병행 프로세스와 동기화

구미 3반 김이랑

병행 프로세스와 동기화

병행 프로세스

- 메모리에 다수의 프로세스가 같이 존재
- 한 개의 CPU 단일처리 시스템의 병행 프로세스 중 한 개만 실행되지만,

사람이 보기엔 동시처리 같음

- 병행 프로세스는 서로 간 비동기적
  - 다른 프로세스들이 어떤 상태, 어떤 자원, 어디까지 실행됐는지 모름

공유하는 자원 등 있을 시 관리 못하면 문제 발생

상호배제 임계 영역에 한 번에 하나의 프로세스만 들어가게 제어하는 것

경쟁 상태(Race condition)

기아 상태 : 프로세스 속도가 맞물려 임계영역으로 진입X

- 공유 데이터에 서로 접근 시도
- 상호배제, 교착상태, 기아상태 문제 발생 가능

임계 자원(Critical Resource)

- 두개 이상 프로세스가 동시 사용 불가한 자원

임계 영역(Critical Section)

- 접근, 실행 프로그램 코드 부분

- 상호 배제 지켜지기
- 다른 프로세스가 임계영역 진입 막으면 안됨
- 임계 영역 비어 있으면 바로 진입
- 기아 발생하지 않기

상호배제를 위한 하드웨어 기법

https://coding-start.tistory.com/201

인터럽트 : CPU가 특정 기능 수행 중 급하게 다른 일 처리하고자 할 때 사용하는 기능

TAS (TestAndSet)

- 공유 변수 수정하는 동안 인터럽트 발생 억제
- 동시성을 제어
- 원자성, 실행 중 인터럽트 받지 않음
  - 바쁜 대기 / 기아 발생 / 교착 가능성 있음

권한을 얻기 위해 계속 확인 작업 실시, CPU 낭비

수준	방법	종류
고급	소프트웨어로 해결	• 데커 알고리즘 • 크누스 알고리즘 • 램포트의 베이커리 알고리즘 • 핸슨 알고리즘 • 다익스트라 알고리즘
	소프트웨어가 제공 : 프로그래밍 언어와 운영체제 수준에서 제공	• 세마포어 • 모니터
저급	하드웨어로 해결	• TestAndSet(TAS)

이 명령어는 test 와 set을 한 번에 수행하는 기계어(machine instruction)입니다.

```
// target을 검사하고, target 값을 true로 변경
boolean TestAndSet (boolean *target) {    //<- 한번에 수행 (machine instruction
  boolean temp = *target;  // 이전 값 기록
  *target = true;          // true로 설정
  return temp;             // 값 반환
}
```

기계어이기 때문에 원자성(atomicity)을 가지며 실행 중 interrupt를 받지 않아 명령어 수행
(preemption)이 되지 않습니다.

몬소리야

```
boolean lock = flase;

do {                        // 프로세스 Pi의 진입 영역
  waiting[i] = true;            // 프로세스 i를 대기열에 넣음.
  key = true;
  while (waiting[i] && key)         // 대기가 풀리거나 lock이 풀릴 때까지 대기함.
    key = TestAndSet(&lock);        // lock 값을 key에 반환. key와 lock은 같은 값으로 보면 됨.

  waiting[i] = false;           // CS에 접근할 수 있으므로 대기를 풀어줌. 이 코드가 없다면 무한 대기

  // Critical Section

  j = (i + 1) % n;
  while ((j != i) && !waiting[j])  // 대기 중인 프로세스를 찾음
    j = (j + 1) % n;

  if (j == i)                   // 대기 중인 프로세스가 없으면
    lock = false;                   // 다른 프로세스의 진입을 허용할 수 있게 lock을 풀어줌.
  else                          // 대기 프로세스가 있으면 다음 순서로 임계 영역에 진입하게 함.
    waiting[j] = false;             // Pj가 임계 영역에 진입할 수 있도록 대기를 풂.

} while (true);
```

https://yoongrammer.tistory.com/62

상호배제를 위한 소프트웨어 기법

데커 알고리즘

- 두개 프로세스만 상호 배제
- Flag : 누가 CS에 진입할지 알리는 변수
- Turn : 누가 CS에 들어갈 차례인지 알림

https://blog.naver.com/PostView.naver?blogId=ecarooce&logNo=140050543483&redirect=Dlog&widgetTypeCall=true&directAccess=false

```
while (1) {
    ...
    flag[i] = true;              // 임계영역에 들어가려고 시도
    while (flag[j]) {            // Pj가 임계영역에 있는지 조사
        if (turn == j) {         // Pj가 들어갈 기회라면
            flag[i] = false;     // 일단 진입 취소
            while (turn == j) ;  // 순서를 기다림
            flag[i] = true;      // 재진입 시도
        }
    }
    // 임계영역 (critical section)
    ...
    turn = j;                    // 진입 순서 양보
    flag[i] = false;             // 임계영역 사용 완료 지정
        ...
}
```

```
while (1) {
        ...
    flag[i] = true;                    // 임계영역에 들어가려고 시도
    turn = j;                          // 상대방에 진입 기회 양보
    while (flag[j] && turn == j);      // 상대방이 진입하려고 한다면 대기
    // 임계영역(critical section)
    ...
    flag[i] = false;                   // 임계영역 사용 완료 지정
    ...
}
```

피터슨 알고리즘

- 두개 프로세스만 상호 배제
- 상대방에게 진입 기회 양보

```
while (1) {
    …
    choosing[i] = true;              // 번호표 받을 준비
    // 다음 번호를 생성하여 할당
    number[i] = max(number[0], number[1], …,number[n-1]) + 1;
    choosing[i] = false;             // 번호표를 받았음.
    for (j = 0; j < n; j++) {        //모든 프로세스에 대한 번호표 비교루프
      while (choosing[j]);           // 비교할 Pj가 번호표 받을 때까지 대기
       while (number[j]&&(number[j], j)<(number[i], I));
            // Pj가 번호표를 갖고 있고 Pj의 번호표가 Pi의 번호표보다
            // 작거나 또는 번호표가 같을 경우 j가 i보다 작다면
            // Pj의 종료(number[j]=0)까지 대기
    }
    // Critical Section
    …
    number[i] = 0;                   // 사용완료로 번호표 취소
    …
}
```

랜포트의 제과점 알고리즘

- N 개의 프로세스에서 상호배제 가능
- 낮은 번호가 먼저 온거라 먼저 수행
- 한계 대기 조건만 만족하면 됨,
- 반드시 들어온 순으로 실행X

소프트웨어 상호배제 단점

- 실행 부하 큼
- 프로그래머가 실수하면..
- 중복 진입 막기 위해 while..
- CPU 낭비임

https://m.blog.naver.com/hirit808/221785171412

동기화 : 세마포어, / 뮤텍스, 모니터 하나의 스레드

세마포어 : 각 프로세스에 제어 신호를 전달하여 순서대로 작업하도록 하는 기법

- 즉시 자원 사용할지, 사용 중이라면 일정 시간 기다려야함

> 세마포어는 동시에 리소스에 접근할 수 있는 '**허용 가능한 Counter의 갯수**'라고 Counter라고 보면된다. Counter 개수만큼 공유자원에 접근할 수 있다. 이 세마포어 개수에 따라 **1개면 이진 세마포어**, **2개 이상의 경우면 카운팅 세마포어** 라고 불린다.

> - 잠금 함수 : P (try)
> - 잠금 해제 함수 : V (increment)
> - sem < 0 : 프로세스가 처리부분을 처리하고 있다는 뜻이므로 대기

동작이 독립적이라,

잘못 사용하면 문제 생길 수 있음

(교착 상태, 상호 배제)

모니터 : 모니터 내부에 들어간 프로세스에게만 공유 데이터를 접근할 수 있는 기능 제공

- 즉시 자원 사용할지, 사용 중이라면 일정 시간 기다려야함
- 진입은 프로시저 호출로 사용
- 공유자원에 접근하기 위한 키 획득과 자원 사용 후 해제를 모두 처리
  - 세마포어는 직접 키 해제와 공유자원 접근 처리가 필요

세마포어 vs 모니터

- 세마포어는 동기화 함수의 제약 조건을 고려해야하는 반면,
- 모니터는 프로시저를 호출하여 간단히 해결할 수 있다.

뮤텍스 : 이진 세마포어와 유사

- Locking과 Unlocking을 사용
- Lock을 가지면 공유 데이터 접근 가능, 공용 데이터는 Locking된 상태
- Lock에 대한 소유권이 있음 : Lock을 가진 게 반납 가능

세마포어와 뮤텍스의 차이

- 세마포어는 뮤텍스가 될 수 있지만, 뮤텍스는 세마포어가 될 수 없습니다.
- 세마포어는 소유할 수 없으며, 뮤텍스는 소유할 수 있고 소유주가 그에 대한 책임을 집니다.
- 뮤텍스의 경우 뮤텍스를 소유하고있는 스레드가 이 뮤텍스를 해제할 수 있습니다. 하지만, 세마포어는 소유하지 않고 있는 스레드가 세마포어를 해제할 수 있습니다.
- 세마포어는 시스템 범위에 걸쳐있고 파일 시스템 상의 파일 형태로 존재합니다. 하지만, 뮤텍스는 프로세스 범위를 가지고 프로그램이 종료될 때 자동으로 지워집니다.
- 세마포어는 동기화 대상이 여러개 일 때, 뮤텍스는 동기화 대상이 오로지 하나 일 때 사용됩니다.

면접 예상 질문

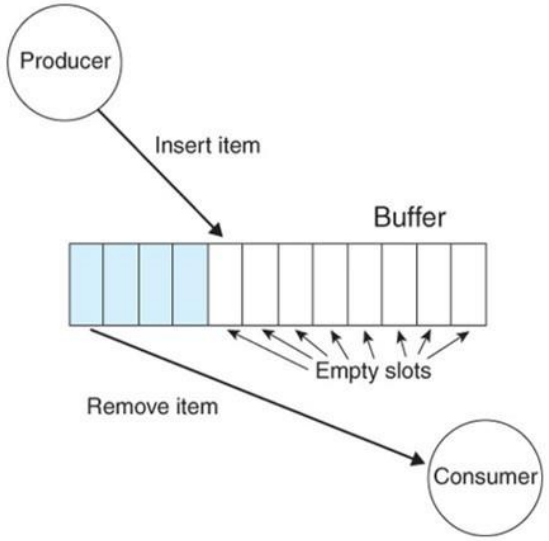
- 동기와 비동기의 차이
- 프로세스 동기화
  - Critical Section
  - 해결책
    - Lock
    - Semaphores
    - 모니터

💡 교착상태 vs 기아상태

- 교착상태는 둘 이상의 프로세스들이 자원을 점유한 상태에서 `서로 다른 프로세스가 점유` 하고 있는 자원을 요구하며 무한정 기다리는 상황입니다.
- 기아상태는 병행 프로세스에서 프로세스가 실행되는데에 필수적인 `자원을 끊임없이 사용하지 못하는 상황` 입니다.

[세마포어(Semaphore) vs 뮤텍스(Mutex) 차이]

뮤텍스는 Locking 메커니즘으로 락을 걸은 쓰레드만이 임계 영역을 나갈때 락을 해제할 수 있습니다. 하지만 세마포어는 Signaling 메커니즘으로 락을 걸지 않은 쓰레드도 signal을 사용해 락을 해제할 수 있습니다. 세마포어의 카운트를 1로 설정하면 뮤텍스처럼 활용할 수 있습니다.



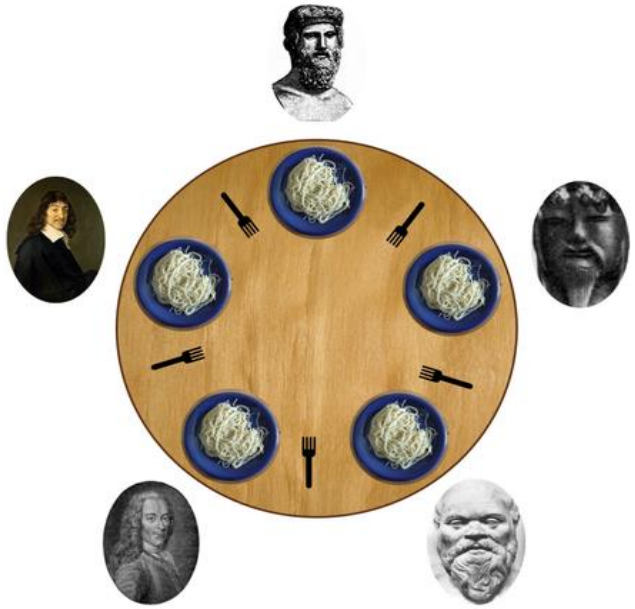
생산자-소비자 문제

- 유한한 버퍼로 인해 발생하는 문제
- 상호배제와 동기화로 해결 가능

https://m.blog.naver.com/hirit808/221785171412

식사하는 철학자 문제

- 한쪽 포크만 선택 : 고착 상태로 인해 멈춤
- 오른쪽 왼쪽 한번에 : 굶어죽는 철학자 발생



해결 방법

- 포크를 많이 두기

(철학자를 줄이거나)

- 홀수철학자는 왼쪽포크 ,

짝수철학자는 오른쪽 포크