운영체제 토론일지

<table>
<tr><td colspan="2">소속 조 : 빈센조</td><td rowspan="2">날짜 : 2021.11.08</td></tr>
<tr><td colspan="2">구성원</td></tr>
<tr><td>학번</td><td>성명</td><td>담당교수 확인</td></tr>
<tr><td>21660021
21960025
21960027</td><td>김승현
신가영
양혜교</td><td></td></tr>
<tr><td colspan="3">토론 주제</td></tr>
<tr><td colspan="3">다양한 상호배제 알고리즘 조사 및 비교</td></tr>
<tr><td colspan="3">토론 내용</td></tr>
<tr><td colspan="3">상호배제 알고리즘이란?
동시 프로그래밍에서 공유 불가능한 자원의 동시 사용을 피하기 위해 사용되는 알고리즘으로, 임계 구역으로 불리는 코드 영역에 의해 구현된다.

방법은 크게 두 가지로 나뉨
① 소프트웨어로 해결
ex) 데커 알고리즘, 램포트의 베이커리 알고리즘 등

② 소프트웨어가 제공
프로그래밍 언어와 운영체제 수준에서 제공한다.
ex) 세마포어, 모니터</td></tr>
</table>

모니터 (우리 조 발표내용)

- 세마포어와 동일한 기능을 제공하지만 세마포어보다 제어가 쉬운 고수준의 동기화 구문
- 프레임워크나 라이브러리 그 자체에서 제공 (C에는 없고, JAVA에는 존재)
- 세마포어와 달리 wait, signal 설정 없이 함수 앞에 synchronized를 붙여주기만 하면 상호배제 하여 함수 작업 수행

[모니터-상호배제 임계구역 구현]

```
synchronized block {
    // 임계영역에 해당하는 코드 블럭을 선언할 때 사용한다.
    // 해당 임계영역에는 모니터 락을 획득해야 진입할 수 있다.
    // 모니터 락을 가진 객체 인스턴스를 지정할 수 있다.
    // 메소드에 선언하면 메소크 블록 전체가 임계영역으로 지정된다.
    // 이 때 모니터락을 가진 객체 인스턴스는 this 객체 인스턴스이다.
}

synchronized (object) {
    // critical section
}
```

참고링크

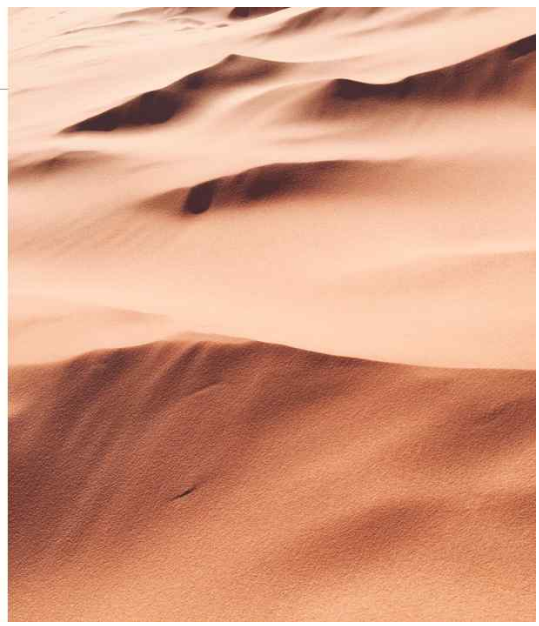
https://coder-in-war.tistory.com/entry/OS-08-%ED%94%84%EB%A1%9C%EC%84%B8%EC%8A%A4-%EB%8F%99%EA%B8%B0%ED%99%94-%EB%AA%A8%EB%8B%88%ED%84%B0Monitor

발표자료

상호배제 기법, 모니터(Monitor)

SW 3B 김승헌
양혜교
신가영

목차 A table of contents.

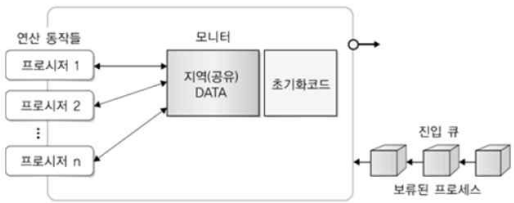


Part 1

모니터의 개요

- 세마포어의 P, V 연산이 프로그램 전체에 퍼져 있으며, 이들 연산이 미치는 영향을 전체적으로 파악하기 쉽지 않아 프로그램 작성이 어려운 단점을 극복하기 위함.
(세마포어와 동일한 기능을 제공하지만 세마포어보다 제어가 쉬운 고수준의 동기화 구문)
- 다수의 병행 프로그래밍 언어에서 구현되어 있다.
(Concurrent-Pascal, Pascal-Plus, Module-2/3, Java 등에서 지원)
- 하나의 프로세스 내의 다른 스레드 간 동기화에 사용된다.
- 모니터는 프레임워크나 라이브러리 그 자체에서 제공된다.(C에는 없고, Java에는 있다)
- 세마포어와 달리 wait, signal 설정 없이 함수 앞에 synchronized를 붙여주기만 하면 상호배제하여 함수 작업 수행

모니터의 구조



하나 이상의 프로시저와
초기화 코드, 공유 데이터로 구성된
소프트웨어 모듈로 이루어진 객체

Part 1

모니터의 개요

모니터의 특징

1. 지역(공유) Data는 모니터의 프로시저를 통해 접근이 가능하다
즉, 외부에서 변수에 대한 직접 접근은 허용되지 않는다.

2. 프로세스는 모니터의 프로시저 중 하나를 호출함으로써 모니터로 들어간다.

3. 한 순간에 오직 하나의 프로세스만이 모니터 내에 존재할 수 있다.
즉, 모니터가 이미 사용 중일 경우 다른 프로세스들은 모니터가 이용 가능해질때까지 대기한다.

- 모니터는 3번째 특징에 의해 상호배제기능을 제공하게 된다.

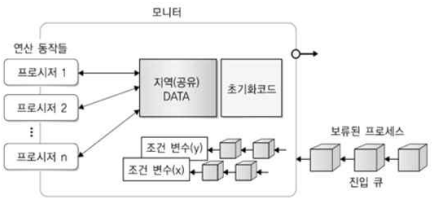
4. 세마포어의 이론적 기반 제공, 타이밍 오류와 P/V 연산 코드 구현

5. 순차적으로만 사용할 수 있는 공유 자원 및 그룹 할당

6. 데이터, 프로시저를 포함하는 병행성 구조

Part 1

모니터의 개요



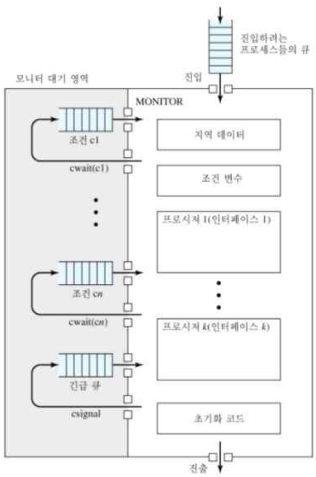
모니터의 조건 변수

- 동기화를 위해 사용된다.
- 모니터 내부에 포함되며, 모니터 내부에서만 접근이 가능함
- 다음 두 인터페이스에 의해서만 접근이 가능하다.
  - o wait 연산
    어떤 프로세스가 signal을 호출할 때까지
    wait을 호출한 프로세스는 연기/중단 된다는 의미
  - o signal 연산
    중단된 프로세스중에서 한 개만 재개하며, 호출 시
    해당 조건 변수와 연관된 큐에서 대기중인 프로세스
    하나를 큐에서 꺼내 모니터로 진입할 수 있도록 함.

    중단된 프로세스가 없으면 효과가 없으며
    연산이 전혀 실행되지 않는 것과 같음.

Part 2

모니터의 동작 개념도/구성 요소



- 지역변수
  : 모니터 내부에서만 접근 가능
- 조건변수
  : 동기화의 부수적 매커니즘 제공
- cwait(c)
  : 호출 프로세스를 조건 c에서 일시 중지
- csignal(c)
  : cwait에 의해 중지된 프로세스 재개
- Next Queue
  : 다른 포레스스가 모니터 내부에 있어서 잠시 대기하는 큐

Part 3

세마포어 / 모니터 상호 관계

구분	세마포어	모니터
주체	OS, 개발자 주체 동시성 지원	프로그래밍 언어 수준 동시성
상호 작용	모니터에 이론적 기반 제공 모니터에 효과적인 기법 제공	타이밍 오류 해결 개발 편의성 보완
특징	s의 타입에 따라 이진/계수 세마포어로 구분	한 시점에 하나의 프로세스만 모니터 내부에서 수행 세마포어와 계산 능력 동일

동기화 구현 사례	Semaphore S;
	P(s); // 감소역할, s-
	임계구역()
	V(s); // 증가역할, s++
		Monitor monitor-name
		{
		Public entry p1(...){}
		Public entry p2(...){}
		}

언어 사례	P, V 연산으로 구현	JAVA의 synchronized Object, .net의 모니터
공통점	동시성 지원을 위한 조정(Coordination) 기능 수행	

세마포어
: 세마포어는 상호 배제/동기화를 프로그래머가 담당, 주의 필요

모니터
: 객체 단위의 설계가 이루어지기 때문에 설계단계에서 상세 설계 필요

동시성
: 필수적인 프로그래밍 요소지만 높은 수준의 프로그래밍 지원 필요

Part 4

유한버퍼에서 모니터를 이용한 생산자/소비자 문제 해결

```
/* 생산자/소비자 프로그램 */
monitor boundedbuffer;
char buffer [N];                    /* N개의 문자가 저장될 수 있는 버퍼 */
int nextin, nextout;                /* 버퍼 포인터 */
int count;                          /* 버퍼 내부에 추가된 문자수 */
cond notfull, notempty;             /* 동기화를 위한 조건 변수 */

void append(char x)
{
   if (count == N)                  /* 버퍼에 문자가 가득 찬 경우, 오버플로 방지 */
     cwait(notfull);
   buffer[nextin] = x;
   nextin = (nextin + 1) % N;
   count++;                         /* 버퍼에 문자 하나를 추가 */
   csignal(notempty);               /* not empty 조건 변수에서 대기하고 있는 프로세스를 깨움 */
}
void take(char x)
{
   if (count == 0)                  /* 버퍼가 빈 경우, 언더플로 방지 */
      cwait(notempty);
   x = buffer[nextout];
   nextout = (nextout + 1) % N;
   count--;                         /* 버퍼에서 문자 하나를 삭제 */
   csignal(notfull);                /* not full 조건 변수에서 대기하고 있는 프로세스를 깨움 */
}
{                                   /* 모니터 몸체(body) */
   nextin = 0;                      /* 버퍼 변수 초기화 */
   nextout = 0;
   count = 0;
}
```

```
void producer()
{
   char x;
   while(true)
   {
      produce(x);
      append(x);
   }
{
void consumer()
{
   char x;
   while (true)
   {
     take(x);
     consume(x);
   }
}
void main()
{
   parbegin(producer, consumer);
}
```

다른 조 발표내용

[데커 알고리즘]

- 공유 메모리를 사용하여 두 프로세스가 하나의 자원을 혼란 없이 공유할 수 있게 함
- 교착 상태, 무기한 연기를 방지
- 바쁜 대기와 프로세스가 임계 영역을 떠나 프로세스 하나 이상이 대기하는 경우(기아 상태) 발생

[스핀 락]

- 임계구역에 진입이 불가능할 때 진입이 가능할 때까지 루프를 돌아 재시도하는 방식으로 구현된 락
- lock 변수를 활용하여 임계 영역의 입출력을 제어함
- 임계영역의 실행 시간이 짧은 경우 효과적임

[피터슨 알고리즘]

- 임계구역과 나머지 구역을 번갈아 가며 실행하는 두 개의 프로세스로 한정하는 방식

[세마포어]

- 현재 공유자원에 접근할 수 있는 스레드의 개수를 두어 상호배제를 달성하는 알고리즘
- P연산(임계 영역에 들어갈 때), V연산(임계 영역에서 나올 때) 활용
- 바쁜 대기 문제가 해결됨
- P연산과 V연산이 생략될 경우 상호 배제가 진행되지 않거나 교착 상태에 빠질 수 있음

[램포트의 베이커리 알고리즘]

- 각 스레드에 티켓 번호를 부여하여 충돌 없이 임계 영역에 진입하고 빠져나오는 원리로 구현됨
- 티켓 번호가 낮을수록 임계 영역에 대한 우선순위를 가짐
- 티켓 번호가 같을 경우 스레드 번호가 낮을수록 우선순위를 가짐