1. Semaphore 를 사용하는 세 프로세스가 deadlock에 빠지는 상황을 예를 들어 설명하시오.
2. Process A 가 코드를 실행해서 semwait(s1) 실행하고 semWait(s2) 를 실행하기 전에 이벤트가 발생해서 process b로 넘어가고 semwait(s2) 를 실행하고 그 때 또 event 가 발생해서 process C로 넘어가고 semwait(s3) 를 실행하고 semWait(s1) 를 실행하고 다시 process A의 작업이 끝나 semWait(s2) 를 실행하면 s2 가 block 이 되고 process B의 작업이 끝나 semWait(s3) 의 작업을 실행하면 block 되고 process C의 semwait(s1) 이 실행되면 결국 깨워줄수있는 process가 실행이 안되기 때문에 그대로 데드락에 걸리게 된다.
3. Deadlock의 발생조건 4가지를 간단히 설명하시오.

* Mutual exclulsion : 한번에 한 프로세스만 자원 사용 가능
* Hold and wait : 어떤프로세스가 자원을 가지고만 있거나 자원을 하나도 가지고 있고 요청만 하는 경우 절대 데드락 x // 자원을 몇 개만 갖고 몇 개만 요청하면 데드락 걸릴 수 있다.
* No pre-emption : 리소스 할당했는데 강제로 뺏는 경우나 프로그램을 뒤로 되돌리는 경우가 안됨 자원을 할당 받기 전으로 못돌림
* Circular wait : 데드락 발생만 프로세스들 사이에 원을 그리면서 기다리고 있는 Circular wait 관계가 생김

1. Deadlock prevention 기법 세 가지를 간단히 설명하시오.

* 데드락이 걸리는 조건 4가지 중 하나만 안지켜도 데드락 발생 안 함

1. Mutual exclusion : 이건 깰 수 없음 왜냐면 이걸 깬다는 것은 프로그램이 정확하지 않게 실행되는 걸 내버려 두겠다는거
2. Hold and Wait : hold랑 wait 을 둘 다 같이 못하게 하는거 // process 가 프로그램을 실행 시작 할 때 자기가 필요한 자원을 전부다 OS에 요청 다 할당 되면 program 실행하고 다 할당 안되면 계속 기다린다. 실행하면서 자원 wait 할 일 x 시작하기 전에는 자원 하나도 없고 이론적으로 데드락 발생x 컴파일 할 때 필요한 자원의 최대치를 가져오므로 자원낭비임
3. No pre-emption : 프로그램 진행하면서 자원 요청 어떤 자원을 중간에 요청했는데 요구가 받아들여지지 않음 예를 들면 중간에 semwait(s2) 해서 실행하고 또 다른 process가 semwait(s2)를 해서 block 되면 그냥 아예 프로그램을 중단시켜버리고 아예 프로그램 다시 실행시킴 물론 중간과정을 저장했으면 그곳부터 시작할 수 있는데 여튼 이런 일이 발생하는 이유는 process가 공통적으로 어떤 자원을 쓰려고 해서임 but 이런일이 발생하는 경우가 적음!! 큰 문제는 이걸 다 다시 실행한다 했을 때 1시간 작업한거를 다시 요청해야 하고 (자원낭비) 프로그램이 안끝날 가능성이 있다.
4. Circular Wait : 그나마 가장 효율적, linear 하고 만들어주면 됨 모든 자원에 번호를 붙여서 작은거부터 큰거 순으로 자원을 가져온다. 먼저 s1잡은 사람이 무조건 처음 할 테니까 번호순서대로 ( os가 가지고 있는 자원 번호) 대로 실행 os가 순서가 바꼈으면 알아서 자원순대로 실행해줌 3번보다 5번이 앞에 있으면 둘다 가져와서 3,5순대로 실행함 이러면 circular wait 발생 x

* 정말 그런지를 생각해보려면 데드락 걸린 상황 만들어서 보고 거기에서 자원에 번호를 매겨서 말이 되는지 아닌지를 확인해봄

1. Deadlock avoidance기법 두 가지를 간단히 설명 하시오.

Deadlock avoidance 실행하는 과정에서 OS가 계속 관찰하면서 자원을 줄지 프로그램을 실행시킬지 결정 여기에서는 hold and wait 처럼 자원을 최대치로 다 가지고 실행하는게 아니라 얼마나 필요한지 파악만 하고 가지고 시작은 x

1. Process initiation denial

* Process 시작하면 데드락 발생할 것 같다라고 생각이 되면 프로그램 실행 x

1. Resource allocation denial

* Program 실행하면서 자원을 주면 deadlock 발생할 거 같다 라고 생각이 되면 os가 자원을 주지 않는다.
* 어차피 엄청 빨리 실행해서 눈에 잘 보이지 않음 os가 필요한 자원을 안주면 block 되고 block 시켰다가 자원을 줘도 된다 싶을 때에 다시 시작하면서 자원을 가져간다.

1. Deadlock avoidance를 위해 제안된 banker’s algorithm 에 관한 다음 질문에 답하시오.
2. 현재 상태는 안전 상태이다. 현재 상태에서 종료 가능한 process 의 순서를 쓰고, 각 process 의 종료 후 변경되는 Available vector 값을 쓰시오.

* P1 종료 후, available vector 는 1,5,3,2
* P4종료 후, available vector 는 1,6,6,2
* P3 종료 후, available vector 는 2,9,6,6
* P2 종료 후, available vector 는 3,9,6,6
* P5 종료 후, available vector 는 3,9,7,10

1. 만약 현재 상태에서 프로세스 P2가 (0,4,2,0) 만큼의 자원을 추가로 요구한다면, 자원은 즉시 할당 가능한가? 그 이유를 자세히 설명해라.

* P2가 최대로 할당 받을 수 있는 자원은 현재 0,7,5,0 이고 할당해 줄 수 있는 available vector 도 1,5,2,0 이므로 자원을 할당을 하게 된다면 p1은 종료가 가능하지만 나머지 P4 P3 P2 P5 는 종료 할 수 없으므로 데드락에 걸리게 된다. 그렇게 안전하지 않은 상태가 되므로 할당 가능하지 않다.
* Deadlock avoidance 언제나 안전한 상태로 이동하는데

1. Process가 자원을 요청할 때 검사하고
2. 현재 상태 x 이미 자원을 주고 난 상태가 safe 한지를 검사 실제로 주는것은 아니고 줬다고 가정
3. System 상태가 safe 한지 검사한 후 safe 하면 그대로 자원을 주고 safe 하지 않으면 해당 process block 시킴 safe 해질 때까지!!
4. Deadlock detection algorithm 을 간단히 설명 하시오.
5. Allocation matrix 의 한 줄 모두가 0이면 마크한다. 마크는 데드락이 아니라고 그 프로세스를 빼는 것
6. Available vector 를 카피해 임시 available vector W를 만든다.
7. 모든 request 값이 w값보다 작거나 같은 즉, 자원을 주면 종료할 수 있는 process Q 값이 W값보다 작은 프로세스를 찾는다.
8. 이러한 열이 발견되면 즉, 이러한 process 가 발견되면 allocation = request + allocation 을 해준다. 그리고 다시 step3으로 돌아가 이 과정을 반복한다.
9. 모두 데드락이라고 가정했을 때 이런식으로 모든 process 가 마크 되면 데드락은 발생하지 않는다.
10. 하지만 각각의 마크되지 않은 process 가 있다면 이것은 데드락이 된다.

* 데드락을 감지했을 떄의 해결 방법

1. 모든 데드락된 process 들을 강제종료시킨다.
2. 데드락 되기 전의 상태를 주기적으로 저장해놓고 모든 진행상태와 모든 process 들을 전부 데드락 발생되기 전 상태로 돌아간다.
3. 더 이상 데드락된 process 가 없을 때 까지 데드락된 process 들을 하나하나 종료시킨다.
4. 데드락 되기 전의 상태를 주기적으로 저장해놓고 그때로 다시 돌아가서 한명씩 다시 실행 시키면서 데드락을 검사한다.
5. 첫번째 방법은 데드락이 발생하지 않았을 경우 비용이 최소 데드락 발생하는 경우 비용 최대 네번째 방법은 그 반대임!
6. Dining philosopher’s problem 을 간단히 설명 하시오.

* 이 방법은 어떻게 하면 데드락이 발생하지 않게 코드를 짜는 거에 대한 해답이다.
* 철학자들이 원탁에 둘러 앉아 포크를 들고 밥 먹는다 했을 때 양손에 포크를 들어야하는데 오른쪽에 포크를 모두 들고 왼쪾 포크를 쓰기위해서 왼쪽 사람이 포크 내려놓을 떄까지 기다린다. 근데 서로 다 왼쪾 포크를 키다리는 상황이므로 그 상태로 데드락이 걸린다.
* 이 문제를 해결 하기 위한 조건

1. 같은 포크를 동시에 두 철학자가 두는 경우는 없다. ( mutual exclusion)
2. 굶어 죽는 philosopher 는 없어야한다. (deadlock 피하고 starvation 피함)