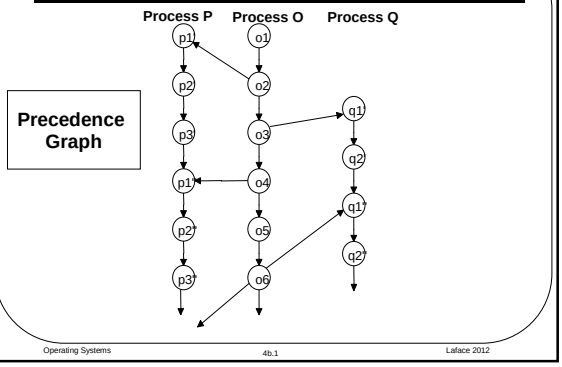
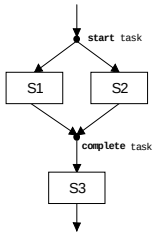
Process Synchronization

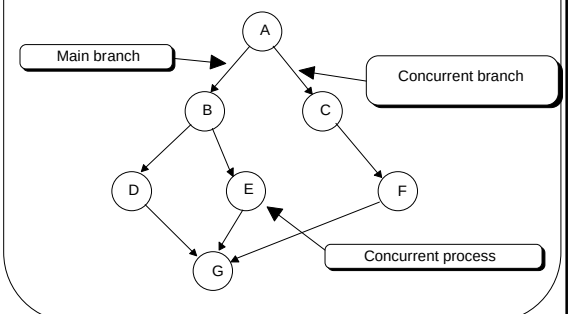


Synchronization Primitives

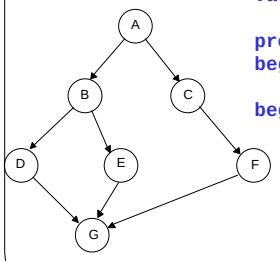



```
var task : response;
begin
  start task do S1;
    S2;
  complete task;
  S3;
```


Synchronization Primitives

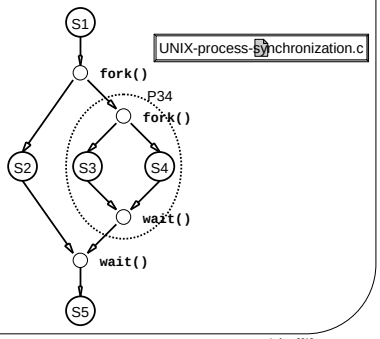


Primitives fork and join




```
var P1, P2 : process;

procedure E1;
begin C;F; end

begin
     A;
     P1 := fork E1;
     B;
     P2 := fork E;
     D;
     join P1;
     join P2;
     G;
end
```


UNIX Process Synchronization



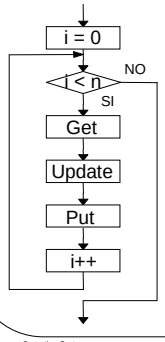
UNIX Process Synchronization Output

```
Prompt > UNIX-process-synchronization.exe

parent - S1 - pid = 1656
parent - S2 - pid = 1656
parent: child pid= 1844, parent pid = 1656
wait - P34 -
child: child pid = 1844, parent pid = 1656
fork - P34 -
- S3 - pid = 1844, parent pid = 1656
- S4 - pid = 1868, parent pid = 1844
end of - P34 -
parent - S5 - pid = 1656
```

UNIX Process Synchronization Output

```
Prompt > UNIX-process-synchronization.exe 1


parent - S1 - pid = 1848
child: child pid = 1352, parent pid = 1848
fork - P34 -
- S4 - pid = 1808, parent pid = 1352
parent - S2 - pid = 1848
parent: child pid= 1352, parent pid = 1848
wait - P34 -
- S3 - pid = 1352, parent pid = 1848
end of - P34 -
parent - S5 - pid = 1848
```


Sequential Process

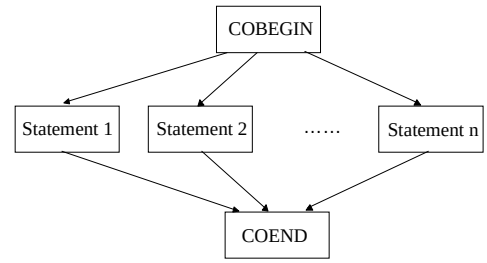



```
typedef …. T;

T this;
int i;
{
  i:=0;
  while (i < n) {
    get (this, reader);
    update (this);
    put (this, printer);
    i++;
  }
}
```

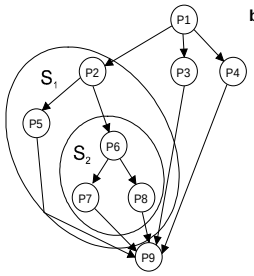

Concurrent Processes

```
if (n >= 1) {
  get (last, reader);
  if (n >= 2) {
    update (last);
    start printing do put (last, printer);
    get (this, reader);
    for(i=2; i < n; i++) {
      start reading do get (next, reader);
      update (this);
      complete printing;
      last = this;
      start printing do put (last, printer);
      complete reading;
      this = next;
    }
    complete printing;
    last := this;
  }
  update (last);
  put (last, printer);
}
```

Concurrent block: Cobegin - Coend



Nested Concurrent Blocks




```
begin
  P1;
  cobegin
    P3;
    P4;
    S1;
  coend;
  P9;
end
```

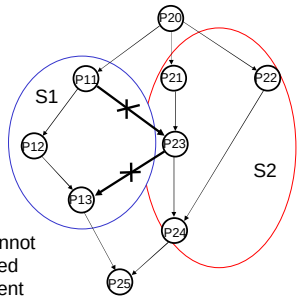
```
S1: begin
    P2;
    cobegin
      P5;
      S2;
    coend;
    end

S2: begin
    P6;
    cobegin
      P7;
      P8;
    coend
    end
end
```


Concurrent Processes

```
if (n ≥ 1) {
  get (this, reader);
  if (n ≥ 2) {
    cobegin
      get (next, reader);
      update (this);
    coend
    for(i=2; i < n; i++){
      last = this;
      this = next;
      cobegin
        get (next, reader);
        update (this);
        put (last, printer);
      coend
    }
      put (this, printer);
      this = next;
    }
    update (this);
    put (this, printer);
  }
```

get-update-print.c

Problems with concurrent blocks



This graph cannot be implemented using concurrent blocks only

Explicit Waits and Signals

```
begin
   P20;
   cobegin
     S1;
     S2;
   coend
   P25;
 end
```

```
S1: begin
     P11;
     signal q1;
     P12;
     wait q2;
     P13;
    end
```

```
S2: begin
    cobegin
      P22;
      begin
        P21;
        wait q1;
        P23;
        signal q2;
      end
    coend
    P24;
    end
```


Correct Use of Cobegin-Coend

```
sum = n1 + n2 + n3 + n4

cobegin
    sum1 = n1 + n2
    sum2 = n3 + n4
coend
sum = sum1 + sum2
```

Incorrect Use of Cobegin-Coend

```
x = 100;
cobegin
  x += 10;                    (1)

  if (x > 100)
    print x;                  (2)
  else
    print x – 50;
coend
```


Concurrency Conditions

```
OBSERVER:
  while (true)
  /*   a sensor detects a car passing */
       COUNT = COUNT + 1;

REPORTER:
  while (true){
       PRINT COUNT;
       COUNT = 0;
  }
```


Observer & Reporter

```
OBSERVER:
  while (true)
  /*    a sensor detects a car passing */
       COUNT = COUNT + 1;

REPORTER:
  while (true){
       PRINT COUNT;
       COUNT = 0;
  }
```

```
print COUNT          (Reporter)
Car                  (Observer)
COUNT = 0            (Reporter)
```

Observer & Reporter

```
OBSERVER:
  while (true)
  /*    a sensor detects a car passing */
        COUNT = COUNT + 1;

REPORTER:
  while (true){
        PRINT COUNT;
        COUNT = 0;
  }
```

```
MOV Reg,COUNT     (Observer)
MOV COUNT,0       (Reporter)
INC Reg           (Observer)
MOV COUNT,Reg     (Observer)
```


Time dependent errors

```
cobegin
    get (next, reader)
    update (this);
    put (this, printer);
coend
```


Concurrency Conditions

- Two processes S_i and S_j can be run in concurrence iff

$$R(S_i) \cap W(S_j) = 0$$
$$R(S_j) \cap W(S_i) = 0$$
$$W(S_i) \cap W(S_j) = 0$$

- Otherwise time dependent errors
- When the concurrency conditions are not fulfilled, the processes must run in *Mutual Exclusion*