# ບົດທີ 5

## ການເຮັດວຽກຮ່ວມກັນຂອງຂະບວນການ (Process Synchronization)

# ເນື້ອໃນຫຍໍ້

- ສະເໜີເບື້ອງຕົ້ນ
- ຊັບພະຍາກອນທີ່ບໍ່ສາມາດໃຊ້ພ້ອມກັນ (Mutual Exclusion)
- ພາກສ່ວນວິກິດ (Critical Section)
- ການແກ້ໄຂບັນຫາທາງ software
- ການແກ້ໄຂບັນຫາທາງ hardware
- ສັນຍາລັກໃນການເຮັດວຽກຮ່ວມກັນ (Semaphores)
- ຕົວຢ່າງບັນຫາໃນການເຮັດວຽກຮ່ວມກັນ
- ຕົວສັງເກດການ (Monitors)
- ວົງຈອນປິດຕາຍ (Deadlock)

# ສະເໜີເບື້ອງຕົ້ນ

- ສິ່ງທີ່ສຳຄັນຂອງການອອກແບບລະບົບປະຕິບັດການແມ່ນການຈັດການ Process ແລະ Thread ໃນສະພາບແວດລ້ອມຕໍ່ໄປນີ້
  - Multiprogramming
  - Multiprocessing
  - Distributed Processing
- ຫລັກການພື້ນຖານໃນການອອກແບບລະບົບປະຕິບັດການດັ່ງກ່າວກໍ່ຄື
  - ການຈັດການໃນສະພາບແວດລ້ອມທີ່ຫລາຍຂະບວນການເຮັດວຽກພ້ອມກັນ (concurrency)
  - ການປະສານເວລາ (synchronization)
  - ການສື່ສານລະຫວ່າງຂະບວນການ
  - ການຍາດແຍ່ງແລະການໃຊ້ຊັບພະຍາກອນຮ່ວມກັນ
  - ການປະສານເວລາຂອງກິດຈະກຳທັງໝົດທີ່ເກີດຂຶ້ນລະຫວ່າງຂະບວນການ
  - ການຈັດສັນເວລາ CPU ໃຫ້ແກ່ຂະບວນການເຫລົ່ານັ້ນ

# ສະເໜີເບື້ອງຕົ້ນ

- ການປະຕິບັດງານພ້ອມກັນ
  - ແມ່ນການເຮັດວຽກພ້ອມກັນຂອງບັນດາ process/thread
  - ອາດຈະເປັນການເຮັດວຽກເປັນອິດສະຫຼະຕໍ່ກັນ ຫຼື ຮ່ວມກັນເຮັດວຽກວຽກໃດໜຶ່ງ
  - process/thread ທີ່ເຮັດວຽກເປັນອິດສະລະຕໍ່ກັນ ແຕ່ມີການສື່ສານກັນ ແລະ ເຮັດວຽກຮ່ວມກັນບາງຄັ້ງຄາວເພື່ອຊ່ວຍກັນເຮັດວຽກບາງຢ່າງເອີ້ນວ່າ ການເຮັດວຽກແບບບໍ່ປະສານກັນ (Asynchronous)
  - ການເຮັດວຽກບໍ່ປະສານກັນແມ່ນສະຫຼັບຊັບຊ້ອນ ແລະ ບໍລິຫານຍາກ
  - ການເຮັດວຽກພ້ອມກັນມັກເກີດຂຶ້ນພາຍໄຕ້ສະພາບດັ່ງນີ້:
    - Application ຫຼາຍຊຸດ
    - ໂຄງສ້າງຂອງ Application
    - ໂຄງສ້າງຂອງລະບົບປະຕິບັດການ

# ຊັບພະຍາກອນທີ່ບໍ່ສາມາດໃຊ້ພ້ອມກັນ

- ການທີ່ສອງ process/thread ເຂົ້າໃຊ້ຂໍ້ມູນໃດໜຶ່ງພ້ອມກັນຈະເຮັດໃຫ້ສະຖານະພາບຂອງຂໍ້ມູນບໍ່ຊອດຄ່ອງກັນ
- ດັ່ງນັ້ນ ການເຂົ້າໃຊ້ຂໍ້ມູນທີ່ໃຊ້ຮ່ວມກັນຈະຕ້ອງບໍ່ເຂົ້າໃຊ້ພ້ອມກັນ (Mutual Exclusion)
- Mutual Exclusion
  - ອານຸຍາດໃຫ້ພຽງໜຶ່ງ process/thread ເຂົ້າໃຊ້ໃນເວລາໃດໜຶ່ງ
  - ສ່ວນ process/thread ອື່ນໆຈະຕ້ອງລໍຖ້າຈົນກ່ວາຜູ້ທີ່ເຂົ້າກ່ອນສຳເລັດ
  - ເປັນການເຂົ້າໃຊ້ແບບຕາມລຳດັບ
  - ຕ້ອງໄດ້ຄວບຄຸມເວລາລໍຖ້າບໍ່ໃຫ້ຫຼາຍເກີນໄປ

# ຊັບພະຍາກອນທີ່ບໍ່ສາມາດໃຊ້ພ້ອມກັນ

- ການອະນຸຍາດໃຫ້ຂະບວນການຫຼືຂະບວນການຍ່ອຍເຂົ້າໃຊ້ຂໍ້ມູນຮ່ວມກັນໃນເວລາດຽວກັນ ອາດເປັນສາເຫດໃຫ້ເກີດຄວາມບໍ່ຊອດຄ່ອງກັນຂອງຂໍ້ມູນ ຫຼື ຂໍ້ມູນມີຄວາມບໍ່ຖືກຕ້ອງ
- ວິທີການໃຊ້ໜ່ວຍຄວາມຈຳຮ່ວມກັນເພື່ອແກ້ບັນຫາ Bounded - buffer ແມ່ນການອະນຸຍາດໃຫ້ມີລາຍການທັງໝົດຢູ່ໃນ buffer ໄດ້ໃນເວລາດຽວກັນ, ການແກ້ບັນຫາບໍ່ເປັນເລື່ອງງ່າຍ
- ຕົວຢ່າງ: ການປ່ຽນແປງລະຫັດຜູ້ຜະລິດ-ຜູ້ບໍລິໂພກ ໂດຍການເພີ່ມຕົວປ່ຽນ counter ກຳໜົດໃຫ້ມີຄ່າເລີ່ມຕົ້ນເທົ່າ 0 ແລະ ເພີ່ມຄ່າຂຶ້ນເທື່ອລະ 1 ຄ່າເມື່ອມີການເພີ່ມລາຍການເຂົ້າໄປໃນ buffer ແຕ່ລະຄັ້ງ, ມີການໃຊ້ຂໍ້ມູນຮ່ວມກັນ

# ຊັບພະຍາກອນທີ່ບໍ່ສາມາດໃຊ້ພ້ອມກັນ

- ການເກີດບັນຫາບໍ່ຊອດຄ່ອງກັນກັນ
  - ສົມມຸດວ່າ counter = 5
  - Producer: register1 = counter (counter = 5)
  - Producer: register1 = register1+1 (register1=6)
  - Consumer: register2 = counter (counter = 5)
  - Consumer: register2 = register2-1 (register2=4)
  - Producer: counter = register1 (counter=6)
  - Consumer: counter = register2 (counter=4)

  - ສັງເກດເຫັນວ່າຄ່າ counter ຈາກທັງສອງພາກສ່ວນກໍ່ກົງກັນ

# ພາກສ່ວນວິກິດ (Critical Section)

- ໃນການຄວບຄຸມການເຂົ້າໃຊ້ຊັບພະຍາກອນທີ່ສາມາດໃຊ້ຮ່ວມກັນໂດຍ ຂະບວນການຫຼືຂະບວນການຍ່ອຍສາມາດເຮັດໄດ້ໂດຍການກຳໜົດໃຫ້ແຕ່ ລະຂະບວນການມີລະຫັດຕ໊ອໜຶ່ງທີ່ຊື່ວ່າ Critical Section
- ແມ່ນຂໍ້ຕົກລົງຮ່ວມກັນຂອງບັນດາຂະບວນການຕ່າງໆທີ່ໃຊ້ໃນການປະສານ ງານກັນ ແລະ ມີຄຸນລັກສະນະດັ່ງນີ້:
  - ການບໍ່ເຮັດພ້ອມກັນ (Mutual Exclusion)
    - ເປັນການອານຸຍາດໃຫ້ພຽງເທຣດດຽວເທົ່ານັ້ນເຂົ້າໄປໃຊ້ຊັບພະຍາກອນດັ່ງກ່າວໃນເວລາໃດ ໜຶ່ງ
  - ການສືບຕໍ່ປະຕິບັດ (Progress)
    - ຖ້າບໍ່ມີເທຣດໃດຢູ່ໃນສ່ວນວິກິດ ແລະ ຍັງມີເທຣດຈຳນວນໜຶ່ງລໍຖ້າຢູ່ແມ່ນອານຸຍາດໃຫ້ເຂົ້າ ໃຊ້ໄດ້ທັນທີ
  - ການລໍຖ້າແບບມີກຳໜົດ (Bounded-waiting)
    - ຖ້າມີເທຣດໃດໜຶ່ງຂໍຮ້ອງເຂົ້າໃຊ້ສ່ວນວິກິດ ຈະຕ້ອງໄດ້ຮັບການຕອບສະໜອງໃຫ້ໄວທີ່ສຸດ ເທົ່າທີ່ເປັນໄປໄດ້

# ການແກ້ໄຂບັນຫາທາງ software

- Dekker ໄດ້ພັດທະນາ Algorithm ຂຶ້ນມາເພື່ອແກ້ໄຂບັນຫາການໃຊ້ຊັບພະຍາກອນຮ່ວມກັນ
- Algorithm ທຳອິດ
  - ສາມາດບັງຄັບບໍ່ໃຫ້ເຂົ້າໃຊ້ຊັບພະຍາກອນຮ່ວມໃນເວລາດຽວກັນ
  - ໃຊ້ຕົວປ່ຽນຄວບຄຸມວ່າຂະບວນການໃດຈະໄດ້ເຂົ້າໃຊ້ໃນເວລາໃດໜຶ່ງ
  - ໄດ້ກວດສອບວ່າຊັບພະຍາກອນຮ່ວມຫວ່າງຫຼືບໍ່ເປັນປະຈຳ
  - ເສຍເວລາລໍຖ້າ
  - ໜ່ວຍປະມວນຜົນໄດ້ເຮັດວຽກຫຼາຍຂຶ້ນ
  - ເກີດເຫດການທີ່ວ່າ ຂະບວນການທີ່ເຮັດວຽກໄວຈະຕ້ອງລໍຖ້າຂະບວນການທີ່ເຮັດວຽກຊ້າ

# ການແກ້ໄຂບັນຫາທາງ software

- Algorithm ທີ່ສອງ
  - ສາມາດແກ້ໄຂບັນຫາຂະບວນການທີ່ເຮັດວຽກໄວລໍຖ້າຂະບວນການທີ່ເຮັດວຽກຊ້າ
  - ບໍ່ໄດ້ຫຼັກການ Mutual Exclusion
  - ເປັນວິທີທີ່ບໍ່ເໝາະສົມ
- Algorithm ທີ່ສາມ
  - ກຳໜົດສັນຍາລັກກ່ອນຈະເຂົ້າໃຊ້ Critical Section ຊຶ່ງຮັບປະກັນຫຼັກການ Mutual Exclusion
  - ອາດຈະເຮັດໃຫ້ເກີດເຫດການປິດຕາຍ (Deadlock) ເຊັ່ນວ່າ:
    - ສອງຂະບວນການອາດກຳໜົດສັນຍາລັກພ້ອມກັນ
    - ຫຼື ອາແມ່ນເຫດການທີ່ບໍ່ສາມາດອອກຈາກການເຮັດວຽກອົ້ນຊ້ຳໄດ້

# ການແກ້ໄຂບັນຫາທາງ software

- Algorithm ທີ່ສີ່
  - ກຳໜົດສັນຍາລັກໃຫ້ເປັນຄ່າ false ໃນຊ່ວງເວລາສັ້ນໆເພື່ອຄວບຄຸມ
  - ສາມາດແກ້ໄຂບຫາໃນອາວກໍຣິດທີ່ມຜ່ານມາ ແຕ່ມີບັນຫາ ອາດຈະເຮັດໃຫ້ບາງ ຂະບວນການໄດ້ລໍຖ້າຕະຫຼອດໄປ
  - ບໍ່ແທດເໝາະກັບວຽກທີ່ສຳຄັນບາງຢ່າງ ຫຼື ວຽກທຸລະກິດທີ່ມີຄວາມແໜ່ນຢຳ
- Algorithm ທີ່ຫ້າ
  - ເປັນວິທີທີ່ເໝາະສົມ
  - ໃຫ້ຂະບວນການທີ່ໄດ້ຮັບສິດພິເສດເຂົ້າໃຊ້ Critical Section
  - ບໍ່ມີບັນຫາຄັດແຍ່ງກັນວ່າຂະບວນການໃດຈະໄດ້ເຂົ້າກ່ອນ
  - ແຕ່ລະຂະບວນການເຂົ້າໃຊ້ Critical Section ໃນຊ່ວງເວລາໃດໜຶ່ງ
  - ສິດພິເສດແມ່ນໄດ້ປ່ຽນໃຫ້ແຕ່ລະຂະບວນການເປັນລຳດັບ
  - ຮັບປະກັນການເຂົ້າຕາມລຳດັບ ແລະ ບໍ່ມີບັນຫາເຫດການປິດຕາຍ ແລະ ລໍຖ້າຕະລອດ

# ການແກ້ໄຂບັນຫາທາງ hardware

- ການແກ້ໄຂບັນຫາ Mutual Exclusion ໂດຍໃຊ້ hardware
  - ເຮັດໃຫ້ປະສິດທິພາບຂອງລະບົບດີຂຶ້ນ
  - ປະຢັດເວລາ
- ໃຊ້ຫຼັກການ Disabling interrupts
  - ໃຊ້ຢູ່ໃນ ລະບົບໜ່ວຍປະມວນຜົນດຽວ
  - ບໍ່ໃຫ້ມີສັນຍານ interrupts ລົບກວນ ປ້ອງກັນໃຫ້ຂະບວນການທີ່ກຳລັງ ເຮັດວຽກເຮັດຈົນສຳເລັດ
  - ອາດຈະເຮັດໃຫ້ເກີດການປົດຕາຍ
  - ເປັນວິທີທີ່ບໍ່ຄອມນິຍົມໃຊ້

# ການແກ້ໄຂບັນຫາທາງ hardware

- Test-and-Set Instruction
  - ເປັນການໃຊ້ຄຳສັ່ງພາສາເຄື່ອງເພື່ອຮັບປະກັນການເຮັດວຽກເປັນລຳດັບ
  - ຄຳສັ່ງດັ່ງກ່າວຈະຕ້ອງເຮັດວຽກອັນໃດອັນໜຶ່ງເທົ່ານັ້ນ (atomic)
  - ສະເພາະຄຳສັ່ງພາສາເຄື່ອງຈະບໍ່ຮັບປະກັນ Mutual Exclusion ຊອບແວຣ໌ຈະຕ້ອງໃຊ້ມັນຢ່າງເໝາະສົມ
  - testAndSet(a, b) ສຳເນົາເອົາຄ່າຂອງ b ໄປເກັບໄວ້ໃນ a, ແລ້ວ sets b ເປັນ true
  - ຕົວຢ່າງຂອງ atomic ແມ່ນຮອບວຽນຂອງ read-modify-write (RMW)

# ການແກ້ໄຂບັນຫາທາງ hardware

- Swap Instruction
  - swap(a, b) ເປັນການສັບປ່ຽນຄ່າລະຫວ່າງ a ແລະ b ເທື່ອລະຄູ່
  - ຄ້າຍຄືກັບຟັງຊັ່ນ test-and-set
  - swap ແມ່ນຖືກໃຊ້ງານຫຼາຍເທິງ architectures ຕ່າງໆ

# ສັນຍາລັກໃນການເຮັດວຽກຮ່ວມກັນ

- Semaphores
  - ເປັນຊອບແວຣ໌ທີ່ໃຊ້ເພື່ອເປັນສັນຍາລັກໃນການປະຕິບັດຫຼັກການ mutual exclusion
  - ມີຕົວປ່ຽນຈຳນວນຖ້ວນທີ່ໃຊ້ປ້ອງກັນການເຂົ້າໃຊ້ຊັບພະຍາກອນຮ່ວມພ້ອມກັນ
  - ສາມາດເຂົ້າໃຊ້ໄດ້ຜ່ານທາງຄຳສັ່ງ wait ແລະ signal ຊຶ່ງສາມາດເອີ້ນວ່າ P ແລະ V ຕາມລຳດັບ

```
wait (S) :
        while S>o do no-op;
        S--;
Signal (S) :
        S++;
```

  - ການແກ້ປ່ຽນແປງຈຳນວນຖ້ວນຂອງຄຳສັ່ງ wait ແລະ signal ຈະຕ້ອງເປັນແບບ atomic

# ສັນຍາລັກໃນການເຮັດວຽກຮ່ວມກັນ

- Semaphores
  - ໃນລະບົບປະຕິບັດການໄດ້ແບ່ງ Semaphore ອອກເປັນ 2 ຊະນິດ
    - ແບບ Binary
    - ແບບນັບ
  - ແບບ Binary ສາມາດໃຊ້ຄວບຄຸມການເຂົ້າໃຊ້ Critical Section ສຳຫລັບຂະບວນການ ແລະ ຂະບວນການຍ່ອຍ ໂດຍຈະມີພຽງຂະບວນການດຽວເທົ່ານັ້ນທີ່ສາມາດເຂົ້າໃຊ້ Critical Section ໄດ້ໃນເວລາໃດໜຶ່ງ
  - ແບບນັບ ສາມາດໃຊ້ຄວບຄຸມການເຂົ້າໃຊ້ຊັບພະຍາກອນທີ່ມີຈຳ ນວນຈຳກັດເທົ່ານັ້ນ ໂດຍ semaphore ຈະກຳໜົດຄ່າຂອງຈຳນວນຊັບພະຍາກອນທີ່ສາມາດໃຊ້ງານໄດ້. ແຕ່ລະຂະບວນການທີ່ເຂົ້າໃຊ້ຊັບພະຍາກອນຈະເອີ້ນໃຊ້ຄຳສັ່ງ P ເພື່ອລຸດຄ່າຈຳນວນຊັບພະຍາກອນ ແລະ ເມື່ອໃຊ້ແລ້ວກໍ່ຈະເອີ້ນໃຊ້ຄຳສັ່ງ V ເພື່ອເພີ່ມຄ່າ

# ສັນຍາລັກໃນການເຮັດວຽກຮ່ວມກັນ

- Semaphores
  - ແບບ Binary ຈະອະນຸຍາດໃຫ້ພຽງຂະບວນການດຽວເທົ່ານັ້ນທີ່ສາມາດເຂົ້າໃຊ້ Critical Section ໄດ້ໃນເວລາໃດໜຶ່ງ
    - ຄຳສັ່ງ Wait
      - ເມື່ອບໍ່ມີເທຣດອື່ນຢູ່ໃນ Critical Section ແມ່ນໃຫ້ມັນເຂົ້າໃຊ້ໄດ້
      - ລຸດຄ່າ Semaphore ລົງໜຶ່ງຄ່າ ເປັນ 0
      - ຖ້າບໍ່ດັ່ງນັ້ນແມ່ນໃຫ້ໄປລໍຖ້າຢູ່ໃນຄິວ
    - ຄຳສັ່ງ Signal
      - ເປັນການບອກໃຫ້ຮູ້ວ່າມີເທຣດໃດໜຶ່ງອອກຈາກ Critical Section ແລ້ວ
      - ເພີ່ມຄ່າ Semaphore ຂຶ້ນໜຶ່ງຄ່າ ຈາກ 0 ເປັນ 1
      - ຖ້າມີເທຣດກຳລັງລໍຖ້າຢູ່ແມ່ນສາມາດໃຫ້ມັນເຂົ້າໃຊ້ Critical Section ໄດ້

# ສັນຍາລັກໃນການເຮັດວຽກຮ່ວມກັນ

- ການປະສານງານກັນຂອງເທຣດໂດຍໃຊ້ Semaphores
  - ສາມາດໃຊ້ Semaphores ເພື່ອແຈ້ງໃຫ້ເທຣດອື່ນຮູ້ຈັກວ່າມີເຫດການໃດໜຶ່ງເກີດຂຶ້ນ
  - ຄວາມສຳພັນລະວ່າງ Producer-consumer
    - Producer ເຂົ້າໃຊ້ critical section ເພື່ອເກັບຄ່າຂໍ້ມູນໃດໜຶ່ງ
    - Consumer ໄດ້ຖືກກັນໄວ້ຈົນກ່ວາ producer ສຳເລັດ
    - Consumer ເຂົ້າໃຊ້ critical section ເພື່ອອ່ານເອົາຄ່າຂໍ້ມູນໃດໜຶ່ງ
    - Producer ບໍ່ສາມາດ update ຂໍ້ມູນຈົນກ່ວາຖືກອ່ານຈົ້ນແລ້ວ
  - Semaphores ເປັນວິທີການທີ່ແກ້ໄຂບັນຫາໄດ້ຢ່າງຊັດເຈນ, ແລະ ງ່າຍຕໍ່ການສ້າງ

# ສັນຍາລັກໃນການເຮັດວຽກຮ່ວມກັນ

- Semaphores ແບບນັບ
  - ມີຄ່າເລີ່ມຕົ້ນຫຼາຍກ່ວາໜຶ່ງ
  - ສາມາດໃຊ້ເພື່ອຄວບຄຸມການເຂົ້າໃຊ້ pool ຂອງຊັບພະຍາກອນທີ່ຄ້າຍຄືກັນ
  - ລຸດຄ່າຕົວນັບຂອງ semaphore ລົງເມື່ອເອົາຂໍ້ມູນອອກຈາກ pool
  - ເພີ່ມຄ່າຕົວນັບ semaphore ເມື່ອເອົາຂໍ້ມູນມາເກັບໄວ້ໃນ pool
  - ເມື່ອບໍ່ມີຂໍ້ມູນຢູ່ໃນ pool, ເທຣດຈະຖືກກັນໄວ້ຈົນກະທັ້ງມີຂໍ້ມູນ

# ສັນຍາລັກໃນການເຮັດວຽກຮ່ວມກັນ

- ການສ້າງ Semaphores
  - ສາມາດສ້າງ Semaphores ໃນລະດັບ application ຫຼື ໃນລະດັບ kernel
    - ໃນລະດັບ Application: ໂດຍປົກກະຕິໄດ້ຖືກສ້າງໂດຍ busy waiting ຊຶ່ງສິ້ນເປືອງເວລາໜ່ວຍປະມວນຜົນ
      - ບໍ່ມີປະສິດທິພາບ(Inefficient)
    - ການສ້າງໃນລະດັບ Kernel ສາມາດຫຼີກລ້ຽງສະພາບການ busy waiting
      - ກັນເທຣດທີ່ກຳລັງລໍຖ້າຈົນກ່ວາມັນກຽມພ້ອມ
    - ການສ້າງໃນລະດັບ Kernel ໂດຍການ disable interrupts
      - ຮັບປະກັນໄດ້ວ່າການເຂົ້າໃຊ້ semaphore ເທື່ອລະເທຣດ
      - ຈະຕ້ອງລະມັດລະວັງບໍ່ໃຫ້ປະສິດທິພາບຕ່ຳ ແລະ deadlock
      - ການສ້າງສຳຫຼັບລະບົບ multiprocessor ຕ້ອງໃຊ້ວິທີທີ່ສະຫຼາດ ແລະ ຫຼັບຊັບຊ້ອນ

# Monitors

- ເປັນການສ້າງພາສາໂປຣແກຣມລະດັບສູງທີ່ເຮັດໜ້າທີ່ເຊັ່ນດຽວກັນກັບ semaphore ແຕ່ຄວບຄຸມງ່າຍກ່ວາ
- Monitor ຖືກນຳໄປໃຊ້ໃນພາສາໂປຣແກຣມຫຼາຍປະເພດເຊັ່ນ: Concurrent Pascal, Pascal-Plus, Modula-2, Modula-3 ແລະ Java ເປັນຕົ້ນ
- ມັນເປັນຊອບເຈັກໂມດູນປະກອບດ້ວຍໜຶ່ງຫຼືຫຼາຍຂະບວນງານ (Procedure) ທີ່ກຳໜົດຄ່າການຈັດລຳດັບ ແລະ ຕົວປ່ຽນຂໍ້ມູນສະເພາະຈຸດ

# Monitors

- ຄຸນລັກສະນະສຳຄັນຂອງ Monitor ປະກອບດ້ວຍ:
  - ຕົວປ່ຽນຂໍ້ມູນສະເພາະບາງອັນ ຈະສາມາດເຂົ້າຫາໄດ້ໂດຍຂະບວນງານສະເພາະບາງຢ່າງຂອງ Monitor ເທົ່ານັ້ນ ແລະ ບໍ່ສາມາດເຂົ້າໃຊ້ໄດ້ໂດຍຂະບວນງານອື່ນໆຈາກພາຍນອກ
  - ຂະບວນການເຂົ້າໃຊ້ Monitor ໄດ້ໂດຍການເອີ້ນໃຊ້ຂະບວນງານສະເພາະບາງຢ່າງຂອງ Monitor
  - ມີພຽງຂະບວນການດຽວເທົ່ານັ້ນທີ່ຖືກກະທຳການໄດ້ໂດຍ Monitor ໃນເວລາໃດໜຶ່ງ, ຖ້າມີຂະບວນການອື່ນຕ້ອງການໃຊ້ Monitor ຈະຕ້ອງຢຸດລໍຖ້າຈົນກ໋ວ Monitor ຫວ່າງ
- ສາມາດນຳໃຊ້ Monitor ເປັນກົນໄກອຳນວຍຄວາມສະດວກກ່ຽວກັບການບ່ໍໃຫ້ເຮັດພ້ອມກັນ

# ວົງຈອມປົດຕາຍ (Deadlock)

- ຂະບວນການໃດໜຶ່ງໃນລະບົບ multiprogramming ຢູ່ໃນສະ ຖານະ deadlock ຖ້າວ່າມັນລໍຖ້າເຫດການໃດໜຶ່ງທີ່ບໍ່ມີໂອກາດເກີດຂຶ້ນ
- ເມື່ອເກີດ Deadlock ຂຶ້ນ, ຂະບວນການທັງໝົດຈະບໍ່ສາມາດເຮັດຫຍັງໄດ້, ຊັບພະຍາກອນຂອງລະບົບຖືກຢຶດໄວ້ ແລະ ກີດກັນບໍ່ໃຫ້ເຮັດວຽກອື່ນໄດ້
- ບັນຫາການກິນເຂົ້າແລງຂອງນັກປັດຊະຍາເປັນຕົວຢ່າງທີ່ສຳຄັນເພື່ອເອົາໄປນຳໃຊ້ກັບການຈັດສັນຊັບພະຍາກອນຂອງລະບົບປະຕິບັດການ
- ໃນລະບົບປະຕິບັດການ, ມີຊັບພະຍາກອນ 3 ປະເພດ:
  - ຊັບພະຍາກອນທີ່ໃຊ້ຮ່ວມກັນໄດ້ໃນເວລາໃດໜຶ່ງ
  - ຊັບພະຍາກອນທີ່ສາມາດສັບປ່ຽນກັນໃຊ້ໄດ້
  - ຊັບພະຍາກອນທີ່ສາມາດໃຊ້ໄດ້ສະເພາະຂະບວນການໃດໜຶ່ງ

# ວົງຈອມປົດຕາຍ (Deadlock)

- ແບບຈຳລອງລະບົບ
  - ລະບົບປະກອບດ້ວຍຊັບພະຍາກອນຈຳນວນຈຳກັດ ຊຶ່ງຈະໃຊ້ແຈກຈ່າຍໃຫ້ແກ່ຂະບວນການທັງໝົດທີ່ຕ້ອງການໃຊ້ພວກມັນ
  - ຊັບພະຍາກອນໃນລະບົບຄອມພິວເຕີ້ມີຫລາຍປະເພດ, ແຕ່ລະປະເພດແບ່ງອອກເປັນຫຼາຍຈຸເຊັ່ນ: ໜ່ວຍຄວາມຈຳ, ຮອບວຽນການໃຊ້ CPU, ແຟ້ມຂໍ້ມູນ ແລະ ອຸປະກອນ I/O ເປັນຕົ້ນ
  - ຂະບວນການຕ້ອງຮ້ອງຂໍກ່ອນທີ່ຈະໃຊ້ຊັບພະຍາກອນ ແລະ ຕ້ອງສະລະຊັບພະຍາກອນເມື່ອໃຊ້ງານແລ້ວ
  - ໃນສະພາບປົກກະຕິຂອງລະບົບປະຕິບັດການ ຂະບວນການອາດຈະໃຊ້ຊັບພະຍາກອນຕາມລຳດັບໃດໜຶ່ງ ຕາມລຳດັບຕໍ່ໄປນີ້:
    - ຮ້ອງຂໍໃຊ້ຊັບພະຍາກອນ (Request)
    - ໃຊ້ຊັບພະຍາກອນ (Use)
    - ສະລະຊັບພະຍາກອນ (Release)

# ວົງຈອມປົດຕາຍ (Deadlock)

- ຄຸນລັກສະນະຂອງ Deadlock
  - ເງື່ອນໄຂຈຳເປັນທີ່ເຮັດໃຫ້ເກີດ Deadlock
    - ການບໍ່ເຮັດພ້ອມກັນ (Mutual Exclusion) ເປັນເງື່ອນໄຂທີ່ບໍ່ສາມາດໃຊ້ຊັບພະຍາກອນຮ່ວມກັນໄດ້, ຊັບພະຍາກອນຈະຖືກເອີ້ນໃຊ້ໄດ້ຈາກຂະ ບວນການດຽວເທົ່ານັ້ນໃນເວລາໃດໜຶ່ງ
    - ຈັບອັນໜຶ່ງໄວ້ແລ້ວຂໍອັນອື່ນ (Hold and Wait) ຂະບວນການທີ່ໄດ້ຮັບຊັບພະຍາກອນທີ່ບໍ່ສາມາດໃຊ້ຮ່ວມກັນໄດ້ ໄດ້ເອົາຊັພະຍາກອນນັ້ນໄວ້ ແລ້ວຮ້ອງຂໍຊັບພະຍາກອນອື່ນທີ່ຍັງມີຜູ້ໃຊ້ຢູ່
    - ບໍ່ມີການບັງຄັບ (No-Preemption) ຊັບພະຍາກອນເລົ່ານັ້ນບໍ່ສາ ມາດບັງຄັບໄດ້ ມັນຈະຖືກປ່ອຍອອກມາເອງເມື່ອຂະບວນການຂະບວນການນັ້ນເຮັດວຽກແລ້ວ
    - ການລໍຖ້າເປັນວົງມົນ (Circular Wait) ແມ່ນຍັນດາຂະບວນການທີ່ລໍຖ້າໃຊ້ຊັບພະຍາກອນນຳກັນເປັນວົງວຽນ
  - Deadlock ຈະເກີດຂຶ້ນຖ້າມີເງື່ອນໄຂທີ່ 4
  - ຈະເຮັດໃຫ້ເກີດເງື່ອນໄຂທີ່ 4 ຖ້າເງື່ອນໄຂ 1, 2, 3 ເກີດຂຶ້ນພ້ອມກັນ

# ວົງຈອມປົດຕາຍ (Deadlock)

- ການຈັດການກັບ Deadlock

  ມີ 4 ວິທີໃນການຈັດການກັບ Deadlock

  - ການປ້ອງກັນ Deadlock (Deadlock Prevention)
    - ເປັນບ່ອະນຸຍາດໃຫ້ 1 ໃນ 4 ເງື່ອນສຳຫລັບ Deadlock ເກີດຂຶ້ນ
  - ການຫລີກລ້ຽງ Deadlock (Deadlock Avoidance)
    - ບ່ຈັດສັນຊັບພະຍາກອນຖ້າເຫັນວ່າຈະເຮັດໃຫ້ເກີດ Deadlock
  - ການກວດຫາ Deadlock (Deadlock Detection)
    - ຈັດສັນຊັບພະຍາກອນຕາມປົກກະຕິ, ແຕ່ຕ້ອງກວດສອບຫາ Deadlock ເປັນໄລຍະ ຖ້າເຫັນ Deadlock ເກີດຂຶ້ນກ່ໍກູ້ຄືນ
  - ການກູ້ຄືນຈາກ Deadlock (Deadlock Recovery)
    - ເປັນການແກ້ໄຂບັນຫາ Deadlock ເຮັດໃຫ້ລະບົບກັບມາເຮັດວຽກຕາມປົກກະຕິໄດ້

# ວົງຈອມປົດຕາຍ (Deadlock)

- ການປ້ອງກັນ Deadlock (Deadlock Prevention)
  - ເປັນການປັບເງື່ອນໄຂໃຫ້ແກ່ລະບົບເຮັດໃຫ້ມັນບໍ່ເກີດມີ Deadlock ໂດຍການເຮັດບໍ່ໃຫ້ມີ 4 ເງື່ອນທີ່ເຮັດໃຫ້ Deadlock ເກີດຂຶ້ນ
  - ສຳຫລັບເງື່ອນໄຂທີ່ບໍ່ໃຫ້ເກີດພ້ອມກັນ (Mutual Exclusion), ການຄອບຄອງຊັບພະຍາກອນທີ່ບໍ່ສາມາດໃຊ້ພ້ອມກັນໄດ້ຈະຕ້ອງເຮັດຕາມລຳດັບ
  - ສຳຫລັບເງື່ອນການຖືຊັບພະຍາກອນໜຶ່ງ ແລ້ວ ລໍຖ້າຊັບພະຍາ ກອນອື່ນ (Hold and Wait), ຈະຕ້ອງຮັບປະກັນວ່າ ຂະບວນການທີ່ຮ້ອງຂໍຊັບພະຍາກອນໃດໜຶ່ງ ຈະຕ້ອງບໍ່ຖືຊະບພະຍາກອນອື່ນໄວ້
  - ສຳຫລັບເງື່ອນບໍ່ສາມາດບັງຄັບຊັບພະຍາກອນທີ່ໄດ້ຖືກຈັດສັນແລ້ວ ຕ້ອງຮັບປະກັນວ່າ ມັນຈະບໍ່ຖືກຈອງໄວ້ໂດຍປະຕິບັດຕາມຫລັກການທີ່ວ່າ ຖ້າຂະບວນການທີ່ກຳລັງຈອງຊັບພະຍາກອນໃດໜຶ່ງຢູ່ ຮ້ອງຂໍຊັບພະຍາ ກອນອື່ນເພີ່ມເຕີມ ແຕ່ຍັງບໍ່ສາມາດຈັດສັນໃຫ້ໄດ້ ມັນຈະຕ້ອງຖືກບັງຄັບໃຫ້ປ່ອຍຊັບພະຍາກອນທັງໝົດທີ່ມັນຈອງຢູ່
  - ເພື່ອຮັບປະກັນວ່າການລໍຖ້າກັນເປັນວົງມົນບໍ່ເກີດຂຶ້ນໃນລະບົບຈະຕ້ອງຈັດລຳດັບຊັບພະຍາກອນທຸກປະເພດ ແລະ ການຮ້ອງຂໍຈາກຂະບວນການ

# ວົງຈອມປົດຕາຍ (Deadlock)

- ການຫລີກລ້ຽງ Deadlock (Deadlock Avoidance)
  - ເປັນເງື່ອນໄຂທີ່ບໍ່ໄດ້ເຂັ້ມງວດຄືກັບການປ້ອງກັນ Deadlock ເພື່ອໃຫ້ລະບົບສາມາດໄດ້ຊັບພະຍາກອນສຸງສຸດ
  - ເປັນການຈັດສັນຊັບພະຍາກອນໃຫ້ແກ່ຂະບວນການກ່ຳຕ່ເມື່ອເຫັນມັນຈະບໍ່ເກີດ Deadlock ຫຼື ເອີ້ນວ່າ “safe state”
  - Safe state
    - ເປັນການຮັບປະກັນວ່າທຸກຂະບວນການສາມາດເຮັດວຽກແລ້ວພາຍໃນເວລາທີ່ກຳໜົດ
  - Unsafe state
    - ບໍ່ໄດ້ໝາຍຄວາມວ່າມັນເກີດ Deadlock ແຕ່ມັນໝາຍຄວາມວ່າ OS ບໍ່ສາມາດຮັບປະກັນວ່າທຸກຂະບວນການສາມາດເຮັດວຽກແລ້ວພາຍໃນເວລາທີ່ກຳໜົດ

# ວິງຈອມປົດຕາຍ (Deadlock)

- ການຫລີກລ້ຽງ Deadlock (Deadlock Avoidance)

| Process | max( $P_i$ ) (maximum need) | loan( $P_i$ ) (current loan) | claim( $P_i$ ) (current claim) |
|---|---|---|---|
| $P_1$ | 4 | 1 | 3 |
| $P_2$ | 6 | 4 | 2 |
| $P_3$ | 8 | 5 | 3 |
| *Total resources, t, = 12* | | *Available resources, a, = 2* | |

Safe state.

| Process | max( $P_i$ ) (maximum need) | loan( $P_i$ ) (current loan) | claim( $P_i$ ) (current claim) |
|---|---|---|---|
| $P_1$ | 10 | 8 | 2 |
| $P_2$ | 5 | 2 | 3 |
| $P_3$ | 3 | 1 | 2 |
| *Total resources, t, = 12* | | *Available resources, a, = 1* | |

Unsafe state.

# ວົງຈອມປົດຕາຍ (Deadlock)

- ການຫລີກລ້ຽງ Deadlock (Deadlock Avoidance)
  - ຕົວຢ່າງ: ການປ່ຽນຈາກ Safe ເປັນ Unsafe state
    - ສົມມຸດວ່າໃນສະຖານະປະຈຸບັນຢູ່ໃນ Safe state ດັ່ງເຊັ່ນໃນຕົວຢ່າງຂ້າງເທິງ
    - ຈຳນວນຊັບພະຍາກອນທີ່ຍັງເຫຼືອແມ່ນ a = 2
    - ສົມມຸດວ່າຂະບວນການທີ 3 (P3) ຮ້ອງຂໍມາໄດ້ຕື່ມ 1 ຊັບພະຍາກອນ

| Process | max( $P_i$ ) (maximum need) | loan( $P_i$ ) (current loan) | claim( $P_i$ ) (current claim) |
|---|---|---|---|
| $P_1$ | 4 | 1 | 3 |
| $P_2$ | 6 | 4 | 2 |
| $P_3$ | 8 | 6 | 2 |
| *Total resources, t, = 12* | | *Available resources, a, = 1* | |

# ວິງຈອມປົດຕາຍ (Deadlock)

- ການຫລີກລ້ຽງ Deadlock (Deadlock Avoidance)
  - ສະຖານະພາບໃນຮູບລຸ່ມນີ້ Safe ຫຼື ບໍ່?

| Process | max( $P_i$ ) | loan( $P_i$ ) | claim( $P_i$ ) |
|---|---|---|---|
| $P_1$ | 5 | 1 | 4 |
| $P_2$ | 3 | 1 | 2 |
| $P_3$ | 10 | 5 | 5 |
| | | $a = 2$ | |

ຄຳຕອບ:

ບໍ່ສາມາດຮັບປະກັນໄດ້ວ່າຂະບວນການເຫຼົ່ານັ້ນຈະສຳເລັດ

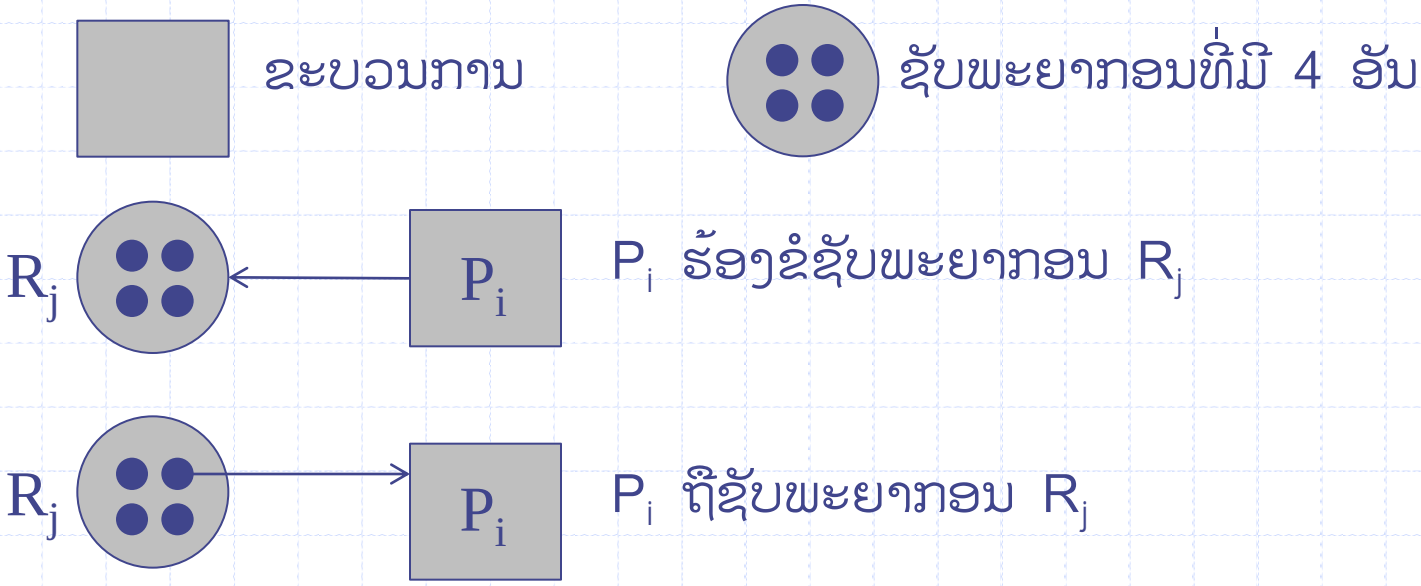
- $P_2$ ສາມາດສຳເລັດໄດ້ໂດຍໃຊ້ 2 ຊັບພະຍາກອນທີ່ມີ
- ເມື່ອ $P_2$ ສຳເລັດ, ຈະມີພຽງ 3 ຊັບພະຍາກອນທີ່ໃຊ້ໄດ້ ຊຶ່ງບໍ່ພຽງພໍທີ່ຈະຈັດສັນໃຫ້ $P_1$ ທີ່ຕ້ອງການ 4 ແລະ $P_3$ ທີ່ຕ້ອງການ 5

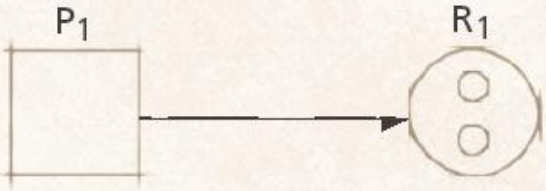
# ວົງຈອມປົດຕາຍ (Deadlock)

- ການກວດສອບຫາ Deadlock (Deadlock Detection)
  - ເປັນຂະບວນການທີ່ໄປຄົ້ນຫາວ່າ Deadlock ມັນເກີດຂຶ້ນຢູ່ບ່ອນໃດ ແລະ ກຳໜົດໃຫ້ໄດ້ວ່າມີຂະບວນການ ແລະ ຊັບພະຍາກອນໃດແດ່ທີ່ກ່ຽວຂ້ອງ
  - ກຸ່ມຂອງຂະບວນການຈະເອີ້ນວ່າເກີດເຫດການ Deadlock ຖ້າວ່າແຕ່ລະຂະບວນການພາຍໃນກຸ່ມລໍ້ຖ້າເຫດການບາງຢ່າງທີ່ມີພຽງຂະບວນການດຽວພາຍໃນກຸ່ມເທົ່ານັ້ນທີ່ສາມາດເຮັດໃຫ້ມັນເກີດຂຶ້ນໄດ້
  - ໃນການກວດຄົ້ນຫາ Deadlock ລະບົບຈຳເປັນຕ້ອງໃຊ້ Graph ຈັດສັນຊັບພະຍາກອນ
  - ການກວດຄົ້ນຫາ Deadlock ຈະຕ້ອງພິຈາລະນາຈາກປັດໃຈດັ່ງນີ້
    - Deadlock ເກີດຂຶ້ນຖີ່ປານໃດ
    - ມີຂະບວນການໃດແດ່ທີ່ຮັບການກະທົບ

# ວົງຈອນປິດຕາຍ (Deadlock)

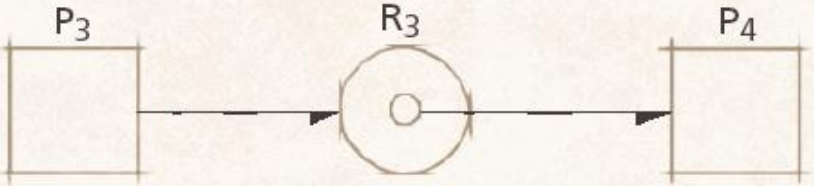
- ການກວດສອບຫາ Deadlock (Deadlock Detection)
  - Graph ຈັດສັນຊັບພະຍາກອນ
    - ຮູບສີ່ແຈສາກແມ່ນໃຊ້ເປັນຕົວແທນຂອງຂະບວນການ
    - ວົງມົນໃຫຍ່ໃຊ້ເປັນຕົວແທນຂອງຂອງກຸ່ມຊັບພະຍາກອນທີ່ຄ້າຍຄືກັນ
    - ວົງມົນນ້ອຍທີ່ຢູ່ທາງໃນວົງມົນໃຫຍ່ໃຊ້ເປັນຕົວແທນຂອງຊັບພະຍາກອນແຕ່ລະອັນ



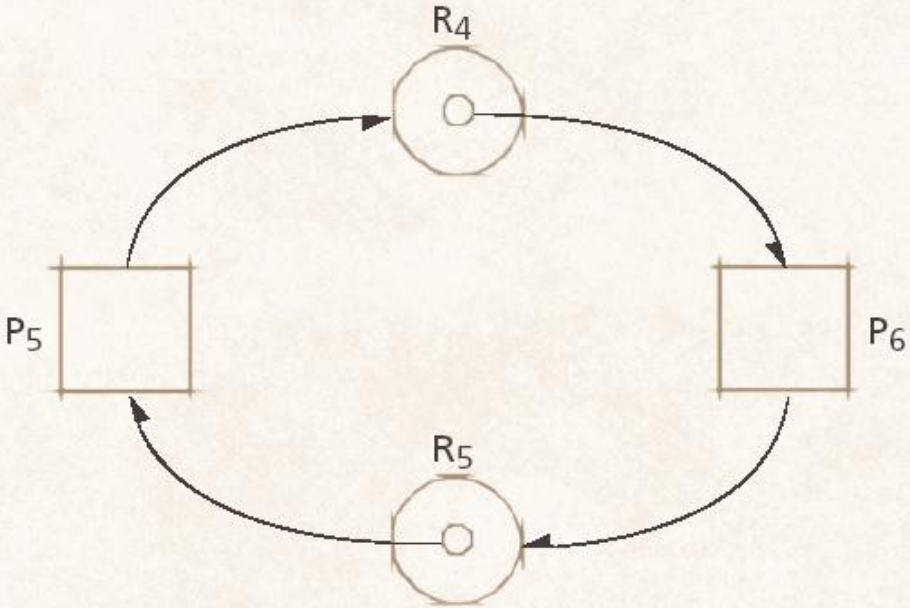



(a) $P_1$ is requesting a resource of type $R_1$, of which there are two identical resources.


(b) One of two identical resources of type $R_2$ has been allocated to process $P_2$.



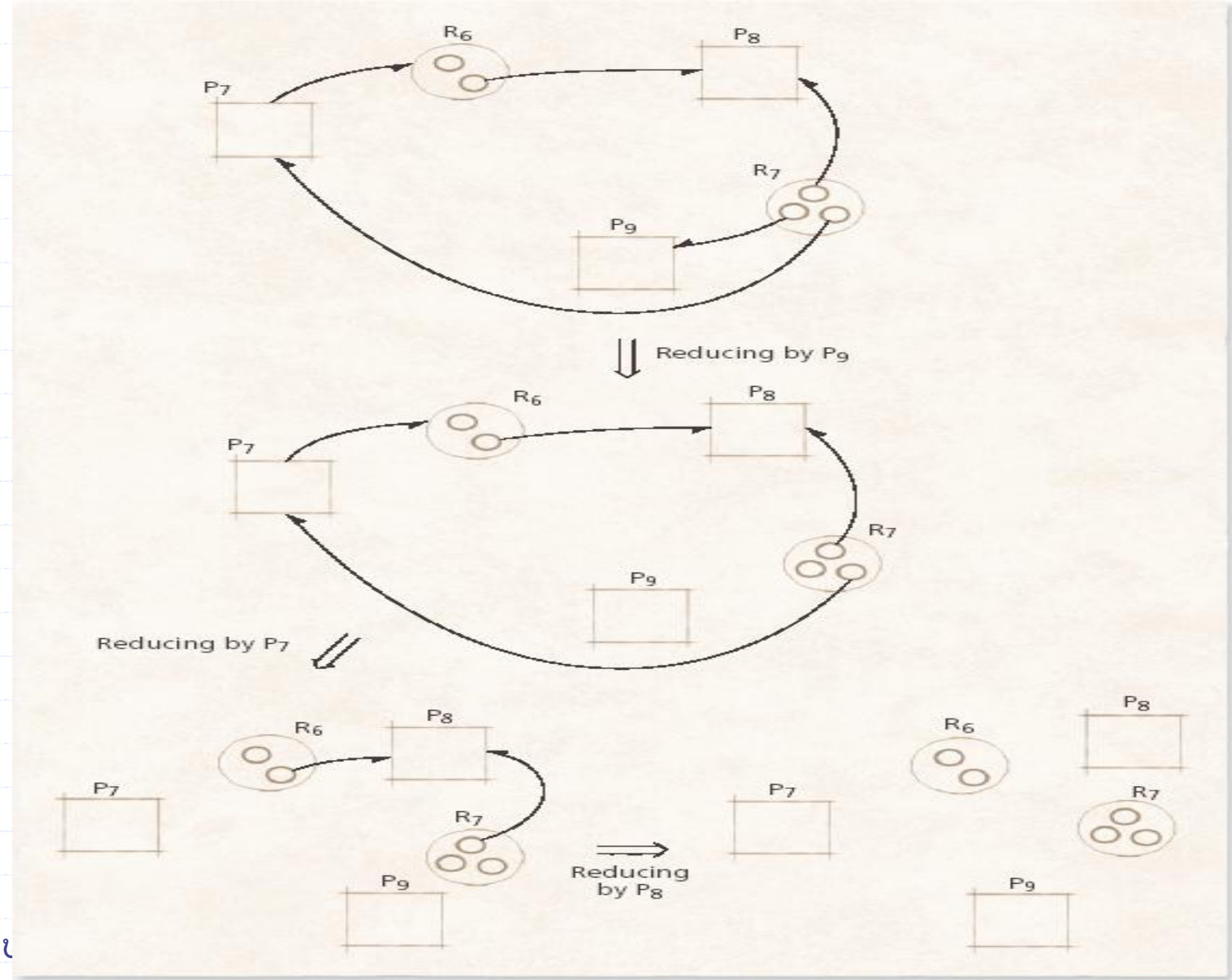

(c) Process $P_3$ is requesting resource $R_3$, which has been allocated to process $P_4$.




(d) Process $P_5$ has been allocated resource $R_5$ that is being requested by process $P_6$ that has been allocated resource $R_4$ that is being requested by process $P_5$ (the classic "circular wait").

# ວົງຈອມປົດຕາຍ (Deadlock)

- ການກວດສອບຫາ Deadlock (Deadlock Detection)
  - ການຕັດຮອມ Graph ຈັດສັນຊັບພະຍາກອນ
    - ຖ້າຊັບພະຍາກອນຂອງຂະບວນການໃດອາດສາມາດຈັດສັນໄດ້ graph ດັ່ງກ່າວຈະຖືກຕັດຮອມຂະບວນການນັ້ນອອກໄປ ສະແດງວ່າຂະບວນການນັ້ນບໍ່ເຮັດໃຫ້ເກີດ deadlock
    - ຖ້າສາມາດຕັດຮອມຂະບວນການອອກຈາກ graph ໄດ້ໝົດສະແດງວ່າ ບໍ່ມີ deadlock ເກີດຂຶ້ນ
    - ຖ້າບໍ່ສາມາດຕັດຮອມຂະບວນການທັງໝົດອອກຈາກ graph ໄດ້ ສະແດງວ່າມັນຈະເຮັດໃຫ້ເກີດ deadlock ຂຶ້ນໃນກຸ່ມຂະບວນການນັ້ນ


R6
P8
P7
R7
P9
Reducing by P9
R6
P8
P7
R7
P9
Reducing by P7
R6
P8
P7
R7
P9
Reducing
by P8
P7
R6
P8
R7
P9

# ວົງຈອມປົດຕາຍ (Deadlock)

- ການກູ້ຄືນ Deadlock (Deadlock Recovery)
  - ເປັນການກຳຈັດ Deadlock ໃຫ້ອອກຈາກລະບົບ ເພື່ອເຮັດໃຫ້ຂະບວນການທີ່ຢູ່ໃນສະພາບ Deadlock ໃຫ້ກັບມາເຮັດວຽກໄດ້ປົກກະຕິຈົນສຳເລັດ ແລະ ປ່ອຍຊັບພະຍາກອນໃຫ້ລະບົບ
  - ສາມາດຈັດການໄດ້ດ້ວຍແນວທາງຕ່າງໆດັ່ງນີ້
    - ບອກໃຫ້ຜູ້ປະຕິບັດງານຮູ້ວ່າເກີດ Deadlock ຂຶ້ນ ແລະ ອານຸຍາດໃຫ້ຜູ້ປະຕິບັດງານຈັດການກັບສະຖານະການດັ່ງກ່າວ
    - ໃຫ້ລະບົບກູ້ຄືນຈາກ Deadlock ໃຫ້ແບບອັດຕະໂນມັດ
  - ການແກ້ໄຂບັນຫາດັ່ງກ່າວສາມາດເຮັດໄດ້ 2 ວິທີ
    - ການຍົກເລີກຂະບວນການໃດໜຶ່ງ ຫຼື ຫຼາຍຂະບວນການເພື່ອຫຍຸດການລໍຖ້າແບບວົງມົນ
    - ການບັງຄັບດຶງຊັບພະຍາກອນທີ່ກ່ຽວຂ້ອງອອກມາຈາກການຍຶດຄອງຂອງບັນດາຂະບວນການ

# ວົງຈອມປົດຕາຍ (Deadlock)

- ການກູ້ຄືນ Deadlock (Deadlock Recovery)
  - ການຍົກເລີກຂະບວນການ
    - ຍົກເລີກຂະບວນການທັງໝົດທີ່ເກີດ Deadlock. ວິທີນີ້ສາມາດຢຸດ deadlock ໄດ້ ແນ່ນອນ ແຕ່ມີຄ່າໃຊ້ຈ່າຍສູງຫລາຍ
    - ຍົກເລີກເທື່ອລະຂະບວນການຈົນກ່ວາຢຸດ deadlock ໄດ້ ຊຶ່ງເປັນວິທີທີ່ສົມເໝາະສົມ ແຕ່ຕ້ອງໄດ້ມີຂັ້ນຕອນວິທີໃນການກວດຫາ ແລະ ຕັດສິນໃຈວ່າຄວນຈະຢຸດຂະບວນການໃດ
  - ປັດໃຈທີ່ສາມາດໃຊ້ໃນການພິຈາລະນາຍົກເລີກມີດັ່ງນີ້
    - ລຳດັບຄວາມສຳຄັນຂອງຂະບວນການນັ້ນ
    - ຂະບວນການນັ້ນຈະໃຊ້ເວລາເທົ່າໃດໃນການຄຳນວນ ແລະ ຕ້ອງໃຊ້ເວລາຄຳນວນອີກດົນເທົ່າໃດຈຶ່ງຈະສຳເລັດ
    - ຈຳນວນ ແລະ ປະເພດຂອງຊັບພະຍາກອນທີ່ຂະບວນການນັ້ນກຳລັງໃຊ້ຢູ່
    - ມີຊັບພະຍາກອນອີກຈຳນວນເທົ່າໃດທີ່ຂະບວນການນັ້ນຕ້ອງໃຊ້ຈົນກ່ວາຈະສຳເລັດ
    - ມີຂະບວນການທີ່ຈຳເປັນຕ້ອງຍົກເລີກຈຳນວນເທົ່າໃດ
    - ຂະບວນການນັ້ນເປັນແບບ Interactive ຫຼື ແບບກຸ່ມ

# ວົງຈອມປົດຕາຍ (Deadlock)

- ການກູ້ຄືນ Deadlock (Deadlock Recovery)
  - ການເອົາຊັບພະຍາກອນອອກຈາກ deadlock
    1. ເລືອກຊັບພະຍາກອນ ແລະ ຂະບວນການທີ່ຈະຍົກເລີກ. ຈະຕ້ອງເລືອກກໍລະນີທີ່ມີຄ່າໃຊ້ຈ່າຍໜ້ອຍທີ່ສຸດ ໂດຍເບິ່ງຈາກຈຳນວນ ແລະ ເວລາທີ່ຂະບວນການນັ້ນໃຊ້ຢູ່
    2. ໃຊ້ວິທີຍ້ອນຂະບວນການນັ້ນກັບໄປຍັງສະຖານະທີ່ປອດໄພ ແລະ ໄປເລີ່ມຕົ້ນຂະບວນການນັ້ນຈາກຈຸດທີ່ປອດໄພ. ການກຳໜົດສະຖານະທີ່ປອດໄພນັ້ນເປັນເລື່ອງຍາກ ສະນັ້ນ ວິທີງ່າຍທີ່ສຸດຄືການຍ້ອນກັບທັງໝົດ ໂດຍການຍົກເລີກຂະບວນການແລ້ວເລີ່ມຕົ້ນໃໝ່
    3. ນະໂຍບາຍການເລືອກຊັບພະຍາກອນ ແລະ ຂະບວນການ ຈະຕ້ອງມັ້ນໃຈວ່າຂະບວນການທີ່ຖືກເລືອກນັ້ນຈະຕ້ອງຖືກເລືອກເປັນໄລຍະເວລາສັ້ນໆເທົ່ານັ້ນເພື່ອຫລີກລ້ຽງການລໍຄອຍແບບບໍ່ສິ້ນສຸດ