# บทที่ 2

# ทฤษฎีที่เกี่ยวข้อง

บทนี้จะอธิบายถึงความหมายของตัวแปลภาษาทั้งสองแบบคือคอมไพเลอร์ (compiler) และอินเตอร์พรีเตอร์ (interpreter) และทฤษฎีที่ใช้ในการพัฒนาตัวแปลภาษา อันได้แก่การอธิบายไวยากรณ์โดยสัญกรณ์บีเอ็นเอฟ (BNF notion) ,แผนภาพของไวยากรณ์ (syntax diagram) และการแปลโดยวิธีเรียกซ้ำตามลำดับขั้น

ในส่วนของทฤษฎีที่เกี่ยวข้องกับการพัฒนาอินเตอร์พรีเตอร์เพื่อให้ประมวลผลพร้อมกัน จะอธิบายถึงการจัดการกระบวนการเพื่อให้ทำงานพร้อมกัน การประสานจังหวะ (synchronization) ของกระบวนการ และการติดต่อระหว่างกระบวนการ

## 2.1 ตัวแปลภาษา (translator)

### 2.1.1 ภาษาธรรมชาติ และภาษาเชิงแบบ (natural language and formal language )

ภาษาธรรมชาติ (natural language) เป็นภาษาที่เราใช้อยู่ในปัจจุบัน ซึ่งเป็นภาษาที่มีสามารถในการสื่อสารกันได้ดี แต่ภาษาธรรมชาติไม่เหมาะสมกับการใช้อธิบายภาษาคอมพิวเตอร์ เพราะมีความคลุมเคลือ ซึ่งนำไปสู่ความกำกวม ส่วนภาษาเชิงแบบ (formal language) เป็นภาษาที่พัฒนาขึ้นเพื่อใช้อธิบายภาษาอื่นเพื่อลดความคลุมเคลือ กำกวม ในการสร้างตัวแปลภาษานิยมใช้ภาษาเชิงแบบในการอธิบายวากยสัมพันธ์ของภาษาเพื่อความชัดเจน และสามารถเขียนโปรแกรมตรวจสอบได้

### 2.1.2 คอมไพเลอร์และอินเตอร์พรีเตอร์ (compiler and interpreter)

ตัวแปลภาษา (translator) คือ โปรแกรมที่ทำหน้าที่แปลภาษาต้นแบบ (source language) ไปเป็นอีกภาษาหนึ่งซึ่งเรียกว่าภาษาเป้าหมาย (target language) ตัวแปลภาษาที่ทำงานโดยการ

แปลภาษาต้นแบบไปเป็นรหัสกลาง (intermediate code) แล้วปฏิบัติการโดยตรงกับรหัสกลางนั้น เรียกว่า อินเตอร์พรีเตอร์ (Interpreter) ส่วนตัวแปลภาษาที่ทำหน้าที่แปลภาษาต้นแบบซึ่งเป็นภาษาระดับสูง (high level language) ไปเป็นภาษาเป้าหมายซึ่งเป็นภาษาระดับต่ำ (low level language) เช่นภาษาแอสเซมบลี หรือภาษาเครื่องเรียกว่าคอมไพเลอร์ (compiler)

### 2.1.3 สัญกรณ์บีเอ็นเอฟ (BNF notation)

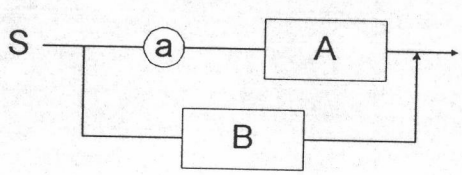
ภาษาที่ใช้เป็นภาษาต้นแบบควรเป็นภาษาที่ชัดเจนไม่คลุมเครือกำกวม สัญกรณ์บีเอ็นเอฟ (BNF notation) เป็นภาษาหนึ่งที่นิยมนำมาใช้ในการอธิบายภาษาเป้าหมายเชิงแบบ สัญกรณ์บีเอ็นเอฟประกอบด้วยกฎเกณฑ์ต่างๆ โดยสัญลักษณ์ที่ถูกแทนที่ได้จะอยู่ทางซ้ายของเครื่องหมาย ::= ส่วนทางด้านขวาของเครื่องหมาย ::= ประกอบด้วยสัญลักษณ์ที่ถูกแทนที่ได้ (non terminal) และ/หรือ สัญลักษณ์ที่ถูกแทนที่ไม่ได้ (terminal) โดยสัญลักษณ์ที่ถูกแทนที่ได้จะเขียนอยู่ในเครื่องหมาย < > ส่วนที่อยู่นอกเครื่องหมาย < > ถือว่าเป็นสัญลักษณ์ที่ถูกแทนที่ได้ เช่น

<S> ::= a<A>|B

คือ S ถูกแทนที่ได้ด้วย a ตามด้วย A หรือ ถูกแทนที่ได้ด้วย B

### 2.1.4 แผนภาพของไวยากรณ์ (syntax diagram)

แผนภาพของไวยากรณ์เป็นเครื่องมืออีกชนิดที่ใช้อธิบายไวยากรณ์ของภาษาเป้าหมาย ประกอบด้วยกฎเกณฑ์ต่างๆ โดยให้วงกลมแทนสัญลักษณ์ที่ถูกแทนที่ได้ สี่เหลี่ยมแทนสัญลักษณ์ที่ถูกแทนที่ไม่ได้ เช่น ไวยากรณ์บีเอ็นเอฟ <S> ::= a<A>|B จะอธิบายโดย แผนภาพของไวยากรณ์ดังนี้




รูปที่ 2.1 แผนภาพแสดงไวยากรณ์ของ <S> ::= a<A>|B

### 2.1.5 เทคนิคการแปลแบบเรียกซ้ำตามลำดับขั้น

เทคนิคที่ใช้ในการแปลคือเทคนิคการแปลแบบเรียกซ้ำตามลำดับขั้น (Recursive descent) คือการเรียกใช้ Non terminal นั้น เพื่อแปลความหมายของ Non terminal ต้องมีกระบวนการรองรับเพื่อดำเนินการอย่างใดอย่างหนึ่งตามหลักเกณฑ์ที่กำหนดไว้เช่น สมมุติกำหนดหลักเกณฑ์ไว้ดังนี้

1. <S> ::= a <A> <B>
2. <S> ::= b
3. <A> ::= c
4. <B> ::= b

โดยที่ <S>,<A> เป็น nonterminal

::= เป็นการแปลความหมายของ nonterminal

a,b,c เป็น terminal

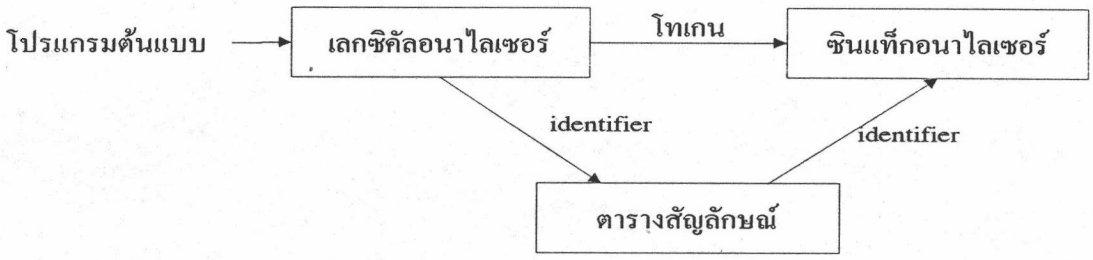
a<A>,<S>,a,b เป็นความหมายของ nonterminal ที่กำหนดไว้ในกฎเกณฑ์ของภาษา

เมื่อมีการตรวจสอบพบ nonterminal <S> หรือ <A> ก็จะมีการเรียกกระบวนความที่เกี่ยวเนื่องสัมพันธ์กับ nonterminal นั้น เช่น หากพบ nonterminal <S> ก็อาจให้ผลลัพท์ได้สองแบบคือ อาจได้ผลลัพท์เป็นแบบที่ 1 คือ <S> ::= a <A> <S> ซึ่งจะต้องมีการเรียกกระบวนการของ nonterminal <A> และ <S> อีก จนกว่าไม่มี nonterminal เหลืออยู่ หรือผลลัพท์แบบที่ 2 คือ terminal <B> ดังนั้นการใช้เทคนิคการแปลแบบเรียกซ้ำตามลำดับขั้น จะทำการแปลและเปรียบเทียบกับกฎเกณฑ์ที่มีอยู่ จนกว่าจะไม่มี non terminal เหลืออยู่

### 2.1.6 ขั้นตอนในการทำงานของตัวแปลภาษา

- เลกซิคัลอนาไลเซอร์ (Lexical analyzer) เป็นขั้นตอนแรกในการดำเนินการแปลคือการอ่านอักขระจากภาษาต้นแบบ และแยกออกมาเป็นกลุ่มคำ (token) โดยแบ่งตาม

ลักษณะและหน้าที่ตามที่ได้ระบุไว้ในโครงสร้างและวากยสัมพันธ์ของภาษาได้แก่ key word (เช่น DO, IF ...) หรือ ตัวระบุ (Identifier) ต่างๆ (เช่น I,J,K,NUMBER...) หรือสัญลักษณ์ของเครื่องหมายทางคณิตศาสตร์ (เช่น < = + - ^ ...) และสัญลักษณ์อื่นๆ ( เช่น , comma ) เพื่อส่งผลลัพท์ให้กับ ซินแทกอนาไลเซอร์ต่อไป




รูปที่ 2.2 แสดงขั้นตอนการทำงานของตัวแปลภาษา

- ซินแทกอนาไลเซอร์ (Syntax analyzer) คือการตรวจสอบโทเกน ว่าถูกต้องตามวากยสัมพันธ์ (Syntax) ของภาษาต้นแบบหรือไม่ หน้าที่ของซินแท็กอนาไลเซอร์ คือรับโทเกนจากสัญลักษณ์เทอร์มินอลจากขั้นตอนเลกซิคัลอนาไลเซอร์ การตรวจสอบวากยสัมพันธ์ทำได้โดยการสร้างพาซเซอร์ (Parser) เช่น จากตัวอย่าง

1. <S> ::= a <B>

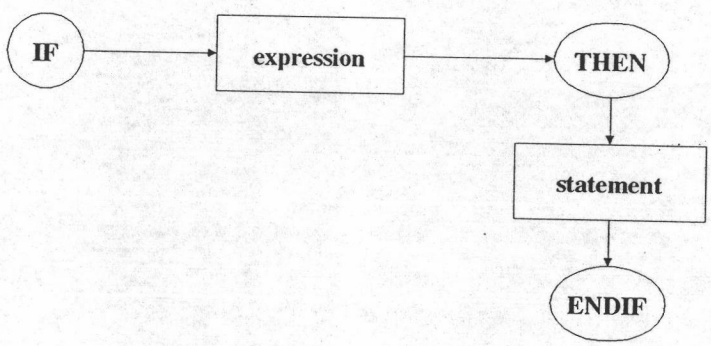
2. | c

3. <B> ::= b

หากต้องการตรวจสอบว่า ab เป็นไปตามไวยกรณ์ข้างต้นหรือไม่ โดยการสร้างพาซเซอร์ดังนี้

| ขั้นตอน | พาซทรี | โทเคนตัวต่อไป | โทเคนที่เหลือ | โปรดักชันที่ใช้ |
|---|---|---|---|---|
| 1 | <S> | | ab | - |
| 2 | <S><br>a <B> | a | b | 1 |
| 3 | <S><br>a <B><br>b | b | - | 3 |

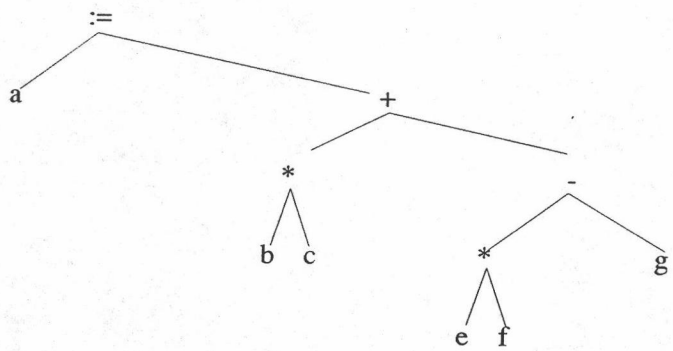
- ซีแมนติกอนาไลเซอร์ (Semantic analyzer) เป็นการตรวจสอบความหมายของภาษาว่าเป็นไปตามนิยามที่กำหนดไว้ในไวยากรณ์หรือไม่ เช่นกำหนดไวยากรณ์ของประโยค IF ไว้ว่าต้องตามด้วยเงื่อนไข คำว่า THEN อนุประโยคปิดท้ายด้วยคำว่า ENDIF จะมีความหมายคือ ไวยากรณ์ IF ต้องตามด้วย เงื่อนไขที่เป็นนิพจน์ (Expression) และในอนุประโยค ระหว่าง THEN กับ ENDIF จะต้องเป็นประโยคเป็นต้น




รูปที่ 2.3 แสดงไวยากรณ์ของคำสั่ง IF-THEN-ENDIF

เมื่อทำการตรวจสอบความหมายของภาษาแล้ว จะสร้างรหัสกลาง ให้อยู่ในรูปแบบขั้นตอนที่ใกล้เคียงกับภาษาเครื่องแต่ยังไม่มีการกำหนดรีจิสเตอร์ในการประมวลผล ทั้งนี้เพื่อ

ประโยชน์ในความสะดวกในการปรับเปลี่ยนโปรแกรมให้สามารถปฏิบัติการได้ในเครื่องหลายเครื่องโดยไม่จำเป็นต้องทำการปรับเปลี่ยนโปรแกรมมากนัก ในการสร้างรหัสกลางทำได้โดยการสร้างซินแทคทรี ตัวอย่างเช่น a := (b * c) + (e * f - g) สร้างซินแทกทรีได้ดังนี้

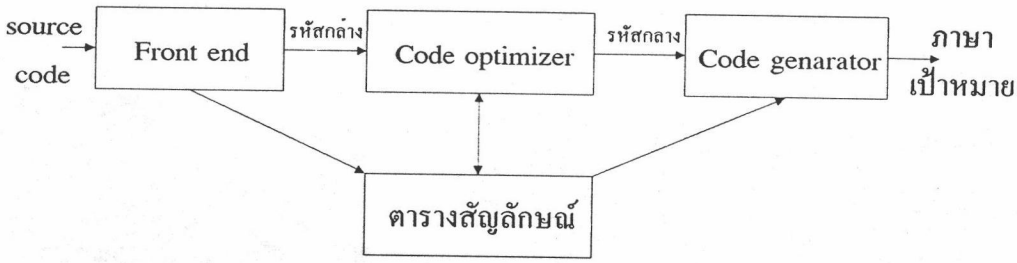



รูปที่ 2.4 Syntax tree ของ a := (b * c) + (e * f - g)

จะได้รหัสกลางดังนี้

| | | | |
|---|---|---|---|
| *(1)* | * | *b* | *c* |
| *(2)* | * | *e* | *f* |
| *(3)* | - | *(2)* | *g* |
| *(4)* | + | *(1)* | *(3)* |
| *(5)* | := | *a* | *(4)* |

- โคดเจนเนอเรเตอร์ (code genarator) เป็นขั้นตอนสุดท้ายในการทำคอมไพเลอร์ คือขั้นตอนการเปลี่ยนรหัสกลางให้เป็นภาษาเป้าหมายสามารถกระทำการ (execute) ได้ เช่นภาษาเครื่องหรือภาษาแอสเซมบลีซึ่งจะขึ้นอยู่กับเครื่องหรือซีพียู (CPU) ที่ต้องการ มีการกำหนดตำแหน่งข้อมูลในหน่วยความจำ รวมทั้งการกำหนดรีจิสเตอร์ที่ใช้เป็นต้น ในขั้นตอนนี้อาจมีการปรับปรุงรหัส ให้มีประสิทธิภาพมากยิ่งขึ้น (code optimizer) คอมไพเลอร์บางตัวอาจไม่มีขั้นตอนนี้ก็ได้ หรืออาจมีขั้นตอนเหล่านี้มากกว่าหนึ่งขั้นตอนก็ได้




รูปที่ 2.5 แสดงขั้นตอนการทำงานของตัวแปลภาษา

## 2.2 การประมวลผลพร้อมกัน (Concurrent Processing)

### 2.2.1 ความหมายของกระบวนการ (process)

กระบวนการ (process) หมายถึง โปรแกรมที่ถูกดำเนินการ (Program in execution) ในการทำงานในระบบการทำงานแบบหลายโปรแกรม (multi programming) จะมีกระบวนการหลายกระบวนการที่ทำงานอิสระ โดยเสมือนว่ามีกระบวนการของตนดำเนินการอยู่ในระบบเพียงกระบวนการเดียว ซึ่งระบบปฏิบัติการ (operating system) จะเป็นผู้จัดการสับหลีกกระบวนการต่างๆและดูแลการทำงานของกระบวนการต่างๆเพื่อให้มีการทำงานที่ประสานจังหวะกันได้ (process synchronization)

### 2.2.2 องค์ประกอบของกระบวนการ

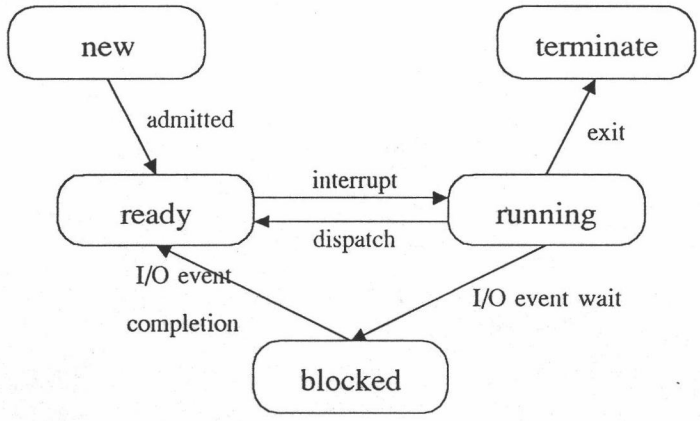
กระบวนการที่สมบูรณ์จะต้องประกอบด้วยส่วนประกอบต่างๆคือ ชื่อและหมายเลขประจำตัวของกระบวนการ (process identifier,PID) , รหัสคำสั่งของกระบวนการ (program code), ข้อมูลที่กระบวนการนั้นต้องจัดการประมวลผล (data) , และบล็อกควบคุมกระบวนการ (process control block)

### 2.2.3 สถานะของกระบวนการ

- สถานะพร้อม (ready state) คือสถานะที่กระบวนการพร้อมที่จะใช้หน่วยประมวลผล กลางทันทีที่ที่ระบบปฏิบัติการมอบหมายให้

- สถานะดำเนินการ (run state) คือสถานะที่กระบวนการกำลังครอบครองหน่วยประมวลผลกลางในการดำเนินการคำสั่งใดๆของกระบวนการนั้น

- สถานะติดขัด (blocked state) คือสถานะที่กระบวนการหยุดรอ หรือกระบวนการนั้นไม่พร้อมที่จะครอบครองหน่วยประมวลผลกลาง




รูปที่ 2.6 แสดงสถานะของกระบวนการ

เมื่อมีการสร้างกระบวนการขึ้นใหม่ กระบวนการจะอยู่ในสถานะพร้อมก่อนและไม่สามารถเข้าไปใช้หน่วยประมวลผลกลางได้ทันที แต่จะถูกนำไปไว้ที่ในลำดับของกระบวนการ (process queue) เมื่อหน่วยประมวลผลกลางว่างลงเนื่องจากกระบวนการที่ครอบครองอยู่ปลดปล่อยซีพียู ระบบปฏิบัติการจะนำเอากระบวนการที่อยู่ต้นคิวในสถานะพร้อมให้เข้ามาใช้หน่วยประมวลผลกลาง และจะเปลี่ยนสถานะของกระบวนการจากสถานะพักเป็นเป็นสถานะดำเนินการ ในสถานะนี้โคดคำสั่งของโปรแกรมจะถูกดำเนินการโดยซีพียู

เมื่อการทำงานของกระบวนการสิ้นสุดลง โดยอาจจะเกิดจากการสั่งหยุดหรือยกเลิกการทำงาน กระบวนการนั้นก็จะถูกระบบปฏิบัติการทำลายไป (kill process)

เนื่องจากมีหลายกระบวนการอยู่ในระบบที่มีหน่วยประมวลผลกลางเพียงตัวเดียว ระบบปฏิบัติการจะแบ่งเวลาในการใช้หน่วยประมวลผลกลางของกระบวนการต่างๆเรียกว่าเวลาควอนตัม (quantum time) หากกระบวนการใดครอบครองหน่วยประมวลผลกลางเกินกว่าเวลาควอนตัม ระบบปฏิบัติการจะทำการย้ายกระบวนการนั้นออกจากหน่วยประมวลผลกลางและเปลี่ยนสถานะไปเป็นสถานะพร้อม และนำเอากระบวนการถัดมา เพื่อประมวลผลต่อไป

หากกระบวนการที่กำลังดำเนินการ และต้องการใช้อุปกรณ์อินพุต-เอาท์พุต ระบบปฏิบัติการจะเปลี่ยนสถานะเป็นติดขัดแทน และย้ายกระบวนการนี้ออกจากหน่วยประมวลผลกลางเพื่อ

ให้กระบวนการที่มีสถานะพร้อมเข้ามาประมวลผลต่อไป ส่วนกระบวนการที่ติดขัดก็จะรอจนกว่า I/O จะเสร็จ จึงสามารถกลับเข้ามาใช้หน่วยประมวลผลกลางได้

### 2.2.4 การประสานจังหวะของกระบวนการ (process synchronization)

- เซมาฟอร์ (semaphore)

เซมาฟอร์คือตัวแปรที่สามารถเปลี่ยนแปรค่าได้โดยผ่านคำสั่งพื้นฐานในการประสานจังหวะ ตัวแปรเซมาฟอร์จะต้องเป็นตัวแปรร่วมระหว่างกระบวนการต่างๆ

คำสั่งพื้นฐานสองคำสั่งที่เป็นคำสั่งที่แบ่งแยกมิได้ (atomic instruction) คือคำสั่ง wait และ signal

- wait (semaphore) : จะทำการรอคอย ตราบใดที่มีค่าของเซมาฟอร์มีค่าน้อยกว่าหรือเท่ากับศูนย์ และหาก semaphore มีค่ามากกว่าศูนย์ จะลดค่าเซมาฟอร์ลงหนึ่ง และจะดำเนินการคำสั่งต่อไป
- signal (semaphore) : คำสั่งนี้จะเพิ่มค่าของเซมาฟอร์ขึ้นอีกหนึ่ง และทำการปลุกกระบวนการที่รออยู่

```
var sem : semaphore type := 1;     { กำหนดค่าเริ่มต้นเท่ากับหนึ่ง }
procedure semaphore-user;
        wait (sem);
        access share resource
        signal (sem);
end;
```

เมื่อมีกระบวนการหนึ่งต้องการเข้าประมวลผลเพื่อใช้ทรัพยากรร่วม จะต้องตรวจสอบว่ามีกระบวนการอื่นใช้อยู่ด้วยหรือไม่ โดยคำสั่ง wait จะทำการตรวจสอบค่า sem หากพบว่าไม่มีกระบวนการอื่นใช้งานอยู่ ก็จะเซตค่า sem ลดลงเหลือศูนย์ เพื่อเป็นสัญญาณให้กระบวนการอื่นทราบว่ามีการประมวลผลอยู่ จนกระทั่งประมวลผลเสร็จจึงจะให้ signal เพื่อเพิ่มค่าเซมาฟอร์เป็นหนึ่ง และปลดปล่อยการครอบครองทรัพยากรร่วม เพื่อให้กระบวนการอื่นที่รอใช้ทรัพยากรร่วมอยู่สามารถเข้าประมวลผลได้

- เขตวิกฤติ (critical region)

อาณาเขตวิกฤต คือบริเวณ หรือส่วนของโปรแกรมที่เข้าใช้ทรัพยากรร่วม ซึ่งจำเป็นจะต้องกำหนดให้กระบวนการนี้ดำเนินการให้เสร็จสิ้นโดยกระบวนการอื่นจะเข้ามาก้าวก่ายไม่ได้ ดังเช่นการใช้คำสั่ง wait และ signal ในเซมาฟอร์ แต่การใช้คำสั่งประสานจังหวะอาจเกิดการผิดพลาดได้ง่ายเนื่องจากตัวแปรเซมาฟอร์ไม่แตกต่างไปจากตัวแปรอื่น การกำหนดเขตวิกฤตเพื่อทำให้โปรแกรมเป็นโครงสร้างมากขึ้น

```
var sem : semaphore type;                    { create semaphore }
region sem do                          {declaration critical section }
begin
        critical region
end;
```

เมื่อคอมไพล์เลอร์พบคำสั่ง region จะแทรกคำสั่งในการประสานจังหวะ wait และ signal ไปในตำแหน่งที่เหมาะสม จะได้ผลลัพท์ดังนี้

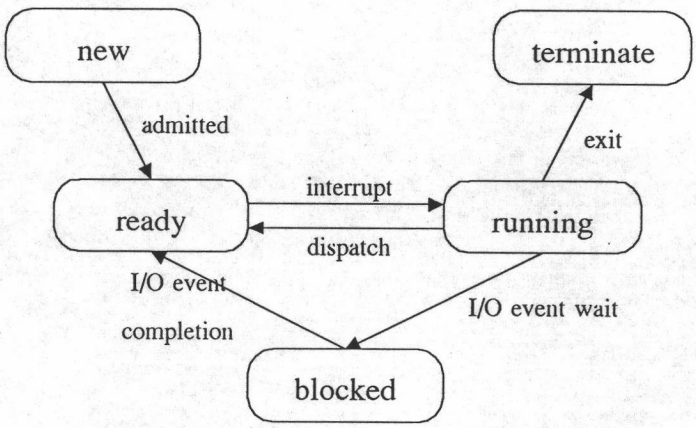
```
wait (sem);
critical region;
signal (sem);
```

### 2.2.5 การจัดการกระบวนการ (Process scheduler)

หน่วยประมวลผลกลาง (control processor unit, CPU) เป็นทรัพยากรประเภทหนึ่งที่อยู่ในระบบ ซึ่งอาจจะมีกระบวนการหลายกระบวนการต้องการเข้าครองครองเพื่อปฏิบัติการ (execution) แต่กระบวนการจะสามารถเข้าครอบครองซีพียูได้เพียงครั้งละหนึ่งกระบวนเท่านั้นดังนั้นจึงจำเป็นต้องกำหนดกลวิธีเพื่อจัดการให้กระบวนการต่างๆใช้ตัวประมวลผลได้อย่างมีประสิทธิภาพ โปรแกรมที่ทำหน้าที่กำหนดให้กระบวนการต่างๆเข้าประมวลผล เรียกว่าโปรแกรมกำหนดงาน (scheduler)

- การจัดคิวแบบมาก่อนได้ก่อน (First Come First Serve, FCFS)

ในการจัดคิวแบบต่างๆการจัดคิวแบบมาก่อนได้ก่อนเป็นการจัดคิวแบบง่ายที่สุด โดยจะจัดลำดับกระบวนการที่มีสถานะพร้อม (ready state) ที่มาถึงก่อนให้เข้าใช้ซีพียูก่อน และจะครอบครองซีพียูจนกระทั่งกระบวนการนั้นประมวลผลเสร็จสิ้น ซึ่งไม่กำหนดระยะเวลาในการประมวลผล เมื่อกระบวนการแรกทำงานเสร็จสิ้นแล้ว กระบวนการนี้จะเป็นสถานะเป็นสถานะจบ (terminate state) และจะปลดปล่อยซีพียูเพื่อให้กระบวนการอื่นสามารถเข้าใช้ซีพียูได้ อย่างไรก็ตามหากในระหว่างการประมวลผล กระบวนการมีการเรียกใช้งานอุปกรณ์อินพุท-เอาท์พุท (input-output device) หรือเกิดการรอเหตุการณ์บางอย่างกระบวนการนั้นจะต้องปลดปล่อยซีพียูและเปลี่ยนจากสถานะดำเนินการ (run~state) ไปสู่สถานะติดขัดและกระบวนการนี้จะย้ายไปต่อท้ายคิวเพื่อให้กระบวนการอื่นที่อยู่ในสถานะพร้อมเข้าใช้ซีพียูต่อไป




รูปที่ 2.7 แสดงสถานะของกระบวนการ

การจัดคิวแบบมาก่อนได้ก่อนมีข้อดีคือสามารถจัดคิวได้ง่ายไม่ยุ่งยากซับซ้อน แต่ถ้าหากมีกระบวน การใดใช้เวลามากและได้ประมวลผลก่อน จะทำให้เกิดผลเสียกับกระบวนการอื่นๆ มากเพราะจะเกิดเวลารอคอย (wait time) มากเช่น

| กระบวนการ | เวลาในการประมวลผล (msec) |
|---|---|
| P1 | 14 |
| P2 | 5 |
| P3 | 1 |
| P4 | 5 |

หากกระบวนการที่อยู่ในลำดับคือ P1 P2 P3 P4
เวลารอโดยเฉลี่ยคือ (0+14+19+20) / 4 = 13.25 msec.

หากกระบวนการที่อยู่ในลำดับคือ P2 P3 P4 P1
เวลารอโดยเฉลี่ยคือ (0+5+6+11) / 4 = 5.5 msec.

จะพบว่าหากกระบวนการที่ใช้เวลาในการประมวลผลนานเข้าประมวลผลก่อนจะทำให้เกิดเวลารอคอยนานกว่ากรณีที่กระบวนการที่ใช้เวลาน้อยเข้าประมวลผลก่อน

- การจัดคิวแบบ Shortest Job First (SJF)

การคัดเลือกกระบวนการด้วย SJF นี้จะเลือกโดยพิจารณาจากกระบวนการที่ใช้เวลาในการประมวลผลน้อยที่สุด จะได้ประมวลผลก่อนกระบวนการที่ใช้เวลาประมวลผลมาก หากกระบวนการสองกระบวนการซึ่งมีเวลาในการประมวลผลเท่ากัน จะพิจารณาจากการจัดคิวแบบมาก่อนได้ก่อนเช่น~

| กระบวนการ | เวลาในการประมวลผล (msec) |
|---|---|
| P1 | 14 |
| P2 | 5 |
| P3 | 1 |
| P4 | 5 |

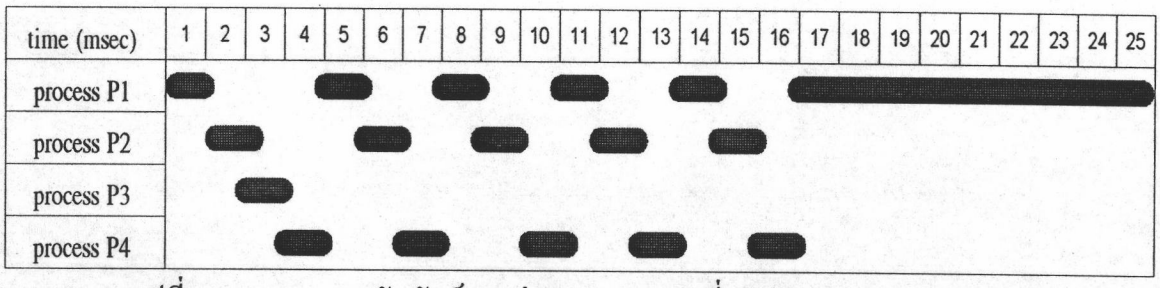
ลำดับในการประมวลผลคือ P3 P2 P4 P1

เวลาในการรอเฉลี่ยคือ (0+1+6+11) / 4 = 4.5 msec.

สำหรับการจัดคิวแบบ SJF เป็นการจัดคิวที่ทำให้เวลาในการรอคอยโดยเฉลี่ยสั้นที่สุด ทั้งนี้เป็นเพราะเวลาที่ใช้ในการรอคอยจะเป็นเวลาที่ใช้ในการประมวลผลของกระบวนการก่อนหน้าซึ่งจะเป็นเวลาในการประมวลผลที่สั้นที่สุดตามลำดับ

- การจัดคิวแบบวนรอบ (Round robin ,RR)

การจัดคิวแบบวนรอบจะคล้ายกับการจัดคิวแบบมาก่อนได้ก่อนแต่จะแก้ปัญหาของการรอคอยคือการจัดคิวแบบวนรอบจะกำหนดให้มีการหมุนเวียนของการครอบครองซีพียู กระบวนการที่อยู่ในสถานะดำเนินการ (run state) จะถูกจำกัดให้ทำงานภายในระยะเวลาหนึ่ง เรียกว่าเวลาควอนตัม (quantum time) กระบวนการต่างๆที่อยู่ในสถานะพร้อมจะหมุนเวียนมาดำเนินการครอบครองซีพียูในช่วงเวลาที่กำหนด เมื่อหมดระยะเวลาควอนตัมจะเปลี่ยนสถานะไปสู่สถานะพร้อม~เพื่อให้กระบวนการอื่นในลำดับการรอคอยเข้ามาใช้ซีพียูต่อไป การจัดคิวแบบวนรอบจะแก้ปัญหาการรอคอยนานของกระบวนการที่ใช้เวลาในการประมวลผลน้อยๆได้




รูปที่ 2.8 แสดงความสัมพันธ์ระหว่างกระบวนการที่ถูกประมวลผลกับเวลา

| ~กระบวนการ | เวลาที่ต้องการใช้ซีพียู | เวลาที่ต้องรออยู่ในคิว | เวลาที่ทำงานเสร็จ |
|---|---|---|---|
| P1 | 14 | 11 | 25 |
| P2 | 5 | 10 | 15 |
| P3 | 1 | 2 | 3 |
| P4 | 5 | 11 | 16 |

เวลาในการรอเฉลี่ยคือ (11+10+2+11) / 4 = 8.5 msec

ดังนั้นอาจสรุปได้ว่าการจัดคิววนรอบจะเป็นผลดีกับกระบวนการที่ใช้เวลาในการประมวลผลน้อย เพราะจะทำให้เวลาในการรอลดลงแต่จะทำให้กระบวนการที่ใช้เวลาในการประมวลผลมากมีระยะเวลาในการรอคอยเฉลี่ยมากขึ้น ทั้งนี้เป็นเพราะกระบวนการที่ใช้เวลาในการประมวลผลมาก(เมื่อเทียบกับเวลาควอนตัม)จะต้องวนกลับเข้ามาสู่สถานะพร้อมหลายครั้ง

- การจัดคิวแบบลำดับความสำคัญ (Priority queue)

ในการจัดคิวแบบลำดับความลำคัญกระบวนการต่างๆจะประกอบด้วยเลขลำดับความสำคัญ (Priority number) ซึ่งจะมีค่าอยู่ในช่วงที่กำหนด เช่นเลขลำดับความสำคัญมีค่าระหว่าง 0 ถึง 127 กระบวนการที่มีเลขลำดับความสำคัญน้อย จะถูกประมวลผลก่อนกระบวนการที่มีเลขลำดับความสำคัญมาก หากกระบวนการสองกระบวนการมีเลขลำดับความสำคัญเท่ากันจะใช้การจัดคิวแบบมาก่อนได้ก่อน

| กระบวนการ | เลขลำดับความลำคัญ | เวลาในการประมวลผล (msec) |
|---|---|---|
| P1 | 1 | 10 |
| P2 | 2 | 5 |
| P3 | 4 | 6 |
| P4 | 3 | 7 |
| P5 | 5 | 2 |

เวลาในการรอคอยโดยเฉลี่ยคือ (0+10+15+17+23) / 5= 13 msec.

ปัญหาของการจัดคิวแบบลำดับความสำคัญคือ กระบวนการที่มีเลขลำดับความสำคัญน้อยสามารถแทรกเข้าประมวลผลก่อนกระบวนการที่มีเลขลำดับความสำคัญมากได้ หากมีกระบวนการที่มีเลขความสำคัญน้อยมาแทรกกระบวนการที่มีเลขความสำคัญมากตลอดเวลา จนทำให้กระบวนนั้นๆไม่สามารถเข้าครอบครองซีพียูได้เลย เราเรียกปรากฏการณ์นี้ว่า Indefinite blocking หรือ starvation การป้องกันการเกิดเหตุการณ์เช่นนี้อาจทำได้โดย กำหนดอายุของกระบวนการให้สัมพันธ์กับลำดับความสำคัญ เช่น ในช่วงระยะเวลาหนึ่งๆหากกระบวนใดยังไม่ถูกดำเนินการเราจะลดเลขลำดับความสำคัญลง เช่น ถ้าเลขลำดับความสำคัญมีค่าระหว่าง 0 ถึง 127 และจะลดค่าเลขลำดับความสำคัญลงด้วยค่า 1 ในทุกๆ 30 นาทีหากกระบวนการอยู่ในสถานะพร้อมนาน เลขลำดับความสำคัญจะลดลงเรื่อยๆ จนกระทั่งเลขลำดับความลำดับความสำคัญลดลงจนเหลือ 0 นั้นคือมีความสำคัญมากที่สุดก็จะถูกประมวลผล ด้วยขบวนการดังกล่าว

เราจะประกันได้ว่า กระบวนการทุกๆกระบวนการ จะถูกประมวลผลในช่วงเวลาไม่เกิน (30 นาที x 128ช่วง) เท่ากับ 64 ชั่วโมงนั้นคือไม่มีกระบวนการใดรอคอยเกินกว่า 64 ชั่วโมง

- การจัดคิวแบบหลายระดับ (Multi level scheduling)

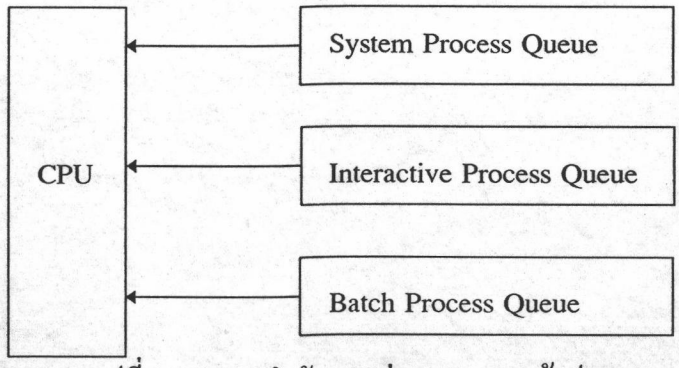
การจัดคิวทั้งสี่แบบที่ได้กล่าวมาแล้วข้างต้นเป็นการจัดคิวในระดับเดียวเท่านั้น เพื่อเพิ่มประสิทธิภาพในการจัดคิวให้มากยิ่งขึ้น โดยเฉพาะในระบบคอมพิวเตอร์ขนาดใหญ่จะต้องมีการจัดลำดับความสำคัญของงานต่างๆไว้ไม่เท่ากัน โดยอาจแบบเป็นกลุ่มได้ดังนี้

-กระบวนการระบบ (system process)

-กระบวนการแบบโต้ตอบ (interactive process)

-กระบวนการแบบกลุ่ม (batch process)

กระบวนการระบบจะมีความสำคัญสูงกว่ากระบวนการแบบโต้ตอบและสูงกว่ากระบวนการในโหมดกลุ่ม ซึ่งในคิวต่างๆ อาจเป็นการจัดคิวแบบใดก็ได้เช่นในกระบวนการระบบ อาจเป็นการจัดคิวแบบลำดับความสำคัญ ในกระบวนการแบบโต้ตอบอาจเป็นการจัดคิวแบบวนรอบก็ได้

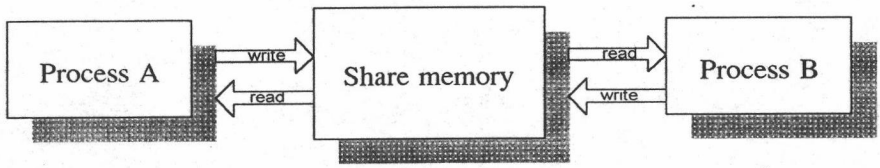



รูปที่ 2.9 แสดงลำดับงานต่างๆขณะรอเข้าประมวลผล

### 2.2.6 การติดต่อระหว่างกระบวนการ (Interprocess-Communication)

โดยปกติกระบวนการต่างๆที่อยู่ในระบบจะทำงานอย่างอิสระต่อกัน โดยไม่รับรู้ความเป็นไปของกระบวนการอื่น แต่ในกรณีที่กระบวนการใดต้องการข้อมูลจากกระบวนการอื่น การที่กระบวนการต่างๆมีการติดต่อกันเช่นนี้เรียกว่าการติดต่อระหว่างกระบวนการ การติดต่อระหว่างกระบวนการทำได้สองวิธีคือ

- การใช้หน่วยความจำร่วม (share memory) เป็ น การติดต่อกันทำได้โดยการนำข้อมูลที่ต้องการติดต่อไปไว้ในหน่วยความจำส่วนหนึ่ง ที่กำหนดให้เป็นหน่วยความจำในการรับส่งข้อมูลระหว่างกระบวนการ หากกระบวนการใดต้องการรับข้อความก็สามารถรับข้อความที่ต้องการติดต่อได้




รูปที่ 2.10 หน่วยความจำร่วม

- ระบบส่งข่าวสาร (message system) เทคนิคหนึ่งที่ใช้ในการติดต่อระหว่างกระบวนการคือระบบการส่งข่าวสาร (message-passing) เทคนิคนี้เป็นการติดต่อระหว่างกระบวนการโดยไม่ใช้หน่วยความจำร่วม แต่จะใช้ตัวดำเนินการอย่างน้อยสองคำสั่งในการติดต่อ คือ send (message) และ receive (message) ถ้ากระบวนใดๆต้องการรับส่งข่าวสารจะต้องมีการกำหนดชื่อผู้รับ-ผู้ส่งอย่างชัดเจนการกำหนดชื่อแบ่งได้ออกเป็นสองแบบคือ

1. การอ้างชื่อโดยตรง (direct communication) เป็นการกำหนดชื่อกระบวนการที่เป็นผู้รับส่งอย่างชัดเจนเช่น

send ( P, message) คือส่งข่าวสารไปให้กระบวนการ P

receive( Q, message) คือการรับข่าวสารจากกระบวนการ Q

2. การอ้างชื่อโดยอ้อม (indirect communication) เป็นการติดต่อที่ไม่ได้กำหนดผู้รับ-ส่งโดยตรง แต่จะอ้างถึงตู้ไปรษณีย์ (mail box) ที่จะรับ-ส่งแทนเช่น

send ( A, message) คือการส่งข่าวสารไปที่ mail box A

receive( A, message) คือการรับข่าวสารจาก mail box A

เมื่อผู้ส่งต้องการส่งข้อความก็จะส่งข้อความไปเก็บไว้โดยระบุชื่อของตู้ไปรษณีย์ ซึ่งในขณะที่ยังไม่มีผู้รับก็จะยังคงเก็บไว้ที่ตู้ไปรษณีย์นี้ เมื่อมีกระบวนการใดต้องการรับข้อความ ผู้รับจะต้องรับโดยอ้างอิงจากชื่อของตู้ไปรษณีย์ เช่น receive (A,message) หมายถึงรับข้อความจากตู้ไปรษณีย์ A ด้วยวิธีการเช่นนี้เราสามารถติดต่อกันได้มากกว่าสองกระบวนการ เพราะว่าไม่จำเป็นต้องอ้างอิงโดยชื่อกระบวนการของผู้รับผู้ส่ง

เนื่องจากระบบส่งข่าวสาร ไม่จำเป็นต้องใช้ตัวแปรร่วม แต่จะมีช่องทาง (channel) เป็นทรัพยากรร่วมที่กระบวนการต่างๆใช้ร่วมกัน แต่ตัวแปรต่างๆที่ใช้ในกระบวนการจะเป็นตัวแปรท้องถิ่นทั้งหมด และระบบนี้ไม่จำเป็นต้องมีกลไกพิเศษเพื่อป้องกันการช่วงชิงทรัพยากร (mutual exclusion) การที่กระบวนการต่างๆไม่จำเป็นต้องใช้ตัวแปรร่วมจะเป็นประโยชน์อย่างยิ่งในการออกแบบโปรแกรมแบบกระจาย (distributed program) การควบคุมการทำงานของกระบวนการต่างๆ จะใช้วิธีส่งข่าวสารจากกระบวนการหนึ่งไปสู่กระบวนการหนึ่ง ซึ่งกระบวนการต่างๆอาจจะประมวลผลโดยโปรเซสเซอร์ที่แยกจากกันในระบบหลายโปรเซสเซอร์(multi-processor) ก็ได้

- Asynchronous message passing

การส่งข่าวสารในระบบ Asynchronous message passing เมื่อข่าวสารถูกส่งเข้าไป เราจะสมมติว่ามีบัฟเฟอร์ (buffer) ที่ใช้ในการเก็บโดยไม่จำกัด เมื่อกระบวนการหนึ่งส่งข้อความไปสู่กระบวนการหนึ่งโดยผ่านช่องทาง (channel) ที่กำหนด ข้อมูลที่ส่งจะเก็บไว้ในบัฟเฟอร์ และกระบวนการผู้ส่งก็สามารถประมวลผลต่อไป เมื่อกระบวนการผู้รับถูกประมวลผลคำสั่งรับข่าวสาร กระบวนการผู้รับจะอ่านข้อมูลจากบัฟเฟอร์ที่เก็บไว้ ดังนั้นในการส่วนแบบนี้ทำให้ไม่เกิดการบล็อกของกระบวนการผู้ส่ง

ตัวอย่างเช่น หากกระบวนการ A ต้องการส่งข่าวสารไปให้กระบวนการ B ข้อความจะส่งไปเก็บไว้ที่บัฟเฟอร์ และกระบวนการ A ก็สามารถประมวลผลต่อไปโดยไม่จำเป็นต้องให้กระบวนการ B มารับข้อมูล เมื่อกระบวนการ B ประมวลผลคำสั่งรับข่าวสารก็จะอ่านข้อมูลในจากบัฟเฟอร์ที่เก็บเอาไว้

ในกรณีกลับกัน หากกระบวนการ B ได้ประมวลผลก่อน เมื่อกระบวนการ B อ่านข้อมูลจากบัฟเฟอร์จะพบว่ายังไม่มีข่าวสารส่งมาถึง ดังนั้นกระบวนการ B จะถูกบล็อก จนกระทั่งกระบวนการ A ประมวลผลและส่งข่าวสารไปให้กระบวนการ B กระบวนการ B จึงสามารถประมวลผลต่อไปได้

- Synchronous message passing

จากระบบ asynchronous message passing จะกำหนดให้มีบัฟเฟอร์ในการรับ-ส่งข่าวสาร แต่จะพบว่าหากต้องการตรวจสอบว่าการรับส่งข่าวสารทำได้ได้ถูกต้อง อาจทำได้โดยถ้ากระบวนการ A ส่งข่าวสารถึงกระบวนการ B เมื่อกระบวนการ B รับข้อมูลได้แล้วก็จะส่งข้อความกลับมาเพื่อยืนยันการรับข่าวสาร

| Process A | Process B | comment |
|---|---|---|
| send ( B , message) | | ส่งข้อมูลไปให้ B |
| | receive ( A , message) | รับข้อมูลจาก A |
| | send ( A , ReceiveOK) | ส่งคำยืนยันว่ารับข้อมูลได้แล้ว |
| receive ( B , confirm) | | รับคำยืนยัน |

ในกรณีที่กระบวนการ A ส่งข้อมูลไปแล้ว แต่ไม่ได้รับการยืนยันจากกระบวนการ B จะเกิดปัญหาขึ้นว่า เกิดความผิดพลาดจากการส่งข้อมูลไปไม่ถึงผู้รับ หรืออาจเกิดจากความผิดพลาดในการส่งข้อความยืนยันจากกระบวนการ B ก็ได้

นอกจากนี้ในทางปฏิบัติ ในการออกแบบบัฟเฟอร์จะพบว่าบัฟเฟอร์จะมีขนาดจำกัด ซึ่งหากข่าวสารที่ส่งไปมีจำนวนมากจะทำให้บัฟเฟอร์เต็มได้ซึ่งจะทำให้เกิดการบล็อกในกระบวนการผู้ส่งได้

ในระบบ synchronous message passing จะหลีกเลี่ยงปัญหาที่เกิดขึ้นข้างต้นโดยการกำหนด

process A :: B ! e

process B :: A ? x

โดยที่

B ! e คือ output statement

A ? x คือ input statement

สเตตเมนต์ทั้งสองนี้เรียกว่า match statement เมื่อกระบวนการ A ต้องการส่งข่าวสาร e ให้กระบวนการ B การส่งข่าวสารจะสำเร็จลุล่วงได้เมื่อกระบวนการ A และกระบวนการ B ถูกประมวลผลพร้อมกัน หากกระบวนการได้มาถึง match statement ก่อนอีกกระบวนการหนึ่ง จะต้องรอจนกว่ากระบวนการทั้งสองมาถึง match statement พร้อมกัน