운영체제 토론일지

소속 조 : 빈센조		날짜 : 2021.11.15
구성원		
학번	성명	담당교수 확인
21660021 21960025 21960027	김승현 신가영 양혜교	

토론 주제

다양한 상호배제 알고리즘 속도 비교

토론 내용

- 모니터 알고리즘 실행 시간을 구한 뒤 다른 상호배제 알고리즘과 비교 분석해보자

- 모니터
  - f(n) = O(n)
  - 세마포어보다 제어가 쉬운 고수준의 동기화 구문
  - wait, signal 설정 없이 함수 앞에 synchronized를 붙여주기만 하면 상호 배제 하여 함수 작업 수행
- 램포트의 베이커리 알고리즘
  - f(n) = O(n^2)
  - 각 스레드에 티켓 번호를 부여하여 충돌 없이 임계영역에 진입하여 빠져 나옴
- 세마포어 알고리즘
  - f(n) = O(n)
  - 공유 자원에 접근할 수 있는 스레드의 개수를 두어 상호배제 실행
  - P연산(임계 영역에 들어갈 때), V연산(임계 영역에 나올 때) 활용

```
package monitor;

import java.util.Scanner;

public class MonitorTest {
        private static int val1 = 0;
        private static int thread = 0;
        synchronized public static void addsafe1() {
            val1 = val1 + 1;
        }
        public static void main(String[] args) throws
InterruptedException {
            Scanner scan = new Scanner(System.in);
            System.out.print("스레드 개수 입력 : ");
            thread = scan.nextInt();
             Runnable test1 = () -> {
                     addsafe1();
             };
             Thread[] add1 = new Thread[thread];
             long beforetime = System.currentTimeMillis();
             for (int i = 0; i < thread; i++){
                 add1[i] = new Thread(test1);

                 add1[i].start();
             }
             for (int i = 0; i < thread; i++){
                 add1[i].join();
             }
             long aftertime = System.currentTimeMillis();
             long time = aftertime - beforetime;
             System.err.println("실행 시간:"+ time);
        }
    }
```

발표자료

상호배제 알고리즘 속도 비교

SW 3B 21660021 김승현
SW 3B 21960025 신가영
SW 3B 21660027 양혜교

©SeabyeolYu. Seabyeol's PowerPoint

목차 A table of contents.

1 모니터 알고리즘

2 램포트의 베이커리 알고리즘

3 세마포어 알고리즘

4 성능 비교

Part 1

모니터 알고리즘

```java
package monitor;
import java.util.Scanner;

public class Monitor3 {
	private static int val1 = 0;
	private static int thread = 0;
	synchronized public static void addsafe1() {
		val1 = val1 + 1;
	}
	public static void main(String[] args) throws InterruptedException {
		Scanner scan = new Scanner(System.in);
		System.out.print("스레드 개수 입력 : ");
		thread = scan.nextInt();

		Runnable test1 = () -> {
			addsafe1();
		};
		Thread[] add1 = new Thread[thread];
		long beforetime = System.currentTimeMillis();
		for (int i = 0; i < thread; i++)
		{
			add1[i] = new Thread(test1);

			add1[i].start();
		}
		for (int i = 0; i < thread; i++)
		{
			add1[i].join();
		}
		long aftertime = System.currentTimeMillis();
		long time = aftertime - beforetime;
		System.err.println("실행 시간:"+ time);
	}
}
```

모니터의 특징

- f(n) = O(n)
- 세마포어보다 제어가 쉬운 고수준의 동기화 구문
- wait, signal 설정 없이 함수 앞에 synchronized를 붙여주기만 하면 상호배제 하여 함수 작업 수행

©Saebyeol Yu, Saebyeol's PowerPoint

Part 2

램포트의 베이커리 알고리즘

- f(n) = O(n^2)
- 각 스레드에 티켓 번호를 부여하여 충돌없이 임계 영역에 진입하고 빠져나오는 원리로 구현

```java
package os.bakery;
import java.util.Scanner;

public class Bakery {
	private static int thread = 0;

	public static void main(String[] args) {

		Scanner scan = new Scanner(System.in);
		System.out.print("스레드 개수 입력 : ");
		thread = scan.nextInt();

		boolean choosing[] = new boolean[thread];
		int ticket[] = new int[thread];

		long beforetime = System.currentTimeMillis(); //코드 실행 전 시간

		for(int i = 0; i < ticket.length; i++) {
			Bakery_Thread bakery_thread = new Bakery_Thread(choosing, ticket);
			bakery_thread.start();
			try{
				bakery_thread.join();
			}catch(Exception e) {}
		}

		long aftertime = System.currentTimeMillis(); //코드 실행 후 시간
		long time = aftertime - beforetime; //코드 실행전후 시간차
		System.err.println("실행 시간 : " + time);
	}
}
```

```java
package os.bakery;

public class Bakery_Thread extends Thread {
	boolean choosing[];	//스레드들이 가진 티켓의 번호를 기록하는 배열
	int[] ticket;		//각 스레드의 티켓 값을 0으로 초기화

	public Bakery_Thread(boolean[] choosing, int[] ticket) {
		this.choosing = choosing;
		this.ticket = ticket;
	}
	@Override
	public void run() {
		int x = threadNumber(ticket);	//현재 스레드 번호 저장함
		choosing[x] = true;			//티켓 선택 프로세스를 시작함
		ticket[x] = maxValue(ticket) + 1;
		choosing[x] = false;			//티켓 선택 프로세스를 마침

		for( int i = 0; i < ticket.length; i++) {
			if( i == x ) {
				continue;
			}
			while( choosing[i] != false ) {
			};
			while( ticket[i] != 0 && ticket[i] < ticket[x]) {
			};
			if( ticket[i] == ticket[x] && i < x ) {
				while( ticket[i] != 0) {
				};
			}
		}
			ticket[x] = 0;
	}
```

©Saebyeol Yu, Saebyeol's PowerPoint

Part 3

세마포어 알고리즘

```java
package os.semaphore;

import java.util.Scanner;
import java.util.concurrent.Semaphore;

public class Sema_1 {
   public static int value, sum = 0;
   public static long startTime;

   public static void main(String[] args) {
      final Work work = new Work(3);
      Thread t;

      Scanner sc = new Scanner(System.in);
      System.out.println("스레드 개수를 입력하세요 : ");

      value = sc.nextInt();
      startTime = System.currentTimeMillis();
      for(int i=1; i<=value; i++) {
         t = new Thread(new Runnable() {
            public void run() {
               work.use();
            }
         });
         t.start();
      }
   }
}
```

```java
class Work{
   private Semaphore semaphore;
   private int maxT;

   public Work(int maxT) {
      this.maxT = maxT;
      this.semaphore = new Semaphore(maxT);
   }

   public void use() {
      try{
         semaphore.acquire();
         Sema_1.sum += 1;

         semaphore.release();

         if(Sema_1.sum==Sema_1.value) {
            System.out.println("실행 시간 : " + (System.currentTimeMillis() - Sema_1.startTime));
         }
      }catch(InterruptedException e) {
         e.printStackTrace();
      }
   }
}
```

세마포어 알고리즘의 특징

- f(n) = O(n)
- 현재 공유자원에 접근할 수 있는 스레드의 개수를 두어 상호배제를 달성하는 알고리즘
- P연산(임계 영역에 들어갈 때), V연산(임계 영역에서 나올 때) 활용

©SaebyeolYu. Saebyeol's PowerPoint

Part 4

성능 비교

모니터	스레드 개수 입력 : 1 실행 시간:1	스레드 개수 입력 : 100 실행 시간:8	스레드 개수 입력 : 1000 실행 시간:75	스레드 개수 입력 : 1500 실행 시간:106	스레드 개수 입력 : 2000 실행 시간:142
베이커리	스레드 개수 입력 : 1 실행 시간 : 1	스레드 개수 입력 : 100 실행 시간 : 44	스레드 개수 입력 : 1000 실행 시간 : 162	스레드 개수 입력 : 1500 실행 시간 : 244	스레드 개수 입력 : 2000 실행 시간 : 317
세마포어	스레드 개수 입력 : 1 실행 시간 : 1	스레드 개수 입력 : 100 실행 시간 : 9	스레드 개수 입력 : 1000 실행 시간 : 73	스레드 개수 입력 : 1500 실행 시간 : 120	스레드 개수 입력 :· 2000 실행 시간 : 140

∴ 처리속도 : 베이커리 << 세마포어 ≒ 모니터

©SaebyeolYu. Saebyeol's PowerPoint

다른 조 발표내용

[박부성 & 구본성 조]

- N개의 스레드를 사용하는 상호배제 알고리즘들의 속도와 효율성을 비교하여 상황별로 어떤 알고리즘이 우수한지 조사
- 램포트의 베이커리 알고리즘
  - 가장 간단한 n-thread 상호 배제 알고리즘
  - 교착상태 무기한 연기 방지
- 다익스트라 알고리즘
  - 이중 for문 구조로 동작
- 세마포어 알고리즘
  - P/V 연산을 통한 상호 배제
- 처리 속도 : 베이커리 = 다익스트라 << 세마포어

[이세영 & 황호현 조]

- 세마포어
  - 현재 공유 자원에 접근 가능한 쓰레드의 개수를 두어 상호배제를 달성하는 기법
  - 임계영역에 들어가려 할 때 acquire로 P연산 수행, 임계영역을 나오려 할 때 release로 V연산 수행
- 베이커리 알고리즘과 세마포어 알고리즘 비교
  - 베이커리 : 100개의 스레드를 실행 시 32ms 소요 , 1000개의 스레드를 실행 시 319ms 소요
  - 세마포어 : 100개의 스레드를 실행 시 16ms 소요, 1000개의 스레드 소요 시 106ms 소요
- 처리 속도 : 베이커리 << 세마포어

[허원석 & 조은새 조]

- 데커 알고리즘
  - 공유 메모리를 사용하여 두 프로세스가 하나의 자원을 혼란 없이 공유할 수 있게 함
  - 교착상태, 무기한 연기 방지
- 피터슨 알고리즘
  - 임계구역과 나머지 구역을 번갈아 가며 실행하는 두 개의 프로세스로 한정하는 방식
- 원자성을 보장해주는 타입을 사용하여 알고리즘 구현
  - Lock을 확보하는 과정의 '원자성'을 보장하는 것으로 Lock을 확보하는 과정에서 다른 스레드의 개입을 막아주는 것
  - Operation System의 CAS(Compare And Swap) Operation을 사용

[김찬혁 & 송지현 조]

- 데커의 알고리즘
  - flag와 true라는 변수로 임계영역에 들어갈 프로세스를 결정하는 방식
  - flag값은 프로세스 중 임계 구역에 들어가길 원하는지 나타내는 변수
  - true 변수는 누가 임계영역에 들어갈 차례인지 나타내는 변수
  - 20000개의 스레드가 돌아갈 때 걸리는 시간 : 15
- 피터슨의 알고리즘
  - 데커 알고리즘과 상당히 유사하지만 상대방에게 진입 기회를 양보한다는 차이가 있음
  - 20000개의 스레드가 돌아갈 때 걸리는 시간 : 1 4
- 램포트의 베이커리 알고리즘
  - 빵집을 비유로 고객들에게 번호를 부여하여, 번호표를 기준으로 먼저 처리해야할 일들의 우선순위를 부여
  - 모든 고객들은 맨처음 번호표를 부여 받고 자기 순서가 올 때 까지 대기, 이로서 두 명 이상의 클라이언트들이 공유자원에 접근하지 못하게 함
  - 20000개의 스레드가 돌아갈 때 걸리는 시간 : 7 1
- 처리 속도 : 베이커리 < 데커 <= 피터슨

[박동환 조]

- 스핀락
  - 임계 구역에 진입이 불가능할 때 진입이 가능할 때까지 루프를 돌면서 재시도하는 방식으로 구현된 락을 가리킴
  - 임계 구역 진입 전까지 루프를 계속 돌고 있기 때문에 busy waiting이 발생
- 뮤텍스
  - 자원에 대한 접근을 동기화하기 위해 사용되는 상호 배제 기술
  - Locking 메커니즘으로 락을 걸은 스레드만이 임계영역을 나갈 때 락을 해제
  - 권한을 획득할 때 가지 busy waiting 상태에 머무르지 않고 sleep 상태로 들어감
- 세마포어
  - 멀티프로그래밍 환경에서 공유된 자원에 대한 접근을 제한하는 방법