8장 교착상태 과제

17311017 김민경.

4. 식사하는 철학자란 어떤 문제를 다루기 위한 문제 제기인가? 누가 처음 든 예이며, 이를 공식화 한 사람은 누구인가? 제기된 문제와 해결하는 방법은 무엇인지 설명하라.

식사하는 철학자 문제는 다중 프로그래밍으로 인한 교착상태 문제를 비유적으로 표현한 방법이다. Edsger Dijstra 교수가 1965년 병렬 처리 강의에서 동기화 관련 문제를 제시하였다. 이어 1971년 Coffman이 논문에서 시스템에서 교착 상태를 일으키는 4가지의 필요 충분 조건을 제시하며 이를 공식화 하였다. 문제 상황과 조건은 다음과 같다.

1. 5명의 철학자가 원탁에서 식사한다. 식사 시간은 서로 다를 수 있다.
2. 자리마다 스파게티 하나와 왼쪽과 오른쪽에 포크가 있다.
3. 철학자는 다른 철학자와 대화할 수 없다.
4. 식사를 위해서는 양 옆의 포크를 모두 들고 있어야 한다.
5. 왼쪽 포크를 먼저 든 후 오른쪽 포크를 순서대로 들며, 다른 사람이 먼저 사용 중이면 대기해야 한다. 왼쪽 포크를 들기 전엔 오른쪽 포크도 잡을 수 없다.

여기서 테이블에 앉아있는 5명의 철학자가 동시에 왼쪽 포크를 들고 오른쪽 포크를 집으려고 하는 상황에서 순환구조가 생겨서 무한대기, 교착상태가 발생한다. (5명의 철학자는 영원히 오른쪽 포크를 집을 수 없게 되는 상황) 이를 해결하기 위해서는 마지막 철학자가 왼쪽 포크 대신 오른쪽 포크를 먼저 들도록 하여 해결할 수 있다. 마지막 철학자가 오른쪽 포크를 먼저 들게 되면, 첫번째 철학자는 양쪽 포크를 집유할 수 있어 스파게티를 먹을 수 있으며, 식사를 종료하게 된다. 이어서 그 다음 부터 4번째까지 순환적으로 식사가 가능해지면서, 마지막 철학자까지 식사를 마무리 하며 테이블에 앉아 있는 철학자 모두가 식사를 할 수 있다.

2. 교착상태를 정의해보라.

교착상태는 자원을 소유한 스레드들 사이에서 각 스레드가 다른 스레드가 소유한 자원을 요청하여 모든 스레드가 무한정 대기하는 현상을 말한다.

3. 자원의 종류 두 가지 유형을 들어보라.

컴퓨터 시스템에는 많은 자원들이 존재한다. 이들은 소프트웨어 자원과 하드웨어 자원으로 나눌 수 있다. 먼저 소프트웨어 자원으로는 뮤텍스, 스핀락, 세마포, 파일, 데이터베이스, 파일락 등이 있다. 그리고 하드웨어 자원으로는 프린터, 메모리, 프로세서 등이 있다.

4. 교착 상태가 발생할 필요조건 4가지란 무엇인가?

1) 상호배제 (Mutual Exclusion) - 자원은 한 스레드에게 할당 가능하다.

2) 소유하면서 대기 (Hold and Wait) - 스레드가 한 자원을 소유하면서 다른 자원을 요청하며 대기한다.

3) 강제 자원 반환불가 (No Preemption) - 스레드에게 할당된 자원은 강제로 빼앗지 못한다.

4) 순환대기 (Circular Wait) - 한 그룹의 스레드들 사이에서, 각 스레드는 다른 스레드가 요청하는 자원을 소유하는 순환 고리를 형성한다.

5. 자원할당 그래프에서 사이클 (cycle)은 무엇이며 왜 중요한가?

자원할당 그래프에서 사이클은 스레드들 간 요청/대기로 이루어진 간선들의 순환 고리를 말한다. 컴퓨터 시스템은 실행되는 동안 계속 자원할당 그래프를 유지 갱신하면서 필요한 시점에 자원할당 그래프를 검사하며 교착상태를 발견하고 교착 상태에 빠진 스레드들과 자원들을 알아낸다. 여기서 사이클의 존재여부에 따라 시스템의 교착 상태를 판단하게 되므로 자원 할당 그래프에서 사이클은 중요하다.

6. 교착상태를 다루는 방법 4가지를 간단히 기술하라. 그중에서 가장 많이 사용되는 방법과 그 이유는 무엇인가?

1) 교착상태 예방 (Prevention) - 코프만의 4가지 조건 중 최소 한 가지 이상 성립하지 못하게 하여 교착 상태를 예방한다. 첫째, 상호배제를 없앤다. 둘째, 소유하면서 대기하는 일이 없도록 기다리지 않게한다. 셋째, 강제 자원 반환을 가능하게 하여 선점을 허용한다. 넷째, 순환 대기를 제거한다.

2) 교착 상태 회피 (Avoidance) - 운영 체제가 자원을 할당할 때 교착상태에 빠지지 않을 것이라고 확신하는 경우에만 자원을 할당하는 방법이다. 자원 할당을 요청받았을 때, 순환대기가 발생할 것인지 판단되면 자원을 할당 하지 않음으로써 교착 상태의 발생을 피한다.

3) 교착상태 감지 및 복구 (Detection and Recovery) - 교착상태의 예방이나 회피 전략을 가동하지 않고, 운영체제가 교착 상태를 감지하는 별도의 프로그램을 백그라운드로 구동시켜, 교착 상태에 빠진 스레드 그룹이 있다면 이들을 교착 상태로부터 해제 시키는 방법이다.

4) 교착상태 무시 (Ignore) - 교착 상태에 대해 아무런 대책을 세우지 않고 교착 상태를 무시하는 전략을 말한다.

교착상태를 다루는 4가지 방법 중 교착상태 무시 방법을 가장 많이 사용한다. 교착상태 예방, 회피, 감지 및 복구 방법은 많은 시간과 공간을 필요로 하며 컴퓨터 시스템의 성능을 떨어뜨리기 때문이다. 그리고 범용 시스템에서는 교착상태가 발생한다고 하더라도 파국을 부를만한 작업을 실행시키지 않기 때문이다.

7. 교착상태의 예방책으로 제안된 방법의 문제점은?

첫째, 상호 배제의 조건을 제거하여 모든 자원을 공유 방식으로 허용한다. 하지만 이는 자원에 대한 접근 제어 알고리즘이 복잡해지고 많은 문제를 야기시킬수 있다.

둘째, 강제 자원 반환 불가 조건을 무시하는 방법이다. 모든 자원에 대해 선점을 허용하는 것이다. 더 높은 스레드가 자원을 요청하며 운영체제가 그 자원을 가진 낮은 순위의 스레드로 부터 강제로 자원을 반환하도록 하는 과정에서 자원을 뺏긴 스레드의 상태를 이전 상태로 되돌려야 하기 때문에 설계가 복잡하다는 단점이 존재한다. (하지만, 이는)

셋째, 소유하면서 대기의 조건을 무시하는 방법이다. 하지만 이는 자원을 필요하지 않는 순간에도 가지고 있기 때문에 다른 스레드는 무한 대기 상태 혹은 기아상태에 빠질수 있고 자원이 낭비되며 자원을 효율적으로 이용하는데 한계가 있다.

넷째, 순환 대기를 제거하는 방법이다. 자원에 순서를 부여하여 순환대기를 제거해주는 방법은 실현 가능하면서 자원 이용률도 크게 저하되지 않고 설계도 그리 복잡하지 않아 큰 문제가 존재하진 않는다.

따라서 전반적으로 예방방법은 심각한 자원 낭비가 발생하고 비현실적인 단점이 있기 때문에 예방을 통한 교착상태방지는 잘 사용하지 않는다.

8. 교착상태를 다루는 방법 중 자원을 할당하는 시점에 문제를 해결하려고 하는것은 어떤 것인가?

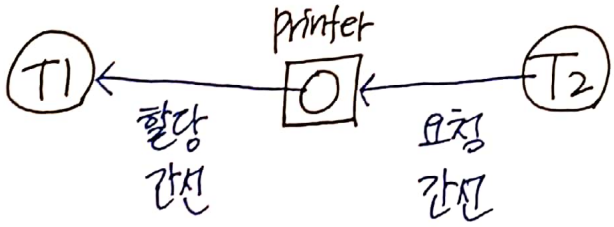
교착 상태 회피 (Avoidance) 방법이다. 회피 방법은 프로세스에게 자원을 할당하려는 시점에서 자원을 할당하는 경우 순환이 발생하는지 판별하여 자원 할당 여부를 판단하여 교착상태를 회피하는 방법이다. 다만, 판별하는 과정에서 자원을 할당할 때마다 순환대기가 발생할 것인지 확인하는 과정에서 필요한 자원 할당량을 인지해야 하는데 이는 현실적으로 구현이 불가능하여 부담이 크다는 단점이 있다.

9. 교착상태의 회피 방법 시, 안전 상태와 불안전 상태는 어떤 상태를 말하는가?

교착상태 회피 방법에서 회피 방법은 각 스레드가 필요로 하는 자원의 개수, 현재 실행중인 각 스레드가 할당받은 자원의 개수, 그리고 시스템 내에 할당가능한 자원의 개수를 토대로 현재 요청된 자원을 할당해도 교착 상태가 발생할 지 판단한다. 회피 방법은 시스템을 계속 감시하여 교착상태를 방지하는 방법인데, 여기서 교착 상태가 발생하지 않을 것 같은 경우 안전한 상태. 발생할 것 같은 경우 불안전한 상태로 나누고, 안전한 상태일 경우 자원을 할당한다. 이 알고리즘을 사용하기 위해서 각 스레드는 필요한 전체 자원의 수를 실행 전에 운영체제에게 알려주어야 하는데, 스레드 실행 전에 필요한 자원의 개수를 아는 것은 사실상 불가능하므로 비현실적인 알고리즘이다. 실행 중인 스레드의 개수는 동적으로 변하기 때문에 미리 스레드의 개수를 정적으로 고정시키는 것 또한 불가능하다.

10. 자원할당 그래프란 어떤 것인가? 한 가지만 사례로 그려보라.

컴퓨터 시스템에서 자원 할당 그래프를 이용하여 교착 상태를 표현하고 이 그래프를 기반으로 교착 상태 예방/회피/감지 등의 알고리즘을 개발한다.
자원 할당 그래프는 다른 그래프와 같이 꼭짓점과 간선으로 이루어지며, 스레드는 원, 자원은 사각형, 자원에서 스레드로 향하는 화살표는 할당 간선, 스레드에서 자원으로 향하는 화살표는 요청 간선이다.
다음은 사례이다.



11. 자원을 요청할 때 마다 교착상태를 탐지하는 것의 단점과 그 해결책은?

시스템에서 상시적으로 백그라운드 프로그램을 이용하여 자원할당 그래프에 순환 대기가 있는지 판단하여 교착상태를 탐지한다. 따라서 이에 대한 해결책으로 회복(recovery) 방법을 사용한다. 그런데 이와 같은 탐지를 위해 실행하는 알고리즘과 교착 상태 해결을 위한 비용이 많이 소모되는 단점이 있다.

12. 교착상태가 발생한 것을 탐지한 후 회복하는 3가지 방법을 제시해보라.

첫째, 스레드 강제 종료이다. 교착 상태에 처한 스레드를 탐지하여 해당 스레드를 제거하여 순환구조를 제거한다. 가장 간단하면서도 효과적인 방법이지만 어떤 스레드를 희생할 지 결정해야 한다. 또한 스레드 제거로 안한 위험도가 높다.

둘째, 자원 강제 선점이다. 교착상태에 처한 스레드의 자원 점유권을 강제로 선점하여 해당 자원을 기다리는 다른 스레드에게 스위칭한다.

셋째, 롤백이다. 롤백은 교착 상태가 발생할 것으로 예측되는 스레드들에 대해 그들의 상태를 주기적으로 저장해 두었다가 교착 상태가 발생하면 가장 최근에 저장했던 상태로 복구시켜 가장 최근에 실행했던 상태로 돌아가게 하는 방법이다. 롤백이후 스레드들이 다시 시작하면 자원이 할당되는 순서가 다르게 되고 교착 상태가 발생하지 않게 되지만, 주기적으로 스레드들의 상태가 저장되어야 하기 때문에 시스템의 시간과 저장공간을 빼앗아 시스템의 성능을 떨어뜨린다는 문제가 있다.

13. Ostrich 알고리즘을 간단히 설명하라. 이 알고리즘이 기본적으로 취하고 있는 전제는 무엇인가?

Ostrich 알고리즘은 교착상태에 대한 특별한 대책을 세우지 않고 문제가 발생하는 경우 시스템 재부팅, 프로세스 kill을 통해 해결하는 방식이다. 이는 시스템에서 교착 상태에 대한 문제가 발생하지 않을 거란 상황을 전제한다. 따라서 언제 발생할지도 모르는 교착 상태에 많은 시간과 비용을 소요하는 대비책이나 해결책을 마련할 필요가 없다고 본다. 데이터 손실이 일어나더라도 이를 감수하여 비용 면에서 이득을 보는 방법이기 때문에 멀티 스레드 응용 프로그램 내에서 자주 사용된다. 하지만 시스템 재시작이나 스레드 강제 종료 등으로 파국이 초래될 수 있는 핵 관련 시스템, 비행기, 미사일, 병원, 환자감시 시스템 등 실시간 시스템에는 부적절하다.

14. 교착 상태 해결을 위해 범용 운영체제인 Unix, Linux, Windows의 기본 해결법은 무엇인가? 그리고 그렇게 하는 이유는 무엇인가?

현재 대다수의 운영체제는 무시 방법을 채용하고 있다. 회피 방식 경우, 언제 사전에 인지하고 있어야하는 정보 (프로세스 수, 자원의 유형 등)을 예측하고 방지한다는 것은 비현실적이다. 그리고 감지의 방법의 경우, 자원의 활용율이 낮다. 따라서 예방/회피/감지 와 같은 방법을 현대 컴퓨터 환경에서 사용하는것은 복잡한 알고리즘이 되며 시스템 부하를 유발할수 있다. 따라서 교착상태에 대한 해결책과 대비책을 수립하는데 소요되는 비용 보다는 프로세스 제거, 재부팅을 통한 해결방법이 소요 비용이 적기 때문에 많은 운영체제들이 무시 방법을 채택하고있다

15. 예제 8-1 을 직접 입력하여 실행해보라. 교착 상태가 발생하면 두 프로세스를 종료시키면 된다. 코드와 실행 결과를 캡쳐하여 함께 제출하라.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

int x=0;
int y=0;
pthread_mutex_t lock1;
pthread_mutex_t lock2;

void *worker1(void* arg){
        pthread_mutex_lock(&lock1);
        printf("%s lock1 locked\n", (char*) arg);
        x++;
        sleep(2);

        pthread_mutex_lock(&lock2);
        printf("%s lock2 locked\n", (char*)arg);
        y++;
        pthread_mutex_unlock(&lock2);
        printf("%s lock2 open\n", (char*) arg);
        pthread_mutex_unlock(&lock1);
        printf("%s lock1 open\n", (char*) arg);

}

void *worker2(void* arg){
        pthread_mutex_lock(&lock2);
        printf("%s lock2 locked\n", (char*)arg);
        y++;
        sleep(2);

        pthread_mutex_lock(&lock1);
        printf("%s lock1 locked\n", (char*)arg);
        x++;
        pthread_mutex_unlock(&lock1);
        printf("%s lock1 open \n", (char*)arg);

        pthread_mutex_unlock(&lock2);
        printf("%s lock open \n", (char*)arg);

}

int main(){

        char *name[] = {"kitae", "hyosoo"};
        pthread_t tid[2];

        pthread_mutex_init (&lock1, NULL);
        pthread_mutex_init (&lock2, NULL);

        pthread_create(&tid[0], NULL, worker1, name[0]);
        pthread_create(&tid[1], NULL, worker2, name[1]);

        pthread_join(tid[0],NULL);
        pthread_join(tid[1],NULL);

        pthread_mutex_destroy(&lock2);
        pthread_mutex_destroy(&lock1);

        printf("x= %d, y= %d \n",x,y);

        return 0;
```