**ปฏิบัติการที่ 10: Process กับ thread และการโปรแกรมแบบขนาน VI**

**1. อธิบายการทํางานของโปรแกรมนี้ ให้บอกว่าใครทําอะไรก่อน หลัง อะไรที่ทําไปพร้อมกันได้ และมีกลไกอะไรบังคับให้เป็นไปตามนี้**

**Ans** การทำงานของ make\_cake สามารถแยกเป็น การทำงาน 3 อย่างใหญ่ๆตามลำดับ ดังนี้

**Ingrediant Preparation ( Egg and Batter Threads )**

--- control by ---

**cond** needIngredients

* โดยในการเตรียม ingrediants สามารถทำได้พร้อมกันโดยไม่ต้องรอกัน ดังนั้น thread batterAdder และ 2 eggBreaker เป็นอิสระต่อกัน
* เมื่อได้วัตภุดิบเพียงพอต่อการ baking a cake ก็จะทำการหยุดทำงานแบบไม่ขึ้นต่อกัน

(2 egg,1 batter)และเข้า queue ของ needIngredients

* ทุกๆครั้งทุก ingredients ทำงานเสร็จก็จะส่ง signal ไปให้ BowlHeater เพื่อตรวจสอบเงื่อนไขว่าสามารถ baking a cake ได้ไหม

**Baking A Cake ( need Ingrediants to fullfill this step )**

--- control by ---

**cond** readyToBake

* cake จะถูกอบได้ก็ต่อเมื่อมีวัตถุดิบครบ (2 egg , 1 batter) ถ้ายังไม่ครบจะถูกเข้า queue ของ readyToBake จนกว่าจะได้ วัตถุดิบครบ
* ทุกครั้งที่ cake ถูกอบออกมาเสร็จจะส่ง signal ไปให้ startEating เพื่อให้กิน cake

**Eating A Cake ( eat the baked cake :D )**

--- control by ---

**cond** startEating

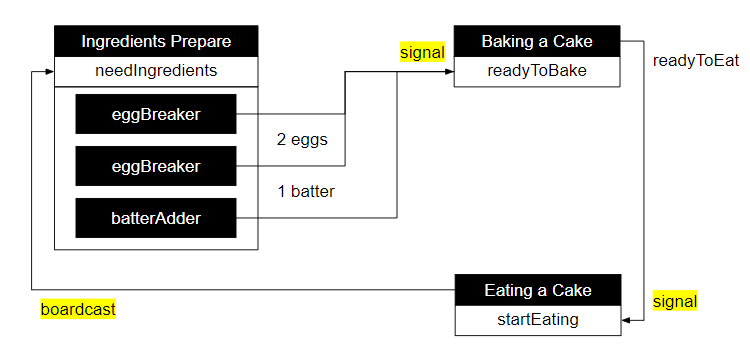
* cake จะถูกกิน ถ้ายังไม่มี cake ให้กิน (readyToEat = 0) ก็จะต่อ queue ของ startEating
* เมื่อกิน cake เสร็จแล้วก็จะ boardcast signal ของ needIngredients เพื่อบอกว่าสามารถทำ ingredients สำหรับ cake ก้อนต่อไปได้เลย

**2. จะเห็นได้ว่าโปรแกรมเรียกใช้ pthread\_mutex\_lock(&lock); โดยไม่มี unlock เลย ทําไมโปรแกรมจึงไม่ deadlock อธิบายว่าใครเป็นคนมาปลด lock นี้ออกทั้งๆที่ไม่มี unlock ที่มาจับคู่กับมันเลย**

**Ans** เพราะการทำงานของ program make\_cake เป็นแบบ cyclic working กล่าวคือเป็นการทำงานเมื่อทำงานของตัวเองเสร็จก็จะส่ง signal ไปให้งานต่อไปทำงานต่อได้ โดย งานจะทำงานต่อได้ก็ต่อเมื่อ condition have been fullfilled ( signal จะส่งไปยัง condition variable ของงานถัดไป )

จากงานดังกล่าวมีการทำงานแบบ cyclic working ดังนี้ (อ้างอิงการแบ่งงานจาก**ข้อ 1**)

**→ Ingrediant Preparation → Baking A Cake → Eating A Cake →**



ตัวที่ปลดล็อค thread คือ wait ในแต่ละ thread เช่นใน batterAdder มีการเช็คว่า ถ้ามีแป้งอยู่แล้วหรือพร้อมที่จะกิน thread จะถูกเข้าไป wait ใน while ที่มี pthread\_cond\_wait(&needIngredients, &lock); อยู่ ซึ่งคือการรอ signal&needIngradients และ unlock &lock นั่นเอง

**3. เขียนโปรแกรมในลักษณะ pseudo code (ในลักษณะคล้ายกับที่เราใช้ตอนเรียนวิชาระบบปฏิบัติการ)ที่ใช้ monitor (การใช้งาน condition variable ร่วมกับ lock) แก้ปัญหาการผลิตรถยนต์ดังต่อไปน**

“ในโรงงานผลิตรถยนต์แห่งหนึ่งจะมีการผลิตรถออกมาจากวัสดุและกระบวนการต่อไปนี้ แชสซี (chasis) ยาง(tires) เบาะนั่ง (seats) เครื่องยนต์ (engine) หลังคารถ (top cover) และการพ่นสี (painting) ดังนั้นจะมี 6 task ในการผลิตรถยนต์แต่ :

• ยาง เบาะนั่ง และเครื่องยนต์ จะยังถูกประกอบเข้ามาไม่ได้ถ้ายังไม่มี แชสซี วางมาบนสายพานประกอบ

• หลังคารถ จะยังถูกประกอบเข้ามาไม่ได้ถ้ายังไม่มี ยาง เบาะนั่ง และเครื่องยนต์

• การพ่นสี จะยังทําไม่ได้ถ้า หลังคารถ ยังไม่ถูกประกอบเข้ามา

มีช่างเทคนิค 4 คน A B C และ D

• A ใส่ยาง และ พ่นสี

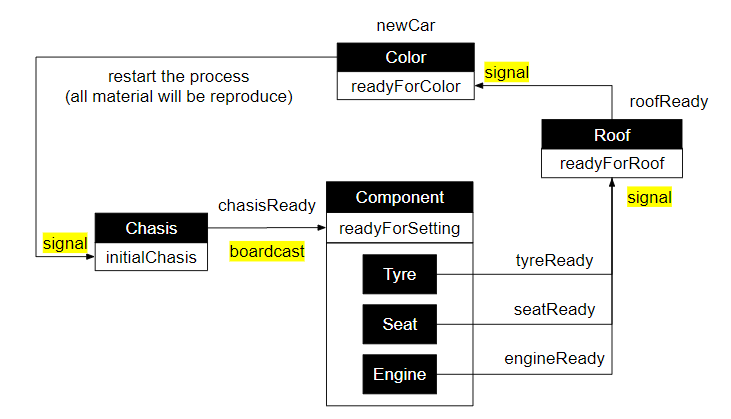
• B ส่ง แชสซี ลงสายพานประกอบ

• C ประกอบ เบาะนั่ง

• D ใส่ เครื่องยนต์ และ หลังคารถ

ใช้ thread 4 ตัว แทนช่างเทคนิคทั้ง 4 คน ให้สามารถผลิตรถยนต์ได้โดยไม่ฝ่าฝืนขั้นตอนการผลิตและบทบาทหน้าที่ของแต่ละคน อย่าลืมอธิบายแนวทางการแก้ปัญหาและให้เหตุผลประกอบความถูกต้องด้วย (อธิบายว่าเราใช้ monitor อะไร เป็นตัวแทนใคร ที่ต้องรออะไรก่อนจะดําเนินกิจกรรมที่ตัวเองได้รับมอบหมาย)”

**Ans**

ใช้หลัก cyclic working ที่ได้อธิบายไปในข้อสอง ที่มี condition variable กำกับว่า flow ตอนนี้อยู่ส่วนไหนของ cycle

**mutex** lock **cond** readyToStart **cond** readyForSetting **cond** readyForRoof

**cond** readyForColor

**Thread A** (tyre setting & spraying color):

monitor\_mutex(&lock);

while ( True ){

while ( !chasisReady || tyreReady ) {

monitor\_cond\_wait( &readyForSetting, &lock );

}

tyreReady = True;

monitor\_cond\_signal( &readyForRoof );

while ( !roofReady ) {

monitor\_cond\_wait( &readyForColor, &lock);

}

// getTheCar !!!

newCar = True

chasisReady = False;

tyreReady = False;

seatReady = False;

engineReady = False;

roofReady = False;

monitor\_cond\_boardcast( &readyToStart );

}

**Thread B** (Chasis transfer):

monitor\_mutex( &lock );

while ( True ){

while ( chasisReady || newCar) {

monitor\_cond\_wait( &readyToStart, &lock )

}

newCar = False;

chasisReady = True;

monitor\_cond\_signal( &readyForSetting );

}

**Thread C** (Seat Setting):

monitor\_mutex(&lock);

while ( True ){

while ( !chasisReady || seatReady ) {

monitor\_cond\_wait( &readyForSetting, &lock )

}

seatReady = True;

monitor\_cond\_signal( &readyForRoof );

}

**Thread D** (Engine Setting and Roof Setting):

monitor\_mutex(&lock);

while ( True ){

while ( !chasisReady || engineReady ) {

monitor\_cond\_wait( &readyForSetting, &lock )

}

engineReady = True;

while ( !tyreReady || !seatReady || roofReady ) {

monitor\_cond\_wait( &readyForRoof, &lock )

}

roofReady = True;

monitor\_cond\_signal( &readyForColor );

}