1 Lý thuyết

Câu 1: 2 nhu cầu trao đổi thông tin của các tiến trình nhằm:

a. Chia sẻ tài nguyên chung; Phối hợp hoạt động

b. Xử lý song song; Phối hợp hoạt động

c. Bảo đảm độc lập; Thông báo lỗi

Câu 2: Race Condition là:

a. Kết quả thực hiện tiến trình phụ thuộc vào kết quả điều phối
b. Hiện tượng các tiến trình chia sẻ tài nguyên chung
c. Kết quả tiến trình thực hiện luôn luôn sai

Câu 3: Critical section là:

a. Tài nguyên dùng chung giữa các tiến trình
b. Cơ chế bảo vệ tài nguyên dùng chung
c. Đoạn chương trình có khả năng gây ra hiện tượng race condition
d. Đoạn chương trình có truy cập tài nguyên dùng chung

Câu 4: 2 nhu cầu đồng bộ tiến trình là :

a. Hò hẹn; Phối hợp hoạt động
b. Trao đổi thông tin; Phối hợp hoạt động
c. Độc quyền truy xuất; Giải quyết tranh chấp
d. Không có câu nào đúng

Câu 5: Cho biết ít nhất 3 điều kiện cho một giải pháp đồng bộ tốt

2 Bài tập

2.1 Bài 1

Sử dụng Semaphore để viết lại chương trình sau theo mô hình xử lý tuần tự.

P1 Y:= W* X1

P2 Z:= W* X2

P3 Ans:= Y+Z

2.2 Bài 2

Sử dụng Semaphore để viết lại chương trình sau theo mô hình xử lý đồng hành

P1 W:= X1*X2

P2 V:=X3*X4

P3 Y:=V*X5

P4 Z:=V*X6

P5 Y:=W*Y

P6 Z:=W*Z

P7 Ans:=Y+Z

Đáp án:

```
process P1{
        W = X1 * X2;
        up(s15);
        up(s16);
}
process P2{
        V = X3 * X4;
        up(s23);
        up(s24);
}
```

```
process P3{
        down(s23);
        Y = V * X5;
        up(s35);
}
procsee P4{
        down(s24);
        X = V * X;
        up(s46);
}
process P5{
        down(s15);
        down(s35);
        Y = W * Y;
        up(s57);
}
process P6{
        down(s16);
        down(s46);
        Z = W * Z;
        up(s67);
}
process P7{
        down(s57);
```

```
        down(s67);
        ANS = Y + Z;
}
```

2.3 Bài 3

Xét giải pháp đồng bộ hoá sau :

```
while (TRUE) {
    int j = 1-i;
    flag[i]= TRUE; turn = i;
    while (turn == j && flag[j]==TRUE);
    critical-section ();
    flag[i] = FALSE;
    Noncritical-section ();
}
```

Đây có phải là một giải pháp bảo đảm được độc quyền truy xuất không?

Đáp án:

Không. Xét tình huống khi flag[0] =1; turn =0=>P0 vào CS,
Nếu lúc đó flag[1]= 1 → P1 có thể gán turn = 1 và vào luôn CS!

2.4 Bài 4

Một biến X được chia sẻ bởi hai tiến trình cùng thực hiện đoạn code sau:

```
do{
    X = X +1;
    if ( X == 20) X = 0;
}while ( TRUE );
```

Bắt đầu với giá trị X = 0, chứng tỏ rằng giá trị X có thể vượt quá 20.

Đáp án:

```
Semaphore mutex = 1;

do{

    down(mutex);

    X = X +1;

    if ( X == 20) X = 0;

    up(mutex);

}while ( TRUE );
```

2.5 Bài 5

Xét hai tiến trình xử lý đoạn chương trình sau :

```
process P1 {
        A1 ;
        A2 ;
}
process P2 {
        B1 ;
        B2 ;
}
```

Đồng bộ hoá hoạt động của hai tiến trình này sao cho cả A1 và B1 đều hoàn tất trước khi A2 hay B2 bắt đầu.

Đáp án:

```
semaphore ab = 0, ba = 0;
process P1 {
        A1 ;
        Down(ab);
        Up(ba);
        A2 ;
}
process P2 {
        B1 ;
        Down(ba);
        Up(ab);
        B2 ;
}
```

2.6 Sản xuất xe

Một hãng sản xuất xe ô tô có các bộ phận hoạt động song song:

- Bộ phận sản xuất khung xe
  void SXKhung(){
  printf("San xuat khung");
  }
- Bộ phận sản xuất bánh xe
  void SXBanhXe(){
  printf("San xuat banh xe");
  }
- Bộ phận lắp ráp: Sau khi có đủ 1 khung và 4 bánh thì tiến hành lắp ráp.
  void LapRapXe(){
  printf("Lap rap xe");
  }

Hãy đồng bộ hoạt động của các bộ phận trên theo nguyên tắc: tại mỗi thời điểm chỉ cho phép sản xuất 1 khung xe, cần chờ đủ 4 bánh xe để gắn vào khung xe hiện tại này trước khi sản xuất một khung xe khác.

Đáp án:

```
semaphore skhung = 0, sbanh = 0, slaprap = 1;

void SXKhung(){

    while(true){

        down(slaprap);
        printf("San xuat khung");
        up(skhung);
    }

}

void SXBanhXe(){

    while(true){

        printf("San xuat banh xe");
        up(sbanh);
    }
```

```
}
void LapRapXe(){
    while(true){
        down(skhung);
        down(sbanh);
        down(sbanh);
        down(sbanh);
        down(sbanh);
        printf("Lap rap xe");
        up(slaprap);
    }
}
```