บทที่ 1

องค์ประกอบของระบบคอมพิวเตอร์และระบบปฏิบัติการ

1.1 ระบบคอมพิวเตอร์พื้นฐาน

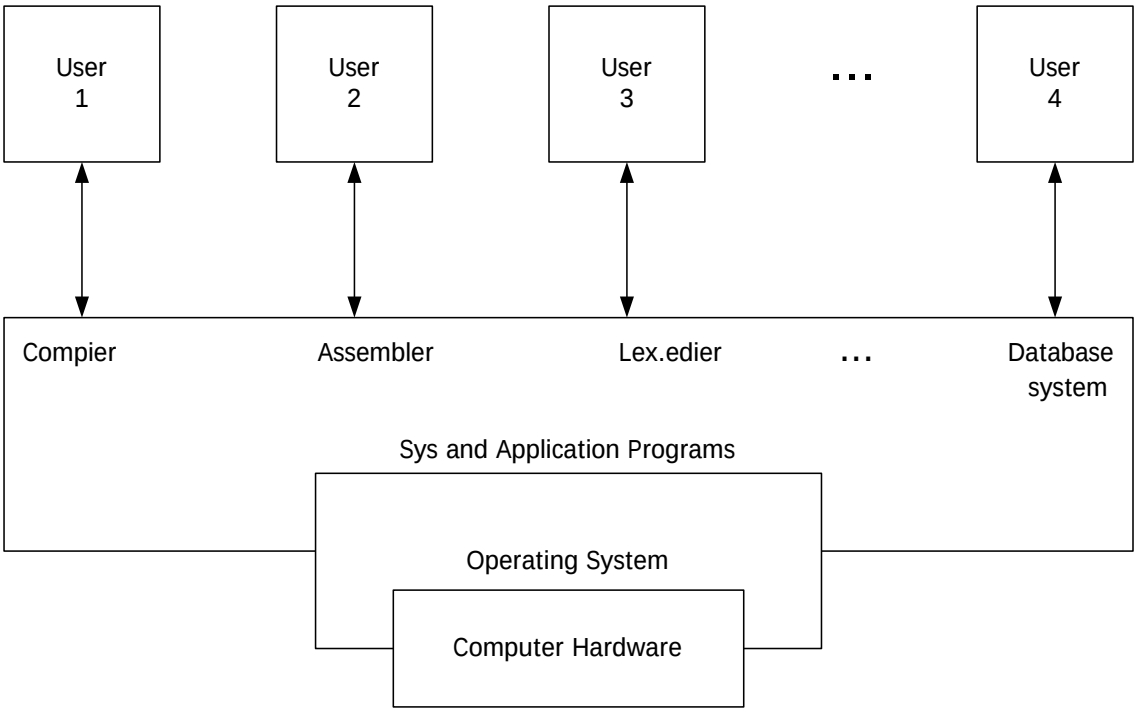
วัตถุประสงค์ของระบบปฏิบัติการคือ

1. เตรียมสิ่งแวดล้อมเพื่อให้ผู้ใช้สามารถ execute โปรแกรมได้
2. ทำให้ระบบคอมพิวเตอร์ใช้ง่ายขึ้น
3. ใช้ฮาร์ดแวร์ให้เกิดประสิทธิภาพอย่างสูงสุด

ระบบปฏิบัติการคืออะไร

ระบบคอมพิวเตอร์แบ่งออกเป็น 4 ส่วน

1. **ฮาร์ดแวร์** สนับสนุนปัจจัยในการคำนวณ เช่น CPU , หน่วยความจำ , อุปกรณ์รับส่งข้อมูล
2. **ระบบปฏิบัติการ** ควบคุมและบริหารการใช้ปัจจัยระหว่างโปรแกรม ต่าง ๆ สำหรับผู้ใช้ เช่น Dos , Unix , Windows 2000 , Windows NT ฯลฯ
3. **โปรแกรมประยุกต์** กำหนดทิศทางในการใช้ปัจจัยสำหรับการแก้ปัญหาต่าง ๆ ของผู้ใช้ เช่น compilers , database systems , video games , business programs
4. **ผู้ใช้** เช่น บุคลากร (people) , เครื่องจักร (machines) , คอมพิวเตอร์เครื่องอื่น ๆ (other computers)



รูปที่ 1.1 ส่วนประกอบของระบบคอมพิวเตอร์

นิยามของระบบปฏิบัติการ

1. **Resource allocator** – บริหารการจัดสรรทรัพยากร เช่น การจัดการ Harddisk , memory , printer ให้เกิดประโยชน์ได้อย่างเต็มที่
2. **Control program** – ควบคุมการ execute โปรแกรมของผู้ใช้ และการทำงานของอุปกรณ์รับส่งข้อมูล
3. **Kernel** (แก่นแท้) – โปรแกรมที่ทำงานอยู่ตลอดเวลาบนคอมพิวเตอร์

โดยทั่วไปแล้วความหมายของระบบปฏิบัติการ คือกลุ่มโปรแกรมซึ่งได้รับการจัดระเบียบให้เป็นส่วนเชื่อมโยงระหว่างเครื่องและผู้ใช้เครื่อง โดยจะเอื้ออำนวยการพัฒนาและการใช้โปรแกรมต่างๆ รวมถึงการจัดสรรทรัพยากร (resource) ต่างๆ ให้มีประสิทธิผลที่ดี หากจะกล่าวให้เฉพาะเจาะจงกว่านี้ หน้าที่ของระบบปฏิบัติการ สามารถแบ่งได้เป็นสองหน้าที่ใหญ่ๆ ด้วยกันคือ

1. ควบคุมการทำงานของโปรแกรมและอุปกรณ์ต่างๆ โดยเฉพาะอุปกรณ์รับข้อมูลและแสดงผล (input/output device) ความหมายนี้รวมถึงการเอื้ออำนวยให้ผู้ใช้สามารถใช้อุปกรณ์ต่างๆ อย่างสะดวก หน้าที่นี้เป็นหลักสำคัญ ซึ่งจะขาดเสียมิได้ในระบบปฏิบัติการทุกรูปแบบ ตั้งแต่เครื่องเล็กไปจนถึงเครื่องใหญ่ สำหรับในเครื่องเล็ก (microcomputer) ระบบปฏิบัติการจะเป็นแบบง่ายๆ และทำหน้าที่ควบคุมในลักษณะนี้แต่เพียงอย่างเดียว จึงมักเรียกกันว่าเป็นโปรแกรมควบคุม (Control program หรือ CP) จุดประสงค์ของหน้าที่นี้ คือการให้ความสะดวกแก่ผู้ใช้เครื่อง

2. จัดสรรทรัพยากรที่ใช้ร่วมกัน (shared resources) ความหมายของหน้าที่นี้จะเห็นได้ชัดในเครื่องระดับใหญ่ (mainframe) ซึ่งจะมีอุปกรณ์ต่างๆ จำพวกหน่วยประมวลผลกลาง หน่วยความจำ ฯลฯ ซึ่งมีสมรรถนะหรือขนาดใหญ่ เกินความจำเป็นของงานแต่ละงาน จึงมีการใช้ทรัพยากรเหล่านี้ร่วมกัน ในลักษณะของระบบมัลติโปรแกรมมิ่ง (multiprogramming) หน้าที่ของระบบปฏิบัติการ จึงต้องครอบคลุมถึงการจัดสรรทรัพยากรเหล่านี้ เพื่อขจัดความขัดแย้ง โดยคำนึงถึงความยุติธรรมต่อผู้ใช้แต่ละคน และประสิทธิผลของเครื่องเป็นหลักสำคัญ

1.1.1 บอกระบบคอมพิวเตอร์ยุคเริ่มต้น

ในสมัยแรกเริ่มรวมปี พ.ศ. 2483-2492 คอมพิวเตอร์มีแต่เครื่องเปล่าๆ ไม่มีระบบปฏิบัติการใดๆเลย ผู้ใช้เครื่องต้องเขียนโปรแกรมเป็นภาษาเครื่องทั้งหมด รวมถึงควบคุมเครื่อง ตระเตรียมงาน ตรวจสอบและทำโปรแกรม แต่ลักษณะการใช้เช่นนี้ ทำให้ประโยชน์ใช้สอย (utilization) ของเครื่องต่ำมาก โดยเฉพาะเมื่อเครื่องในสมัยก่อนมีราคาแพงมากเมื่อเทียบกับเครื่องในสมัยนี้ ซึ่งมีความสามารถทัดเทียมกัน ดังนั้น จึงมีการจ้างพนักงานคุมเครื่อง (operator) เพื่อลดเวลาที่เสียไปในการตระเตรียมงาน (set - up time) และเวลาที่ต้องเก็บกวาด (tear-down time) ซึ่งนอกจากพนักงานคุมเครื่องอาชีพจะชำนาญกว่าผู้ใช้แล้ว ยังสามารถจัดงานที่มีลักษณะคล้ายคลึงกันไว้พวกเดียวกัน เช่นงานที่ต้องใช้ตัวแปลภาษา (translator หรือ compiler) ตัวเดียวกัน ลักษณะนี้เรียกว่า เป็นการป้อนงานแบบกลุ่มด้วยมือ (manual batch system) ระบบนี้ก็ยังไม่ต้องอาศัยระบบปฏิบัติการแต่อย่างใด

1.1.1.1 <u>การป้อนงานแบบกลุ่มโดยอัตโนมัติ</u>

แม้ว่าจะมีการใช้พนักงานคุมเครื่องมืออาชีพ แต่เวลาของเครื่องก็ยังทิ้งเสียเปล่าในขณะที่พนักงานตรวจสอบความต้องการของงาน หางาน (ซึ่งโดยปกติอยู่ในรูปของบัตรเจาะรู และเทปแม่เหล็ก) และป้อนงานเข้าสู่เครื่อง (เช่นใส่บัตรในเครื่องอ่านบัตร หรือใส่เทปในตู้เทป) รวมถึงการนำงานนั้นๆ ออกจากเครื่อง (เช่น เก็บเทป เก็บบัตร หรือฉีกกระดาษผลลัพธ์ เป็นต้น) ดังนั้นในช่วงต้นทศวรรษที่ 5 (พ.ศ. 2493 - 2497) General Motors Research Laboratories ได้พัฒนาระบบปฏิบัติการรุ่นแรกออกมาโดยใช้กับเครื่อง IBM 701 ที่ใช้กันอยู่ในห้องทดลองนั้น

ระบบปฏิบัติการรุ่นแรกนี้ เรียกว่าเป็นการประมวลผลแบบกลุ่มโดยอัตโนมัติ (automatic batch processing) ระบบปฏิบัติการรุ่นแรกนี้ เป็นเพียงโปรแกรมเล็กๆ ซึ่งจะอยู่ในเครื่องตลอดเวลา (resident monitor) และจะส่งมอบการควบคุมเครื่องให้กับโปรแกรมของผู้ใช้ทีละโปรแกรมเรียงตามลำดับตามกันไป ซึ่งในกรณีนี้ จะต้องมีข้อมูลปะหน้าและท้ายโปรแกรม เพื่อแยกงานออกจากกัน รวมทั้งบอกระบบปฏิบัติการถึงลักษณะงาน เช่นตัวแปลภาษาที่ต้องใช้ ตู้เทป และเลขหมายของม้วนเทป เป็นต้น จึงเกิดเป็นภาษาใหม่ขึ้น คือ ภาษาคุมงาน (job control language หรือ JCL)

ปัญหาที่สำคัญอีกประการหนึ่ง คือความแตกต่างของความเร็ว ระหว่างหน่วยประมวลผลกลาง (central processing unit หรือ cpu) กับอุปกรณ์รับข้อมูลและแสดงผล แม้ว่าจะได้มีการพัฒนาอุปกรณ์รับข้อมูลและแสดงผลอย่างไรก็ตาม แต่ขีดจำกัดของเครื่องกลไกก็ยังทำให้อุปกรณ์เหล่านี้ช้ากว่าหน่วยประมวลผลกลางซึ่งทำงานด้วยความเร็วของวงจรอิเล็กทรอนิกส์เป็นหลายพันเท่า ความแตกต่างนี้ ทำให้การใช้ประโยชน์ของหน่วยประมวลผลกลางต่ำมากตัวอย่างเช่น การแปลภาษาเครื่องของงานหนึ่ง ใช้เวลาของหน่วยประมวลผลกลางเพียง 4.8 วินาที ขณะที่การอ่านโปรแกรมนั้น (1579 บัตร ด้วยความเร็ว 1,200 บัตรต่อนาที) ใช้เวลา 78.9 วินาที ดังนั้นหน่วยประมวลผลกลางจะต้องรอเครื่องอ่านบัตร 74.1 วินาที หรือร้อยละ 93.9 ของเวลาที่ใช้ในการทำงานชิ้นนี้ เรียกได้ว่าการใช้ประโยชน์ (utilization) ของหน่วยประมวลผลกลางเป็นเพียงร้อยละ 6.1 เท่านั้น ซึ่งหากรวมความล่าช้าในการแสดงผลเข้าไปด้วยแล้ว การใช้ประโยชน์ของหน่วยประมวลผลกลางก็ยิ่งต่ำลงไปอีก วิธีแก้ปัญหานี้ นับจากสมัยแรกเริ่ม ได้แก่ระบบ buffering ระบบ off-time และระบบ spooling

<u>การทำงานแบบ buffering</u>

แนวความคิดนี้คือ ให้หน่วยรับข้อมูลและแสดงผล ทำงานขนานไปพร้อมกันกับหน่วยประมวลผลกลางมากที่สุดเท่าที่จะทำได้ วิธีการคือ ขณะที่หน่วยประมวลกลาง ประมวลผลข้อมูลจำนวนหนึ่ง หน่วยรับข้อมูลจะอ่านข้อมูลถัดไปเข้ามาไว้ในหน่วยความจำ ส่วนที่เตรียมไว้เพื่อการนี้ ซึ่งเรียกว่าบัฟเฟอร์ (buffer) ซึ่งหากการอ่านข้อมูล (หรือการพิมพ์ผลลัพธ์) สำหรับข้อมูลแต่ละหน่วยพอดี อุปกรณ์ทั้งสองประเภทก็ไม่ต้องรอซึ่งกันและกัน ทำให้ได้ประโยชน์ใช้สอยเต็มที่คือร้อยละร้อย แต่ในความเป็นจริงแล้วจะเกิดความเหลื่อมล้ำ (mismatch) ของเวลาการทำงานสำหรับข้อมูลแต่ละหน่วย ความเหลื่อมล้ำนี้ขึ้นกับสาเหตุที่สำคัญสองประการคือ อัตราความเร็วของอุปกรณ์ต่างๆ และประเภทของงานเนื้อหาสำหรับสาเหตุแรกนั้น ไม่ว่าเครื่องคอมพิวเตอร์ประเภทใด หน่วยประมวลผลกลางจะมีความเร็วสูงกว่าหน่วยรับข้อมูลและแสดงผลมาก แม้ว่าจะมีบัฟเฟอร์ แต่ละหน่วยประมวลผลก็ยังต้องรออยู่ดี ส่วนสาเหตุประการที่สองนี้ หากงานเป็นพวกที่

ใช้หน่วยรับข้อมูลและแสดงผลมากๆ (I/O bounded) หน่วยประมวลผลกลางจำทำงานน้อย เพราะต้องรอข้อมูลจากหน่วยรับข้อมูล (หรือรอให้หน่วยแสดงผลนำผลที่ได้ไปแสดง) ในทำนองกลับกัน หากงานเป็นประเภทที่ใช้หน่วยประมวลผลกลางมากๆ (cpu bounded) ช่วงเวลาที่หน่วยประมวลผลกลางจะว่างก็ลดลงจนอาจถึงกับไม่ต้องรอเลย กลายเป็นว่า หน่วยรับข้อมูลและแสดงผลต้องเป็นฝ่ายรอหน่วยประมวลผลกลาง

ในเครื่องสมัยแรกเริ่มนั้น (หรือแม้แต่ในปัจจุบันเองก็ตาม) ความเหลื่อมล้ำระหว่างหน่วยประมวลผลกลางต่ำมาก วิธีการแก้ทางหนึ่งคือ เพิ่มความเร็วของหน่วยรับข้อมูลและแสดงผลแต่วิธีนี้ทำได้ยาก เพราะอุปกรณ์เหล่านั้น เป็นเครื่องกลไกซึ่งมีข้อจำกัดทางด้านความเร็วเป็นธรรมดาอยู่แล้ว จึงได้มีการนำอุปกรณ์ที่มีความเร็วสูงขึ้นอีกระดับ เช่น เทปและจานแม่เหล็กมาคั่นระหว่างหน่วยประมวลผลกลาง และหน่วยรับข้อมูลและแสดงผล ส่วนอีกวิธีหนึ่งคือ การทำงานหลายๆ งานพร้อมกัน เพื่อไม่ให้หน่วยประมวลผลกลางต้องอยู่เฉย ขณะรอการรับข้อมูลหรือแสดงผลของงานหนึ่งงานใด ซึ่งลักษณะนี้ คือการทำมัลติโปรแกรมมิ่ง (multiprogramming) ซึ่งจะกล่าวถึงต่อไป

<u>การทำงานแบบ off - line</u>

ปัญหาความเหลื่มล้ำของความเร็ว สามารถผ่อนหนักให้เป็นเบาได้วิธีหนึ่งคือ การใช้เทปแม่เหล็กมาแทนเครื่องอ่านบัตรหรือเครื่องพิมพ์ที่มีความเร็วต่ำมาก วิธีการคือ การจำลองข้อมูลจากบัตรลงบนเทปแม่เหล็ก เมื่อโปรแกรมต้องการอ่านบัตร ระบบปฏิบัติการจะเปลี่ยนไปอ่านเทปให้แทน ส่วนการพิมพ์ผลก็ทำทำนองเดียวกัน โดยการพิมพ์ลงเทปแม่เหล็กก่อน แล้วนำเทปนั้นไปถ่ายออกเครื่องพิมพ์อีกที การถ่ายเทข้อมูลผ่านเทปนี้กระทำได้สองวิธีคือ ใช้เครื่องอ่านบัตรและเครื่องพิมพ์ที่ออกแบบมาเป็นพิเศษ สามารถถ่ายข้อมูลกับเครื่องอ่านเทปแม่เหล็กได้โดยตรง ไม่ต้องผ่านหน่วยประมวลผลกลาง ส่วนอีกวิธีหนึ่งคือ ใช้อุปกรณ์มาตรฐานปกติแต่ใช้เครื่องคอมพิวเตอร์ขนาดเล็ก เป็นตัวถ่ายเทข้อมูล แทนที่จะใช้เครื่องใหญ่ การทำงานโดยอาศัยเทปแม่เหล็กนี้ จะต้องได้รับความช่วยเหลือจากระบบปฏิบัติการในอันที่จะทำให้คำสั่งรับข้อมูลหรือแสดงผล (input / output operation) ในโปรแกรมของผู้ใช้สามารถเปลี่ยนไปใช้กับอุปกรณ์ใดก็ได้ขึ้นกับความเหมาะสมในการบริหารระบบ ลักษณะการทำงานเช่นนี้เรียกว่า อิสรภาพจากอุปกรณ์ (device independence)

ข้อเสียของระบบ off-line คือ โปรแกรมจะต้องผ่านขั้นตอนมากขึ้น และในการเก็บข้อมูลเทปแม่เหล็ก ต้องรอให้มีหลายๆ โปรแกรมเสียก่อน จึงค่อยนำเข้าเครื่องคอมพิวเตอร์ใหญ่เสียทีหนึ่ง ทำให้ผู้ใช้ต้องรอนานขึ้น แม้ว่าประโยชน์ใช้สอยของหน่วยประมวลผลกลางจะดีขึ้นก็ตาม

<u>การทำงานแบบ spooling</u>

เมื่อเทคโนโลยีของจานแม่เหล็กได้รับการพัฒนามากขึ้น ระบบปฏิบัติการก็เริ่มหันมาใช้จานแม่เหล็กแทนเทปแม่เหล็กด้วยเหตุผลสำคัญประการหนึ่งที่เห็นได้ชัดเจนคือ การที่ไม่สามารถทำการประมวลผลข้อมูลในเทป ไปพร้อมๆ กัน กับที่ถ่ายเทข้อมูลจากเครื่องอ่านบัตร ลงเทปม้วนเดียวกันนั้นได้

หลักการใช้จานแม่เหล็กมีลักษณะคล้ายกับเทปแม่เหล็ก ข้อแตกต่างที่สำคัญมีด้วยกันสองประการคือ

1. เนื่องจากการเข้าถึง (access) จานแม่เหล็กเป็นแบบตรง (direct) ไม่ใช่แบบเรียงลำดับ (sequential) อย่างเทปแม่เหล็ก จึงทำให้สามรถแยกงานออกจากกันได้ โดยสร้างตารางบ่งบอกว่าข้อมูล (หรือผลลัพธ์) ของงานใดอยู่ในส่วนใดของจานบันทึก

2. เมื่อการใช้จานแม่เหล็กเป็นแบบตามสาย หรือต่อตรง (on-line) หน่วยประมวลผลที่ใช้ในการถ่ายเทข้อมูลระหว่างจานแบะอุปกรณ์รับข้อมูลและแสดงผล จึงต้องเป็นตัวเดียวกับที่ใช้ในการประมวลงานของผู้ใช้ หรือกล่าวอีกนัยหนึ่งคือ ต้องมีโปรแกรมพิเศษตัวหนึ่ง ทำงานคู่ขนานไปกับโปรแกรมของผู้ใช้ เพื่อทำการถ่านเทข้อมูลกับจานแม่เหล็ก จึงเกิดเป็นการทำมัลติโปรแกรมมิ่งแบบพื้นฐานขึ้น

หลักการใช้จานแม่เหล็กแทนอุปกรณ์รับข้อมูล และแสดงผลนี้ เรียกว่า spooling ซึ่งย่อมาจาก Simultaneous Peripheral Operation On - Line

ข้อดีที่สำคัญของ spooling คือความจำเป็นที่ต้องพัฒนาระบบมัลติโปรแกรมมิ่งแบบพื้นฐานขึ้น ซึ่งก่อให้เกิดความก้าวหน้าต่อวงการ โดยเฉพาะทางศาสตร์ด้านระบบปฏิบัติการระบบนี้ทำให้สามารถเหลื่อมการประมวลผลของงานหนึ่งกับการรับข้อมูลและแสดงผล (โดยผ่านโปรแกรม spool) ของอีกงานหนึ่งได้ จุดนี้ต่างกับการใช้บัฟเฟอร์ ตรงที่การใช้บัฟเฟอร์นั้นเป็นการเหลื่อมกันระหว่างการประมวลผล และการรับและแสดงข้อมูลของโปรแกรมเดียวกันซึ่งก็ไม่อาจทำได้มากเท่าไรนัก ด้วยจำกัดอยู่ที่ขั้นตอนการทำงานของโปรแกรมนั้นๆ

ข้อดีอีกประการหนึ่งของ spooling คือ ลักษณะการเข้าถึงแบบตรงของจานแม่เหล็กงานที่ถูกป้อนเข้ามาแบบเรียงลำดับ สามารถถูกจัดแยกเป็นอิสระ เกิดเป็น job pool ซึ่งระบบปฏิบัติการสามารถเลือกงานเข้าประมวลผล ตามความเหมาะสมได้ เช่นตามความสำคัญของงานซึ่งกำหนดโดยผู้ใช้หรือผู้บริหารระบบ หรือตามระดับและลักษณะการใช้งานอุปกรณ์ต่างๆ ในระบบ เช่น เลือกงานที่ใช้เทป เมื่อมีตู้เทปว่าง เป็นต้น ทำให้เกิดเป็นระบบคัดเลือกงาน (job scheduling) แบบพื้นฐานขึ้น

1.1.1.2 <u>ระบบมัลติโปรแกรมมิ่ง</u>

แม้ว่า spooling จะเป็นการทำงานแบบมัลติโปรแรมมิ่งอย่างง่ายๆ โดยมีโปรแกรมวิ่ง

ขนานกันอยู่สองโปรแกรม คือโปรแกรม spool และโปรแกรมของผู้ใช้ (ในลักษณะการประมวลผลแบบกลุ่ม) แต่ลักษณะของโปรแกรมที่เกี่ยวข้อง ก็ยังไม่อาจใช้ประโยชน์องค์ประกอบต่างๆ ของคอมพิวเตอร์ได้เต็มที่ เหตุเพราะโปรแกรมของผู้ใช้ได้เต็มที่ เหตุเพราะโปรแกรมของผู้ใช้อาจทำงานร่วมกับเทปแม่เหล็ก ซึ่งมีความเหลื่อมล้ำทางความเร็วกับหน่วยประมวลผลกลางมากอยู่ดี ในสมัยต่อมาจึงได้มีการขยายการทำมัลติโปรแกรมมิ่งออกไป เพื่อให้ประโยชน์ใช้สอยของระบบสูงขึ้น

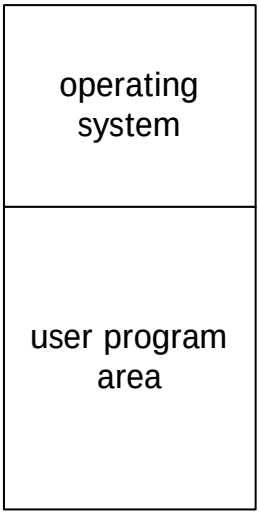
หลักการของระบบมัลติโปรแกรมมิ่ง คือการให้มีโปรแกรมอยู่ในหน่วยความจำหลักพร้อมที่จะถูกประมวลผลได้หลายๆ โปรแกรม ระบบปฏิบัติการจะเลือกโปรแกรมมาตัวหนึ่งให้หน่วยประมวลกลางทำการประมวลผลไปเรื่อยๆ ในที่สุดโปรแกรมนั่นต้องหยุดคอยอะไรสักสิ่งหนึ่ง เช่น รอให้พนักงานคุมเครื่องใส่เทปแม่เหล็กเข้าตู้เทป หรือรอให้เครื่องอ่านบัตรอ่านข้อมูลชิ้นถัดไปเข้ามา ซึ่งในลักษณะมัลติโปรแกรมมิ่งนี้ หน่วยประมวลผลกลางจะไม่อยู่เฉยโดยระบบปฏิบัติการจะคัดเลือกงานอื่นที่พร้อมจะถูกประมวลผล มาให้หน่วยประมวลผลกลางทำการดำเนินต่อได้ หน่วยประมวลผลกลางจะเปลี่ยนไปทำงานที่พร้อมจะให้ทำเช่นนี้เรื่อยๆ ไปและในที่สุด งานแรกที่ค้างไว้จะกลับมาพร้อมให้ทำอีก เนื่องจากสิ่งที่รออยู่นั้นสำเร็จลุล่วงแล้วซึ่งเมื่อหน่วยประมวลผล
ว่างก็จะหันกลับมาทำงานแรกนั้นต่อไป

ลักษณะของมัลติโปรแกรมมิ่งนี้ มีตัวอย่างที่เห็นได้ชัดในชีวิตประจำวันโดยทั่วไป เช่น ผู้จัดการบริษัทสั่งเลขานุการ ให้ติดต่อกับผู้จัดการของอีกบริษัทหนึ่ง ขณะที่เลขาฯ หมุนโทรศัพท์ติดต่อ ผู้จัดการผู้นั้นก็สามารถหันไปทำงานอื่น รอจนเลขาฯ ติดต่อได้แล้ว จึงหันกล้องมาคุยโทรศัพท์

อาจถือได้ว่า ระบบมัลติโปรแกรมมิ่ง เป็นต้นกำเนิดของศาสตร์ทางระบบปฏิบัติการก็ว่าได้ เนื่องจากระบบมัลติโปรแกรมมิ่ง ไม่ว่าจะเป็นแบบใดจะซับซ้อนมาก การที่จะทำงานหลายๆ งานพร้อมๆ กัน ระบบปฏิบัติการต้องคอยควบคุม และจัดการองค์ประกอบต่างๆ เช่น การจัดสรรเนื้อที่ในหน่วยความจำหลักที่มีจำกัด ให้แก่งานเหล่านั้น ทั้งต้องสับหลีกงาน เมื่อมีงานหลายๆ งานพร้อมที่จะให้ทำการประมวลผล และรวมถึงการจัดการอุปกรณ์ข้างเคียงให้มีประสิทธิภาพในการใช้งานสูง นอกจากนี้ความต้องการทรัพยากรของงานต่างๆ อาจเกิดความขัดแย้ง และสับสน ในลักษณะนี้เรียกว่า deadlock ซึ่งระบบปฏิบัติการจำเป็นต้องหาทางป้องกันของงานแต่ละงานต่างๆ เหล่านี้เป็นต้น ซึ่งจะเห็นได้ในหัวข้อต่อๆ ไปว่า หัวข้อต่างๆ ในศาสตร์ด้านระบบปฏิบัติการ จะเกี่ยวพันกับจุดประสงค์ของการทำงานระบบมัลติโปรแกรมมิ่ง

1.1.2 ลักษณะระบบคอมพิวเตอร์แบบแบทช์

คอมพิวเตอร์ยุคแรก ๆ มีขนาดใหญ่ทำการรับจาก console อุปกรณ์นำเข้า (input) เป็นพวก card reader และ tape drive อุปกรณ์แสดงผล (output) เป็นพวก line printer, tape drive และ บัตรเจาะรู ผู้ใช้จะไม่ได้ติดต่อกับระบบคอมพิวเตอร์โดยตรง วิธีการติดต่อคือ ผู้ใช้ไม่ได้ติดต่อกับระบบคอมพิวเตอร์โดยตรง วิธีติดต่อคือ ผู้ใช้เตรียม job พวกโปรแกรม, ข้อมูล, และสารสนเทศเพื่อการควบคุมเกี่ยวกับธรรมชาติของงาน (control card) และนำไปให้ operator งานดังกล่าวมักอยู่ในรูปบัตรเจาะรู (punched card) จึงต้องมี card reader เพื่อประมวลผล ระบบปฏิบัติการในยุคแรก งานหลักจะถูก execute ทีละงาน โดยระบบปฏิบัติการจะฝังในหน่วยความจำตลอดเวลา ดังรูป



รูปที่ 1.2 ระบบคอมพิวเตอร์แบบแบทช์

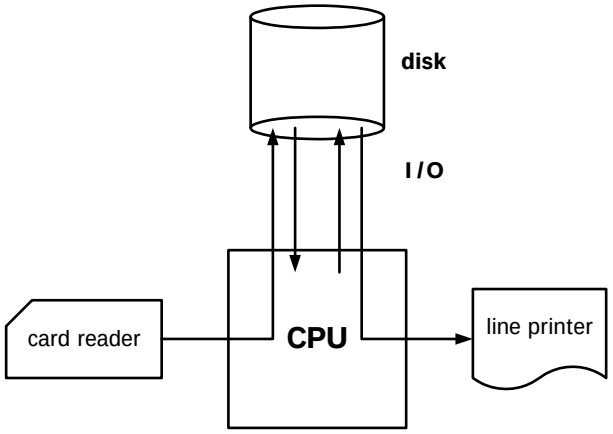
เพื่อที่จะให้กระบวนการเร็วขึ้น งานที่คล้าย ๆ กันก็จะถูกเอามารวมกัน แล้ว run เป็นกลุ่ม ดังนั้นโปรแกรมเมอร์ก็จะเอาโปรแกรมของพวกเขาไปให้ operator operator จะเรียงโปรแกรมเหล่านั้นเป็นกลุ่มที่คล้าย ๆ กน แล้วให้คอมพิวเตอร์ run ทีละกลุ่ม (batch) ผลลัพธ์ (output) ที่ได้โปรแกรมเมอร์จะพอใจกว่าแบบเดิม

ระบบปฏิบัติการแบบกลุ่มจะอ่าน card reader ซึ่งแต่ละ control card ก็จะทำงานไป เมื่องานเสร็จก็ print ออกมา จะเห็นว่าระบบแบบกลุ่ม (batch) จะไม่มีการปฏิสัมพันธ์ (interaction) ระหว่างผู้ใช้และงาน (job) ในขณะ execute เพราะฉะนั้น delay ระหว่างช่วงที่รังงานจนถึงงานเสร็จ จะเรียกว่า turnaround time

จะเห็นได้ว่า CPU จะว่างบ่อยมาก เนื่องจาก speed ของเครื่องจักร อุปกรณ์รับส่งข้อมูลช้ากว่าอุปกรณ์อิเล็กทรอนิกส์ ตัวอย่างเช่น CPU ทำงานช้า ๆ ก็เป็น ไมโครวินาที หรือทำงานตามคำสั่งได้ 1000 ชุดคำสั่งใน 1 วินาที แต่ card reader อย่างเร็วก็ทำได้เพียง 20 card ในวินาที

ปัญหานี้ถูกแก้ไขโดย disk technology แทนที่ card จะถูกอ่านจากเครื่องอ่านเข้าสู่หน่วยความจำและงานจะถูก process เราควรอ่าน card เข้าสู่ disk ตำแหน่งของ card image จะถูกบันทึกไว้เป็น

ตาราง โดยระบบปฏิบัติการ เมื่องานถูก execute ระบบปฏิบัติการจะร้องขอให้เครื่องอ่านทำการอ่านจาก disk เช่นเดียวกับ output เมื่องานร้องขอ printer ให้พิมพ์ 1 บรรทัด บรรทัดนั้นจะถูกคัดลอกไปไว้ใน buffer และถูกเขียนใน disk เมื่องานเสร็จ output จะถูก print จริง ๆ รูปแบบของกระบวนการนี้เรียกว่า **spooling** ดังรูป



รูปที่ 1.3 Spooling

Spooling ผ่าน remote site (ทางไกลได้) CPU จะส่งข้อมูลผ่านเส้นทางติดต่อสื่อสารไปสู่ remote printer

Spooling overlap

ในระบบอย่างง่าย spooler อาจอ่าน input จากงานหนึ่งในขณะกำลัง print output ของอีกงานหนึ่ง ในระหว่างนั้นยังคงมีอีกงานหนึ่ง execute อยู่

ประโยชน์ของ Spooling

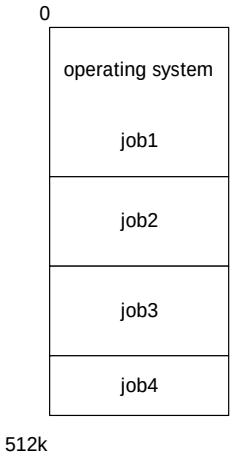
1. Overlap job ได้
2. ทำให้ CPU และ อุปกรณ์รับส่งข้อมูลทำงานได้ในอัตราที่สูงขึ้น

1.1.3 ลักษณะระบบคอมพิวเตอร์แบบไทม์แชร์ริ่ง

ในสมัยแรก ผู้ใช้คอมพิวเตอร์ต้องเข้าจองและครอบครองเครื่อง ตลอดช่วงเวลางานของเขาเรียกได้ว่าเครื่องเป็นของผู้ใช้คนนั้นๆ โดยสมบูรณ์ ในช่วงเวลานั้นเข้าสามารถทำอะไร ๆ เกี่ยวข้องกับงานได้ตามใจประสงค์ เช่นแก้ไขโปรแกรม ตรวจสอบดูข้อมูลในหน่วยความจำหลัก ฯลฯ แต่เมื่อมีการผลักดันให้ใช้เครื่องอย่างมีประสิทธิภาพ เพราะเครื่องมีราคาแพงมากเกินกว่าจะปล่อยให้ผู้ใช้มานั่งขบคิดแก้ไขโปรแกรมอยู่หน้าเครื่องได้ ความสะดวกในการใช้เครื่องของผู้ใช้ก็ลดลงเรื่อยๆ ยิ่งเมื่อมีระบบป้อนงานแบบกลุ่มออกมาใช้เป็นที่แพร่หลาย ผู้ใช้ส่วนมากก็แทบไม่ได้

เห็นเครื่องคอมพิวเตอร์เลย แต่เมื่อการใช้เครื่องแพร่หลายขึ้น ผนวกกับความก้าวหน้าของศาสตร์ด้านโปรแกรมระบบ ทำให้เกิดแรงผลักดันที่จะให้ผู้ใช้ได้เป็นเจ้าของเครื่องอีก หรืออย่างน้อยก็เสมือนว่า ผู้ใช้ได้เป็นเจ้าของเครื่องแต่ผู้เดียว

ระบบปฏิบัติการจะเก็บงานหลาย ๆ งานไว้ในหน่วยความจำ ดังรูป



รูปที่ 1.4 การเก็บงานไว้ในหน่วยความจำ

กลุ่มของงานดังกล่าวเป็นเพียงสับเซตของงานที่เก็บไว้ในบ่องาน (pool job) ระบบปฏิบัติการจะนำงานแรกมาทำในหน่วยความจำ จากนั้นอาจต้องรอ tapes กำลังอ่านหรือรอการทำงานของ I/O ในระบบแบบ multiprogram ระบบปฏิบัติการจะ switch ไปทำงานอีกงานเมื่องานที่ 2 ต้องรอ CPU จะ switch ไปทำงานอีกงาน เมื่องานที่ 2 ต้องรอ CPU จะ switch ไปอีกงานไปเรื่อย ๆ จนวนมาถึงคิวของงานแรก CPU ก็จะไม่มีทางว่าง

ตัวอย่าง คนไข้มาหาหมอหลายคน คนแรกหมอก็คุย คนอื่นก็ไปทำประวัติหรืออ่านหนังสือพิมพ์ หรือนั่งรอเฉย ๆ เมื่อคนไข้คนแรกต้องไป x-ray หมอก็เรียกคนไข้อีกคนมารักษา เช่นนี้หมอก็จะไม่เสียเวลาไปโดยเปล่าประโยชน์

1. <u>การทำงานแบบโต้ตอบ</u>

ลักษณะการทำงานแบบป้อนงานเป็นกลุ่ม (batch processing) นั้น ผู้ใช้และเครื่องติดต่อกันโดยผ่านภาษาควบคุมงาน และผ่านผลลัพธ์ที่ได้ออกมาจากเครื่อง เรียกได้ว่าเป็นการสื่อสารแบบไร้สาย (off-line) คือไม่ได้ทำงานโดยตรงกับเครื่อง แต่เป็นลักษณะการทำงานโต้ตอบ (interactive system) ผู้ใช้ติดต่อกับเครื่องในลักษณะตามสาย (on-line) โดยผู้ใช้ได้รับผลสนองตอบจากเครื่อง หรือจากระบบปฏิบัติการหรือจากโปรแกรมของผู้ใช้เอง โดยทันทีหรือเกือบจะทันที หลังจากผู้ใช้ป้อนคำสั่งใดๆ เข้าไป ลักษณะการสื่อสารเช่นนี้ มักกระทำผ่าน

เทอร์มินัล (terminal) ซึ่งประกอบด้วยอุปกรณ์รับข้อมูล คือแป้นพิมพ์ดีด (keyboard) และอุปกรณ์แสดงผล คือจอภาพหรือเครื่องพิมพ์ (display / printer) ผนวกรวมอยู่เป็นอุปกรณ์เดียวกัน ในลักษณะนี้ผู้ใช้สามารถแก้ไขโปรแกรมสักเล็กน้อย แล้วสั่งให้ระบบปฏิบัติการแปลโปรแกรม และลองประมวลผลดู ซึ่งจุได้ผลลัพธ์ของการแปลและการประมวลผลออกมาบนจอภาพทันทีทันใด (ในระบบจริง จะล่าช้าบ้าง ขึ้นกับองค์ประกอบหลายประการ เช่น ขนาดของงาน ขีดความสามารถของเครื่อง ปริมาณงานในระบบ และลำดับความสำคัญของงานที่ป้อนเข้าไป เป็นต้น)

2. <u>ระบบโต้ตอบแบบมัลติโปรแกรมมิ่ง</u>

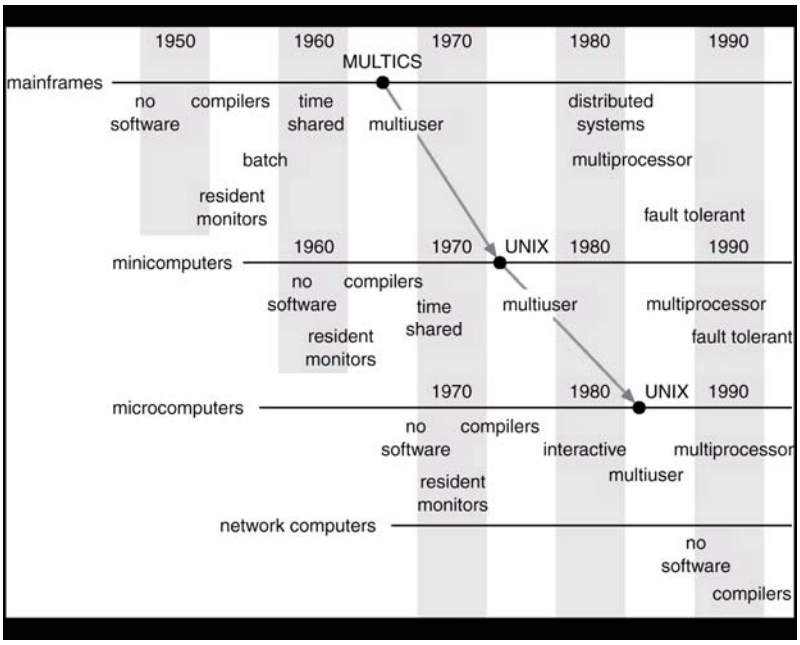
ลักษณะการทำงานแบบโต้ตอบ จะเห็นได้ชัดในระบบปฏิบัติการของเครื่องขนาดเล็ก (microcomputer) ซึ่งปกติไม่มีการทำงานแบบมัลติโปรแกรมมิ่งอยู่แล้ว เนื่องจากผู้ใช้เป็นเจ้าของระบบทั้งหมด (ในขณะที่ทำงานอยู่) ความเร็วในการตอบสนอง (response time) ที่ผู้ใช้รู้สึกได้จะขึ้นกับชนิดและขนาดของงานที่ทำกับความเร็วของอุปกรณ์ต่างๆ ของเครื่องไมโครคอมพิวเตอร์เอง แต่ในการใช้ระบบตอบโต้กับเครื่องใหญ่ ที่เป็นแบบมัลติโปรแกรมมิ่ง เครื่องมิได้เป็นของผู้ใดโดยเฉพาะ ทว่าต้องเฉลี่ยการใช้งานออกไป และยังอาจมีการทำงานแบบกลุ่ม รวมถึงระบบ spooling อยู่ด้วย ดังนั้นหากจะให้ผู้ใช้ผู้ใดผู้หนึ่งได้รับการตอบสนองสูงสุด ก็เป็นการปิดกั้นผู้ใช้และงานอื่นๆ ไป ดังนั้นในระบบมัลติโปรแกรมมิ่ง จะต้องมีการประนีประนอมระหว่างอัตราการตอบสนองและความยุติธรรม (ทั้งระหว่างผู้ใช้ระบบโต้ตอบด้วยกันเอง และระหว่างระบบโต้ตอบกับระบบทำงานแบบอื่นๆ)

หลักการสร้างความยุติธรรมภายในกลุ่มผู้ใช้ระบบโต้ตอบคือ การแบ่งเวลา (timesharing) ของหน่วยประมวลผลกลางออกเป็นส่วนๆ สำหรับผู้ใช้แต่ละคน เช่น หากมีหน่วยประมวลผลกลางที่สามารถประมวลผลได้หนึ่งล้านคำสั่งต่อวินาที ในลักษณะนี้ผู้ใช้แต่ละคนจะมีความรู้สึกว่ากำลังทำงานโต้ตอบกับเครื่องคอมพิวเตอร์ ซึ่งสามารถประมวลคำสั่งได้หนึ่งแสนคำสั่งในแต่ละวินาที หรือเป็นเครื่องซึ่งมีขีดความสามารถสูง อัตราการตอบสนองผู้ใช้จะสูงตามไปด้วย แต่หากมีจำนวนผู้ใช้ระบบโต้ตอบมาก อัตราตอบสนองจะลดลง เพราะเครื่องต้องถูกแบ่งกระจายไปบริการผู้ใช้จำนวนมาก

3. <u>Personal Computer System</u>

PC เริ่มในยุค 1970 ซึ่งก็คือ microcomputer ในช่วงแรก ๆ CPU ยังขาดส่วนที่จะป้องกันระบบจากโปรแกรมของผู้ใช้ ระบบปฏิบัติการของ PC ก็เป็น multiuser หรือไม่ก็ multitasking วัตถุประสงค์ของระบบปฏิบัติการของ PC คือ ทำให้ผู้ใช้ได้รับการตอบสนองและมีความสะดวกมากที่สุด ระบบที่ run บน PC ก็เช่น Microsoft Windows , Apple Macintosh ระบบปฏิบัติการของ microcomputer

ได้รับการพัฒนามาจาก Mainframe รูปต่อไปนี้เป็นการทดสอบระบบปฏิบัติการเปรียบเทียบกันของ mainframe ,minicomputer และ microcomputer



รูปที่ 1.5 การเปรียบเทียบกันของ mainframe ,minicomputer และ microcomputer

ตัวอย่างที่ดีของความเปลี่ยนแปลงนี้คือระบบปฏิบัติการ MULTICS MULTICS ถูกพัฒนามาจาก ปี 1965 – 1970 โดยสถาบัน Massachusetts Institute of Technology (MIT) ซึ่ง run บน mainframe ที่ซับซ้อนและมีขนาดใหญ่ (GE645) แนวความของคิด MULTICS ถูกพัฒนาต่อโดย Bell Laboratory จนเป็น UNIX ระบบปฏิบัติการ UNIX ถูกออกแบบในยุค 1970 ใช้บน PDP-11 minicomputer ในยุค 1980 UNIX บน minicomputer ถูกพัฒนาเป็น UNIX บน microcomputer ได้ ต่อมาจึงเริ่มเป็น Windows NT, IBM OS/2 , และ Macintosh จะเห็นว่า ระบบปฏิบัติการจะถูกพัฒนามาจาก mainframe ขนาดใหญ่จนมาเป็น microcomputer

1.1.4 ลักษณะระบบคอมพิวเตอร์แบบขนาน

คือระบบ multiprocessor ที่มี CPU มากกว่า 1 ตัว ในการติดต่อสื่อสาร และเป็น Tightly coupled system คือ processor มีการ share memory และ clock การติดต่อสื่อสารจะผ่านทาง share memory

ข้อดีของ Parallel System คือ

1. เพิ่ม throughput แต่ไม่ใช่ว่าเราเพิ่มจำนวน processor ไป n ตัว แล้วจะทำให้อัตรเร็วที่เพิ่มขึ้นเป็น n เท่าไปด้วย ตัวอย่างเช่น ถ้าเราโปรแกรมเมอร์ n คน ไม่จำเป็นว่าจะได้ผลลัพธ์เป็น n เท่าของงานที่จะเสร็จ
2. ประหยัด เพราะสามารถ share ทรัพยากรกันได้
3. เพิ่มความน่าเชื่อถือ เช่น ถ้าเรามี 10 processor แล้วเสียไป 1 ที่เหลือก็ยังคงทำงานได้ แต่อาจช้าลงหน่อย สิ่งนี้เป็นการช่วยระดับของความอยู่รอดของฮาร์ดแวร์ซึ่งถูกเรียกว่า graceful degradating (การแตกตัวอย่างสวยงาม) ระบบที่ออกแบบมาสำหรับ graceful degradation เรียกว่า fault-tolerant (ความทนทานต่อความผิดพลาด)

Symmetric-multiprocessing model

- ใช้ระบบปฏิบัติการเดียวกันทุก ๆ processor
- Processor ทุกตัวทำงานพร้อมกันได้โดยไม่มีการลดประสิทธิภาพ (performance deterioration)
- ระบบปฏิบัติการในปัจจุบันมีการสนับสนุน symmetric-multiprocessing model

Asymmetric-multiprocessing model

- มีการกำหนดงานแต่ละชิ้นให้กับ processor มี processor ตัวหลักคอยบริหารและจัดสรรทรัพยากร เป็นคุณสมบัติแบบ master-slave
- นิยมใช้ในระบบขนาดใหญ่

1.1.5 ลักษณะระบบคอมพิวเตอร์แบบกระจายงาน

เป็นการแจกจ่ายงานให้กับ processors ที่มีอยู่ เราเรียก Distributed Systems อีกชื่อว่า Loosely coupled system คือ processor แต่ละตัวจะมีหน่วยความจำเป็นของตัวเอง การสื่อสารระหว่าง processor ก็ทำได้หลายวิธี เช่น ผ่านทาง high-speed buses หรือสายโทรศัพท์

ข้อดีของ Distributed system คือ

1. Resource Sharing เป็นการลดค่าใช้จ่ายในการซื้อเครื่องมือ เช่น share กันใช้ printer ร่วมกัน
2. Computation Speedup (การทำงานทำได้เร็วขึ้น) หรืออีกชื่อคือ load sharing ถ้า site ที่เราทำงานอยู่มีงานที่ overload อยู่ ก็จะมีการส่งงานบางส่วนไปยัง site อื่น ที่ ไม่ค่อยมีงานหนัก การย้ายงานแบบนี้เราเรียก load sharing
3. Reliability (ความน่าเชื่อถือ) ถ้าเครื่อง A เสีย เราสามารถโอนข้อมูลไปเครื่อง B เพื่อทำงานต่อโดยไม่ต้องรอได้
4. Communication ในการแลกเปลี่ยนข้อมูล ไม่ว่าการ share หรือ transfer ข้อมูล เราก็ทำได้โดยใช้ Electronic mail

1.1.6 ลักษณะระบบคอมพิวเตอร์แบบเรียวไทม์

ใช้ส่วนใหญ่ในระบบเจาะจงพิเศษ เช่น งานทดลองทางวิทยาศาสตร์ ระบบภาพทางการแพทย์ (Medical imaging systems), งานควบคุมทางอุตสาหกรรม , งานแสดงผลบางอย่าง Real-time systems จะทำงานได้ดี ถ้ามีข้อกำหนดในเรื่อง fixed-time

ระบบ Real-time systems มี 2 ระบบดังนี้

1. **Hard Real-Time System** (ทำงานได้เสร็จตรงตามเวลา)

   ไม่มี Harddisk หรือมีขนาดเล็ก การเก็บข้อมูลจะเก็บใน short-term memory หรือ ROM ข้อเสียคือ มีปัญหากับระบบ time-sharing และ ไม่มีการ support จากระบบปฏิบัติการทั่วไป

2. **Soft Real-Time System** (ขาด deadline)

- มีการจำกัด utility และเสียงในการใช้ในอุตสาหกรรมควบคุมหรือหุ่นยนต์
- มีประโยชน์ในการประยุกต์ใช้ใน multimedia , virtual reality และการสำรวจใต้น้ำหรือดาวเคราะห์
- ระบบนี้ต้องการ features ของระบบปฏิบัติการขั้นสูง

ระบบ Real-time systems มักจะมีเวลาในการทำงานจำกัด ถ้าระบบไม่ให้ผลลัพธ์ได้ทันในช่วงเวลานั้น ระบบก็จะล้มเหลว ซึ่งต่างจาก time sharing ซึ่งตอบสนองได้ทันทีหรือระบบ Batch ซึ่งไม่มีการจำกัดเวลาในการตอบสนองเลย

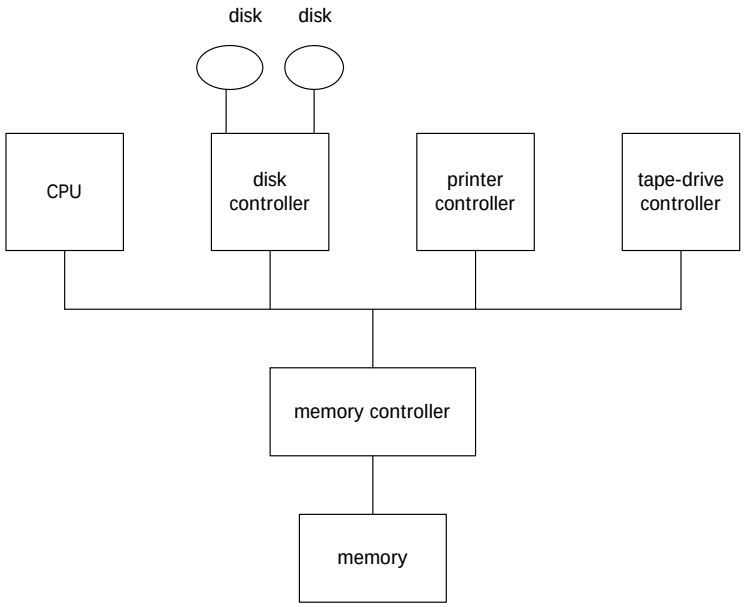
ในงานบางประเภท ความต้องการในอัตราตอบสนองจะสูงมาก นั่นคือเมื่อมีข้อมูลเข้ามาสู่เครื่อง จะต้องคำนวณประมวลผลให้เสร็จสิ้นและส่งผลลัพธ์ออกไปให้รวดเร็วที่สุดเท่าที่จะทำได้ เพราะงานที่ใช้ระบบประเภทนี้เป็นงานที่วิกฤตในด้านการตอบสนอง ซึ่งโดยปกติจะเป็นการควบคุมเครื่องจักรกล เพราะการควบคุมนี้ต้องกระทำให้ทันกับสภาพที่แปรผันไปในงานมิเช่นนั้นงานจะเสียหายได้ ตัวอย่างเช่น การควบคุมน้ำเข้าถังเก็บ ข้อมูลที่เข้ามาจะเป็นสัญญาณจากอุปกรณ์ตรวจสอบระดับน้ำ (เช่นลูกลอย) ซึ่งหากน้ำเต็มถัง เครื่องคอมพิวเตอร์ต้องประมวลออกเป็นผลลัพธ์ในลักษณะของสัญญาณไฟฟ้าไปควบคุมให้เครื่องสูบน้ำเข้าถังหยุดทำงาน

งานประเภทระบบตอบสนองฉับพลัน (real - time system) นี้ คำนึงถึงอันตราตอบสนองเหนือสิ่งอื่นใด ลักษณะการใช้หน่วยประมวลผลกลางจึงมักจะมีประสิทธิภาพต่ำมาก เพราะหน่วยประมวลกลางต้องว่างตลอดหรือเกือบตลอดเวลา เพื่อที่จะได้ประมวลงานได้ทันทีเมื่อมีข้อมูลเข้ามาในลักษณะนี้หน่วยงานที่ควบคุมจะกันคอมพิวเตอร์ทั้งระบบเอาไว้สำหรับงานแบบตอบสนองฉับพลันนี้โดยเฉพาะ หรือหากผนวกเข้ากับระบบประมวลผลอื่นๆ จะกำหนดให้งานประเภทนี้มีความสำคัญสูงสุด ตัวอย่างของงานประเภทนี้ได้แก่ การบันทึกข้อมูลสำหรับการทดลองทางวิทยาศาสตร์ การตรวจสอบดูแลคนไข้ การควบคุมระบบโรงงาน และการแสดงผลบางประเภท เช่น real - time simulation เป็นต้น

1.2 โครงสร้างของระบบคอมพิวเตอร์ (COMPUTER SYSTEM STRUCTURES)

1.2.1 ระบบการทำงานของคอมพิวเตอร์ (Computer-System Operation)

ระบบคอมพิวเตอร์ยุคใหม่จะมี CPU, device controller ซึ่งเชื่อมโยงกันผ่าน common bus ซึ่ง share memory ร่วมกัน ดังรูป



รูปที่ 1.6 ระบบคอมพิวเตอร์

- Device controller แต่ละตัวรับผิดชอบ device ของตัวเอง
- CPU และ device controller สามารถทำงานพร้อมกันได้
- มี memory controller เพื่อจัดลำดับการ share memory

เมื่อคอมพิวเตอร์เริ่ม running ตัวอย่างเช่น เมื่อเปิดสวิตช์ หรือ reboot ก็จำเป็นที่จะต้อง run โปรแกรมเริ่มต้น (initial program) หรือ boot strap program ซึ่งเป็นการนำ register ของ CPU และ device controller เข้าสู่หน่วยความจำ bootstrap program ต้องถูก load ไปไว้ใน kernel ของระบบปฏิบัติการ ระบบปฏิบัติการจึงจะเริ่ม execute โปรแกรมแรก เช่น “init” และรอผลลัพธ์ ผลที่

ได้มักเป็นสัญญาณขัดจังหวะ จากฮาร์ดแวร์หรือซอฟแวร์ ฮาร์ดแวร์อาจขัดจังหวะโดยส่งสัญญาณผ่านทาง system bus มาที่ CPU โดยซอฟแวร์ ขัดจังหวะโดยการปฏิบัติการบางอย่างที่พิเศษเรียกว่า system call หรือ monitor call

เมื่อ CPU ถูกขัดจังหวะ ก็จะหยุดทำงานที่กำลังทำอยู่ แล้วหันไปทำงานตามสัญญาณขัดจังหวะนั้นทันที โดยย้ายไปทำงานยังตำแหน่งในหน่วยความจำที่บรรจุโปรแกรมสำหรับดำเนินการกับสัญญาณนั้น (interrupt service routine) เมื่อดำเนินการเสร็จ CPU ก็จะกลับไปทำงานเดิมที่ค้างไว้หน้าที่โดยทั่วไปของสัญญาณขัดจังหวะ

- สัญญาณขัดจังหวะจะส่งการควบคุมไปยัง interrupt service routine ผ่านทาง ตาราง สัญญาณขัดจังหวะ (interrupt vector) (array ของ address ของ service routine ต่าง ๆ)
- สถาปัตยกรรมของสัญญาณขัดจังหวะ จะต้องบันทึกตำแหน่งของชุดคำสั่งที่ถูกขัดจังหวะไว้
- สัญญาณขัดจังหวะที่เข้าสู่ระบบจะถูก disable ถ้ามีการทำงานของสัญญาณขัดจังหวะตัวอื่นอยู่ก่อนแล้ว เพื่อป้องกันการสูญหายของสัญญาณขัดจังหวะ (lost interrupt)
- ในระบบที่ซับซ้อนขึ้นอาจยอมให้มีการขัดจังหวะซ้อน ๆ กันได้โดยเรียงตามศักดิ์(Priority) สัญญาณที่มีศักดิ์สูงกว่าอาจขัดจังหวะสัญญาณที่มีศักดิ์ต่ำกว่า แต่ถ้ามีศักดิ์เท่ากันต้องรอ interrupt พร้อมกันไม่ได้
- ระบบปฏิบัติการยุคใหม่ ใช้ ตัวขับสัญญาณขัดจังหวะ (interrupt driver) ถ้าไม่มีการ process ไม่มีการเรียกใช้อุปกรณ์รับส่งข้อมูล ไม่มีการตอบสนองผู้ใช้ ระบบปฏิบัติการก็ไม่ต้องทำอะไร นั่นเป็นสัญญาณบอกเหตุว่าเกิด interrupt หรือ trap (กับดัก)
- trap คือ software-generated interrupt (การเกิดการขัดจังหวะของซอฟแวร์) ซึ่งเกิด error หรือไม่ก็เกิดจาก การร้องขอของโปรแกรมของผู้ใช้

<u>การจัดการสัญญาณขัดจังหวะ</u>

1. ระบบปฏิบัติการจะรักษาสถานภาพ (state) ของ CPU ด้วยการเก็บ registers และ program counter ไว้
2. ตรวจสอบดูว่าเกิดสัญญาณขัดจังหวะชนิดไหน
   - อาจเป็น polling (การร้องขอของอุปกรณ์รับส่งข้อมูล)
   - หรือ vectored interrupt system (การระบุตำแหน่ง (address) ผิด)
3. แยกส่วนของ code เพื่อกำหนดว่าควรทำอะไรกับแต่ละชนิดของการขัดจังหวะนั้น

1.2.2 ระบบอินพุต-เอาต์พุต

I/O Interrupts

เมื่อ I/O เริ่มทำงาน CPU จะ load register ที่จำเป็นมาไว้ใน device controller ซึ่ง device controller จะทำการตรวจสอบ register เหล่านั้น เพื่อกำหนดว่าจะทำงานอะไร เช่น ถ้าพบร้องขอให้อ่านข้อมูล controller จะเริ่มโอนย้ายข้อมูลจาก device ไปไว้ที่ local buffer เมื่อโอนย้ายข้อมูลเสร็จ device controller จะบอก CPU ว่าทำงานเสร็จแล้ว การติดต่อสื่อสารนี้จะสำเร็จได้โดยสัญญาณขัดจังหวะ

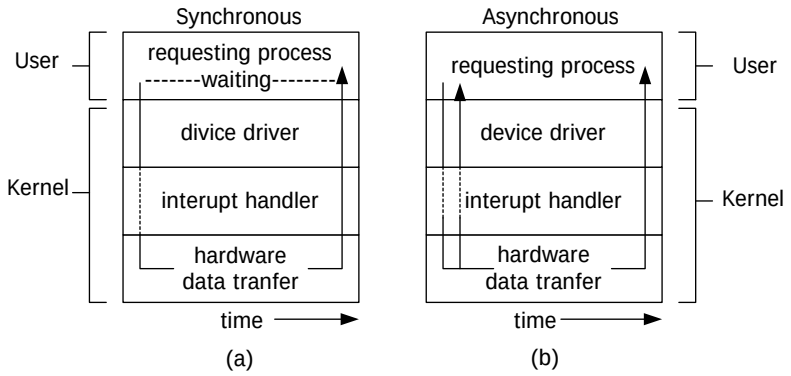
Synchronous I/O

- เมื่อการรับส่งข้อมูลเริ่มขึ้น การจะโยกย้ายการควบคุมให้กับโปรแกรมของผู้ใช้ จะทำได้หลังจากเสร็จสิ้นการรับส่งข้อมูลเท่านั้น
- ในการรอรับส่งข้อมูลเสร็จ มี 2 วิธี
    1. คอมพิวเตอร์บางเครื่องมีชุดคำสั่ง wait พิเศษ ซึ่งปล่อยให้ CPU ว่าง จนกระทั่งเกิดสัญญาณขัดจังหวะถัดไป
    2. เครื่องจักที่ไม่มีชุดคำสั่งดังกล่าวอาจจะมี wait loop

loop : jmp loop

loop นี้จะวนรอบไปเรื่อย ๆ จนกระทั่งเกิดสัญญาณขัดจังหวะก็จะโอนย้ายการควบคุมไปส่วนอื่น ๆ ของระบบปฏิบัติการ

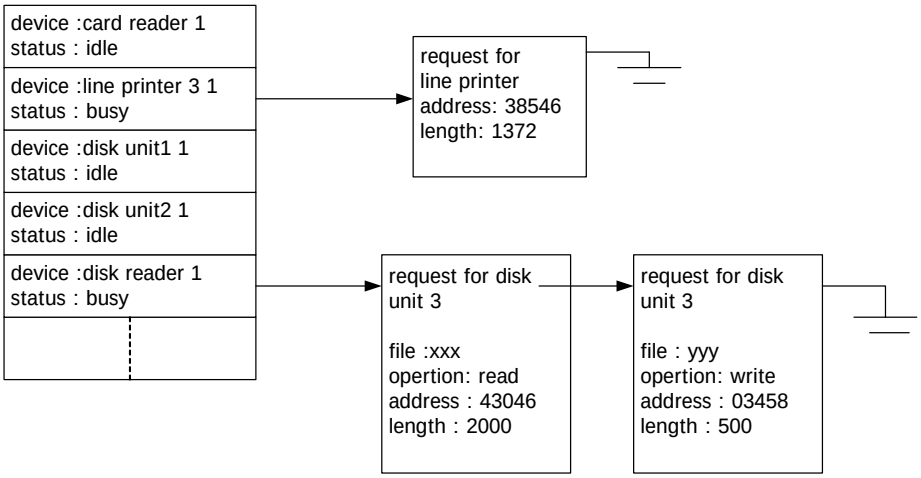
- ถ้า CPU ต้องรอให้รับส่งข้อมูลเสร็จงานก่อนเสมอ แสดงว่าต้องมีการร้องขอของ I/O อยู่หนึ่งตัวที่เด่นอยู่ตลอดเวลา ดังนั้นเมื่อเกิดสัญญาณการขัดจังหวะการรับส่งข้อมูล ระบบปฏิบัติการจะรู้ทันทีว่า device กำลังถูกขัดจังหวะ แต่ไม่สามารถประมวลผล I/O device หลาย ๆ ตัวพร้อมกันได้
- ตัวอย่างของระบบนี้คือ ระบบปฏิบัติการ MS-DOS เวลาสั่งพิมพ์ ต้องรอเสร็จงานก่อนถึงจะทำงานต่อไปได้



รูปที่ 1.7 Synchronous And Asynchronous

Asynchronous I/O

- เมื่อการรับส่งข้อมูลเริ่มขึ้นการโยกย้ายการควบคุมให้กับโปรแกรมของผู้ใช้ทำได้โดยไม่ต้องรอให้การรับส่งข้อมูลเสร็จ
- System call (คำสั่งของระบบปฏิบัติการ) อนุญาตให้โปรแกรมของผู้ใช้รอคอยให้รับส่งข้อมูลเสร็จ
- ตารางที่ระบบปฏิบัติการใช้เก็บบันทึกของอุปกรณ์รับส่งข้อมูลแต่ละตัว คือ device-status table ซึ่งใช้แสดงชนิดของอุปกรณ์ ที่อยู่(address) และ สถานภาพ(state) (ว่าง , กำลังทำงาน หรือเสีย) ดังรูป



รูปที่ 1.8 ตารางที่ระบบปฏิบัติการใช้เก็บบันทึกของอุปกรณ์รับส่งข้อมูล

เมื่อเกิดสัญญาณขัดจังหวะ ระบบปฏิบัติการจะตรวจสอบดูว่าสัญญาณขัดจังหวะมาจากอุปกรณ์รับส่งข้อมูลตัวไหน แล้วชี้ไปยังตารางของอุปกรณ์รับส่งข้อมูลนั้นเพื่อตรวจสอบสถานะของอุปกรณ์นั้น เพื่อปรับปรุงค่าในตารางให้ถูกต้องตามสัญญาณขัดจังหวะนั้น ถ้ามีคิวของอุปกรณ์อยู่ ระบบปฏิบัติการก็จะทำงานตามการร้องขอถัดไป เมื่อเสร็จก็จะคืนการควบคุมไปทำงานเดิมที่ถูกขัดจังหวะ เช่น โปรแกรมกำลังรอผลลัพธ์จากอุปกรณ์รับส่งข้อมูลอยู่ก็จะทำงานต่อไปได้เลย

- ข้อดีของ asynchronous I/O คือ การเพิ่มประสิทธิภาพของระบบ

ตัวอย่างของระบบนี้ เช่น Windows เวลากำลังพิมพ์งาน เราจะทำการยกเลิกงานที่พิมพ์อยู่ทันทีได้

DMA Structure

ถ้าผู้ใช้ป้อนข้อมูลผ่านทางแป้นพิมพ์ซึ่งมีอัตรารับ 9600 baud (9600 สัญญาณต่อวินาที) การส่งอักษร 1 ตัวจะใช้เวลาประมาณ 1 มิลลิวินาที หรือ 1000 ไมโครวินาที โปรแกรมสำหรับดำเนินการกับสัญญาณขัดจังหวะ (interrupt service routine) จะอ่านอักษรไปเก็บไว้ที่ที่พักข้อมูล (buffer) อาจใช้เวลาทำการประมาณ 2 ไมโครวินาที/ตัวอักษร ดังนั้นหน่วยประมวลผลกลางเมื่อถูกขัดจังหวะด้วยโปรแกรมนี้จะยังคงมีเวลาเหลืออีก 1000 – 2 = 998 ไมโครวินาที ต่อทุก ๆ 1000 ไมโครวินาที (เวลาที่เหลืออาจใช้ทำงานอื่น ๆ หรือตอบสนองการขัดจังหวะอื่นได้) สัญญาณที่มาจากอุปกรณ์รับส่งข้อมูลประเภทนี้จึงมักถูกกำหนดให้มีศักดิ์ (priority) ต่ำ เพื่อให้สัญญาณประเภทอื่นได้มีโอกาสทำก่อน สำหรับอุปกรณ์รับส่งข้อมูลที่มีความเร็วสูง (high-speed device) เช่น เทป , จานแม่เหล็ก (disk) หรือ ข่ายงานสื่อสาร (communication network) อาจมีความเร็วในการส่งข้อมูลได้ใกล้เคียงกับความเร็วของหน่วยความจำ CPU อาจต้องใช้เวลา 2 ไมโครวินาทีในการตอบสนองสัญญาณที่มาขัดจังหวะ 1 ครั้ง แต่สัญญาณที่มาถึงอาจมีความเร็ว 4 ไมโครวินาทีต่อครั้งก็ได้ ซึ่งก็จะเกิดปัญหาทันที จึงมีการใช้ Direct Memory Access (การเข้าถึงหน่วยความจำโดยตรง) มาแก้ปัญหา สำหรับอุปกรณ์ที่มีความเร็วสูงเหล่านี้ ตัวควบคุมอุปกรณ์จะส่งข้อมูลจาก buffer ของตนมายังหน่วยความจำหลักโดยตรงทีละชุด (ไม่ใช่ทีละ 1 อักษร) โดยไม่ได้อาศัยหน่วยประมวลผลกลางเลย สัญญาณก็จะลดลงเหลือเพียง 1 ครั้งต่อชุด ไม่เหมือนอุปกรณ์ความเร็วต่ำที่ส่งสัญญาณ 1 ครั้ง/อักษร การทำงานก็เหมือนเดิม คือ เมื่อโปรแกรมของผู้ใช้ต้องการรับส่งข้อมูลไปยังอุปกรณ์เหล่านี้ ระบบก็จะจัด buffer (อาจเป็น empty buffer ในกรณีรับ input หรือเป็น full buffer ในกรณีต้องการส่ง output) ขนาดของ buffer โดยทั่วไปจะอยู่ระหว่าง 128 – 4096 ไบต์ ทั้งนี้ขึ้นอยู่กับชนิดของอุปกรณ์นั้น ๆ จากนั้นส่วนของระบบปฏิบัติการที่เรียกว่า device driver จะ set ตัวควบคุมหน่วยความจำ (DAM controller) ให้เก็บค่าตำแหน่งที่รับ , ส่ง และขนาดของข้อมูลไว้ใน register ของตน แล้วส่งสัญญาณไปยังอุปกรณ์ที่ต้องการผ่าน register ควบคุม (โดยใช้บิตควบคุม (control bit)) เพื่อให้อ่านข้อมูลเข้ามาที่ buffer ของตน แล้วตัวควบคุมก็จะส่งข้อมูลต่อไปที่หน่วยความจำหลัก หลังจากส่งข้อมูลเรียบร้อยแล้ว ตัวควบคุมจะส่งสัญญาณไปยังหน่วยประมวลผลกลาง

1.2.3 ระบบจัดเก็บข้อมูล (Storage Structures)

ในอุดมคติ เราต้องการให้โปรแกรมและข้อมูลอาศัยอยู่ใน main memory อย่างถาวร แต่ทำไม่ได้ด้วยเหตุผล 2 ประการ

1. Main Memory เล็กเกินไปที่จะเก็บโปรแกรมและข้อมูลที่จำเป็นทั้งหมดอย่างถาวร
2. Main Memory เป็นอุปกรณ์ที่ใช้เก็บข้อมูลแบบชั่วขณะ เพราะเมื่อปิดเครื่องข้อมูลก็หาย ดังนั้นคอมพิวเตอร์จึงต้องมี secondary storage (หน่วยเก็บข้อมูลสำรอง) เพื่อเก็บข้อมูลมาก ๆ ได้อย่างถาวร

ในเรื่อง storage structures เราจะพูดถึง main memory, จานแม่เหล็ก (magnetic disk) และ magnetic tape

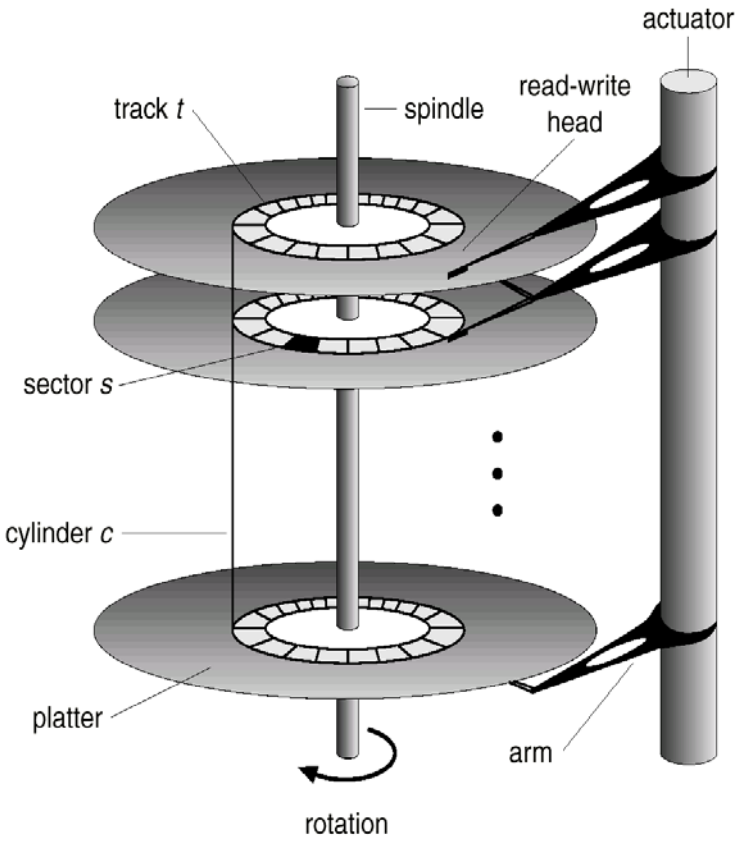
Main Memory

- เป็นสื่อที่ใช้เก็บข้อมูลขนาดใหญ่ที่อยู่ใน CPU ซึ่ง CPU สามารถจะเข้าถึงได้โดยตรง
- คำสั่งและข้อมูลต้องอยู่ใน memory ก่อนที่จะทำงาน ถ้าข้อมูลไม่อยู่ใน memory จะต้องย้ายเข้ามาก่อนที่ CPU จะดำเนินการกับข้อมูลเหล่านั้น
- Memory-mapped I/O (ใช้เพื่ออำนวยความสะดวกในการเข้าถึง I/O device) คือช่วงของที่อยู่(address) ของ memory ที่ใช้ map ไปยัง device register

การอ่าน/เขียน address ของ memory เป็นเหตุให้ข้อมูลถูกโยกย้ายไป/กลับจาก device register วิธีนี้ใช้ได้ดีกับอุปกรณ์ที่ตอบสนองได้รวดเร็ว เช่น video controller

Magnetic Disks

คือโลหะแข็งหรือ แผ่นเสียงแก้ว (platter) ที่ปกคลุมไปด้วยสารแม่เหล็ก (ดูรูปประกอบ)



รูปที่ 1.8 Magnetic Disks

แต่ละจานแผ่นเสียงมีรูปร่างกลมแบนเหมือน CD เส้นผ่านศูนย์กลางประมาณ 1.8 – 5.25 นิ้ว พื้นผิวทั้ง 2 หน้า ปกคลุมไปด้วยสารแม่เหล็ก เหมือนเทปแม่เหล็ก (magnetic tape) เราเก็บข้อมูลข่าวสาร (Information) โดยบันทึกบน platter

- หัวอ่าน-เขียนอยู่เหนือแต่ละพื้นผิวของทุก ๆ platter และ หัวอ่าน-เขียนอยู่บน disk arm
- เราแบ่งพื้นผิวของ platter เป็นวงกลมเรียกว่า track ซึ่งแบ่งเป็นส่วน ๆ เรียกว่า sector
- ตั้ง (กลุ่ม) ของ track ซึ่งตรงกับตำแหน่ง arm เดียวกัน เรียกว่า cylinder
- หน่วยความจุของ disk drive คือ gigabyte
- 1 kilobyte = 1024 bytes
- 1 megabyte = 1024^2 bytes
- 1 gigabyte = 1024^3 bytesแต่โรงงานมักปัดเศษทิ้งดังนั้น 1 megabyte = 1 ล้านไบต์
  1 gigabyte = 1 พันล้านไบต์
- ความเร็วในการหมุนประมาณ 60 – 150 รอบ/วินาที
- ความเร็วของ disk มี 2 ส่วน
- transfer rate คือ อัตราการไหลของข้อมูลระหว่าง drive และ คอมพิวเตอร์
- positioning time (เวลาในการระบุตำแหน่ง) บางครั้งเรียกว่า random access time (เวลาในการเข้าถึงข้อมูลอย่างสุ่ม) ประกอบไปด้วยเวลาในการเคลื่อน disk arm ไปสู่ cylinder ที่ต้องการ ซึ่งเรียกว่า seek time และเวลาที่ใช้ในการหมุน sector ที่ต้องการไปสู่หัวอ่าน-เขียน เรียกว่า rotational latency
- head crash คือ อุบัติเหตุที่เกิดจากหัวอ่าน-เขียนไปถูกพื้นผิวของ disk
- disk controller เป็นตัวประสานระหว่าง device กับคอมพิวเตอร์

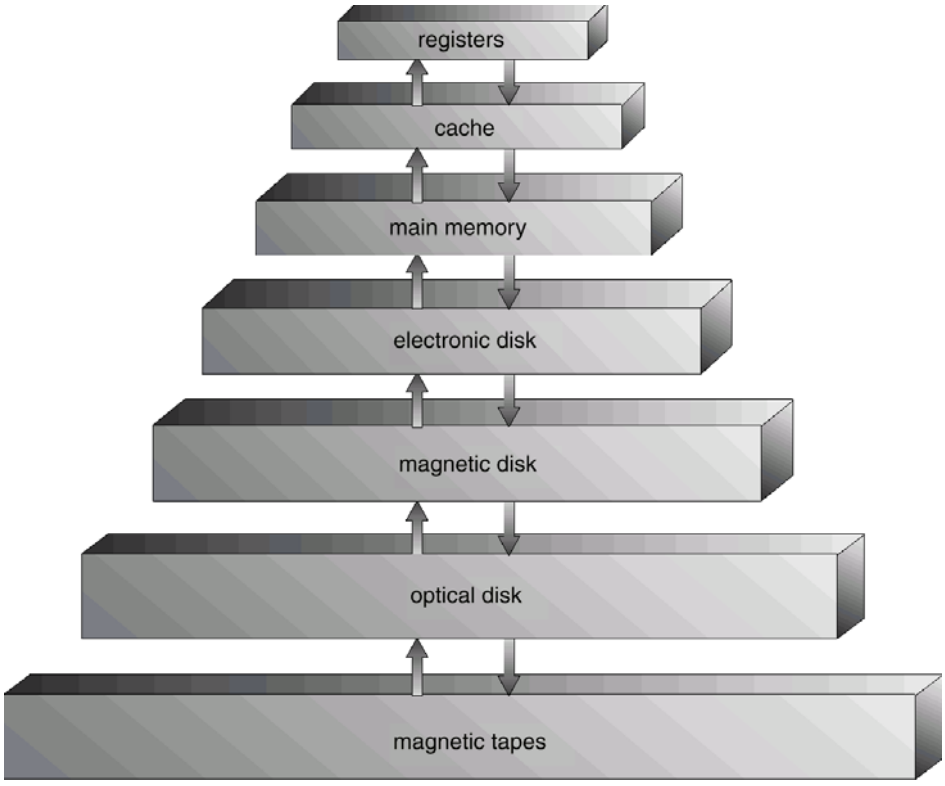
Magnetic Tapes

- เก็บข้อมูลได้มาก ถาวร
- เวลาในการเข้าถึงข้อมูลช้าเมื่อเทียบกับเวลาที่ main memory ใช้
- random access time ช้ากว่า magnetic disk
- มักใช้ back up information ที่ใช้ไม่ค่อยบ่อย
- ความจุของเทปมากกว่า disk 20 เท่า
- ความกว้างของเทปมีหลายขนาด คือ 4 , 8 , 9 มิลลิเมตร และ ¼ , ½ นิ้ว

1.2.4 ระบบจัดลำดับการทำงานของอุปกรณ์

แบ่งตาม speed ราคา และ Volatility (ความเปลี่ยนแปลงได้ง่าย) ระดับบนสุดก็แพงแต่ทำงานได้รวดเร็ว ข้อมูลในหน่วยเก็บข้อมูลแบบชั่วขณะ (volatile storage) จะหายไปเมื่อปิดเครื่อง main memory , cache และ registers คือ volatile storage แต่ magnetic disk , optical disk ,magnetic tapes เป็น nonvolatile storage (หน่วยเก็บข้อมูลแบบถาวร)ส่วน electronic disk เป็นแบบชั่วขณะและถาวร

ในการปฏิบัติการปกติ electronic disk จะเก็บข้อมูลใน array ของ DRAM ซึ่งเป็นแบบชั่วขณะ แต่ electronic disk device จะมี magnetic hard disk ซ่อนอยู่และมี battery สำหรับ backup power ถ้าไฟดับ disk controller จะคัดลอกข้อมูลจาก RAM ไปไว้ใน magnetic disk เมื่อไฟมา controller จะคัดลอกข้อมูลกลับไปยัง RAM



รูปที่ 1.9 ลำดับขั้นของหน่วยเก็บข้อมูล

Caching

ปกติ information จะถูกเก็บไว้ใน storage (เช่น main memory) เมื่อถูกเรียกใช้ มันจะถูกคัดลอกไปไว้ใน storage ที่เร็วกว่า นั่นคือ cache เมื่อเราต้องการ information ซักชิ้น เริ่มต้นเราควรหาใน cache ถ้าเจอเราก็จะใช้มันได้เลย จาก cache ถ้าไม่เจอ ค่อยไปหาจาก main storage แล้วเก็บสำเนาไว้ใน cache ภายใต้สมมติฐานที่ว่า มีความน่าจะเป็นสูงที่มันจำเป็นจะต้องนำมาใช้อีก Main memory ก็

เหมือนเป็น cache อย่างเร็วสำหรับหน่วยความจำสำรอง เพราะข้อมูลต้องถูกคัดลอกจากหน่วยเก็บข้อมูลสำรองไปยัง main memory เพื่อใช้ และข้อมูลต้องอยู่ใน main memory ก่อนจะถูกย้ายไปยังหน่วยเก็บข้อมูลสำรองเพื่อเก็บ

Coherency and Consistency

ถ้าต้องการเพิ่มค่าของ A ซึ่งอยู่ในไฟล์ B อีก 1 ซึ่งไฟล์ B อยู่ในเทปแม่เหล็ก การเพิ่มค่านี้ทำได้โดยเริ่มจากการปฏิบัติการของ I/O จะคัดลอก disk block ที่ A อยู่ไปไว้ใน main memory และทำสำเนาของ A ไปยัง cache และยังไว้ใน internal register อีก ดังนั้น สำเนาของ A ก็มีอยู่หลายที่ เมื่อเพิ่มค่าของ A ใน internal register ค่าของ A ก็จะแตกต่างกันในอีกหลาย ๆ ที่ ค่าของ A จะมาเหมือนกันหลังจากค่าใหม่ของ A ถูกเขียนกลับไปยัง magnetic disk ในสิ่งแวดล้อมแบบมี 1 งาน ในเวลาหนึ่งการทำงานก็ไม่ยาก เพราะการเข้าถึง A ก็เพียงทำสำเนาไปยังระดับที่สูงที่สุดของ hierarchy แต่ในสิ่งแวดล้อมแบบ multitasking ซึ่ง CPU ต้อง switch ไป switch มา 4 งาน ต้องระวัง ถ้าหลาย ๆ กระบวนการต่างต้องการเข้าถึง A ซึ่งแต่ละ process จะได้รับค่าของ A ที่ update ที่สุด

ในสิ่งแวดล้อมของ multiprocessor สำเนาของ A อาจมีในหลาย ๆ cache ซึ่ง CPU หลายตัวสามารถทำงานได้พร้อมกัน เราต้องทำให้มั่นใจว่า ค่าของ A ที่ update ใน 1 cache จะสะท้อนไปทุก ๆ caches ที่ A อยู่อย่างทันที ปัญหานี้เรียกว่า **cache coherency**

ในสิ่งแวดล้อมของ distributed สำเนาของไฟล์เดียวกันสามารถเก็บในคอมพิวเตอร์ต่าง ๆ สำเนาเหล่านั้นอาจถูก access และ update ได้ในเวลาเดียวกัน เราต้องทำให้แน่ใจว่าเมื่อสำเนาถูก update ในที่หนึ่ง ในที่อื่น ๆ ทั้งหมดก็ต้อง update ด้วย ในทันที

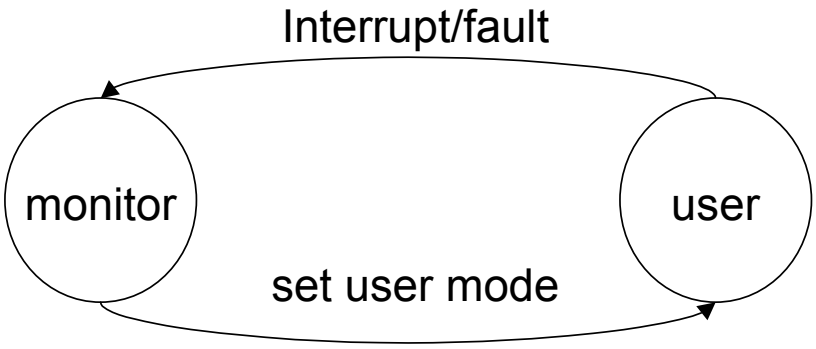
การป้องกันอุปกรณ์(Hardware Protection)

ข้อผิดพลาดหลายอย่างมักจะตรวจสอบได้โดยฮาร์ดแวร์ ซึ่งสามารถควบคุมได้โดยระบบปฏิบัติการ เช่น ถ้างานหนึ่งอ่านหรือเขียนข้อมูลออกนอกขอบเขตของตน หรือร้องขอคำสั่งที่ไม่ถูกต้อง ฮาร์ดแวร์ก็จะส่งสัญญาณไปขัดจังหวะ แล้วย้ายการควบคุมไปที่ระบบปฏิบัติการ ซึ่งจะทำการจัดการข้อผิดพลาดเหล่านั้น ระบบอาจหยุดการทำงานของโปรแกรมที่มีข้อผิดพลาดนั้นไปเลย และแสดงข้อความบอกความผิดพลาด พร้อมกับพิมพ์ค่าต่าง ๆ ในหน่วยความจำให้ดูข้อผิดพลาดที่อยู่ในหน่วยความจำมักเก็บไว้เป็นไฟล์ เพื่อให้ผู้ใช้ตรวจสอบหรือแก้ไขเพื่อเริ่มโปรแกรมใหม่

Dual-Mode Operation (การทำงานแบบ 2 ช่วง)

ระบบที่ดีต้องสามารถป้องกันตัวระบบเอง โปรแกรมอื่นและข้อมูลจากโปรแกรมอื่น ๆ ไม่ให้เกิดข้อผิดพลาด การป้องกันนี้ต้องอาศัยฮาร์ดแวร์ให้สามารถแยกความแตกต่างของ ช่วงการ

ทำงาน (mode of operation) ออกเป็น 2 ช่วง คือ user mode และ monitor mode (บางทีเรียกว่า supervisor mode, system mode หรือ privileged mode (ช่วงสงวน)) อาจกำหนดบิตขึ้นมา 1 บิต เรียกว่า บิตแสดงช่วง (mode bit) ในฮาร์ดแวร์ เพื่อให้ทราบว่าขณะนี้เป็นการทำงานใน mode ใด monitor(bit=0) หรือ user (bit=1)



รูปที่ 1.10 การทำงานแบบ 2 ช่วง

เมื่อเริ่ม boot เครื่อง ฮาร์ดแวร์จะเริ่มด้วย monitor mode ระบบปฏิบัติการจะถูก load มา และเริ่มการทำงานของผู้ใช้ใน user mode เมื่อเกิดสัญญาณ interrupt ฮาร์ดแวร์จะ switch จาก user mode มาเป็น monitor mode (bit จาก 1 เป็น 0) ดังนั้นเมื่อระบบปฏิบัติการได้การควบคุมมาจากคอมพิวเตอร์ มันก็อยู่ใน monitor mode และก่อนที่จะย้ายการควบคุมไปให้ผู้ใช้หรือโปรแกรมระบบก็จะสั่งให้ฮาร์ดแวร์เปลี่ยนเป็น user mode (bit เป็น 1)

การทำงานแบบ Dual-Mode นี้ สามารถป้องกันระบบหรืองานอื่น ๆ จากงานบางงานซึ่งมีข้อผิดพลาดได้ โดยการกำหนดให้คำสั่งฮาร์ดแวร์ (machine instruction) บางคำสั่งเป็นคำสั่งสงวน (privileged instruction) ซึ่งฮาร์ดแวร์จะยอมให้ทำงานตามคำสั่งสงวนนี้ได้ เมื่ออยู่ใน monitor mode เท่านั้น ถ้ามีการใช้คำสั่งสงวนนี้ใน user mode ฮาร์ดแวร์ก็จะไม่ปฏิบัติตามและถือว่าเป็นข้อผิดพลาด ซึ่งจะทำให้เกิดการส่งสัญญาณขัดจังหวะไปยังระบบปฏิบัติการ

I/O Protection

โปรแกรมของผู้ใช้บางคนอาจก่อกวนระบบโดยใช้คำสั่งร้องขออุปกรณ์อย่างผิดๆ หรืออ้างอิงตำแหน่งในหน่วยความจำที่อยู่ในส่วนของระบบปฏิบัติการ หรือ ไม่ยอมคืนการควบคุม CPU มาสู่ระบบ เพื่อป้องกันอุปกรณ์รับส่งข้อมูล (I/O) ต่างๆ เรากำหนดให้คำสั่งสำหรับร้องขออุปกรณ์ทุกชนิดเป็นคำสั่งสงวน (Privileged instruction) ดังนั้นผู้ใช้จะไม่สามารถสั่งไปยังอุปกรณ์ได้โดยตรง แต่ต้องร้องขอผ่านระบบปฏิบัติการเสมอต้องทำให้แน่ใจว่า โปรแกรมของผู้ใช้จะไม่มีทาง

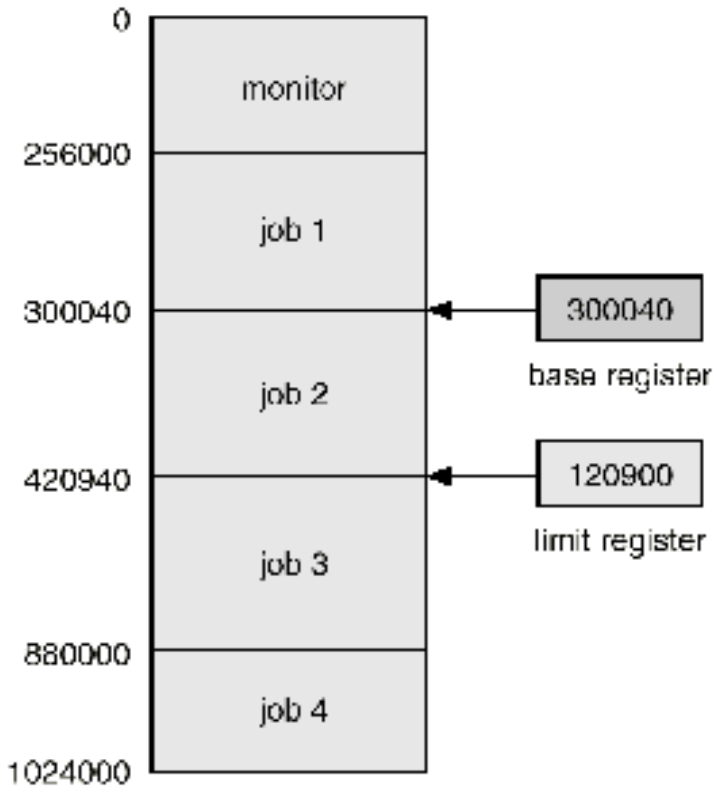
ทำงานใน monitor mode ได้ ขณะที่โปรแกรมของผู้ใช้ทำงานอยู่ใน user mode เมื่อเกิดข้อผิดพลาด ระบบจะเปลี่ยนบิต เป็น monitor mode แล้วย้ายการทำงานไป ณ ตำแหน่งที่กำหนดจากตาราง สัญญาณ (interrupt vector)

สมมติว่าผู้ใช้แก้ไขค่าในตารางสัญญาณเพื่อให้ชี้ไปที่ตำแหน่งใหม่ ซึ่งเป็นโปรแกรมของผู้ใช้เอง แล้วปล่อยให้เกิดข้อผิดพลาด เพื่อให้ระบบเปลี่ยนเป็น monitor mode แล้วย้ายการทำงานไป ณ ตำแหน่งในตาราง (ซึ่งขณะนี้ชี้ไปที่โปรแกรมของผู้ใช้) ดังนั้นโปรแกรมของผู้ใช้จะสามารถทำงานใน monitor mode ได้

Memory Protection

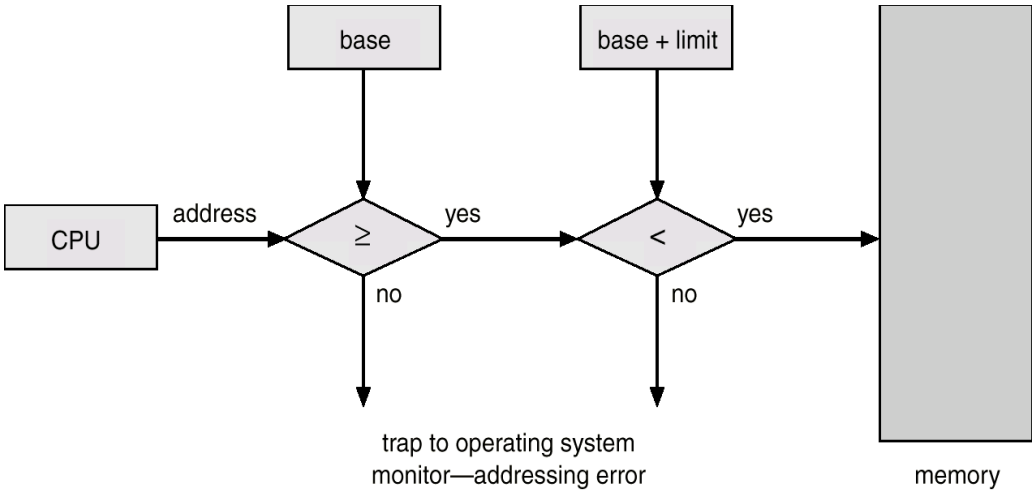
เราต้องป้องกันตารางสัญญาณ (interrupt vector) ไม่ให้ผู้ใช้เข้ามาแก้ไขได้ รวมทั้งโปรแกรมของระบบปฏิบัติการ (interrupt service routine) ก็ต้องมีการป้องกันด้วย

วิธีการป้องกันหน่วยความจำ คือ ต้องแบ่งแยกหน่วยความจำสำหรับงานแต่ละงาน และป้องกันไม่ให้งานหนึ่งเข้าไปอ่านหรือแก้ไขข้อมูลในส่วนของงานอื่นได้ การป้องกันนี้ทำได้โดย register 2 ตัว เรียกว่า ฐาน (base) และ ขอบเขต (limit) ดังรูปประกอบ



รูปที่ 1.11 การแบ่งแยกหน่วยความจำสำหรับงานแต่ละงาน

- Base register จะเก็บค่าตำแหน่งสุดท้ายที่ยอมให้ใช้ได้ใน memory
- Limit register จะเก็บขนาดของเนื้อที่ทั้งหมดของงาน ตัวอย่างเช่น ถ้า base register เก็บค่า 300040 และ limit register เก็บค่า 120900 โปรแกรมสามารถใช้หน่วยความจำในตำแหน่งตั้งแต่ 300040 ถึง 420940 วิธีการป้องกันแบบนี้ ฮาร์ดแวร์ของ CPU จะต้องตรวจสอบค่าตำแหน่งทุกครั้งที่มีการอ้างอิงหน่วยความจำ เมื่ออยู่ใน user mode ถ้ามีการอ้างอิงนอกขอบเขตที่กำหนด ก็จะเกิดการส่งสัญญาณขัดจังหวะไปสู่ระบบปฏิบัติการ ซึ่งถือว่าเป็นข้อผิดพลาดอย่างรุนแรง (fatal error) ดังรูป

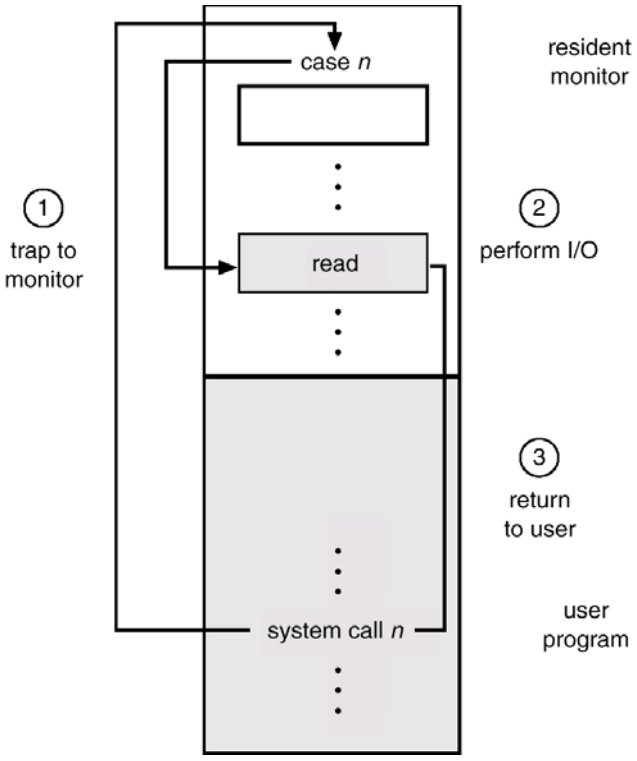


รูปที่ 1.12 ขอบเขตและการป้องกันอุปกรณ์ขั้นพื้นฐาน

CPU Protection

ในการประกันว่า การควบคุมจะย้ายกลับมายังระบบเสมอ แม้ว่าจะมีงานบางงานทำงานวนรอบอย่างไม่มีที่สิ้นสุด (infinite loop) เราสามารถทำได้โดยการใช้นาฬิกาจับเวลา (timer) นาฬิกานี้จะส่งสัญญาณไปขัดจังหวะฮาร์ดแวร์ เมื่อถึงช่วงเวลาที่กำหนด ซึ่งอาจมีค่าคงที่ (เช่น 1/60 วินาที) หรือมีค่าเปลี่ยนแปลงได้ (เช่น จาก 1 มิลลิวินาที ถึง 1 วินาที โดยเพิ่มทีละ 1 มิลลิวินาที) โดยปกติแล้วเราใช้นาฬิกาจริง (ในฮาร์ดแวร์) ร่วมกับตัวนับ (counter) เพื่อสร้างนาฬิกาจับเวลา ระบบปฏิบัติการจะเป็นผู้กำหนดค่าให้กับตัวนับ เมื่อนาฬิกาจริงเดินไป 1 ติ๊ก ตัวนับก็จะนับถอยหลังลง 1 จนกระทั่งตัวนับมีค่าเป็น 0 ก็จะส่งสัญญาณไปขัดจังหวะฮาร์ดแวร์ ก่อนที่ระบบจะส่งการควบคุมไปยังโปรแกรมของผู้ใช้ ระบบก็จะตั้งนาฬิกาจับเวลาไว้ เมื่อถึงช่วงเวลาที่กำหนด นาฬิกาจับเวลาจะส่งสัญญาณมาขัดจังหวะฮาร์ดแวร์ ทำให้การควบคุมย้ายกลับมาที่ระบบปฏิบัติการ ซึ่งระบบอาจจะ

แก้ไขข้อผิดพลาด (fatal error) หรือต่อเวลาให้กับโปรแกรมผู้ใช้อีกก็ได้ คำสั่งในการกำหนดค่านาฬิกาจับเวลานี้ต้องเป็นคำสั่งสงวนด้วย จะเห็นว่า นาฬิกาจับเวลาสามารถป้องกันเวลาสามารถป้องกันไม่ให้โปรแกรมของผู้ใช้ทำงานหรือใช้ CPU นานเกินไปได้ วิธีง่าย ๆ ก็คือ กำหนดตัวนับเป็นค่าเวลาสูงสุดที่งานนั้น ๆ ต้องการใช้ เช่น โปรแกรมงานซึ่งมีกำหนดเวลาไม่เกิน 7 นาที เราอาจตั้งตัวนับให้เท่ากับ 420 และนาฬิกาจับเวลาก็ตั้งไว้ที่ 1 วินาที ดังนั้นทุก ๆ 1 วินาที นาฬิกาจับเวลาจะส่งสัญญาณไปขัดจังหวะการทำงานของโปรแกรม เราก็เพียงแต่นับถอยหลังไป 1 ตราบใดที่ค่าตัวนับยังเป็นบวก การควบคุมก็จะถูกส่งกลับให้โปรแกรมผู้ใช้ เมื่อตัวนับมีค่าติดลบ การควบคุมจะไม่ถูกส่งไป แต่ระบบจะหยุดการทำงานของโปรแกรมผู้ใช้ เนื่องจากทำงานเกินกว่าเวลาที่ได้รับอนุญาต ปกติแล้วนาฬิกาจับเวลา จะใช้ระบบปันส่วน (time sharing) โดยอาจกำหนดให้ส่งสัญญาณทุก ๆ N มิลลิวินาที N คือ ค่าส่วนแบ่งเวลา (time slice) ที่ให้ผู้ใช้แต่ละคนทำงานก่อนคนอื่น ระบบปฏิบัติการจะได้การควบคุมทุกครั้งที่นาฬิกาจับเวลานี้ส่งสัญญาณไปขัดจังหวะการทำงาน ระบบก็จะจัด , เก็บ กำหนดค่าใน register , buffer , ตัวแปรภายใน , ปรับค่าอื่น ๆ ที่จำเป็นสำหรับการย้ายให้งานอื่นได้มาใช้ CPU บ้าง โดยทำงานต่อจากจุดที่หยุด (ถูก interrupt เพราะหมดเวลาตามส่วนแบ่ง (time slice)) ครั้งก่อน



รูปที่ 1.13 การใช้ของการเรียกระบบเพื่อกระทำ I/O

1.3 โครงสร้างของระบบปฏิบัติการ

1.3.1 องค์ประกอบของระบบปฏิบัติการ

1. <u>การจัดการโปรเซส (Process Management)</u>

โปรเซส คือ โปรแกรมที่กำลังทำงานอยู่ ได้แก่ งานแบบกลุ่ม (batch job) โปรแกรมของผู้ใช้ในระบบปันส่วน (time-shared user program) งาน spooling เป็นต้น

โปรเซส ต้องใช้ทรัพยากรต่าง ๆ ในการทำงานหนึ่ง ๆ เช่น เวลาประมวลผล , หน่วยความจำ , แฟ้มข้อมูล และอุปกรณ์รับส่งข้อมูล ซึ่งการโปรเซส อาจได้รับทรัพยากรเหล่านี้ตั้งแต่ตอนที่ถูกสร้างขึ้น หรือได้มาระหว่างทำงาน ทั้งยังสามารถส่งผ่านทรัพยากรไปสู่โปรเซส อื่นได้อีกด้วย เช่น โปรเซส หนึ่งมีหน้าที่แสดงสถานะของแฟ้มข้อมูลหนึ่ง บนจอภาพ ก็จะได้รับข้อมูลเข้าเป็นชื่อแฟ้มข้อมูล และก็จะทำคำสั่งบางอย่างเพื่อให้ได้ข้อมูลสถานะของแฟ้มนั้นมาเพื่อแสดงต่อไป

โปรเซสเป็นหน่วยย่อยของงานในระบบ ระบบประกอบไปด้วยโปรเซส หลาย ๆ โปรเซส โปรเซส บางอันเป็นของระบบปฏิบัติการเอง บางอันเป็นของผู้ใช้ระบบ โปรเซส เหล่านี้สามารถทำงานไปเสมือนพร้อมกัน (concurrent) ได้ โดยการสลับกันใช้หน่วยประมวลผลกลาง

ระบบปฏิบัติการมีหน้าที่ในการจัดการโปรเซส ดังต่อไปนี้

1. สร้าง และ ทำลายโปรเซสทั้งของระบบและผู้ใช้
2. ให้โปรเซสหยุดชั่วคราว (suspension) หรือ กลับคืนสภาพ (resumption)
3. จัดกลไกในการประสานงานระหว่างโปรเซส (process synchronization)
4. จัดกลไกในการจัดการปัญหาวงจรอับ (deadlock handling)

2. <u>การจัดการหน่วยความจำหลัก</u> (Main-Memory Management)

หน่วยความจำ คือ array of words or bytes ซึ่งมีตำแหน่ง (address) เฉพาะ หน่วยความจำเป็นที่พักข้อมูล (ซึ่งสามารถเรียกใช้ได้อย่างรวดเร็ว) ระหว่างหน่วยประมวลผลกลาง และ อุปกรณ์รับส่งข้อมูล หน่วยประมวลผลกลางอ่านคำสั่งจากหน่วยความจำหลักใน “ช่วงเวลาอ่านคำสั่ง” (the instruction – fetch cycle) การรับส่งข้อมูลโดยใช้ตัวควบคุมหน่วยความจำ (DMA) จะอ่านหรือเขียนข้อมูลจากหน่วยคามจำหลักเช่นกัน

การที่จะทำให้โปรแกรมทำงาน เริ่มจากการนำโปรแกรมลงสู่ตำแหน่งสัมบูรณ์ (Absolute address) ในหน่วยความจำหลัก แล้วให้หน่วยประมวลผลกลางอ่านคำสั่ง และดำเนินการตามคำสั่ง

ของโปรแกรมในหน่วยความจำหลัก เมื่อโปรแกรมสิ้นสุด หน่วยความจำหลักจะถูกคืนสู่ระบบ ระบบก็จะนำโปรแกรมต่อไปมาทำงานได้อีก

ระบบปฏิบัติการมีหน้าที่ในการจัดการหน่วยความจำ ดังนี้

1. บันทึกข้อมูล ว่าโปรแกรมใดอยู่ในหน่วยความจำตำแหน่งใด
2. เลือกโปรเซสที่จะนำลงหน่วยความจำต่อไป เมื่อมีหน่วยความจำว่าง
3. จัดสรร และ คืน หน่วยความจำตามที่ระบบขอร้อง

3. <u>การบริหารแฟ้มข้อมูล</u> (File Management)

แฟ้มข้อมูล คือ ที่เก็บรวบรวมข้อมูลที่เกี่ยวข้องกัน โดยผู้สร้างเป็นผู้กำหนดขึ้น ตัวอย่างเช่น แฟ้มข้อมูลสำหรับโปรแกรม ข้อมูลดิบ ข้อมูลดิบอาจเป็นตัวเลข , อักษร หรือ อักขระ แฟ้มข้อมูลอาจจะไม่มีโครงสร้าง (free – form) เช่น แฟ้มข้อความ (text files) หรือมีโครงสร้างอย่างซับซ้อน แฟ้มข้อมูลประกอบด้วย บิท , ไบท์ , เรคคอร์ด ซึ่งผู้สร้างเป็นผู้กำหนดความหมาย
ระบบปฏิบัติการมีหน้าที่บริหารแฟ้มข้อมูล ดังนี้

1. สร้าง และ ทำลายแฟ้มข้อมูล
2. สร้าง และ ทำลายไดเรคทอรี่
3. ให้บริการการใช้งานแฟ้มข้อมูลและไดเรคทอรี่ ขั้นพื้นฐาน
4. อ้างอิงข้อมูลจากแฟ้มกับข้อมูลจริงในหน่วยความจำสำรอง
5. ทำสำเนาแฟ้มข้อมูลลงในหน่วยความจำถาวร (ไม่ลบเมื่อไฟดับ)

4. <u>การบริหารระบบรับส่งข้อมูล</u> (I/O System Management)

หน้าที่สำคัญอันหนึ่งของระบบปฏิบัติการคือ การซ่อนรายละเอียดของอุปกรณ์เฉพาะอย่างจากผู้ใช้ เช่น ในระบบปฏิบัติการ UNIX รายละเอียดต่าง ๆ ของอุปกรณ์รับส่งข้อมูลจะถูกเก็บไว้ในระบบรับส่งข้อมูลเอง แยกต่างหากจากระบบปฏิบัติการหลัก ระบบรับส่งข้อมูลประกอบไปด้วย

1. องค์ประกอบการจัดการหน่วยความจำประกอบด้วย buffering , caching และ spooling
2. วิธีการเรียกใช้ตัวควบคุมอุปกรณ์ (A general device – driver interface)
3. ตัวควบคุมอุปกรณ์พิเศษ (drivers for specific hardware devices)

ตัวควบคุมอุปกรณ์ (device drivers) เท่านั้นที่ทราบรายละเอียดของอุปกรณ์ที่มันควบคุมอยู่

5. <u>การจัดการหน่วยเก็บข้อมูลสำรอง</u> (Secondary – Storage Management)

จุดประสงค์หลักของระบบคอมพิวเตอร์ คือ การทำงานตามโปรแกรมผู้ใช้ ซึ่งโปรแกรมเหล่านี้ รวมทั้งข้อมูลที่ใช้จะต้องอยู่ในหน่วยความจำหลักในขณะทำงาน แต่เนื่องจากหน่วยความจำหลักมีขนาดจำกัด จึงจำเป็นต้องมีที่เก็บข้อมูลสำรอง คอมพิวเตอร์สมัยใหม่นิยมใช้จานบันทึก

(disk) เป็นหน่วยเก็บข้อมูลสำรองด่านแรก สำหรับเก็บทั้งโปรแกรมและข้อมูล โปรแกรมส่วนมาก เช่น ตัวแปลภาษา (compilers) ตัวแปรภาษาเอสแซมบลี้ (assemblers) โปรแกรมจัดเรียง (sort routines) ตัวแก้ไขข้อมูล (editors) และ ตัวจัดรูปแบบข้อมูล (formatters) ถูกเก็บไว้ในจานบันทึก จนกระทั่งมีการนำลงสู่หน่วยความจำหลัก ทั้งยังใช้จานบันทึกเป็นแหล่งข้อมูล และ ที่เก็บผลลัพธ์ในการทำงานอีกด้วย ดังนั้น การจัดการหน่วยเก็บข้อมูลสำรองจึงเป็นสิ่งสำคัญยิ่งสำหรับระบบคอมพิวเตอร์เช่นกัน

ระบบปฏิบัติการมีหน้าที่บริหารจานบันทึก ดังนี้

1. การบริหารพื้นที่ว่าง
2. การจัดหาเนื้อที่เก็บข้อมูล
3. การจัดตารางการใช้จานบันทึก

6. <u>เครือข่าย</u> (Networking)

ในระบบกระจายอำนาจ (distributed system) หน่วยประมวลผลแต่ละตัวจะมีหน่วยความจำและนาฬิกาของตนเอง การติดต่อระหว่างหน่วยประมวลผลจะทำได้โดยผ่านทางเครือข่ายการสื่อสาร เช่น บัสความเร็วสูง , สายโทรศัพท์ เป็นต้น หน่วยประมวลผลแต่ละแห่งอาจมีขนาดและหน้าที่แตกต่างกันมาก เช่น คอมพิวเตอร์ส่วนบุคคล , สถานีงาน (work station) คอมพิวเตอร์ขนาดกลาง และ คอมพิวเตอร์ขนาดใหญ่ เป็นต้น

เครือข่ายการสื่อสารระหว่างหน่วยประมวลผลเหล่านี้ อาจปรับเปลี่ยนวิธีการต่อเชื่อมได้หลายแบบ เช่น แบบต่อเชื่อมสมบูรณ์แบบ หรือ บางส่วน การออกแบบต้องคำนึงถึง การหาเส้นทาง และ วิธีการติดต่อ ตลอดจนปัญหาคอขวด และปัญหาด้านความปลอดภัย

ระบบกระจายอำนาจ ทำให้ผู้ใช้สามารถใช้ทรัพยากรต่าง ๆ ได้มากมาย ซึ่งเป็นผลให้สามารถทำงานได้เร็วขึ้น , เข้าถึงข้อมูลได้มากขึ้น , และระบบมีความน่าเชื่อถือมากขึ้น ระบบปฏิบัติการแบบนี้ มักแปลงการติดต่อผ่านเครือข่ายให้อยู่ในรูปการใช้แฟ้มข้อมูล ซึ่งจะช่วยซ่อนรายละเอียดในการติดต่อไว้ในตัวควบคุมอุปกรณ์เครือข่าย

7. <u>ระบบป้องกัน หรือ ระบบความปลอดภัย</u> (Protection System)

ระบบต้องมีการป้องกัน ความผิดพลาดที่เกิดจากโปรเซสหนึ่งไปกระทบอีกโปรเซสหนึ่ง โดยสร้างกลไกบางอย่างเพื่อป้องกันแฟ้มข้อมูล , หน่วยความจำส่วนหนึ่ง, หน่วยประมวลผลกลาง และทรัพยากรต่าง ๆ ในระบบให้สามารถใช้ได้โดยโปรเซสที่มีสิทธิจะใช้ได้เท่านั้น

ตัวอย่างเช่น กลไกในการอ้างอิงหน่วยความจำหลัก ป้องกันโปรเซสให้ใช้หน่วยความจำหลักได้แต่ในส่วนของโปรเซสนั้นเท่านั้น การใช้นาฬิกาจับเวลาเพื่อควบคุมไม่ให้โปรเซสใดทำงานใน

หน่วยประมวลผลตลอดไป (ไม่ยอมคืนการควบคุมกลับมาที่ระบบ) การไม่อนุญาตให้ผู้ใช้ระบบทำการรับส่งข้อมูลเองโดยตรง เพื่อป้องกันความผิดพลาดในการใช้งานของอุปกรณ์รับส่งข้อมูล
การป้องกัน คือ กลไกการควบคุมโปรแกรม , โปรเซส หรือ ผู้ใช้ระบบ ในการใช้ทรัพยากรของระบบคอมพิวเตอร์ โดยที่กลไกเหล่านี้ต้องกำหนดวิธีการใช้งานทรัพยากรต่าง ๆ และวิธีการบังคับให้ผู้ใช้ปฏิบัติตาม

การป้องกันช่วยให้ระบบมีความเชื่อมั่นสูงขึ้น โดยการตรวจจับข้อผิดพลาดที่อาจเกิดขึ้นในการติดต่อระหว่างระบบย่อย การตรวจจับแต่เนิ่นๆ สามารถช่วยป้องกันระบบย่อยที่ดีจากระบบย่อยที่เสียได้ ทรัพยากรที่ไม่มีการป้องกัน อาจเสียหายจากการใช้งานที่ผิดพลาด หรือ จงใจจากผู้ใช้ที่ไม่มีสิทธิ หรือ ไม่รู้เรื่องได้ ระบบที่มีการป้องกันดีจะสามารถแบ่งแยกได้ว่า การใช้งานเป็นไปตามสิทธิ (authorized) หรือไม่มีสิทธิ (unauthorized)

8. <u>ระบบแปลคำสั่ง</u> (Command – Interpreter System)

โปรแกรมระบบที่สำคัญที่สุดอันหนึ่ง คือ ตัวแปลคำสั่ง (Command Interpreter) ซึ่งเป็น interface ระหว่างผู้ใช้กับระบบปฏิบัติการ ระบบปฏิบัติการบางระบบ ได้รวมเอาตัวแปลคำสั่งไว้ในแกนกลางของระบบ (kernel) ระบบปฏิบัติการอื่น เช่น MS-DOS และ UNIX ถือว่า ตัวแปลคำสั่งเป็นเพียงโปรแกรมพิเศษอันหนึ่งซึ่งจะทำงานเมื่อมีงานเข้าในระบบ หรือ เมื่อผู้ใช้เริ่มเข้าสู่ระบบ (log on) (บนระบบ time-sharing)

คำสั่งส่วนใหญ่เป็นประโยคควบคุม (Control statements) เมื่อมีงานใหม่เริ่มทำงานในระบบการทำงานแบบกลุ่ม (batch) หรือ เมื่อผู้ใช้เข้าสู่ระบบในระบบการทำงานแบบปันส่วน (time-sharing) จะมีโปรแกรมหนึ่งคอยอ่านคำสั่ง และ ทำงานตามคำสั่งเหล่านั้นโดยอัตโนมัติ โปรแกรมนี้มีชื่อเรียกได้หลายอย่าง เช่น ตัวแปลบัตรควบคุม ตัวแปรบรรทัดคำสั่ง เชลล์ (shell in UNIX) เป็นต้น มันมีหน้าที่ง่าย ๆ คือ อ่านคำสั่งและทำงานตามคำสั่ง

ระบบปฏิบัติการอาจถูกแบ่งตามลักษณะของตัวแปลคำสั่งได้ เช่น มีตัวแปรคำสั่งที่ใช้ง่าย (user – friendly) พร้อมคำแนะนำในตัว อย่างในระบบ Macintosh ตัวแปรคำสั่งที่มีระบบหน้าต่าง (Windows) และรายการให้เลือก (Menu system) ตลอดจนใช้ตัวชี้ หรือ เม้าส์ ได้ด้วย ผู้ใช้เพียงแต่ใช้ตัวชี้ตำแหน่งไปที่รูปภาพบนจอ ซึ่งมีทั้งรูปของแฟ้มข้อมูล , โปรแกรมต่าง ๆ และกดปุ่มบนตัวชี้ โปรแกรมที่ถูกชี้ก็จะทำงานทันที หรือให้ตารางรายการเลือกคำสั่งด้วยตัวชี้ ส่วนอีกระบบหนึ่ง อาจมีตัวแปลคำสั่งที่ซับซ้อน และใช้งานได้มากมาย แต่เรียนรู้ได้ยาก ซึ่งเหมาะสำหรับผู้ใช้ที่มีความเชี่ยวชาญเท่านั้น ผู้ใช้ต้องพิมพ์คำสั่งทางแป้นพิมพ์ และกดแป้นส่งคำสั่ง (Enter key) เพื่อที่จะส่งคำสั่งเข้าระบบ ระบบนี้ได้แก่ MS-DOS , Shell in UNIX

1.3.2 การให้บริการของระบบปฏิบัติการ (Operating – System Service)

ระบบปฏิบัติการ เป็นผู้จัดสภาพแวดล้อมให้โปรแกรมทำงาน โดยให้บริการต่าง ๆ แก่โปรแกรม และผู้ใช้ระบบ ระบบปฏิบัติการต่าง ๆ มักมีการให้บริการที่แตกต่างกัน แต่จะมีส่วนหนึ่งที่เหมือนกัน เพื่อให้ความสะดวกต่อผู้ใช้ หรือ ผู้เขียนโปรแกรม ในการทำงานต่าง ๆ ให้ง่ายและรวดเร็ว บริการเหล่านี้ ได้แก่

- การให้โปรแกรมทำงาน (Program Execution) ระบบต้องสามารถนำโปรแกรมลงสู่หน่วยความจำหลัก และให้โปรแกรมทำงาน โดยที่การทำงานต้องมีวันสิ้นสุด ไม่ว่าจะเป็นปกติหรือไม่ปกติก็ตาม
- การรับส่งข้อมูล (I/O Operation) โปรแกรมของผู้ใช้อาจต้องการรับส่งข้อมูล โดยผ่านแฟ้มข้อมูล หรือ อุปกรณ์รับส่งข้อมูล อุปกรณ์รับส่งข้อมูลบางชนิดต้องการคำสั่งช่วยพิเศษ เช่น เครื่องขับเทป ต้องการการถอยหลังกลับเมื่อเต็ม หรือจอภาพต้องการคำสั่งล้างจอเมื่อเริ่มต้นทำงาน เนื่องจากผู้ใช้ไม่สามารถใช้อุปกรณ์รับส่งข้อมูลได้โดยตรง ดังนั้น ระบบจึงต้องจัดหาวิธีการเพื่อเป็นตัวกลางใช้แทน
- การใช้ระบบแฟ้มข้อมูล (File – system Manipulation) ระบบแฟ้มข้อมูลเป็นสิ่งจำเป็นอย่างยิ่ง โปรแกรมต้องการอ่าน หรือ เขียนข้อมูลลงในแฟ้มข้อมูล นอกจากนี้ ยังต้องการสร้าง หรือ ลบแฟ้มข้อมูลด้วยการใช้ชื่อแฟ้ม
- การติดต่อสื่อสาร (Communications) บางครั้งโปรเซสหนึ่งอาจต้องการส่งข้อมูลให้อีกโปรเซสหนึ่ง โดยที่โปรเซสทั้งสองนั้น อาจอยู่ในเครื่องคอมพิวเตอร์เดียวกันหรือคนละเครื่องกัน แต่ติดต่อผ่านเครือข่ายคอมพิวเตอร์ การติดต่อสื่อสารนี้อาจทำได้โดยใช้ หน่วยความจำร่วม (share memory) หรือ การส่งผ่านข้อความ (message passing) โดยมีระบบปฏิบัติการเป็นตัวกลาง
- การตรวจจับข้อผิดพลาด (Error detection) ระบบปฏิบัติการจำเป็นต้องมีกลไกในการตรวจจับข้อผิดพลาดที่อาจจะเกิดขึ้นได้ เช่น ในหน่วยประมวลผลกลาง (เครื่องเสีย , ไฟดับ) ในอุปกรณ์รับส่งข้อมูล (เทปเสีย , การติดต่อผ่านเครือข่ายล้มเหลว , หรือกระดาษพิมพ์หมด) หรือในโปรแกรมของผู้ใช้ (เช่น คำนวณผิด , ระบุตำแหน่งในหน่วยความจำผิด หรือ ใช้ CPU time มากไป) สำหรับข้อผิดพลาดแต่ละชนิด ระบบปฏิบัติการจะจัดการด้วยวิธีที่เหมาะสมเพื่อแก้ข้อผิดพลาดเหล่านั้น

นอกจากระบบปฏิบัติการจะมีหน้าที่อำนวยความสะดวกให้แก่ผู้ใช้ ยังต้องประกันประสิทธิภาพในการปฏิบัติการของระบบเองอีกด้วย ในระบบผู้ใช้หลายคน เราสามารถเพิ่มประสิทธิภาพได้โดยใช้ทรัพยากรร่วมกัน

การจัดสรรทรัพยากร (Resource allocation) เมื่อมีผู้ใช้หลายคนหรืองานหลายงานทำงานพร้อมกันในช่วงเวลาหนึ่ง ทรัพยากรต่าง ๆ ก็ต้องถูกจัดสรรให้กับคนหรืองานเหล่านั้น ชนิดของทรัพยากรต่าง ๆ จะถูกจัดการด้วยระบบปฏิบัติการ ทรัพยากรบางอย่าง (เช่น รอบการใช้ CPU , หน่วยความจำหลัก และ ที่เก็บแฟ้มข้อมูล) อาจจะมีรหัสในการจัดสรรพิเศษ โดยที่ทรัพยากรอย่างอื่น (เช่น อุปกรณ์รับส่งข้อมูล) อาจจะมีรหัสร้องขอ และปลดปล่อยพิเศษ

การทำบัญชี (Accounting) เราต้องเก็บรวบรวมการทำงานของผู้ใช้ โดยเก็บบันทึกไว้เป็นบัญชีหรือทำเป็นสถิติการใช้ทรัพยากรต่างๆ แบบสะสม สถิติการใช้เหล่านี้จะเป็นเครื่องมือที่มีค่าสำหรับนักวิจัยซึ่งหวังจะ reconfigure ระบบเพื่อปรับปรุงบริการในด้านการคำนวณ

การป้องกัน (Protection) information ที่เก็บในระบบคอมพิวเตอร์ที่มีผู้ใช้หลายคนอาจจะต้องการควบคุมการใช้งานด้วยตัวมันเอง เมื่อมีโปรเซสหลาย ๆ โปรเซสทำงานพร้อมกัน เราต้องไม่ให้โปรเซสหนึ่งไปแทรกแซงโปรเซสอื่น ๆ หรือ แม้แต่ตัวระบบปฏิบัติการเอง การป้องกันเป็นการประกันว่า การเข้าถึงทรัพยากรของระบบทั้งหมดต้องถูกควบคุม การรักษาความปลอดภัยของระบบจากภายนอกเป็นสิ่งสำคัญ การรักษาความปลอดภัยเริ่มด้วย ผู้ใช้แต่ละคนต้องได้รับการรับรองตัวเองต่อระบบ โดยทั่วไปก็จะหมายถึงรหัสผ่าน เพื่ออนุญาตให้ใช้ทรัพยากรต่าง ๆ ซึ่งรวมไปถึงการป้องกันอุปกรณ์รับส่งข้อมูล ซึ่งประกอบไปด้วย โมเด็มและการ์ดเครือข่าย (network adapters) ถ้าระบบถูกป้องกันและรักษาความปลอดภัยก็เท่ากับว่าเป็นการป้องกันไว้ก่อน

1.3.3 การติดต่อระหว่างโปรแกรมกับระบบปฏิบัติการ

1. System Calls (การเรียกระบบ)

การทำงานของคำสั่ง “เรียกระบบ”

System Call จัดเตรียมส่วนต่อประสาน (interface) ระหว่างโปรเซสหนึ่งกับระบบปฏิบัติการ การเรียกระบบมักเป็นคำสั่งภาษา assembly บางระบบอาจอนุญาตให้เรียกระบบได้โดยตรงจากโปรแกรมภาษาระดับสูง ซึ่งกรณีนี้การเรียกระบบจะถูกกำหนดเป็นหน้าที่ หรือ subroutine call (การเรียกระบบย่อย) ภาษาหลาย ๆ ภาษา เช่น C , Bliss , BCPL , PL/360 , และ PERL ถูกนำมาใช้แทน assembly สำหรับการเขียนโปรแกรมระบบ

ตัวอย่างการเรียกระบบ ลองดูการโปรแกรมซึ่งอ่านข้อมูลจากแฟ้มข้อมูลหนึ่งแล้วเขียนลอกลงในอีกแฟ้มหนึ่ง ขั้นแรกโปรแกรมต้องทราบชื่อของแฟ้มข้อมูลทั้งสองนั้นเสียก่อน (input file และ output file) ถ้าเป็นระบบที่ใช้สัญลักษณ์ภาพ และตัวชี้ (icon –based และ mouse – based) มักมีรายชื่อแฟ้มบนจอภาพให้ผู้ใช้เลือก โดยใช้ตัวชี้ และ กดปุ่มบนตัวชี้ เพื่อกำหนดแฟ้มรับ และ แฟ้มส่งข้อมูล

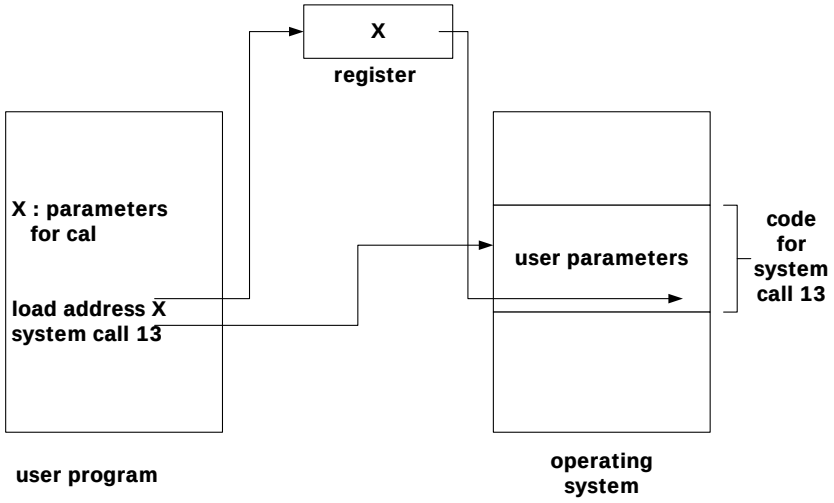
เมื่อได้ชื่อแฟ้มทั้งสองแล้ว โปรแกรมก็ต้องเรียกระบบ เพื่อให้เปิดแฟ้มส่งข้อมูล และ สร้างแฟ้มรับข้อมูล อาจมีข้อผิดพลาดเกิดขึ้นในการเรียกระบบนี้ เช่น ไม่มีแฟ้มข้อมูลที่ผู้ใช้ขอเปิดเป็นแฟ้มส่งข้อมูล (หาชื่อไม่พบ) หรือ ผู้ใช้ไม่มีสิทธิใช้แฟ้มข้อมูลดังกล่าว (protected) ซึ่งโปรแกรมต้องแสดงข้อความทางจอภาพ (โดยการเรียกระบบอีกคำสั่งหนึ่ง) หรือ สั่งหยุดการทำงาน (โดยเรียกระบบอีกคำสั่งหนึ่ง) โดยมีข้อผิดพลาด ถ้าพบแฟ้มส่งข้อมูล และ สามารถเปิดได้ ต่อไปก็เป็นการสร้างแฟ้มรับข้อมูลใหม่ (เรียกระบบเช่นกัน) อาจมีข้อผิดพลาดได้เหมือนกัน เช่น ชื่อแฟ้มซ้ำกับแฟ้มที่มีอยู่แล้ว ซึ่งอาจจะทำให้โปรแกรมต้องหยุดการทำงาน (โดยการเรียกระบบ) หรือ ลบแฟ้มเก่าทิ้งไป (โดยการเรียกระบบอีก) แล้วสร้างแฟ้มใหม่ขึ้น (เรียกระบบอีก) หรือ (ในระบบโต้ตอบ) แสดงข้อความถามผู้ใช้ (เรียกระบบเพื่อแสดงข้อความ เรียกระบบเพื่อรับคำสั่งจากแป้นพิมพ์) ว่าจะเขียนทับหรือยกเลิกการทำงาน

หลังจากที่จัดเตรียมแฟ้มทั้งสองแล้ว โปรแกรมก็ต้องวนเวียนอ่านจากแฟ้มส่งข้อมูล (เรียกระบบ) แล้วเขียนลงในแฟ้มรับข้อมูล (เรียกระบบอีก) ในการอ่าน และ เขียนนี้ ระบบต้องส่งสถานะไปให้โปรแกรมด้วยว่าทำงานเสร็จ หรือ มีข้อผิดพลาดเกิดขึ้น ในการอ่านอาจพบว่าอุปกรณ์ผิดพลาด หรือสิ้นสุดแฟ้มข้อมูล ในการเขียนอาจพบว่า เนื้อที่เต็มแล้ว หรือ กระดาษหมดแล้ว หรือ ถึงสุดหางเทปแล้ว เป็นต้น (ทั้งนี้ขึ้นอยู่กับอุปกรณ์รับข้อมูลนั้น)

เมื่อคัดลอกแฟ้มข้อมูลจนหมดแล้ว โปรแกรมจะปิดแฟ้มทั้งสอง (เรียกระบบอีก) และ แสดงข้อความบนจอภาพ (เรียกระบบอีก) ว่าเสร็จแล้ว และสุดท้ายหยุดโปรแกรม (เรียกระบบเป็นครั้งสุดท้ายเช่นกัน) จากตัวอย่างนี้ จะเห็นว่าโปรแกรมหนึ่ง ๆ อาจเรียกระบบบ่อยครั้งมาก การติดต่อระหว่างโปรแกรมกับสิ่งแวดล้อม จำเป็นต้องทำผ่านระบบปฏิบัติการทั้งสิ้น

<u>การส่งตัวแปรไปให้ระบบมี 3 วิธี คือ</u>

1. ส่งผ่านทางรีจีสเตอร์ วิธีนี้ง่ายและเร็วที่สุด
2. ถ้ามีตัวแปรมากกว่าจำนวนรีจีสเตอร์ของคอมพิวเตอร์ อาจส่งผ่านทางหน่วยความจำหลัก โดยส่งข้อมูลเป็นชุดหรือตารางไว้ในหน่วยความจำหลัก และส่งตำแหน่งที่อยู่ไปทางรีจีสเตอร์ ดังรูป



รูปที่ 1.14 การส่งตัวแปรผ่าน รีจีสเตอร์

3. ส่งโดยการใช้ stack โปรแกรมจะดัน (push) ข้อมูลเข้าใน stack ของหน่วยความจำ และให้ระบบดึง (pop) ข้อมูลออกมาเอง

2 วิธีหลังนี้เป็นที่นิยมมากกว่า เพราะสามารถส่งตัวแปรได้ไม่จำกัดจำนวน เราสามารถจัดคำสั่งเรียกระบบได้เป็น 5 กลุ่มใหญ่ ๆ คือ การควบคุมโปรเซส (process control) การใช้งานแฟ้มข้อมูล (file manipulation) การใช้งานอุปกรณ์ (device manipulation) การใช้งานข้อมูลของระบบ (Information maintenance) และ การสื่อสาร (communication) ตารางต่อไปนี้ แสดงสรุปคำสั่งเรียกระบบแต่ละกลุ่ม ในระบบปฏิบัติการทั่วไป

<u>ตารางแสดงประเภทของคำสั่งเรียกระบบ (Types of system calls)</u>

การควบคุมโปรเซส

- หยุด , ยกเลิก (End , Abort)
- นำเข้าในหน่วยความจำ , ให้ทำงาน (load , execute)
- สร้างและทำลายโปรเซส (Create and Terminate Process)
- อ่านและเขียนข้อมูลจำเพาะของโปรเซส (Get and Set Process Attributes)
- รอจนถึงเวลาที่กำหนด (Wait for Time)

- รอคอยสัญญาณ , ส่งสัญญาณ (Wait Event , Signal Event)
- ขอ และ คืน หน่วยความจำ (Allocate and Free Memory)

การใช้งานแฟ้มข้อมูล (File Manipulation)

- สร้าง และ ลบ แฟ้มข้อมูล (Create and Delete File)
- เปิด , ปิด (Open , Close)
- อ่าน , เขียน , ย้ายตำแหน่ง (Read , Write , Reposition)
- อ่าน และ เขียนข้อมูลจำเพาะของแฟ้มข้อมูล (Get and Set File Attributes)

การใช้งานอุปกรณ์ (Device Manipulation)

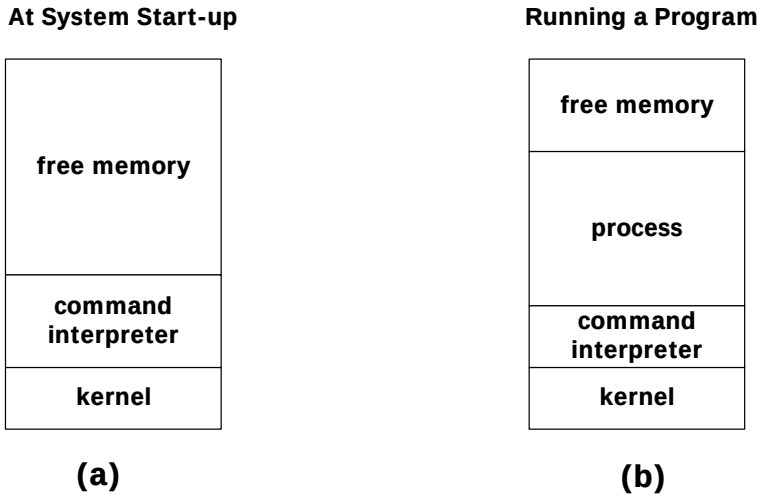
- การร้องขอ และ การคืนอุปกรณ์ (Request and Release Device)
- อ่าน , เขียน , ย้ายตำแหน่ง (Read , Write , Reposition)
- อ่าน และ เขียนข้อมูลจำเพาะของอุปกรณ์ (Get and Set Device Attributes)
- การต่อเชื่อม หรือ ตัดขาดทางตรรกะจากอุปกรณ์ (logically Attach or Detach Devices)

การใช้งานข้อมูลของระบบ (Information Maintenance)

- การอ่าน และ การกำหนดเวลาหรือวันที่
- การอ่าน และ เขียน ข้อมูลจำเพาะของระบบ (System Data)
- การอ่าน และ เขียน ข้อมูลจำเพาะของโปรเซส , แฟ้มข้อมูล , อุปกรณ์

2. <u>การควบคุมโปรเซสและงาน</u> (Process and Job Control)

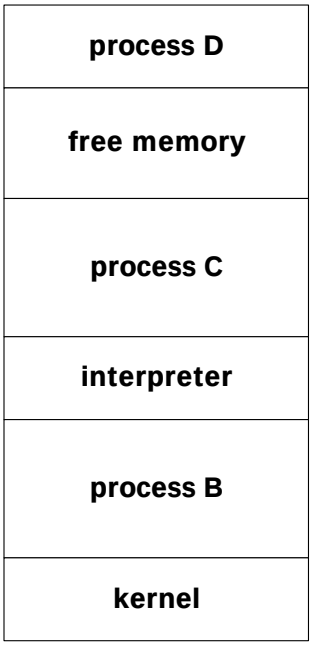
ระบบปฏิบัติการแต่ละระบบ มีระบบย่อยในการควบคุมโปรเซสแตกต่างกัน ตัวอย่างเช่น MS-DOS ซึ่งเป็นระบบปฏิบัติการแบบทำงานเดียว (single – tasking) เมื่อเริ่มเปิดเครื่อง ระบบจะเรียกตัวแปลคำสั่งขึ้นมาเตรียมพร้อมรับคำสั่งจากผู้ใช้ (รูป a)



รูปที่ 1.15 ระบบปฏิบัติการแบบทำงานเดียว (single – tasking)

เมื่อผู้ใช้สั่งให้โปรแกรมทำงาน ระบบจะไม่สร้างโปรเซสใหม่ เพียงแต่นำโปรแกรมที่ต้องการลงสู่หน่วยความจำ โดยเขียนทับตัวแปลคำสั่งเองจนเกือบหมด เพื่อให้ผู้ใช้มีพื้นที่ใช้งานมากที่สุด (รูป b) และกำหนดค่าตัวชี้คำสั่งไปที่บรรทัดแรกในโปรแกรมนั้น เมื่อโปรแกรมนั้นทำงานเสร็จ หรือ เกิดข้อผิดพลาด ระบบก็จะเก็บค่าระดับความผิดพลาดใช้หน่วยความจำของระบบและระบบจะย้ายการควบคุมไปที่ตัวแปลคำสั่งส่วนที่ยังเหลืออยู่ในหน่วยความจำ ตัวแปลคำสั่งก็จะนำส่วนของตัวเองที่ถูกเขียนทับไปกลับมาลงในหน่วยความจำใหม่อีกครั้ง (อ่านมาจากจานบันทึก) แล้วแสดงค่าระดับความผิดพลาด ต่อผู้ใช้หรือโปรแกรมต่อไป

ส่วนระบบปฏิบัติการ Berkeley UNIX ซึ่งเป็นแบบหลายโปรแกรม (multitasking) เมื่อผู้ใช้เข้ามาในระบบ (log on) ตัวแปลคำสั่ง หรือ shell ก็จะเริ่มทำงาน shell โปรแกรมนี้ก็เหมือนกันตัวแปลคำสั่งใน MS-DOS (ที่จริงแล้ว MS-DOS ออกแบบมาจาก UNIX) คือจะรับคำสั่งจากผู้ใช้และทำงานตามคำสั่งนั้น แต่เนื่องจาก UNIX เป็นระบบหลายโปรแกรม ดังนั้น shell อาจทำงานไปพร้อม ๆ กับโปรแกรมที่ผู้ใช้สั่งก็ได้ ดังรูป



รูปที่ 1.16 ระบบปฏิบัติการ แบบหลายโปรแกรม (multitasking)

Shell จะเรียกระบบเพื่อสร้างโปรเซสใหม่ และนำโปรแกรมที่ต้องการลงสู่หน่วยความจำเพื่อให้ทำงาน (โดยการเรียกระบบ) การทำงานจะเป็นแบบใดขึ้นกับคำสั่งของผู้ใช้ว่าจะให้ shell คอย หรือให้โปรเซสใหม่ทำงานอยู่เบื้องหลังขนานกันไปเลย (background mode) ถ้าเป็นการทำงานแบบขนาน shell ก็จะกลับมาคอยคำสั่งจากผู้ใช้ต่อไปทันทีโดยไม่รอให้โปรเซสใหม่ทำงานเสร็จ โปรเซสใหม่จึงไม่สามารถใช้จอภาพเดิมเป็นที่รับหรือส่งข้อมูลได้ แต่จะผ่านทางแฟ้มข้อมูลแทน ผู้ใช้ก็สามารถสั่งให้ทำงานโปรแกรมอื่น ๆ ได้ หรือคอยติดตามตรวจดูการทำงานา ของโปรแกรมเดิม เช่น เปลี่ยนแปลงศักดิ์ เป็นต้น เมื่อโปรเซสที่ทำงานอยู่เบื้องหลังทำงานเสร็จมันก็จะเรียกระบบเพื่อหยุดการทำงานและส่งค่าระดับความผิดพลาด (error code) เป็น 0 หรือเลขอื่น ๆ ซึ่ง shell โปรแกรมนี้หรือโปรแกรมอื่น ๆ ก็จะนำไปใช้ได้

3. <u>การใช้งานแฟ้มข้อมูล</u> (File Manipulation)

ในหัวข้อนี้จะอธิบายโดยกว้าง ๆ เท่านั้น เริ่มจากการสร้าง (create) และการลบ (delete) แฟ้มข้อมูล โดยการเรียกระบบพร้อมส่งตัวแปรชื่อแฟ้ม กับข้อมูลจำเพาะอื่น ๆ (file attribute) เมื่อมีแฟ้มข้อมูลแล้วก็ต้องมีการเปิด (open) ขึ้นมาใช้ โดยการอ่าน (read) เขียน (write) ย้ายตำแหน่ง (reposition) และสุดท้ายคือการปิด (close) แฟ้มข้อมูลเมื่อไม่ต้องการใช้อีก

4. <u>การบริหารอุปกรณ์</u> (Device Management)

โปรแกรมหนึ่ง ๆ อาจต้องการทรัพยากรเพิ่มเติมขณะทำงานเพื่อให้สามารถทำงานต่อไปได้ เช่น หน่วยความจำเพิ่ม , เครื่องขับเทป , การใช้แฟ้มข้อมูล เป็นต้น ถ้ามีทรัพยากรว่างอยู่ โปรแกรมอาจได้ทรัพยากรตามที่ร้องขอ และทำงานต่อไปได้ ถ้าไม่มี โปรแกรมจะต้องรอจนกว่าจะมี แฟ้มข้อมูลสามารถนับอุปกรณ์ทางตรรกแบบหนึ่ง ดังนั้นคำสั่งเรียกระบบสำหรับแฟ้มข้อมูลก็จะต้องมีสำหรับอุปกรณ์ด้วย และส่งคืน (release) เมื่อใช้เสร็จ คำสั่งนี้คล้ายกับคำสั่งเปิดและปิดแฟ้มข้อมูล

เมื่อผู้ใช้ได้รับอุปกรณ์ (หลังจากที่ร้องขอแล้ว) ก็สามารถอ่าน , เขียน , ย้ายตำแหน่งอุปกรณ์เหมือนในแฟ้มข้อมูลได้ จะเห็นได้ว่าอุปกรณ์และแฟ้มข้อมูลมีลักษณะคล้ายกันมากจนระบบปฏิบัติการหลายอันรวมทั้งสองสิ่งไว้เป็นระบบย่อยเดียวกัน โดยกำหนดชื่อแฟ้มพิเศษสำหรับอุปกรณ์แต่ละชนิด

5. <u>การใช้งานข้อมูลของระบบ</u> (Information Maintenance)

ยังมีคำสั่งเรียกระบบหลายคำสั่งที่เป็นเพียงการส่งข้อมูลระหว่างโปรแกรมผู้ใช้กับระบบปฏิบัติการเท่านั้น เช่น จำนวนผู้ใช้รุ่นปัจจุบัน , รุ่น (version) ของระบบ , จำนวนหน่วยความจำที่ยังว่างอยู่ , พื้นที่ในจานบันทึกที่ยังว่างอยู่ เป็นต้น

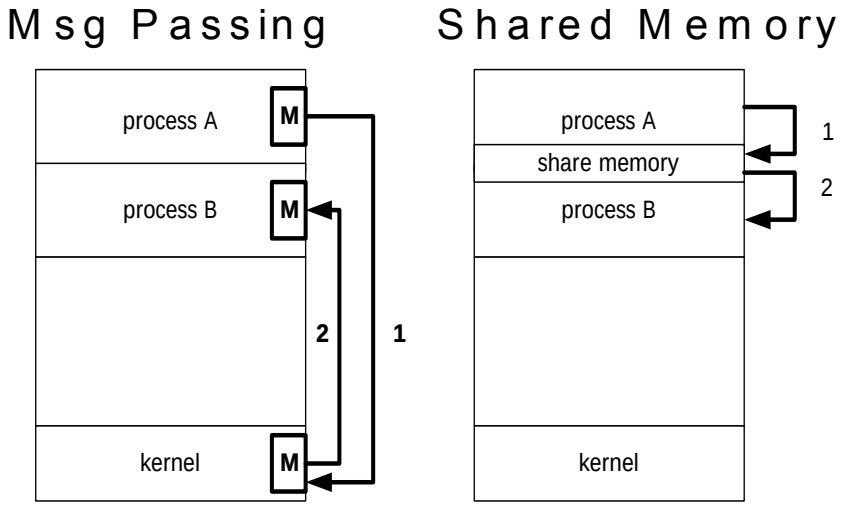
ระบบปฏิบัติการมีข้อมูลของงานทั้งหมดในระบบ รวมทั้งโปรเซสทุกโปรเซสด้วยและผลก็มีคำสั่งเรียกระบบเพื่อดูข้อมูลเหล่านี้ด้วย (อ่านข้อมูลจำเพาะของโปรเซส (get process attribute) และกำหนดข้อมูลจำเพาะของโปรเซส (set process attributes))

6. <u>การสื่อสาร</u> (Communication)

การสื่อสารระหว่างโปรเซสมี 2 วิธีคือ

- การส่งผ่านข้อความ (message passing) ข้อมูลจะถูกส่งผ่านระบบส่งข้อมูลในระบบปฏิบัติการ ก่อนที่จะส่งข้อมูลได้จะต้องมมีการต่อเชื่อม (connection) และต้องทราบชื่อของโปรเซสที่จะติดต่อด้วย รวมทั้งชื่อเครื่องคอมพิวเตอร์ที่โปรเซสนั้นทำงานอยู่ (ในกรณีที่อยู่ต่างเครื่องกัน) คอมพิวเตอร์แต่ละเครื่องในเครือข่ายจะมีชื่อเครื่อง (host name) โปรเซสแต่ละอันก็จะมีชื่อโปรเซส (process name) เช่นกัน ชื่อนี้อาจเป็นตัวเลขก็ได้มีคำสั่งเรียกระบบขอชื่อเครื่อง (get hostid) และคำสั่งขอชื่อโปรเซส (get processid) เพื่อให้ผู้ใช้ทราบชื่อ และใช้คำสั่งเรียกระบบเปิดและปิด (ในกรณีที่ใช้ระบบแฟ้มข้อมูลแทนการติดต่อด้วย) หรือคำสั่งต่อเชื่อม (open connection) และตัดสาย (close connection) โปรเซสที่เป็นผู้รับจะต้องเตรียมรับข้อความ โดยใช้คำสั่งรับสาย (accept connection) ระบบมักสร้างโปรเซสพิเศษขึ้นเป็นตัวแทน (daemons) ในการรับสายนี้ซึ่งจะเรียกคำสั่ง “รอคอยการติดต่อ” (wait for connection) รอจนมีสัญญาณติดต่อมาถึงผู้ที่เริ่มติดต่อหรือเรียกว่า client และผู้รับการติดต่อหรือเรียกว่า server เมื่อต่อเชื่อมกันแล้ว จะส่งข้อมูลกันโดยคำสั่งเรียกระบบอ่านข้อความ (read message) และเขียนข้อความ (write message) จนสุดท้ายคำสั่งตัดสาย (close connection) เพื่อสิ้นสุดการติดต่อ
- การใช้หน่วยความจำร่วม (share memory) โปรเซสหนึ่งใช้คำสั่งเรียกระบบ map-memory เพื่อที่จะสามารถใช้เนื้อที่ในหน่วยความจำส่วนที่เป็นของอีกโปรเซสหนึ่ง (โดยปกติระบบจะป้องกันไม่ให้โปรเซสแต่ละอันใช้หน่วยความจำซ้ำกัน) หลังจากนี้โปรเซสก็สามารถติดต่อได้ทางหน่วยความจำร่วมส่วนนี้ รูปแบบของข้อมูล และตำแหน่งต่าง ๆ เหล่านี้อยู่นอกเหนือการควบคุมของระบบปฏิบัติการ ดังนั้นโปรเซสที่ติดต่อกันต้องรับผิดชอบในการอ่านเขียนข้อมูลเองว่าไม่มีการเขียนข้อมูลลงไปในที่เดียวกันพร้อมกัน

ระบบปฏิบัติการส่วนใหญ่ใช้วิธีหนึ่งใน 2 วิธีข้างต้นนี้ บางระบบอาจมีให้ทั้ง 2 วิธี วิธีส่งผ่านข้อความเหมาะกับเมื่อมีข้อมูลจำนวนน้อยที่ต้องการส่ง เพราะไม่ต้องกังวลเรื่องปัญหาการใช้ข้อมูลพร้อมกัน ทั้งยังเหมาะกับกรณีที่เป็นการติดต่อระหว่างคอมพิวเตอร์ด้วย ส่วนวิธีใช้หน่วยความจำร่วมสามารถส่งข้อมูลได้รวดเร็ว และสะดวกมากเนื่องจากเป็นการอ่านเขียนจาก หน่วยความจำโดยตรง แต่ก็มีปัญหาในการป้องกันการซ้ำซ้อน และการประสานความเร็วในการใช้



รูปที่ 1.17 การส่งผ่านข้อความ และ การใช้หน่วยความจำร่วม

1.3.3.2 โปรแกรมระบบ (System Programs)

โปรแกรมระบบช่วยให้การพัฒนาและการทำงานของโปรแกรมอื่นๆง่ายและสะดวกเราสามารถแบ่งได้หลายประเภทดังนี้

- การใช้งานแฟ้มข้อมูล (File manipulation) โปรแกรมเหล่านี้ทำหน้าที่สร้าง , ลบ , ทำสำเนา , เปลี่ยนชื่อ , พิมพ์ , แสดงรายชื่อ , พิมพ์ข้อมูลทั้งหมด และทำการอื่นๆ เกี่ยวกับแฟ้มข้อมูลและสารบัญ
- ข้อมูลสถานะของระบบ (Status Information) โปรแกรมบางตัวเพียงต้องการข้อมูลจากระบบเช่น วันที่ , เวลา , จำนวนหน่วยความจำที่เหลือ , เนื้อที่ว่างบนจานบันทึก ,จำนวนผู้ใช้ เป็นต้น โปรแกรมระบบเหล่านี้จะแสดงข้อมูลที่ต้องการในรูปแบบที่เหมาะสมบนจอภาพ หรือแฟ้มข้อมูลหรืออุปกรณ์แสดงข้อมูลอื่น

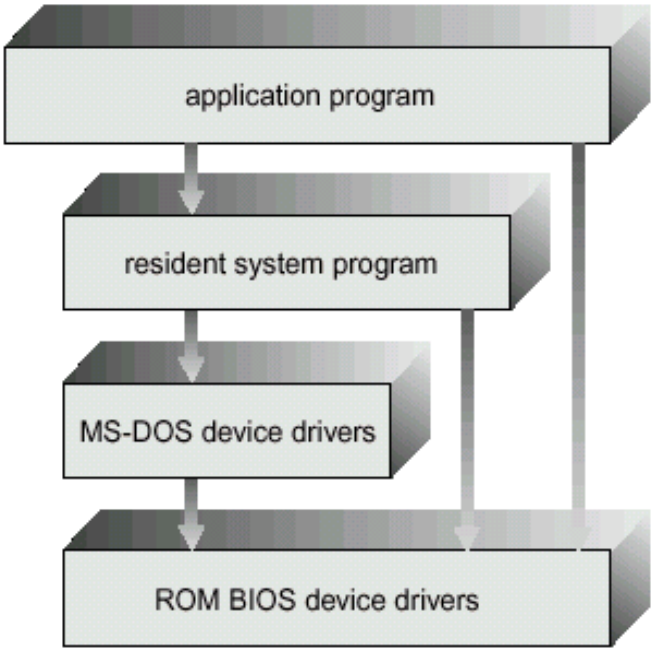
- การแก้ไขแฟ้มข้อมูล (File Modification) อาจมี “ตัวแก้ไขแฟ้มข้อความ” (บรรณาธิการ หรือ editor) หลาย ๆ โปรแกรมในระบบเพื่อให้ผู้ใช้ใช้
- ตัวแปลภาษา (Programming-language Support) ตัวแปลภาษาแบบ Compilers ตัวแปลภาษา assembly ตัวแปรภาษาแบบ Interpreter อาจนับเป็นส่วนหนึ่งของโปรแกรม บางระบบ อาจแยกโปรแกรมเหล่านี้ขายต่างหาก
- ตัวนำโปรแกรมลงหน่วยความจำและทำงาน (Program loading and Execution) เมื่อโปรแกรมภาษาต้นฉบับ (ฟอร์แทรน , เบสิก , ซี , โคบอล , ปาสคาล , ลิสป์ เป็นต้น) ถูกแปลเป็นภาษาเครื่องแล้ว ยังต้องมีตัวเชื่อม (linkage editor) ตัวนำโปรแกรมลงหน่วยความจำ (absolute or relocatable loader or overlay loader) และอาจมีตัวช่วยแก้ไขโปรแกรม (Debugger) ทั้งในภาษาเครื่องด้วย
- การสื่อสาร (Communication) เช่นการติดต่อระหว่างผู้ใช้แต่ละคน (electronic mail) หรือ การส่งแฟ้มข้อมูลข้ามเครื่อง (File transfer) หรือการติดต่อระหว่างโปรเซสต่าง ๆ ในระบบ

โปรแกรมระบบที่ดูเหมือนจะสำคัญที่สุดน่าจะเป็นตัวแปลคำสั่ง (Command interpreter) ซึ่งมีหน้าที่อ่านคำสั่งต่อไปของผู้ใช้ และทำงานตามคำสั่งนั้น

1.3.3.3 โครงสร้างของระบบ (System Structure)

1. <u>โครงสร้างอย่างง่าย</u> (Simple Structure)

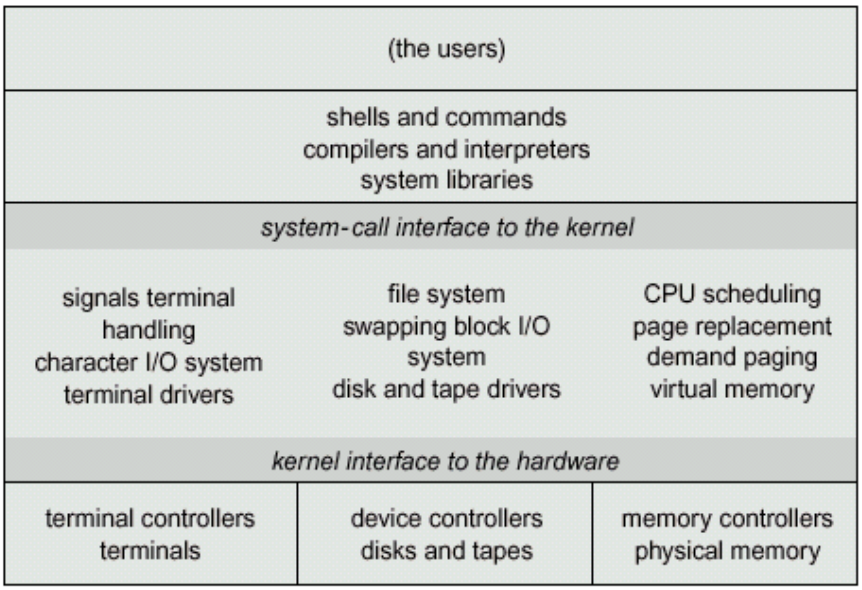
MS-DOS ถูกออกแบบให้สามารถทำงานได้มากที่สุดในเนื้อที่ที่น้อยที่สุด ดังนั้นจึงไม่มีการแบ่งเป็นโมดูล (modules) อย่างระมัดระวัง โครงสร้างของ MS-DOS ในปัจจุบันแสดงได้ดังรูป



รูปที่ 1.18 โครงสร้างของ MS-DOS

แม้ว่า MS-DOS จะมีโครงสร้างแต่ส่วนเชื่อมต่อ (interface) และหน้าที่ต่าง ๆ ไม่มีการแบ่งแยกชัดเจน ตัวอย่างเช่น โปรแกรมประยุกต์สามารถใช้โปรแกรมรับส่งข้อมูลพื้นฐาน (basic I/O routine) เพื่อที่จะเขียนข้อมูลโดยตรงไปที่จอภาพหรือจานบันทึก ซึ่งการยอมให้โปรแกรมอื่น ๆ ทำได้ดังนี้ ระบบอาจเสียหาย หรือข้อมูลในจานบันทึกอาจถูกลบได้ ที่เป็นเช่นนี้เพราะ MS-DOS ไม่มีฮาร์ดแวร์ช่วยในการป้องกัน หรือการทำงานแบบ 2 ช่วง

อีกตัวอย่างหนึ่งคือ ระบบยูนิกซ์รุ่นแรก ยูนิกซ์ก็ถูกจำกัดโดยฮาร์ดแวร์สมัยแรก ๆ เช่นกัน ระบบประกอบด้วย 2 ส่วนคือ แกนกลางของระบบ (kernel) และโปรแกรมระบบ แกนกลางของระบบถูกแบ่งออกเป็น ส่วนต่อเชื่อมและตัวควบคุมอุปกรณ์ ดังรูป



รูปที่ 1.19 โครงสร้างของ UNIX

จะเห็นได้ว่าทุกสิ่งทุกอย่างอยู่ใต้ส่วนต่อเชื่อมในการเรียกระบบ (system-call interface) และอยู่เหนือฮาร์ดแวร์ คือ แกนกลางของระบบ ซึ่งมีระบบแฟ้มข้อมูล , การจัดตารางการใช้หน่วยประมวลผลกลาง , การบริหารหน่วยความจำหลัก และหน้าที่อื่น ๆ อีก โดยผ่านทางคำสั่งเรียกระบบ เห็นได้ชัดว่ามีงานมากมายรวมกันอยู่ใน 1 ระดับชั้นเท่านั้น โปรแกรมระบบที่อยู่เหนือขึ้นไปทำหน้าที่ให้บริการต่าง ๆ ต่อโปรแกรมอื่นได้แก่ การแปลภาษา การใช้งานแฟ้มข้อมูล เป็นต้น

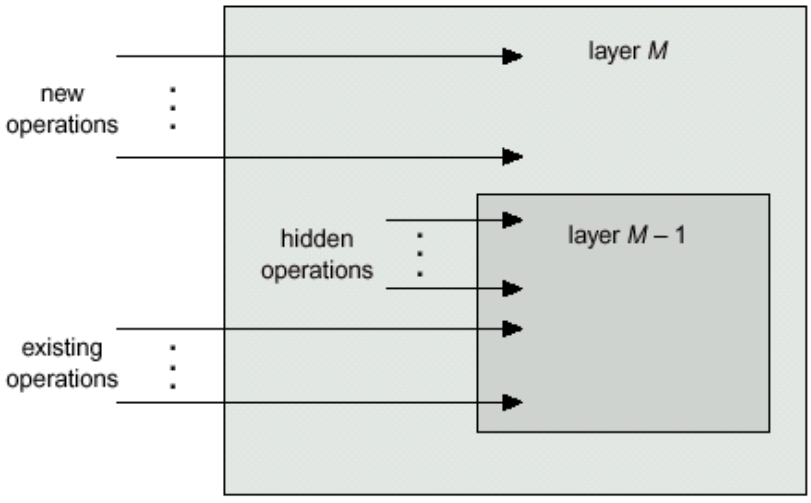
ตัวเชื่อมต่อกับผู้เขียนโปรแกรม (programmer interface) ในระบบยูนิกซ์ก็คือคำสั่งเรียกระบบ (system call) ส่วนตัวเชื่อต่อกับผู้ใช้ระบบ (user interface) ก็คือโปรแกรมระบบ (system program) ทั้งโปรแกรมระบบ และ คำสั่งเรียกระบบเป็นตัวกำหนดบริการของแกนกลางระบบยูนิกซ์รุ่นต่อๆ มา

2. <u>แนวคิดในการแบ่งชั้น</u> (Layer Approach)

ยูนิกซ์รุ่นใหม่ ๆ ได้รับการออกแบบให้สามารถใช้ฮาร์ดแวร์ใหม่ ๆ ได้โดยใช้แนวคิดแบบบนลงล่าง (Top-down Approach) เราสามารถกำหนดหน้าที่ต่าง ๆ แบ่งเป็นระบบย่อยได้อย่างสมบูรณ์ การซ่อนรายละเอียด (Information Hiding) เป็นสิ่งสำคัญอย่างยิ่งในการออกแบบ ทำให้ผู้เขียนโปรแกรมสามารถทำงานได้ตามที่แจ้งไว้

การแบ่งระบบเป็นระบบย่อย (modular) อาจทำได้หลายวิธี แต่วิธีหนึ่งที่น่าสนใจที่สุดคือแนวคิดในการแบ่งชั้น (Layered Approach) ซึ่งแบ่งระบบออกเป็นชั้น ๆ แต่ละชั้นสร้างอยู่บนชั้นล่าง ชั้นล่างสุด (ชั้นที่ 0) คือ ฮาร์ดแวร์ ชั้นบนสุด (ชั้นที่ N) คือผู้ใช้

ชั้นต่าง ๆ ของระบบเป็นการสร้างขึ้นโดยใช้หลักการของ object (abstract object) ซึ่งห่อหุ้มข้อมูล (encapsulation) และการทำงานเกี่ยวกับข้อมูลนั้นไว้ ดังรูป



รูปที่ 1.20 แนวคิดในการแบ่งชั้น (Layered Approach)

แสดงแนวความคิดในการแบ่งชั้นของชั้นที่ M ภายในชั้นจะมีโครงสร้างข้อมูล และโปรแกรมย่อยซึ่งเอาให้ชั้นที่สูงขึ้นไปเรียกใช้ ชั้นที่ M ก็สามารถเรียกโปรแกรมย่อยของชั้นล่างลงไปได้

ข้อดีของหลักการแบ่งชั้นนี้คือ การเป็นระบบย่อย (Modularity) ชั้นต่าง ๆ ถูกแบ่งโดยหลักการที่ว่าชั้นนั้น ๆ ต้องใช้คำสั่ง หรือบริการเฉพาะจากชั้นต่ำกว่าเท่านั้น ซึ่งวิธีนี้จะช่วยให้การหาข้อผิดพลาดหรือตรวจสอบระบบทำได้ง่ายมากขึ้น เราอาจแก้ไขข้อผิดพลาดต่างๆ ของชั้นที่ 1 ได้โดยไม่ต้องคำนึงถึงชั้นอื่น ๆ เลย เพราะชั้นที่ 1 ใช้คำสั่งจากชั้นที่ 0 หรือฮาร์ดแวร์เท่านั้น (สมมติว่าไม่มีข้อผิดพลาดในฮาร์ดแวร์) เมื่อตรวจสอบชั้นที่ 1 แล้ว ชั้นที่ 2 ก็สามารถทำได้ในทำนองเดียวกัน และเรื่อยๆ ขึ้นไป ถ้ามีข้อผิดพลาดเกิดขึ้นที่ชั้นใด เราจะทราบได้ทันทีว่า ข้อผิดพลาดเกิดจากชั้นนั้น ๆ เพราะชั้นล่าง ๆ ลงไปได้ตรวจสอบแล้ว ดังนั้นการออกแบบและสร้างระบบปฏิบัติการจะง่ายขึ้น เมื่อระบบถูกแบ่งเป็นชั้นๆ มีการใช้แนวคิดในการแบ่งชั้นนี้ ครั้งแรกในระบบปฏิบัติการ (THE – Technische Hogeschool Eindhouen) ซึ่งแบ่งเป็น 6 ชั้น (ดังรูป) ชั้นล่างสุดเป็นฮาร์ดแวร์

ชั้นที่ 5 : โปรแกรมผู้ใช้ (user programs)
ชั้นที่ 4 : ที่พักข้อมูลสำหรับอุปกรณ์รับส่งข้อมูล (buffering for I/O device)
ชั้นที่ 3 : ตัวควบคุมอุปกรณ์ของหน้าปัทม์ของผู้ควบคุมเครื่อง (operator-console device driver)
ชั้นที่ 2 : การบริหารหน่วยความจำ (memory management)
ชั้นที่ 1 : การจัดตารางการทำงานของหน่วยประมวลผลกลาง (CPU Scheduling)
ชั้นที่ 0 : ตัวเครื่อง (Hardware)

รูปที่ 1.21 THE – Technische Hogeschool Eindhouen

ชั้นที่ 1 เป็นการจัดตารางการใช้หน่วยประมวลผลกลาง (CPU Scheduling) ชั้นที่ 2 เป็นการจัดการหน่วยความจำหลัก ซึ่งใช้ระบบหน่วยความจำเสมือน (Virtual Memory) ชั้นที่ 3 เป็นตัวควบคุมอุปกรณ์หน้าปัทม์ของผู้ควบคุมเครื่อง (Operator-console device driver) ชั้นที่ 4 เป็นที่พักข้อมูลใน

การรับส่งข้อมูล เนื่องจากที่พักข้อมูลในชั้นนี้ และที่พักข้อมูลของอุปกรณ์ในชั้นที่ 3 อยู่เหนือชั้นที่ 2 เราจึงสามารถให้เนื้อที่พักข้อมูลเก็บในหน่วยความจำเสมือน ข้อผิดพลาดที่เกิดขึ้นในที่พักข้อมูลสามารถแสดงในหน้าปัทม์ของผู้ควบคุมเครื่องได้ เพราะอยู่ในชั้นที่สูงกว่า

แนวคิดนี้สามารถนำไปใช้ได้หลายแบบ เช่น ระบบปฏิบัติการวีนัส (Venus) ซึ่งใช้แนวคิดการแบ่งชั้นเช่นกัน ชั้นที่ 0 ถึง 4 เป็นการจัดตารางการใช้หน่วยประมวลผลกลาง และการจัดการหน่วยความจำหลัก ถูกโปรแกรมลงในฮาร์ดแวร์ ซึ่งทำให้ความเร็วในการประมวลผลสูงขึ้นทั้งยังเป็นการออกแบบที่สมบูรณ์ เนื่องจากใช้วิธีการแบ่งชั้น ดังรูป

ชั้นที่ 6 : โปรแกรมผู้ใช้ (user programs)

ชั้นที่ 5 : ตัวควบคุมอุปกรณ์ และ ตัวจัดตาราง (device drivers and schedulers)

ชั้นที่ 4 : หน่วยความจำเสมือน (virtual memory)

ชั้นที่ 3 : ช่องรับ-ส่ง ข้อมูล (I/O channel)

ชั้นที่ 2 : การจัดตารางการทำงานของหน่วยประมวลผลกลาง (CPU scheduling)

ชั้นที่ 1 : ตัวแปรคำสั่งภาษาเครื่อง (instruction interpreter)

ชั้นที่ 0 : ตัวเครื่อง (Hardware)

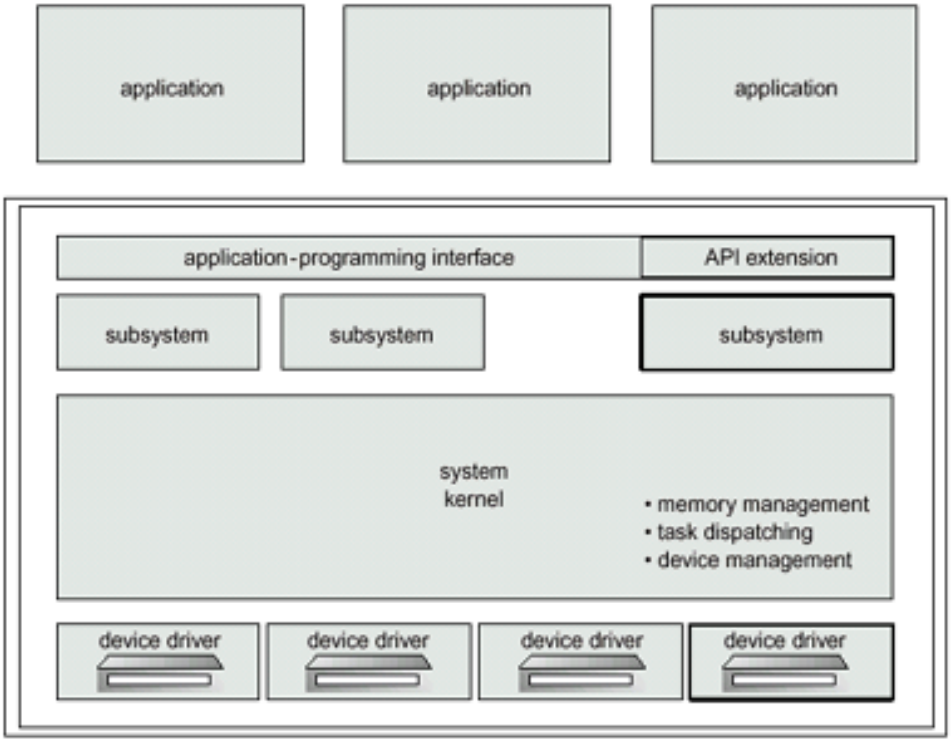
รูปที่ 1.22 ระบบปฏิบัติการวีนัส (Venus)

ปัญหาหลักสำหรับแนวคิดในการแบ่งชั้นคือ การกำหนดชั้น เพราะแต่ละชั้นสามารถใช้คำสั่งของชั้นล่างลงไปเท่านั้น จึงต้องมีการกำหนดอย่างรอบคอบ เช่น ตัวควบคุมอุปกรณ์จานบันทึก

(สำหรับเป็นหน่วยความจำสำรองในระบบหน่วยความจำเสมือน) ต้องอยู่ในชั้นต่ำกว่าชั้นของการจัดการหน่วยความหลัก เพราะการจัดการหน่วยความจำหลักต้องใช้หน่วยความจำสำรอง

การจัดชั้นบนหรือล่างก็เป็นปัญหาเช่นกัน เช่น ตัวควบคุมอุปกรณ์จานบันทึกอยู่เหนือตัวจัดตารางหน่วยประมวลผลกลาง เพราะตัวควบคุมต้องมีการคอย เพื่อรับส่งข้อมูล และหน่วยประมวลผลก็จะย้ายไปทำงานอื่นก่อน แต่ในระบบใหญ่ ๆ ตัวจัดตารางหน่วยประมวลผลกลางอาจต้องเก็บข้อมูลจำเพาะเกี่ยวกับโปรเซสเป็นจำนวนมาก ซึ่งไม่สามารถเก็บไว้ในหน่วยความจำหลักทั้งหมดได้จำเป็นต้องย้ายออกไปเก็บไว้ในหน่วยความจำสำรองบ้าง ดังนั้นตัวควบคุมหน่วยความจำสำรองจะต้องอยู่ใต้ชั้นของตัวจัดตาราง

การออกแบบระบบใหม่ ๆ จึงพยายามลดจำนวนชั้นลง (แต่ต้องการให้มีบริการมากขึ้น) เพื่อหลีกเลี่ยงปัญหาในการกำหนดชั้น ระบบปฏิบัติการ OS/2 ได้เพิ่มการทำงานแบบหลายโปรแกรมและการทำงานแบบ 2 ช่วง เข้าไปพร้อมทั้งบริการใหม่ ๆ ด้วย ทำให้ระบบมีความซับซ้อนมากขึ้นและต้องการฮาร์ดแวร์ช่วยมากขึ้นด้วย ดังรูป



รูปที่ 1.23 โครงสร้างของ ระบบปฏิบัติการ OS/2

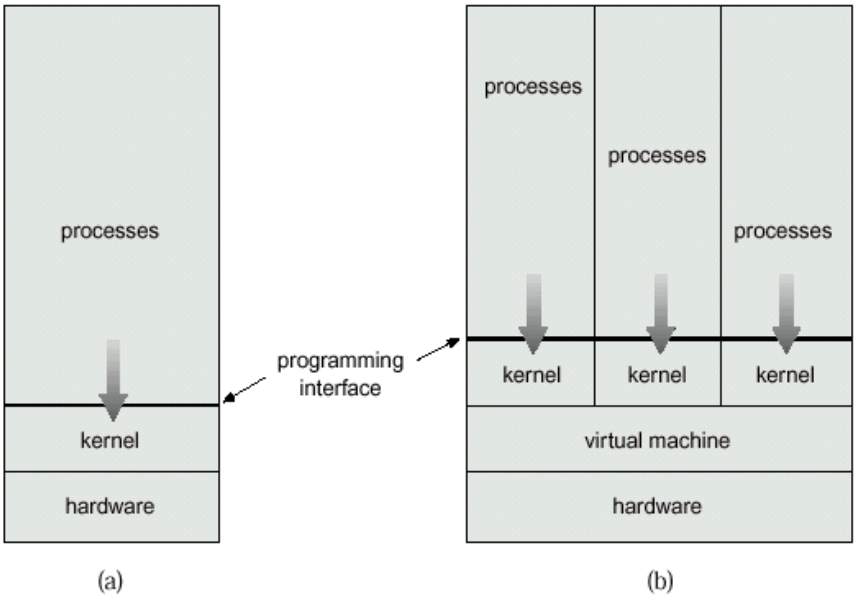
จะเห็นได้ว่า ผู้ใช้ไม่อาจเข้าถึงอุปกรณ์ในชั้นล่าง ๆ ได้ ทำให้ระบบสามารถควบคุมฮาร์ดแวร์ได้ดี และรู้ความเป็นไปของผู้ใช้โดยละเอียด

1.3.4 ลักษณะของ เครื่องจักรเสมือน (Virtual Machines)

ระบบคอมพิวเตอร์อาจแบ่งได้เป็นชั้น ๆ ชั้นล่างสุดเป็นฮาร์ดแวร์ ซึ่งต่อมาเป็นแกนกลางของระบบ (kernel) ซึ่งใช้ฮาร์ดแวร์สำหรับสร้างคำสั่งเรียกระบบให้ผู้ใช้ในชั้นต่อไปเรียกใช้ โปรแกรมระบบ ซึ่งอยู่เหนือชั้นของแกนกลางจึงสามารถใช้ทั้งคำสั่งเรียกระบบ หรือคำสั่งในฮาร์ดแวร์ โดยไม่เห็นความแตกต่างของคำสั่งทั้ง 2 ประเภท และถึงแม้ว่าจะถูกเรียกใช้ต่าง ๆ กัน แต่ต่างก็ทำงานได้ดีเท่ากัน สำหรับโปรแกรมระบบแล้วฮาร์ดแวร์กับคำสั่งเรียกระบบจึงเสมือนอยู่ในชั้นเดียวกัน (ดังรูป a)

บางระบบยังใช้แนวคิดนี้ในชั้นที่สูงขึ้นไปอีกโดยให้โปรแกรมประยุกต์สามารถเรียกโปรแกรมระบบได้โดยง่าย ทั้ง ๆ ที่โปรแกรมอาจอยู่ในชั้นที่สูงกว่าคำสั่งเรียกระบบ หรือคำสั่งในฮาร์ดแวร์แต่โปรแกรมประยุกต์ก็จะเห็นว่าคำสั่งที่สามารถใช้ได้ทั้งหมดเป็นส่วนหนึ่งของเครื่องคอมพิวเตอร์ แนวคิดเรื่องการแบ่งชั้นนำไปสู่แนวคิดของเครื่องจักรเสมือน (Virtual Machine) IBM เป็นผู้ริเริ่มนำแนวคิดนี้มาใช้ในระบบปฏิบัติการ VM

โดยใช้การจัดตารางหน่วยประมวลผลกลาง และหน่วยความจำเสมือน ระบบปฏิบัติการหนึ่งสามารถสร้างสภาพการทำงานแบบหลายโปรเซส แต่ละโปรเซสทำงานบนหน่วยประมวลผลของตน และมีหน่วยความจำเสมือนของตน ทั้งยังมีระบบย่อยอื่น ๆ อีกเช่น คำสั่งเรียกระบบ (system call) และระบบแฟ้มข้อมูล (file system) ซึ่งไม่สามารถทำได้ด้วยฮาร์ดแวร์เพียงอย่างเดียว ในทางตรงกันข้าม แนวคิดของเครื่องจักรเสมือน ไม่ได้ช่วยให้มีระบบย่อย ๆ หรือฟังก์ชันเพิ่มขึ้นเลย เพียงแต่จัดให้ทุก ๆ โปรเซสเสมือนมีฮาร์ดแวร์เป็นของตนเอง (ดังรูป b)



รูปที่ 1.24 โครงสร้างของ OS ที่มี Virtual Machines

ทรัพยากรต่าง ๆ ที่เป็นฮาร์ดแวร์ในระบบจะถูกใช้ร่วมกัน ให้ผู้ใช้แต่ละคนรู้สึกเสมือนมีหน่วยประมวลผลของตนเอง การทำ Spooling และระบบแฟ้มข้อมูลจะสร้าง เครื่องพิมพ์เสมือน (Virtual Line Printer) และ เครื่องอ่านบัตรเสมือน (Virtual Card Readers) หน้าจอของผู้ใช้แต่ละคนในระบบปันส่วนจะทำหน้าที่เป็นหน้าปัทม์เสมือนของผู้ควบคุม (Virtual-operator's console) ตัวอย่างเช่น การแบ่ง partition ของ Harddisk

โปรแกรมเครื่องจักรเสมือน (virtual-machine software) มีหน้าที่จัดการ การทำงานแบบหลายโปรแกรมบนเครื่องจักรเสมือนหลายเครื่องให้ทำงานได้บนเครื่องจักรจริงเครื่องเดียวโดยไม่ต้องคำนึงถึงโปรแกรมช่วยเหลือผู้ใช้อื่นใด (user-support software)

1.3.4.1 <u>การนำไปใช้</u> (Implementation)

แม้ว่าแนวคิดของเครื่องจักรเสมือนจะมีประโยชน์ แต่ก็สร้างได้ค่อนข้างยาก ต้องใช้ความพยายามอย่างมากในการทำให้เสมือนว่ามีเครื่องจักรจริงเหมือนกันหลาย ๆ เครื่อง เช่น การที่ฮาร์ดแวร์สามารถทำงานแบบ 2 ช่วงได้ คือ ช่วงของผู้ใช้ และ ช่วงของผู้ควบคุม โปรแกรมเครื่องจักรเสมือน (virtual-machine software) สามารถจะทำงานในช่วงของผู้ควบคุม เพราะเป็นโปรแกรมในระบบปฏิบัติการ แต่ตัวเครื่องจักรเสมือน (virtual machine) ที่สร้างขึ้นจะทำงานได้ในช่วงผู้ใช้เท่านั้น เพราะเป็นเสมือนผู้ใช้ระบบ เพื่อให้เครื่องจักรเสมือนทำงานได้ 2 ช่วงเหมือนเครื่องจักรจริง เราจึงต้องสร้างช่วงผู้ใช้เสมือน (virtual user mode) และ ช่วงผู้ควบคุมเสมือน (virtual monitor mode) ด้วย ซึ่งทั้ง 2 ช่วงนี้จะทำงานอยู่ในช่วงผู้ใช้ตัวจริง การทำงานใด ๆ ที่เกิดการย้ายช่วงจากช่วงผู้ใช้ ไปสู่ช่วงผู้ควบคุม ในเครื่องจักรจริง (เช่น การเรียกระบบ หรือ การพยายามใช้คำสั่งสงวน) จะต้องทำให้เกิดการย้ายช่วง จากช่วงผู้ใช้เสมือน (virtual user mode) ไปสู่ช่วงผู้ควบคุมเสมือน (virtual monitor mode) ในเครื่องจักรเสมือนด้วย

1.3.4.2 <u>คุณประโยชน์</u> (Benefits)

แนวคิดของเครื่องจักรเสมือนนี้ ยังมีข้อดีข้อเสียหลายข้อ เช่น

- ไม่มีปัญหาด้านความปลอดภัย ในระบบที่ต้องการความปลอดภัยโดยสมบูรณ์ ระบบเครื่อง
- จักรเสมือน จะทำงานเสมือนมีเครื่องจักรแยกเป็นอิสระหลาย ๆ เครื่อง
- ช่วยให้งานวิจัยและพัฒนาระบบปฏิบัติการทำงานได้สะดวกขึ้น
- ผู้เขียนโปรแกรมระบบสามารถใช้เครื่องจักรเสมือนของตนแก้ไขงานระบบส่วนใหญ่

เสมือนว่าทำบนเครื่องจริง ผู้ใช้ระบบทั่วไปก็สามารถใช้งานระบบได้ตามปกติ

- ใช้แก้ปัญหาการเข้ากันไม่ได้ของระบบ (system compatibility problems) เช่น โปรแกรมในระบบ MS-DOS มักใช้บนระบบ CPU ของ Intel แต่ SUN และ DEC ใช้ระบบปฏิบัติการอื่น แต่บริษัทเหล่านี้ก็ต้องการ support ลูกค้าให้สามารถใช้งานโปรแกรมบน MS-DOS ได้ด้วย จึงมีการใช้เครื่องจักรเสมือนเข้ามาช่วย

1.3.4.3 จาวา (Java)

ภาษาจาวา ที่นิยมกันมากถูกออกแบบโดย Sun Microsystems เวลา compile compiler จะสร้างผลลัพธ์ออกมาเป็น bytecode bytecode เหล่านี้เป็นชุดคำสั่งซึ่ง run บน Java Virtual Machine (JVM) เพราะฉะนั้นถ้าจะเขียนโปรแกรมโดยใช้ JAVA ต้องมี JVM run อยู่ก่อนแล้ว JVM สามารถ run บน IBM , Macintosh , Unix , IBM minicomputer และ mainframe นอกจากนี้บน Internet Explore และ Netscape Communicator ก็มี

การทำงานของ JVM จะใช้กลุ่มคำสั่ง stack พื้นฐาน ซึ่งประกอบไปด้วยการคำนวณทางคณิตศาสตร์ , ตรรกะศาสตร์ , การเคลื่อนย้ายข้อมูล , และชุดคำสั่งควบคุมการไหลของข้อมูล (flow control) เพราะว่ามันเป็นเครื่องจักรเสมือน (Virtual Machine) ดังนั้นจึงดูซับซ้อนเกินไปในการที่จะนำคำสั่งต่าง ๆ นั้นไปทำเป็นฮาร์ดแวร์

ข้อดีในการออกแบบของจาวา คือ ทำให้การใช้งานของเครื่องจักรเสมือนอยู่ในสภาพที่สมบูรณ์ ตัวอย่างเช่น bytecodes จะถูกตรวจสอบเพื่อเป็นการรักษาความปลอดภัยหรือความน่าเชื่อถือของเครื่องจักร โดยโปรแกรมภาษาจาวาจะ run ไม่ได้ถ้าไม่ผ่านการตรวจสอบนี้

1.3.5 การออกแบบระบบปฏิบัติการ (System Design)

1.3.5.1 จุดมุ่งหมายของการออกแบบ (Design Goals)

ปัญหาแรกในการออกแบบระบบ คือ การกำหนดจุดประสงค์ และ คุณลักษณะเฉพาะของระบบ ในขั้นแรก การออกแบบขึ้นอยู่กับฮาร์ดแวร์ที่ใช้ และประเภทของระบบที่ต้องการสร้าง ได้แก่ ระบบทำงานแบบกลุ่ม , ระบบปันส่วน , ระบบผู้ใช้คนเดียว , ระบบหลายผู้ใช้ , ระบบกระจายอำนาจ , ระบบโต้ตอบทันที หรือ ระบบใช้งานทั่วไป

ก่อนหน้านั้น การกำหนดความต้องการยิ่งยากขึ้นไปอีก เราอาจแบ่งความต้องการเป็น 2 พวก คือ ความต้องการของผู้ใช้ (user goals) และความต้องการของระบบ (system goals)
ความต้องการของผู้ใช้ได้แก่

- ระบบควรจะใช้สะดวก
- เรียนง่าย
- ใช้ง่าย
- มีความน่าเชื่อถือ

- ปลอดภัยและทำงานได้เร็ว เป็นต้น

ความต้องการเหล่านี้ไม่ค่อยมีประโยชน์ ในการออกแบบเท่าไรนัก เพราะไม่มีมาตราฐานการวัดที่แน่นอนว่าเท่าใดถึงจะเพียงพอ บุคคลอีกกลุ่มหนึ่ง นอกจากผู้ใช้ คือ ผู้ออกแบบ , ผู้สร้าง , ผู้ควบคุม และ ผู้ดูแลรักษาระบบ อาจมีความต้องการคล้าย ๆ กัน ได้แก่

- ระบบควรจะออกแบบได้ง่าย
- สร้างง่าย
- ดูแลรักษาง่าย
- เชื่อถือได้
- ไม่มีข้อผิดพลาด และมีประสิทธิภาพสูง

เช่นเดียวกันความต้องการเหล่านี้ ยังคลุมเคลือ และไม่มีข้อกำหนดที่แน่นอนไม่มีวิธีการที่แน่นอนในการกำหนดความต้องการของระบบปฏิบัติการ ระบบปฏิบัติการที่แตกต่างกันมากย่อมมีความต้องการที่แตกต่างกัน และ การแก้ปัญหาก็แตกต่างกันมากด้วย เช่น ความต้องการในระบบ MS-DOS ซึ่งเป็นระบบผู้ใช้คนเดียวบนเครื่องคอมพิวเตอร์ส่วนบุคคล ย่อมแตกต่างอย่างมากกับระบบ Windows ซึ่งเป็นระบบทำงานหลายโปรแกรม

1.3.5.2 วิธีการและนโยบาย (Mechanisms and Polices)

หลักการสำคัญอันหนึ่ง คือ การแยกนโยบาย (policy) ออกจากวิธีการ (mechanism) วิธีการอธิบายว่าจะทำคำสั่งอย่างไร ส่วนนโยบายเป็นตัวกำหนดว่าควรจะทำอย่างไร
ตัวอย่าง เช่น วิธีการป้องกันการใช้หน่วยประมวลผลกลางนานเกินไป คือ การใช้นาฬิกาจับเวลา แต่การกำหนดว่าผู้ใช้คนหนึ่งควรจะมีเวลาทำงานนานเท่าใดเป็นเรื่องของนโยบาย

การแยกนโยบายออกจากวิธีการ เป็นเรื่องสำคัญต่อความยืดหยุ่นของระบบ เพราะนโยบายอาจเปลี่ยนแปลงได้ ตามสถานที่และเวลา ในกรณีที่แย่ที่สุด การเปลี่ยนแปลงนโยบายแต่ละครั้งอาจทำให้ต้องเปลี่ยนเครื่องจักรไปด้วย วิธีการแบบทั่วไปน่าจะนำมาพิจารณาได้ เพราะเมื่อมีการเปลี่ยนแปลงนโยบาย อาจปรับเปลี่ยนตัวแปรเพียงบางตัว แต่ใช้วิธีการเดิมได้ ตัวอย่างเช่น ในระบบคอมพิวเตอร์หนึ่งต้องการให้ โปรแกรมที่มีการรับส่งข้อมูลมาก (I/O - Intensive) มีศักดิ์สูงกว่าโปรแกรมที่ใช้หน่วยประมวลผลกลางมาก (CPU – Intensive) ส่วนอีกระบบหนึ่ง ต้องการตรงกันข้าม การแยกนโยบายออกจากวิธีการ จะทำให้สามารถใช้วิธีการเดียวกันได้กับระบบทั้งสองนี้ (โดยปรับปรุงเล็กน้อย)

1.3.5.3 การสร้างระบบ (Implementation)

สมัยแรก ๆ เราจะใช้ภาษา assembly ในการเขียนระบบปฏิบัติการ แต่ปัจจุบันเราอาจใช้ภาษาขั้นสูงเขียนแทน ประโยชน์ของการใช้ภาษาชั้นสูงในการเขียน (สร้าง) ระบบปฏิบัติการเห็นได้ชัดเจน คือ สามารถเขียนได้เร็วกว่า , กะทัดรัดกว่า , เข้าใจง่ายกว่า และง่ายต่อการแก้ไขกว่าด้วย

ข้อดีอีกอย่างคือ ทำให้ง่ายในการย้าย (port) ระบบปฏิบัติการไปทำงาน บนเครื่องคอมพิวเตอร์แบบอื่น (ถ้าเราเขียนด้วยภาษาขั้นสูง) ตัวอย่างเช่น ระบบ MS-DOS รุ่นแรก ใช้ภาษา assembly ของ Intel 8088 เขียน MS-DOS รุ่นต่อ ๆ มา ก็มีใช้เพียงบนเครื่องคอมพิวเตอร์ระบบ Intel เท่านั้น ต่างจากระบบยูนิกซ์ ใช้ภาษา C เขียน จึงมีการใช้คอมพิวเตอร์หลายแบบ เช่น Intel , Motorola , sparc

ข้อเสีย คือ อาจทำงานได้ช้ากว่า ใช้หน่วยประมวลผลกลางมากกว่า และมีความต้องการพื้นที่ในการใช้งานมากขึ้น

ในระบบปฏิบัติการ ก็เหมือนกับโปรแกรมอื่น ๆ ที่ประสิทธิผลของโปรแกรมขึ้นอยู่กับโครงสร้างข้อมูลที่ดีและขั้นตอนวิธีที่ถูกต้อง มากกว่าการเขียนโปรแกรมที่รอบคอบรัดกุม เพื่อที่จะหาส่วนที่เป็นจุดวิกฤติหรือ คอขวด (bottlenecks) ของระบบ เราต้องมีวิธีการสังเกตประสิทธิภาพของระบบ โดยการแทรกคำสั่งบางอย่างเข้าไปในโปรแกรม เพื่อคำนวณและแสดงข้อมูลภายในระบบ ข้อมูลที่สำคัญจะถูกบันทึกพร้อมกับเวลาที่ใช้ลงในแฟ้มข้อมูล (log file) ซึ่งข้อมูลเหล่านี้สามารถนำไปวิเคราะห์ต่อได้ เราอาจแสดงผลการบันทึกในขณะทำงานจริง (โดยไม่ต้องเก็บไว้ในแฟ้มข้อมูล) ซึ่งจะช่วยให้ผู้ควบคุมระบบ เข้าใจการทำงานของระบบได้ดีขึ้นและอาจสามารถแก้ไขระบบได้ในขณะทำงานเลย

1.3.6 การนำระบบปฏิบัติการไปใช้งาน

<u>การติดตั้งระบบ (System Generation)</u>

ข้อมูลในการติดตั้งระบบ ควรมีดังนี้

- ใช้หน่วยประมวลผลอะไร ? มีส่วนประกอบพิเศษหรือไม่ ? มีหน่วยประมวลผลกี่ตัว ? อะไรบ้าง ? (ในกรณีเป็นระบบหลายหน่วยประมวลผล)
- ขนาดหน่วยความจำหลักที่มี ? บางระบบอาจหาขนาดได้เองโดยการทดลองเขียนและอ่านข้อมูลลงไปที่ตำแหน่งต่างๆ จนกระทั่งเกิดข้อผิดพลาด ซึ่งตำแหน่งผิดพลาดจะทำให้เราทราบถึงตำแหน่งสูงสุดที่สามารถใช้ได้ ซึ่งก็น่าจะเท่ากับขนาดของหน่วยความจำหลัก
- มีอุปกรณ์อะไรบ้าง ? ระบบปฏิบัติการจำเป็นต้องรู้จักอุปกรณ์ต่าง ๆ ทั้งหมดที่มีอยู่ ข้อมูลต่าง ๆ เกี่ยวกับอุปกรณ์ เช่น หมายเลขอุปกรณ์ , ตำแหน่งอ้างอิง , หมายเลขสัญญาณ , ชนิดและรุ่น ตลอดจนคุณสมบัติพิเศษอื่น ๆ ถ้ามี

ข้อกำหนดพิเศษมีอะไรบ้าง ? หรือตัวแปรต่าง ๆ ของระบบมีค่าเท่าใด ? ข้อมูลเหล่านี้ได้แก่ จำนวน , ขนาดของที่พักข้อมูล , เลือกวิธีการจัดตารางการทำงานของหน่วยประมวลผลกลาง , จำนวนโปรเซสสูงสุดที่ยอมให้มีในขณะหนึ่ง ๆ เป็นต้น

เราสามารถติดตั้งระบบได้หลายวิธี วิธีหนึ่งคือ ใช้ข้อมูลเหล่านี้แก้ไขปรับเปลี่ยนโปรแกรมต้นฉบับ (source code) ของระบบปฏิบัติการโดยตรง เช่น การประกาศข้อมูล , ค่าคงที่ , ค่าเริ่มต้น ,

การแปลอย่างมีเงื่อนไข (condition compilation) เป็นต้น แล้วแปลเป็นภาษาเครื่องชุดใหม่ ผลที่ได้จะเป็นระบบปฏิบัติการ (ในภาษาเครื่อง) ที่มีการปรับแต่งให้เหมาะแก่การติดตั้ง ณ ที่ที่กำหนดข้อมูลข้างต้น

อีกวิธีหนึ่ง (งานน้อยลงกว่าวิธีแรก) คือ ข้อมูลข้างต้นทำให้เกิดตาราง และการเลือกโปรแกรมย่อย ๆ (ในภาษาเครื่อง) จากห้องสมุดโปรแกรม (library) ซึ่งสามารถนำมาเชื่อมต่อกันกับโปรแกรมหลักเป็นผลลัพธ์รวม การเลือกโปรแกรมย่อย ๆ นี้อาจเป็นการเลือกเฉพาะตัวควบคุมอุปกรณ์ที่มีในระบบนี้เท่านั้น เพื่อมาเชื่อมต่อกันเป็นระบบปฏิบัติการสำหรับสถานที่นี้ วิธีนี้เร็วกว่าวิธีแรก แต่โปรแกรมที่ได้อาจมีขนาด และการทำงานเป็นกลาง ๆ มากกว่าวิธีแรก (วิธีแรกจะให้ผลลัพธ์ที่ดีกว่า)

อีกวิธีหนึ่ง คือ สร้างระบบปฏิบัติการที่มีลักษณะเป็นระบบขับเคลื่อนด้วยตาราง (table driven) ทั้งระบบ โปรแกรมทุกแบบจะอยู่ในระบบทั้งหมด การเลือกแบบเกิดขึ้นในขณะทำงานจริง (execution time) ไม่ได้เกิดขึ้นในขณะแปลโปรแกรม (compile time) หรือขณะเชื่อมต่อโปรแกรม (link time) การติดตั้งระบบทำโดยการสร้างตารางเก็บข้อมูลต่าง ๆ ของระบบที่ต้องการ