Today's Plan

Upcoming:

- Assignment 1
- Practice Quiz 1

Last time:

- Processes
- Precedence & Concurrency

Today's topics:

- Process Creation
  - Process flow graphs
  - Cobegin/Coend
  - Fork/Join
- The Critical Section Problem

Process Flow Graph Examples

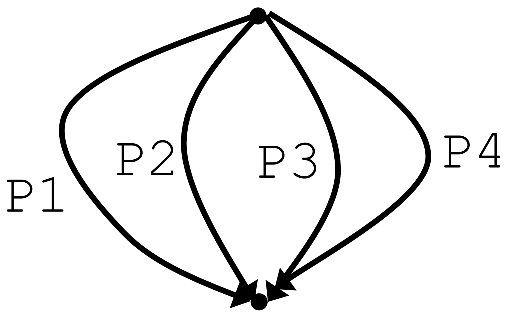
```
s(s(1,p(s(p(2,3),4),5)),6)
```

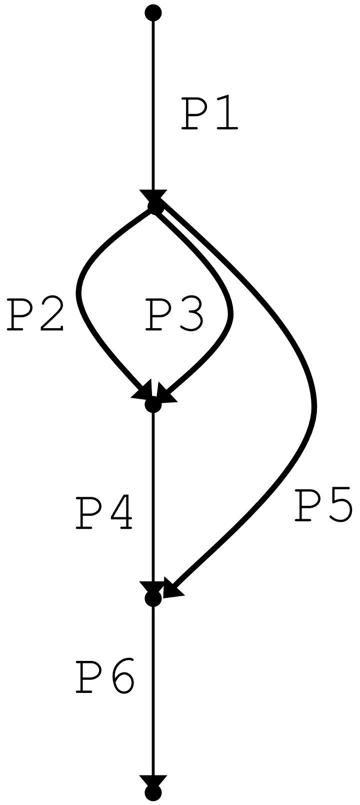
- S & P compositions are difficult to read and write, and are unable to describe non-properly nested situations

Cobegin/Coend Construct

- This is just another way of writing S() and P() functions
  - Only appropriate for use with properly nested graphs
- Statements written between a `cobegin/coend` pair are executed in parallel
  - If statements are nested, then they all begin immediately after the cobegin statement, and the last one to finish does so immediately before the coend statement
- Statements written between a `begin/end` pair are executed in serial, in the order they appear

Cobegin/Coend Examples

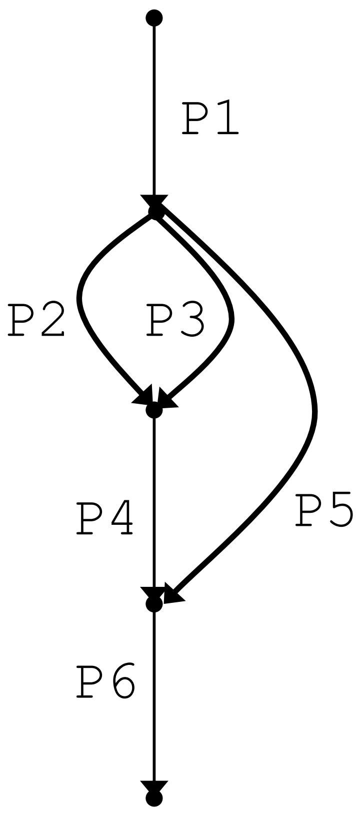




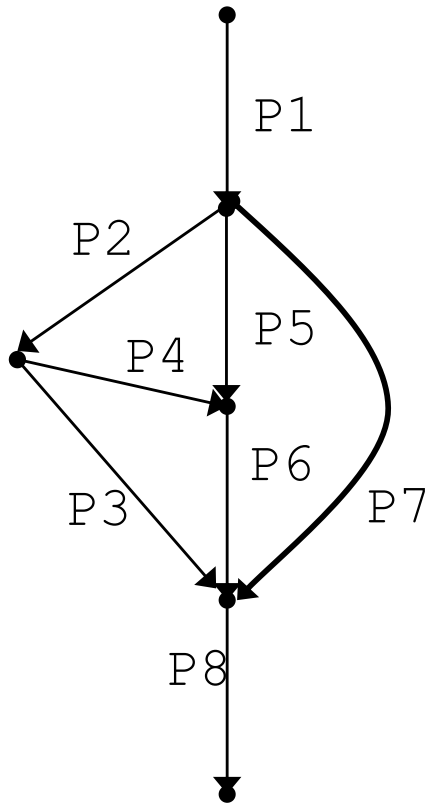
Process Creation Constructs

- One mechanism for creating processes is called **fork and join**
- `Fork(label L)` produces 2 concurrent processes, one starts immediately after the fork statement, and one starts at label L
  - Has the effect of splitting a single process execution into two concurrent processes
- `Join(int x)` recombines x processes into 1, effectively throwing away the first x-1 processes that reach it, and continuing execution after the Join statement, when the xth process reaches it

Fork and Join Example



Fork and Join Example



Critical Sections

- Problem Definition
- Software Solutions
- Hardware Solutions
- Semaphores
- Monitors
- Inter-Process Communication

The Critical Section Problem

- *Critical Sections:*
    - Sections of code in separate processes that do not obey Bernstein’s conditions
- A solution will provide some method of only allowing one process to access their critical section at a time.
- Two critical sections are said to be *related* if they are in separate processes and do not obey Bernstein’s conditions.

E.g.

```
P1: x = 1;      P2: x = 2;          P3: y = 3;
                    y = 2;
```

Example: Producer / Consumer

```
Common data structure:
typedef struct node {
    int item;
    node *next; } NODE;
```

```
Producer:
while (1) {
    /* produce a new item */
        (big piece of code)
    newnode = (NODE *)malloc(sizeof(NODE));
    newnode->item = NewItem;
    newnode->next = first;
    first = newnode;
}
```

```
Consumer:
while (1) {
    while (!first);
    mynode = first;
    first = first->next;
    item = mynode->item;
    /* consume an item */
        (some other big piece
         of code)
}
```

Example: Producer / Consumer

Producer's item ignored

```
C: mynode = first
P: newnode->next = first
P: first = newnode
C: first = first->next
```

Consumer's deletion ignored:

```
C: mynode = first
P: newnode->next = first
C: first = first->next
P: first = newnode
```