ Windows ซึ่งผู้ใช้สามารถเรียกดูรายละเอียดในไดเร็กทอรี่โดยใช้เมาส์เป็นเครื่องมือในการสั่งงาน



รูปที่ 2-12: Graphical User Interface ในระบบปฏิบัติการ Windows

2.5 การเรียกใช้ระบบ (System Calls)

การเรียกใช้ระบบ (System calls) คือ ชุดโปรแกรมที่ระบบปฏิบัติการ จัดเตรียมไว้บริการให้แก่ผู้ใช้ในการเข้าถึงฮาร์ดแวร์และทรัพยากรต่าง ๆ ของระบบคอมพิวเตอร์ โดยทั่วไปถูกพัฒนาด้วยภาษา C/C++ เพื่อ

1 เปิด command prompt ใน administrator mode → cmd

2 สร้างไดเร็กทอรี่ Test → C:\Users> md Test

3 เข้าสู่ไดเร็กทอรี่ Test → C:\Users> cd Test

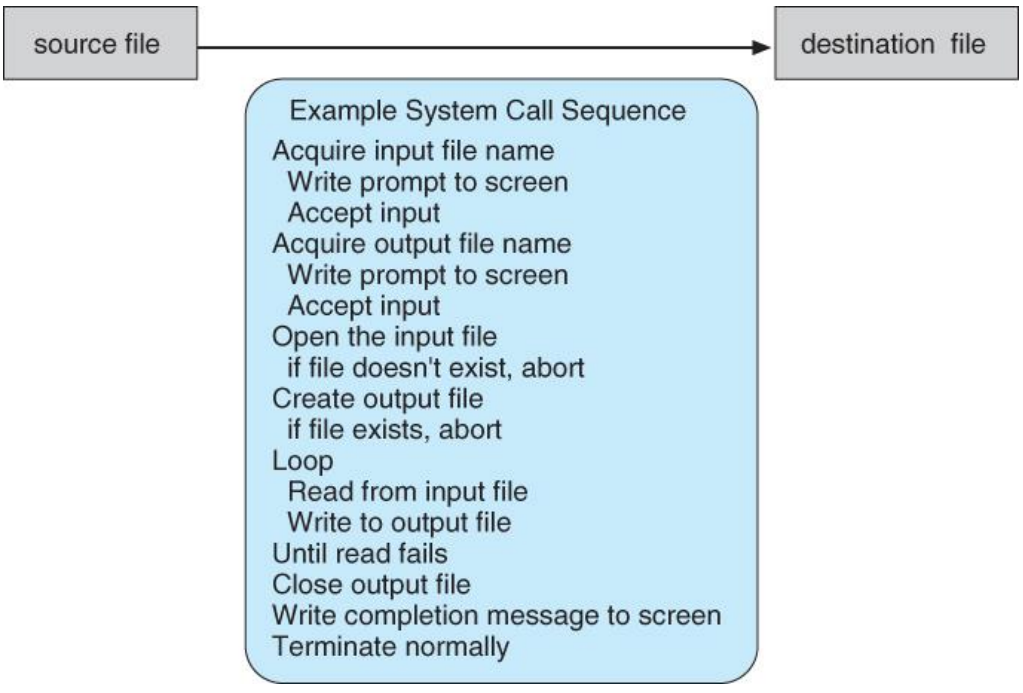
4 สร้างไฟล์และใส่ข้อความในไฟล์ src.txt → C:\Users\Test> echo Welcome to OS > src.txt

5 สำเนาไฟล์ src.txt ไป dst.txt → C:\Users\Test> copy src.txt dst.txt

ทำความเข้าใจ system calls ในการเข้าใช้ทรัพยากร พิจารณาจากตัวอย่างการเรียกใช้ระบบ ผ่าน shell ในระบบปฏิบัติการ ms-dos เพื่อทำสำเนาไฟล์ ซึ่งมีขั้นตอนดังนี้

ในตัวอย่างด้านบนนี้ผู้ใช้สร้างไดเร็กทอรี่ใหม่ชื่อ Test ด้วยคำสั่ง md (บรรทัดที่ 2) จากนั้นเข้าไปยังไดเร็กทอรี่ที่สร้างไว้เพื่อจะสร้างไฟล์ชื่อ src.txt ผู้ใช้สร้างไฟล์ด้วยคำสั่ง echo โดยให้ไฟล์นี้มีข้อความ Welcome to OS ไว้ที่ต้นของไฟล์ (บรรทัดที่ 4) ระบบปฏิบัติการจะสร้างไฟล์ชื่อดังกล่าวเก็บไว้ในตำแหน่งไดเร็กทอรี่ปัจจุบัน จากนั้นผู้ใช้ทำสำเนาข้อมูลจากไฟล์ src.txt ไปยังไฟล์ใหม่ชื่อ dst.txt ด้วยคำสั่ง copy (บรรทัดที่ 5) ระบบปฏิบัติการจะทำการสำเนาข้อมูลและเก็บข้อมูลดังกล่าวไว้ให้ตามที่ต้องการ

สิ่งผู้ใช้สั่งการและรับผลการทำงานผ่าน user interface (UI) นั้นถือว่าอยู่ในระดับด้านบน การทำงานของระบบปฏิบัติการด้านล่างผ่าน system calls สามารถอธิบายขั้นตอนการทำสำเนาจากไฟล์ต้นทาง (source file) ไปยังไฟล์ปลายทาง (destination file) ตามลำดับในตัวอย่างได้ดังรูปที่ 2-13



รูปที่ 2-13: ตัวอย่างการเรียกใช้งาน system calls

ที่มา: (Silberschatz, Galvin, & Gagne, 2010)

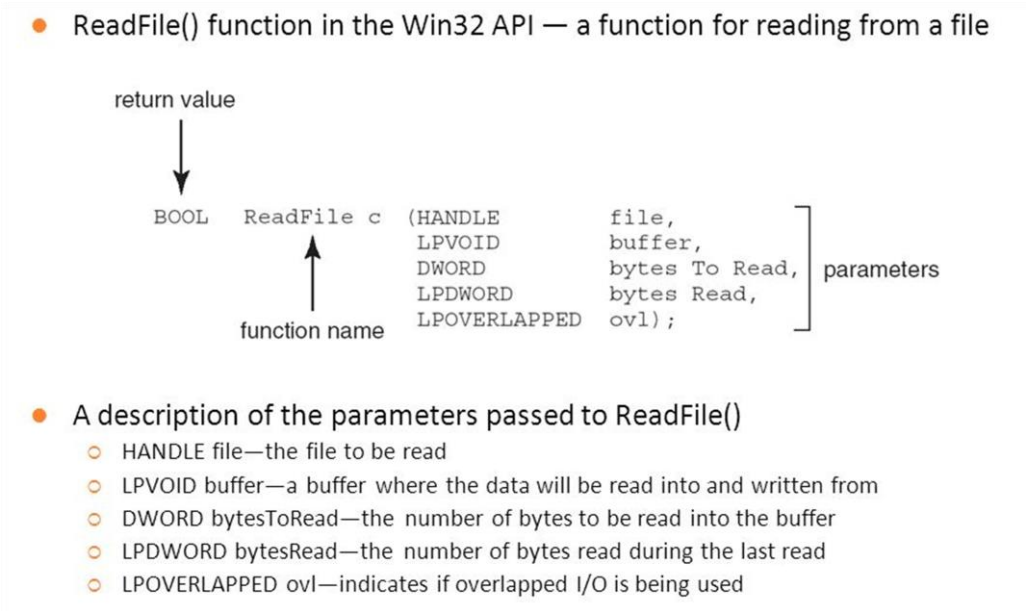
เริ่มต้นสิ่งที่ระบบปฏิบัติการต้องการคือระบุไฟล์ต้นทาง และไฟล์ปลายทางที่ต้องการทำสำเนา ดังนั้นระบบปฏิบัติการจึงเรียกใช้การเรียกใช้ระบบ เพื่อเตรียมการรับชื่อไฟล์จากผู้ใช้ (พิจารณาบรรทัดที่ 5 ประกอบ) จากนั้นระบบจะเรียกการเรียกใช้ระบบ เพื่อทำการเปิดไฟล์ต้นทางเช่น Open() หากไม่พบไฟล์ดังกล่าวระบบ

จะเรียกการเรียกใช้ระบบ แสดงข้อความไปยังผู้ใช้และหยุดกระบวนการสำเนา หากไม่พบความผิดปกติ CreateFile() ซึ่งเป็นการเรียกใช้ระบบ เพื่อสร้างไฟล์ปลายทาง จากนั้นระบบปฏิบัติการทำการทะยอยอ่านข้อมูลจากไฟล์ต้นทาง (src.txt) และเขียนข้อมูลไปยังไฟล์ปลายทาง (dst.txt) ด้วย system calls, Read(), และ Write() ตามลำดับ จะสังเกตุเห็นว่านอกจาก system calls ในการเขียนและอ่านไฟล์แล้วนั้น ระบบยังมีการเรียกใช้ system calls อื่น ๆ ในการจัดการกับอุปกรณ์อินพุทเอาท์พุท ซึ่งเป็นช่องทางสู่หน่วยเก็บข้อมูล (hard disk) ด้วยเช่นกัน การกระทำเช่นนี้วนรอบไปจนกระทั้งหมดทั้งไฟล์

2.6 ส่วนต่อประสานโปรแกรมประยุกต์ (Application Programming Interface : API)

อย่างไรก็ตามโปรแกรมเมอร์ (programmer) ซึ่งพัฒนาโปรแกรมประสานการใช้งานระหว่างผู้ใช้และระบบปฏิบัติการนั้นไม่สามารถเรียกใช้ system calls ได้โดยตรง แต่กระทำผ่านตัวกลาง Application Programming Interface (API) ที่ระบบปฏิบัติการจัดเตรียมไว้ให้ ยกตัวอย่างเช่น ตัวกลาง Win32 APIs ใช้งานสำหรับระบบปฏิบัติการ Windows ตัวกลาง POSIX APIs ใช้งานสำหรับระบบปฏิบัติการ UNIX และ Linux และตัวกลาง Java API ใช้งานสำหรับระบบปฏิบัติการ Java Virtual Machine สาเหตุที่ไม่เรียกใช้ system calls โดยตรงนั้น เป็นเพราะความสะดวกในการเรียกใช้งาน ลดความยุ่งยากในการทำความเข้าใจการทำงานในรายละเอียดของ system calls และยังสามารถใช้ได้กับทุก ๆ ระบบปฏิบัติการที่มี API เหมือนกัน ดังนั้น APIs จึงทำหน้าที่จุดเชื่อมต่อระหว่างโปรแกรมเมอร์กับการเรียกใช้ระบบ system calls ซึ่งอยู่ในส่วนที่เรียกว่า OS kernel ผู้ที่เรียกใช้งาน API ไม่จำเป็นต้องรู้ว่าระบบทำงานลักษณะใด หรือแม้แต่เรียกใช้ system calls ใดบ้าง เพียงแต่ทราบวิธีการใช้งาน API และทำตามเงื่อนไขกฎเกณฑ์ที่กำหนดเท่านั้น

ตัวอย่างการใช้ API ของฟังค์ชั่น ReadFile() ใน Win32 ของระบบปฏิบัติการ Windows ดังแสดงใน



รูปที่ 2-14: ตัวอย่าง AIP ใน Win32

ที่มา: (Silberschatz, Galvin, & Gagne, 2010)

Error! Reference source not found. ผู้เรียกใช้ API ในการอ่านไฟล์ต้องทำตามเงื่อนไขในการส่งค่าพารามิเตอร์และรับผลตามที่กำหนด ซึ่งฟังก์ชันมีรายละเอียดพารามิเตอร์ดังนี้

1) HANDLE file: ชื่อไฟล์ที่ต้องการอ่าน

2) LPVOID buffer: บัฟเฟอร์หรือความจุของขนาดพื้นที่ใช้ในการอ่านเขียนข้อมูล

3) DWORD bytesToread: จำนวนข้อมูลหน่วยเป็นไบท์ (bytes) ที่จะอ่านเข้าสู่บัฟเฟอร์

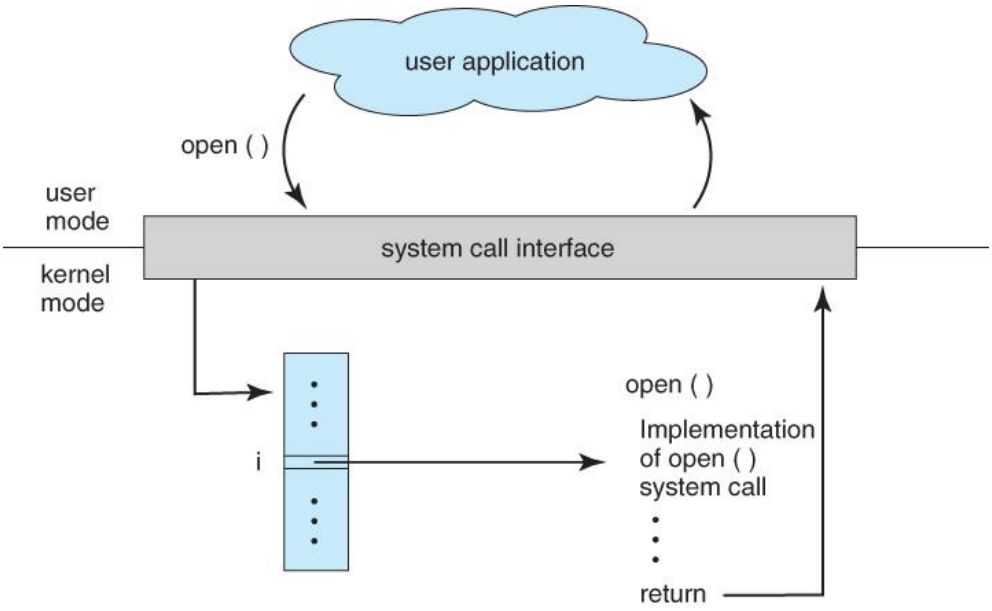
4) LPDWORD bytesRead: จำนวนข้อมูลหน่วยเป็นไบท์ ที่ได้อ่านแล้วในการอ่านล่าสุด

5) LPOVERLAPPED ovl: สถานะบ่งบอกการใช้งานอุปกรณ์อินพุทเอาท์พุททับซ้อนกับการใช้งานโปรแกรมอื่น

ทั้งนี้ผลของการอ่านข้อมูลจากไฟล์สำเร็จหรือไม่ฟังฟค์ชั่น ReadFile() จะส่งค่ากลับมาให้ในรูปแบบค่าตรรกะ (Boolean)

2.7 ความสัมพันธ์ระหว่างการเรียกใช้ระบบและส่วนต่อประสานโปรแกรม (Relation between System calls and APIs)

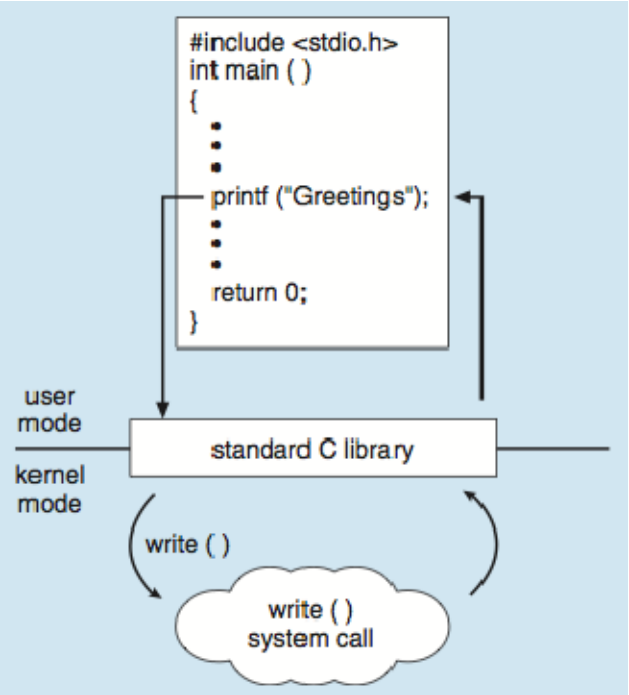
ความสัมพันธ์ระหว่างการเรียกใช้ระบบ (system call) และส่วนต่อประสานโปรแกรมประยุกต์ (API) แสดงในรูปที่ 2-15 ซึ่งเป็นการเรียกใช้บริการการเปิดไฟล์ด้วยฟังก์ชัน open() จะเห็นได้ว่ากระบวนการทำงานแบ่งออกเป็น 2 ส่วน คือ user mode และ kernel mode ในส่วนบนเป็นส่วนที่โปรแกรมเมอร์ผู้พัฒนาโปรแกรมมองเห็นและเรียกใช้งานระบบ การเรียกใช้บริการของระบบกระทำผ่านตัวกลางคือ system call interface หรือ APIs ซึ่งมีรูปแบบและข้อกำหนดต่าง ๆ ดังที่อธิบายไว้แล้ว โปรแกรมเมอร์เรียกใช้ฟังก์ชัน open() ซึ่งเป็น API ในการเปิดไฟล์ (ในรายละเอียดโปรแกรมเมอร์ต้องใส่ค่าพารามิเตอร์เช่น ชื่อไฟล์ เป็นต้น) จากนั้นระบบปฏิบัติการจะทำการเรียกใช้ระบบ ซึ่งอาจเป็นกระบวนการหลายขั้นตอน และต้องเรียกใช้ระบบหลายส่วน เช่นการร้องขอใช้ช่องทางติดต่ออุปกรณ์อินพุทเอาท์พุท เพื่อให้การเปิดไฟล์บรรลุตามวัตถุประสงค์



รูปที่ 2-15: ความสัมพันธ์ระหว่าง API และ System call กรณีตัวอย่าง Open()

ที่มา: (Silberschatz, Galvin, & Gagne, 2010)

หากเราพิจารณาในรายละเอียดเพิ่มมากยิ่งขึ้น ดังเช่นในตัวอย่างรูปที่ 2-16 เป็นการเขียนโปรแกรมในภาษา C โดยต้องการแสดงข้อความออกทางหน้าจอคอมพิวเตอร์ด้วยคำสั่ง printf() การแสดงข้อความออกทางหน้าจอซึ่งเป็นอุปกรณ์ประเภทเอาท์พุท (output device) จำเป็นต้องใช้การเรียกใช้ระบบ ซึ่งเข้าถึงอุปกรณ์ดังกล่าว คอมไพเลอร์ในภาษา C จัดเตรียมการเรียกใช้ระบบ ไว้ในไลบรารี่ชื่อ stdio.h นักโปรแกรมเมอร์จึงสามารถเรียกใช้บริการดังกล่าวด้วย API ชื่อ printf() ด้วยรูปแบบที่กำหนดคือข้อความที่จะพิมพ์ภายใต้เครื่องหมาย “ ” ส่งค่าผ่านไปยังส่วนของ system kernel ภายใต้การทำงานส่วนนี้ถือเป็น kernel mode ที่ต้องนำข้อความลำเลียงออกไปแสดงที่จอภาพได้อย่างถูกต้อง สุดท้ายส่งสัญญาณกลับมาบอกโปรแกรมผู้เรียกใช้ให้ทราบถึงผลการทำงาน



รูปที่ 2-16: ตัวอย่างการเรียกใช้ system call ผ่าน API ใน Library ของภาษา C

ที่มา: (Silberschatz, Galvin, & Gagne, 2010)

2.8 ประเภทของการเรียกใช้ระบบ (Types of System calls)

ประเภทของการเรียกใช้ระบบ (system calls) สามารถจำแนกออกเป็นกลุ่มได้ดังนี้

1) Process Control: เป็น system calls ที่มีเป้าหมายในการควบคุม process ต่าง ๆ เช่น load-execute ใช้เมื่อต้องการประมวลผล, end-abort เมื่อพบการสิ้นสุดการทำงานหรือพบข้อผิดพลาด เป็นต้น

2) File Management: เป็นการเรียกใช้ระบบ สำหรับจัดการกับ file เช่น create-delete ใช้เมื่อต้องการสร้างหรือลบไฟล์, open-close ใช้เมื่อต้องการเปิดหรือปิดไฟล์ เป็นต้น

3) Device management: เป็นการเรียกใช้ระบบ สำหรับจัดการทรัพยากรของคอมพิวเตอร์ฮาร์ดแวร์ เช่น read-write อุปกรณ์ฮาร์ดดิสก์, request-release ซึ่งเป็นการจองหรือปล่อยพื้นที่ memory

4) Communication: เป็นการเรียกใช้ระบบ จัดการกับการสื่อสารซึ่งมี 2 รูปแบบคือ sharing และ passing models

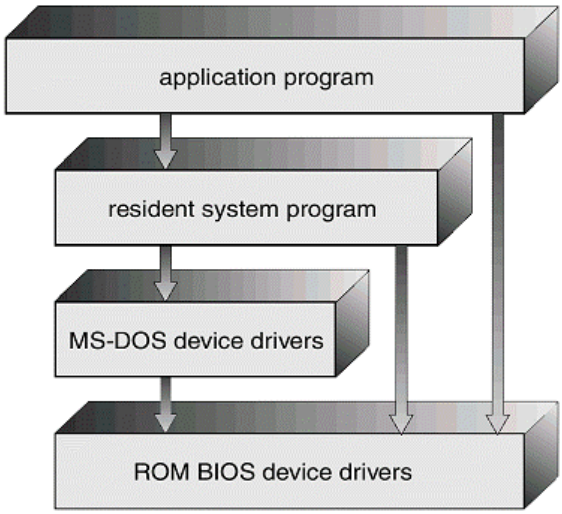
2.9 โครงสร้างระบบปฏิบัติการ (Operating-System Structure)

ระบบปฏิบัติการถือเป็นระบบที่มีขนาดใหญ่และการทำงานซับซ้อน ดังนั้นการออกแบบระบบปฏิบัติการ ให้มีประสิทธิภาพจึงต้องมีการกำหนดหน้าที่อย่างเป็นสัดส่วน เพื่อให้แต่ละส่วนทำงานสอด

ประสานกันอย่างดี โครงสร้างของระบบปฏิบัติการ (OS Structure) จึงเป็นเรื่องสำคัญที่ต้องทำความเข้าใจและมีรูปแบบที่แตกต่างกันออกไปตามวัตถุประสงค์ของการออกแบบ ซึ่งมีรายละเอียดดังต่อไปนี้

2.9.1 โครงสร้างอย่างง่าย (Simple Structure)

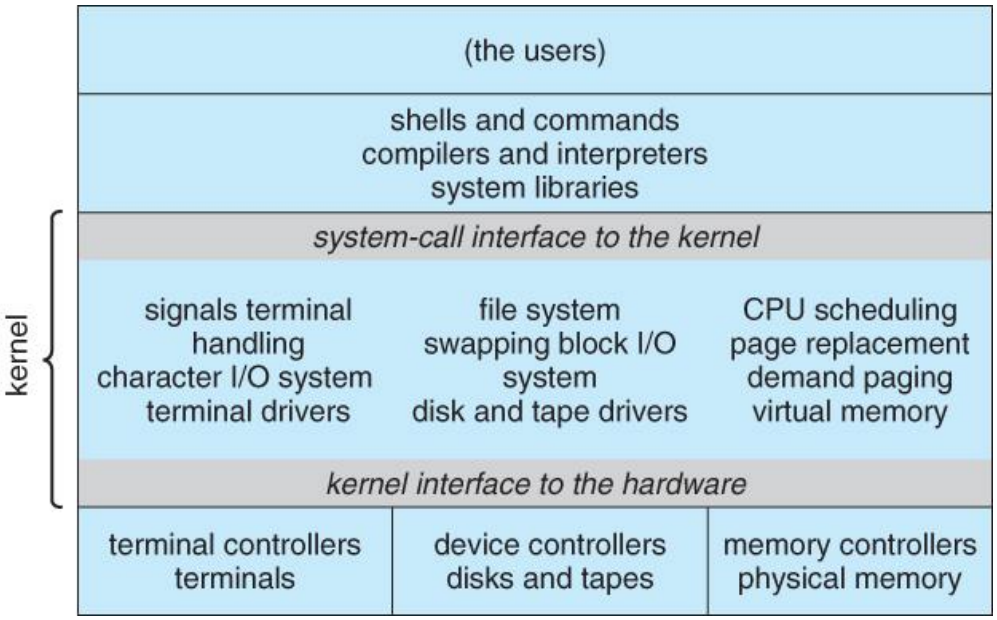
ระบบปฏิบัติการที่มีโครงสร้างเป็นแบบ simple structure คือ MS-Dos ซึ่งได้รับความนิยมมากในอดีตก่อนหน้าระบบปฏิบัติการ Windows ในอดีต MS-Dos ถูกพัฒนาขึ้นโดยไม่ได้จัดแบ่งลำดับการทำงานของแต่ละส่วนไว้ให้ชัดเจน ดังนั้นโปรแกรมต่าง ๆ ในระดับของผู้ใช้ (application program) จึงสามารถเข้าถึงฮาร์แวร์ได้โดยตรง โดยไม่ผ่านตัวกลางหรือ system kernel แต่อย่างใด ตัวอย่างเช่น application program บน MS-Dos สามารถเข้าถึงอุปกรณ์แสดงผลบนจอภาพหรือเข้าถึงฮารด์ดิสก์ได้โดยตรง ซึ่งถือว่าเป็นช่องโหว่ที่อาจทำให้บุคคลที่ไม่มีสิทธิ์เข้าถึงทรัพยากรได้ หรืออาจทำการโจมตีมายังระบบคอมพิวเตอร์ได้โดยง่าย โครงสร้างระบบปฏิบัติการลักษณะนี้แสดงในรูปที่ 2-17



รูปที่ 2-17: โครงสร้างแบบ Simple ใน MS-Dos

ที่มา: (Silberschatz, Galvin, & Gagne, 2010)

ระบบปฏิบัติการแบบดั่งเดิมของ UNIX มีโครงสร้างอย่างง่ายเช่นกัน ดังแสดงในรูปที่ 2-18 โดยมีเคอร์เนลทำหน้าที่จัดการทรัพยากรทุกอย่างในคอมพิวเตอร์ให้แก่ผู้ใช้ ผู้ใช้มีช่องทางติดต่อสู่ระบบด้วยส่วนต่อประสานแบบคำสั่งทีละบรรทัด (command-line interface) คำสั่งจากผู้ใช้จะทำการเรียกใช้ระบบผ่าน system-call interface เพื่อสั่งการไปยังเคอร์เนล จากนั้นเคอร์เนลทำหน้าที่ติดต่อกับฮาร์ดแวร์และทรัพยากรอื่น ๆ เพื่อให้ทำงานตามคำสั่งได้ อย่างไรก็ตามการพัฒนาและดูแลโครงสร้างลักษณะนี้ทำได้ยากเนื่องจากความซับซ้อนในเคอร์เนล

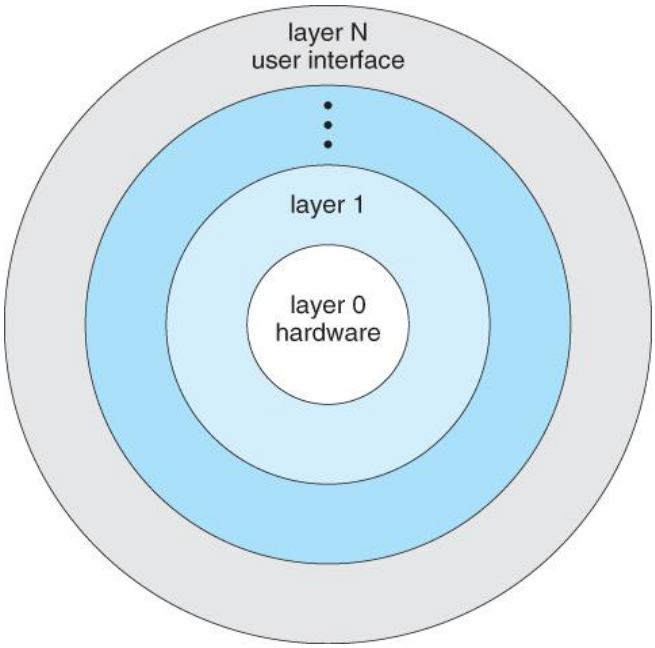


รูปที่ 2-18: โครงสร้างในระบบปฏิบัติการ UNIX แบบดั่งเดิม

ที่มา: (Silberschatz, Galvin, & Gagne, 2010)

2.9.2 โครงสร้างแบบชั้น (Layered Structure)

ระบบปฏิบัติการในลักษณะนี้ถูกแบ่งการทำงานเป็นชั้น ๆ ดังรูปที่ 2-19 โดยกำหนดให้แต่ละชั้นมีหน้าที่ชัดเจน จากชั้นในสุด (layer 0) คือฮาร์ดแวร์ จนกระทั้งชั้นนอกสุด (layer N) ซึ่งเป็นชั้นของผู้ใช้ ตัวอย่างการแยกชั้นเช่น ชั้น CPU scheduling และชั้น memory management เป็นต้น หลักการทำงานคือ ชั้นที่อยู่นอกมากกว่าเป็นผู้เรียกใช้งานชั้นที่อยู่ด้านในมากว่าหนึ่งขั้น ซึ่งชั้นที่ถูกเรียกใช้จะเป็นผู้ให้บริการและส่งผลการดำเนินงานกลับไปให้ผู้เรียก โดยไม่สนใจว่าชั้นอื่นจะทำงานอย่างไร ข้อดีสำหรับโครงเป็นชั้นคือ รายละเอียดการทำงานของแต่ละชั้นถูกปกปิด จึงมีความปลดภัยสูง แต่มีข้อเสียคือไม่มีประสิทธิภาพเนื่องจากการทำงานต้องผ่านหลายขั้นตอนหลายชั้น



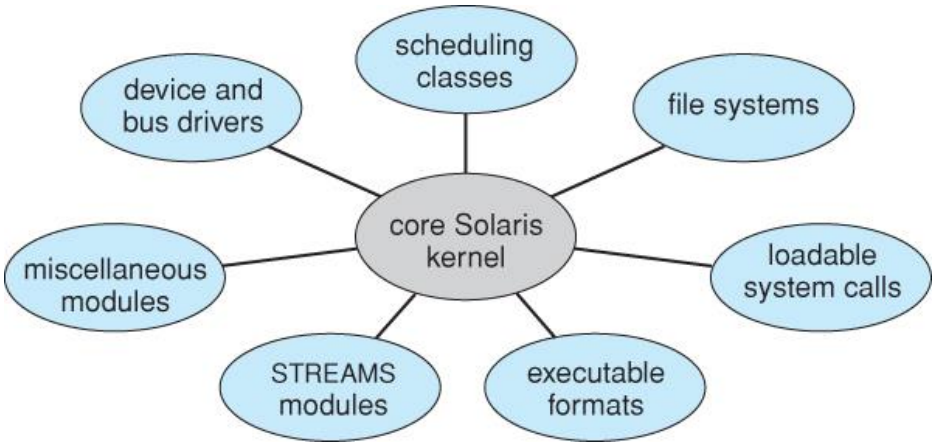
รูปที่ 2-19: โครงสร้างระบบปฏิบัติการแบบแบ่งเป็นชั้น (layered structure)

2.9.3 โครงสร้างแบบไมโครเคอร์เนล (Microkernels)

โครงสร้างระบบปฏิบัติการแบบไมโครเคอร์เนล (microkernels) มีหลักการคือพยายามลดส่วนที่ไม่จำเป็นออกจากเคอร์เนล แต่ให้ย้ายไปไว้ในส่วนของผู้ใช้ (user) หรือ ส่วนบริการ (service) แทน ดังนั้นส่วนของเคอร์เนลจึงเป็นส่วนที่ทำการจัดการหน่วยความจำ (memory management) และส่วนของบริหารการสลับงานให้แก่หน่วยประมวลผล (CPU scheduling) ผลที่ได้จากหลักการดังกล่าวคือความปลอดภัยเพิ่มมากขึ้น เนื่องจากส่วนให้บริการต่าง ๆ ที่จำเป็นต่อผู้ใช้ services ส่วนใหญ่อยู่ในส่วนของ user mode นอกจากนี้การขยายระบบทำได้ง่ายเนื่องจากมากขึ้น โดยขยายเพียง service โดยไม่ต้องแก้ไขในส่วนของเคอร์เนลแต่อย่างใด ส่วนสำคัญอีกประกานหนึ่งคือ massage passing ซึ่งเป็นการสื่อสารเพื่อส่งค่าพารามิเตอร์ และผลการทำงานต่าง ๆ ระหว่างส่วนของ user และส่วนของ kernel

2.9.4 โครงสร้างแบบโมดูล (Modules)

โครงสร้างของระบบปฏิบัติการในลักษณะนี้มีการออกแบบจัดแบ่งเคอร์เนลให้เป็นชุดโมดูล (modules) หรือเรียกว่า Objected-oriented ซึ่งนับว่าเป็นโครงสร้างที่ดีที่สุดในปัจจุบัน ระบบปฏิบัติการที่ออกแบบด้วยโครงสร้างนี้เช่น Solaris, Linux และ Mac OS X โดยโครงสร้างมี core kernel เป็นแกนหลัก และมี modular kernel ซึ่งมีลักษณะเป็น loadable module ซึ่งมีหน้าที่ทำงานเฉพาะตัวและผู้ใช้สามารถเลือกมาติดตั้งเพิ่มเติมได้ภายหลัง ตัวอย่างระบบปฏิบัติการโซลารีส (Solaris) ดังรูปที่ 2-20 ซึ่งมีการแบ่งชุดโมดูลสามารถจำแนกได้เป็นกลุ่มดังนี้



รูปที่ 2-20: โครงสร้างแบบโมดูลในระบบปฏิบัติการโซลารีส (Solaris)

1) Scheduling classes : เป็นโมดูลสำหรับจัดสรรทรัพยากร เช่น การจัดสรรการทำงานของซีพียู เป็นต้น (Oracle, Introduction to Oracle, 2014)

2) File Systems : เป็นโมดูลจัดการระบบไฟล์ให้กับระบบปฏิบัติการโซลารีส ในด้านชนิดไฟล์ โครงสร้างไฟล์ในระบบ โครงข้อมูล และการแบ่งพื้นที่ เป็นต้น (Oracle, Devices and File Systems, 2012)

3) Loadable system calls : เป็นโมดูลสำหรับจัดการชุดการเรียกใช้ระบบ (system call) ในรูปแบบของตาราง ชุดการเรียกใช้ระบบจำนวนหนึ่งถูกพัฒนาให้มีลักษณะบรรจุแบบพลวัต (dynamically loadable system calls) ซึ่งจะทำการบรรจุเมื่อมีการเรียกใช้ในครั้งแรก โดยระบบปฏิบัติการจะจัดเก็บไว้ ณ ตำแหน่ง /kernel/sys และ /usr/kernel/sys (Mauro & McDougall, 2001)

4) Executable formats : เป็นโมดูลสำหรับจัดการรูปแบบของไฟล์ต่าง ๆ ในระบบปฏิบัติการโซลารีส โดยอ็อปเจกต์ไฟล์ (object file) จะถูกแปลโปรแกรม (compile) ให้อยู่ในรูปแบบ ELF (Excecuable and Linking Format) ซึ่งเป็นรูปบบไฟล์ของโซลารีส (Mauro & McDougall, 2001)

5) STREAMS modules : เป็นโมดูลสำหรับควบคุมการรับส่งข้อมูลสู่อุปกรณ์ประเภทอินพุทและเอาท์พุท โดยการควบคุมเกิดขึ้นระหว่างโปรแกรมผู้ใช้ (user program) และโปรแกรมขับอุปกรณ์ (driver program) (Oracle, STREAMS Programming Guide, 2012)

6) Miscellaneous modules : เป็นโมดูลสำหรับจัดการกับสิ่งต่าง ๆ รวมถึงจัดการอุปกรณ์ต่อพ่วง เช่น เมาส์ เป็นต้น (InterSyster, 2017)

7) Device and bus drivers : เป็นโมดูลสำหรับจัดการโปรแกรมขับอุปกรณ์ต่าง ๆ และระบบบัสของระบบ เช่น การติดตั้ง ปรับให้เป็นปัจจุบัน และการถอดโปรแกรมขับ ออกจากระบบ (Oracle, Installing, Updating, and Removing Drivers, 2013)

2.10 กระบวนการเริ่มต้นการทำงานระบบปฏิบัติการ (System Boot)

เมื่อระบบปฏิบัติการถูกติดตั้งสู่เครื่องคอมพิวเตอร์แล้ว คำถามคือเมื่อเราเปิดการทำงานของเครื่องพิวเตอร์แล้วจะเริ่มทำงานและกระบวนการอย่างไร กระบวนการเริ่มต้นการทำงานของระบบปฏิบัติการเพื่อสามารถควบคุมคอมพิวเตอร์ เริ่มโดยการโหลดเคอร์เนลเรียกว่า “booting the system” โดยมี Bootstrap program ซึ่งเป็นชุดคำสั่งขนาดเล็กที่ถูกโหลดเข้าสู่หน่วยความจำในตำแหน่งที่ถูกกำหนดให้เป็นตำแหน่งแรกของการอ่านและประมวลเป็นลำดับแรก ซึ่งผลการประมวลจะทำให้คอมพิวเตอร์สามารถโหลดเคอร์เนลของระบบปฏิบัติการ ที่มีความซับซ้อนตามมาได้ ตำแหน่งของ bootstrap อยู่ที่ initial bootstrap ซึ่งเป็นตำแหน่งที่ระบุไว้ใน หน่วยความจำ ROM ทั้งนี้ Bootstrap program ทำหน้าที่ตรวจสอบความพร้อมของฮาร์ดแวร์ หากพบว่าฮาร์ดแวร์พร้อมก็จะเข้าสู่การ system boot ต่อไป หากไม่พร้อมเช่นไม่พบหน่วยความจำหลักก็จะส่งสัญญาณเสียงบอกให้ทราบ

บทสรุป

บทนี้ได้อธิบายแนวทางการพัฒนาระบบปฏิบัติการ และหน้าที่ระบบปฏิบัติการในการบริหารระบบคอมพิวเตอร์ฮาร์ดแวร์ เพื่อบริการให้แก่ผู้ใช้ วิวัฒนาการในการออกแบบระบบปฏิบัติการซึ่งอธิบายควบคู่กับการพัฒนาและความสามารถของระบบคอมพิวเตอร์ฮาร์ดแวร์ และความต้องการใช้งานเพิ่มขึ้น

หน้าที่สำคัญหนึ่งของระบบปฏิบัติการคือรองรับการใช้งานของผู้ใช้ ดังนั้นการออกแบบส่วนประสานระหว่างผู้ใช้กับระบบคอมพิวเตอร์จึงต้องถูกออกแบบให้รองรับการใช้งานที่มีประสิทธิภาพ โครงสร้างของระบบปฏิบัติการจึงถูกออกแบบไว้เป็นส่วน ๆ โดยแต่ละส่วนมีหน้าที่การทำงานเฉพาะ เช่น ส่วนจัดการชุดโปรแกรมขับ (prodgram drivers) สำหรับควบคุมอุปกรณ์ฮาร์ดแวร์ในระบบคอมพิวเตอร์ หรือส่วนจัดการโปรแกรมผู้ใช้ (user program) เป็นต้น ดังนั้นโปรแกรมระดับของผู้ใช้เมื่อต้องการเข้าใช้ทรัพยากรของระบบจึงต้องเรียกใช้ผ่านการเรียกใช้ระบบ (system call) ทั้งนี้โปรแกรมเมอร์ผู้พัฒนาโปรแกรมผู้ใช้ จำเป็นต้องเข้าใจส่วนประสานของระบบ (application programming interfaces: APIs) ที่เตรียมไว้

แบบฝึกหัดท้ายบท

2.1. จงอธิบายหน้าที่ของ Operating System (OS)

2.2. จงยกตัวอย่างฟังค์ชั่นของระบบปฏิบัติการ มา 3 ตัวอย่าง

2.3. จงบอกพัฒนาการของระบบปฏิบัติการ มีกี่ยุคอะไรบ้าง

2.4. User Interface มีกี่แบบ อะไรบ้าง

2.5. จงอธิบายการเรียกใช้ระบบ (System Call) คืออะไร พร้อมยกตัวอย่าง

2.6. จงอธิบาย Application Programming Interface (API) คืออะไร

2.7. จงอธิบาย Service ของระบบปฏิบัติการ ในด้าน Error Detection มาให้เข้าใจ

2.8. OS Structure มีกี่แบบ อะไรบ้าง

บรรณานุกรม

Degnan, B. (2554). *Sage II UCSD Pascal Televideo Terminal*. Retrieved September 20, 2560, from http://vintagecomputer.net: http://vintagecomputer.net/browse_thread.cfm?id=419

InterSyster. (2017). *Miscellaneous Solaris Settings*. Retrieved from InterSystems Documentation: https://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=GCI_unixparms_sun_zfs_misc

Introduction to Operating System. (2558). Retrieved September 20, 2560, from Transtutor: http://www.transtutors.com

Mauro, J., & McDougall, R. (2001). *Solaris Internals Core Kernel Architecture.* Sun Microsystem Press.

Network Topology. (2559). Retrieved September 20, 2560, from ConceptDrawn: http://www.conceptdraw.com/examples/network-topology

Operating System Evolution. (2557). Retrieved September 19, 2560, from California State University: http://bpastudio.csudh.edu/fac/lpress/471/hout/misc/osgenerations.htm

Oracle. (2012). *Devices and File Systems*. Retrieved from Oracle Document: https://docs.oracle.com/cd/E23824_01/html/821-1459/fsoverview-51.html

Oracle. (2012). *STREAMS Programming Guide*. Retrieved from Oracle Document: https://docs.oracle.com/cd/E26502_01/html/E35856/overvw1-38936.html

Oracle. (2013). *Installing, Updating, and Removing Drivers*. Retrieved from Oracle Document: https://docs.oracle.com/cd/E26505_01/html/E27000/loading-85890.html

Oracle. (2014). *Introduction to Oracle*. Retrieved from Oracle Document: https://docs.oracle.com/cd/E36784_01/html/E36848/gejen.html

Silberschatz, A., Galvin, P. B., & Gagne, G. (2010). *Operating System Concepts* (8th ed.). Asia: John Wiley & Sons.

Timeline of Computer History. (2558). Retrieved September 19, 2560, from Computer History: http://www.computerhistory.org/timeline/