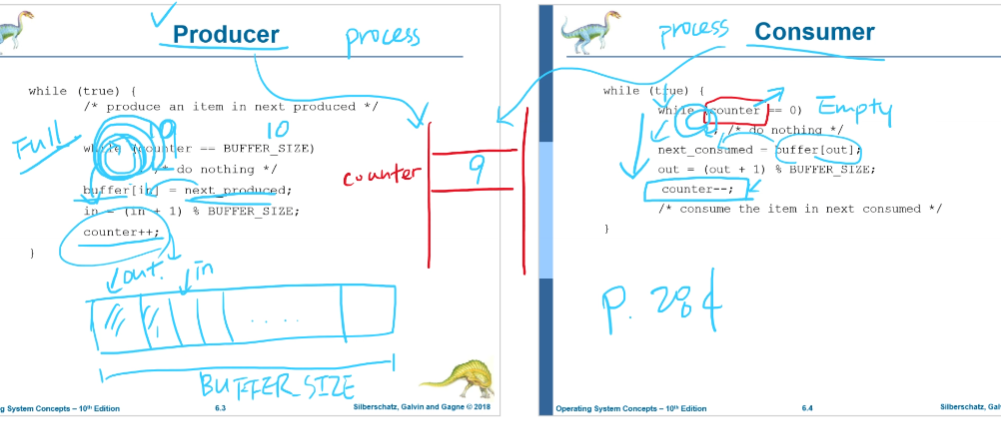
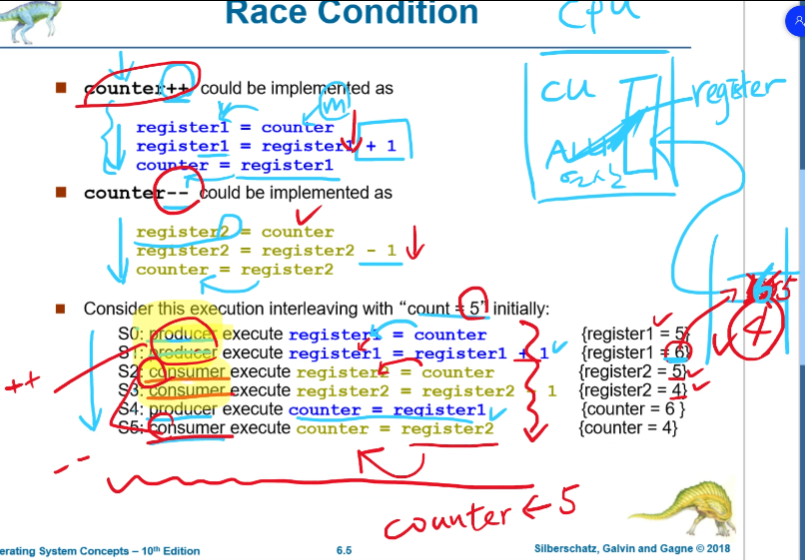
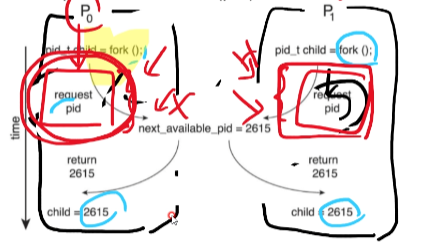
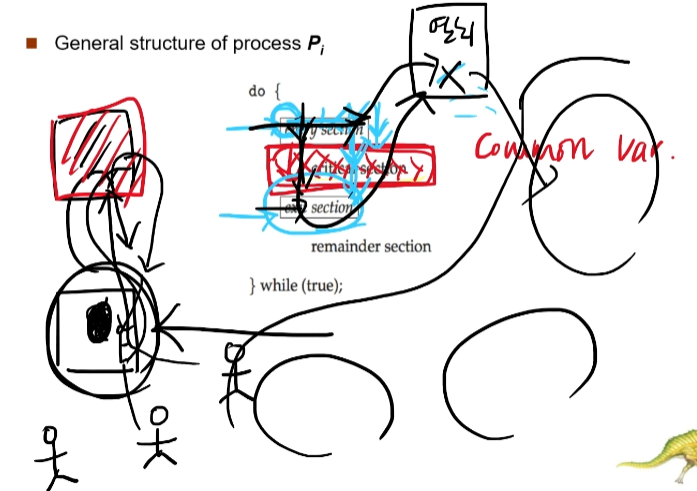
Chapter 6: Process Synchronization; 프로세스 동기화

Synchronization Tolls; 동기화 도구들

* Processes can execute concurrently  
  프로세스는 동시에 실행할 수 있다
  + May be interrupted at any time, partially completing execution  
    언제든지 중단될 수 있으며 부분적으로 완료될 수 있다
* Concurrent access to shared data may result in data inconsistency  
  공유 데이터 동시 접속 시 데이터 불일치가 발생할 수 있다
* Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes  
  데이터의 일관성을 유지하기 위해서는 협력 프로세스의 질서 있는 실행을 보장하는 메커니즘이 필요하다
* Illustration of the problem:  
  Suppose that we wanted to provide a solution to the consumer-producer problem that fills all the buffers.  
  We can do so by having an integer counter that keeps track of the number of full buffers.  
  initially, counter is set to 0. It is incremented by the producer after it produces a new buffer and is decremented by the consumer after it consumes a buffer.

1. Background; 배경  
     
   
2. The Critical-Section Problem; 임계구역 문제
   1. Process P0 and P1 are creating child process using the fork() system call
   2. Race condition on kernel variable next\_available\_pid which represents the next available process identifier (pid)  
      
   3. Unless there is mutual exclusion, the same pid could be assigned to two different processes!  
      상호 배제가 없는 한, 동일한 pid를 두 개의 다른 프로세스에 할당할 수 없다
   4. Consider system of n processes {P0, P1, Pn-1}
   5. Each process has critical section segment of code  
      각 프로세스의 코드에는 크리티컬 섹션이 있다
      1. Process may be changing common variables, updating table, writing file, etc  
         공통변수 변경, 표 갱신, 파일 작성 등
      2. When one process in critical section, no other may be in its critical section  
         한 프로세스가 자기의 크리티컬 섹션에 들어가 있다면 다른 프로세스는 아무도 크리티컬 섹션에 들어갈 수 없다
   6. Critical section problem is to design protocol to solve this
   7. Each process must ask permission to enter critical section in entry section, may follow critical section with exit section, then remainder section  
      각 프로세스는 엔트리 섹션에서 크리티컬 섹션으로 진입하려면 허가를 요청해야한다. 크리티컬 섹션 뒤에는 엑시트 섹션이 따라올 수 있고 코드의 나머지 부분들은 총칭하여 리메인더 섹션이라고 부른다  
      
   8. Solution to Critical-Section Problem
      1. Mutual Exclusion – If process Pi is executing in its critical section, then no other processes can be executing in their critical sections  
         상호 배제 – 만약 프로세스가 크리티컬 섹션에서 실행 중이라면, 어떤 프로세스든 그들의 크리티컬 섹션에서 실행될 수 없다
      2. Progress – If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely  
         진행 – 만약 크리티컬 섹션에서 실행 중인 프로세스가 없고 몇몇 프로세스들이 크리티컬 섹션에 존재하려고 한다면, 다음으로 크리티컬 섹션에 진입할 프로세스의 선정을 무기한 연기할 수 없다
      3. Bounded Waiting – A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted  
         한정된 대기 – 프로세스가 자신의 크리티컬 섹션에 진입하려는 요청을 한 후부터 해당 요청이 허가되기까지 다른 프로세스가 해당 크리티컬 섹션에 진입할 수 있는 횟수에 한계가 있어야 한다
         1. Assume that each process executes at a nonzero speed  
            각 프로세스의 실행 속도가 0이 아닌 것으로 가정한다
         2. No assumption concerning relative speed on the n processes  
            n개 프로세스의 상대 속도에 대한 가정은 없다
3. Peterson’s Solution; Peterson의 해결안
   1. Not guaranteed to work on modern architectures! (But good algorithmic description of solving the problem)  
      현대 구조에서 올바르게 작동한다고 보장할 수 없다
   2. Two process solution
   3. Assume that the load and store machine-language instructions are atomic; that is, cannot be interrupted  
      load와 store 같은 기본적인 기계어를 수행하는 방식이 아토믹하다고 가정; 중단될 수 없다
   4. The two processes share two variables:
      1. int turn;
      2. Boolean flag[2]
   5. The variable turn indicates whose turn it is to enter the critical section
   6. The flag array is used to indicate if a process is ready to enter the critical section. flag[i] = true implies that process Pi is ready!  
      