



# REPORT

## 운영체제 과제 #3

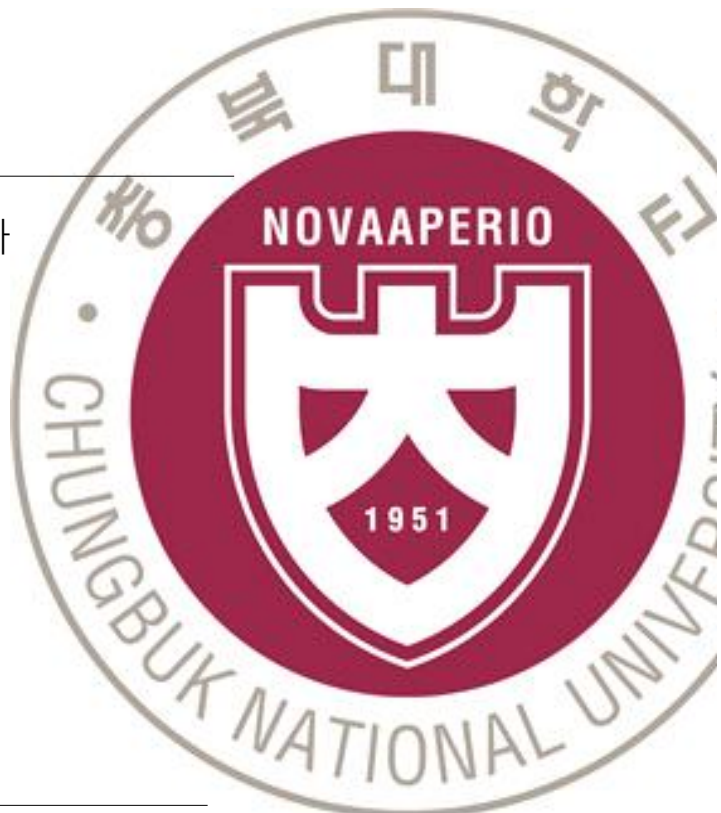
제출일 2022.04.07

학과 소프트웨어학과

학번 2020039009

이름 차현아

담당교수 이건명 교수님



## 4장. 쓰레드

### 1. POSIX 쓰레드 코드

```
#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>
int sum;
void *runner(void *param); // 함수단위 쓰레드
int main(int argc, char** argv)
{
    pthread_t tid;
    pthread_attr_t attr;
    if (argc != 2) {
        fprintf(stderr, "Usage: a.out <integer value>\n");
        return -1;
    }
    if (atoi(argv[1]) < 0) {
        fprintf(stderr, "%d must be >= 0 \n", atoi(argv[1]));
        return -1;
    }
    pthread_attr_init(&attr); // 속성구조체 생성 및 초기화
    pthread_create(&tid, &attr, runner, argv[1]); // 새로운 쓰레드 생성
    pthread_join(tid, (void **)NULL); // tid를 가진 쓰레드 기다리기
    printf("sum = %d\n", sum);
}

void *runner(void *param)
{
    int i, upper = atoi(param);
    sum = 0;
    for (i = 1; i <= upper; i++)
        sum += i;
    pthread_exit(0);
}
```

### 실행 결과

```
chahyeona@chahyeona-virtual-machine:~/os_homework$ ./a.out
Usage: a.out <integer value>
chahyeona@chahyeona-virtual-machine:~/os_homework$
```

## 2. 윈도우 스레드

```

DWORD WINAPI ThreadFunction(void* arg) // 이 함수를 스레드로 만들
{
    int i;
    for (i=0; i < 5; i++) {
        Sleep(500);
        printf("Running Threads %d \n",i);
    }
    return 0;
}

```

```

int main( )
{
    HANDLE hThread;
    DWORD dwThreadId , dw;
    hThread = (HANDLE)_beginthreadex(NULL,0,
    (unsigned int(__stdcall*)(void*))ThreadFunction, NULL, 0, (unsigned*)&dwThreadId);
    if (hThread ==0) {
        puts("_beginthreadex() error");
        exit(1);
    }

    printf("Generated Thread Handle : %d\n",hThread);
    printf("Generated Thread ID : %d\n",dwThreadId);
    dw = WaitForSingleObject(hThread,3000); // 스레드가 종료할 때까지 대기 -> 3초 초과시 종료
    if (dw == WAIT_FAILED) {
        puts("Abnormal termination of thread");
        exit(1);
    }
    else {
        printf("Exit main function, %s exit\n", (dw==WAIT_OBJECT_0)?"Normal":"abnormalities");
    }
    return 0;
}

```

## 실행 결과

```

> Executing task: cmd /C 'c:\Users\user\
Generated Thread Handle : 224
Generated Thread ID : 5576
Running Threads 0
Running Threads 1
Running Threads 2
Running Threads 3
Running Threads 4
Exit main function, Normal exit

```

## 3. 리눅스 쓰레드

```

#define _GNU_SOURCE
#include<signal.h>
#include<unistd.h>
#include<sys/wait.h>
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#include<linux/sched.h>
#include<sys/utsname.h>
#include<pthread.h>
#define STACK_SIZE (1024 * 1024)
#define errExit(msg) do { perror(msg); exit(EXIT_FAILURE); } while (0)
static int childFunc(void *arg) // 함수단위 쓰레드
{
    struct utsname uts;
    // child의 uts네임스페이스에서 호스트명 설정
    if (sethostname(arg, strlen(arg)) == -1)
        errExit("sethostname");
    // 호스트 이름 검색
    if (uname(&uts) == -1)
        errExit("uname");
    printf("uts.nodename in child: %s\n", uts.nodename);
    sleep(200);
    return 0;
}

```

```

int main(int argc, char *argv[])
{
    char *stack; // 스택 버퍼 시작
    char *stackTop;
    pid_t pid;
    struct utsname uts;
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <child-hostname>\n", argv[0]);
        exit(EXIT_SUCCESS);
    }
    stack = malloc(STACK_SIZE); //스택 할당
    if (stack == NULL)
        errExit("malloc");
    stackTop = stack + STACK_SIZE;
    pid = clone(childFunc, stackTop, CLONE_NEWUTS | SIGCHLD, argv[1]);
    if (pid == -1)
        errExit("clone");
    printf("clone() returned %ld\n", (long) pid);
    sleep(1); // 호스트 이름 변경 시간 제공
    if (uname(&uts) == -1)
        errExit("uname");
    printf("uts.nodename in parent: %s\n", uts.nodename);
    if (waitpid(pid, NULL, 0) == -1)
        errExit("waitpid");
    printf("child has terminated\n");
    exit(EXIT_SUCCESS);
}

```



## 리눅스 쓰레드 결과

```

chahyeona@chahyeona-virtual-machine:~/os_homework$ ./a.out
Usage: ./a.out <child-hostname>
chahyeona@chahyeona-virtual-machine:~/os_homework$

```

## 4. 자바 쓰레드1 코드

```

1  import java.lang.*;
2  public class Sum
3  {
4      private int sum;
5      public int getSum() // sum값 읽어오기
6      { return sum; }
7      public void setSum(int sum) // sum값 바꾸는 접근자함수
8      { this.sum = sum; }
9  }
10
11  class Summation implements Runnable
12  {
13      private int upper;
14      private Sum sumValue;
15      public Summation(int upper, Sum sumValue) {
16          this.upper = upper;
17          this.sumValue = sumValue;
18      }
19      public void run() { // 쓰레드로 동작
20          int sum = 0;
21          for (int i = 0; i <= upper; i++) sum += i;
22          sumValue.setSum(sum);
23      }
24  }

```

```

> public class Main {
> @ public static void main(String[] args) {
>     if (args.length > 0) {
>         if (Integer.parseInt(args[0]) < 0)
>             System.err.println(args[0] + " must be >= 0.");
>         else {
>             Sum sumObject = new Sum();
>             int upper = Integer.parseInt(args[0]);
>             // Summation 클래스 내부에 run()이 포함되어있음 - 쓰레드 클래스 객체 생성
>             Thread thrd = new Thread(new Summation(upper, sumObject));
>             thrd.start(); // run() 직접 호출되는 것이 아니라 start()메소드를 통해서 내부적으로 호출됨
>             try {
>                 thrd.join(); // 대기
>                 System.out.println("The sum of " + upper + " is " + sumObject.getSum());
>             } catch (InterruptedException ie) {}
>         }
>     }
>     else
>         System.err.println("Usage: Summation <integer value>");
> }

```

## 자바 쓰레드1 실행화면

```

Main x
"C:\Program Files\Java\jdk-16.0.2\bin\java"
Usage: Summation <integer value>
Process finished with exit code 0

```

```

Main x
"C:\Program Files\Java\jdk-16.0.2\bin\java"
The sum of 5 is 15
Process finished with exit code 0

```

## 5. C# 쓰레드 코드

```

using System;
using System.Threading;
namespace ConsoleApplication
{
    2 references
    public class Top
    {
        2 references
        private int limit = 0;
        1 reference
        public void sayHello() // 쓰레드
        {
            while (limit < 10)
            {
                Thread th = Thread.CurrentThread;
                Console.WriteLine("Thread " + th.GetHashCode() + ": " + limit++);
            }
        }
    }
}

```

```

class SimpleThreadTest
{
    0 references
    static void Main(string[] args)
    {
        Top t = new Top();
        ThreadStart ts = new ThreadStart(t.sayHello); // 객체 생성
        Thread thread = new Thread(ts);
        thread.Start();
        Console.WriteLine("Thread " + Thread.CurrentThread.GetHashCode() + " Main terminated");
    }
}

```

## C# 쓰레드 결과

```

PS C:\Users\user\Desktop\Test> dotnet run
Thread 1 Main terminated
Thread 4: 0
Thread 4: 1
Thread 4: 2
Thread 4: 3
Thread 4: 4
Thread 4: 5
Thread 4: 6
Thread 4: 7
Thread 4: 8
Thread 4: 9

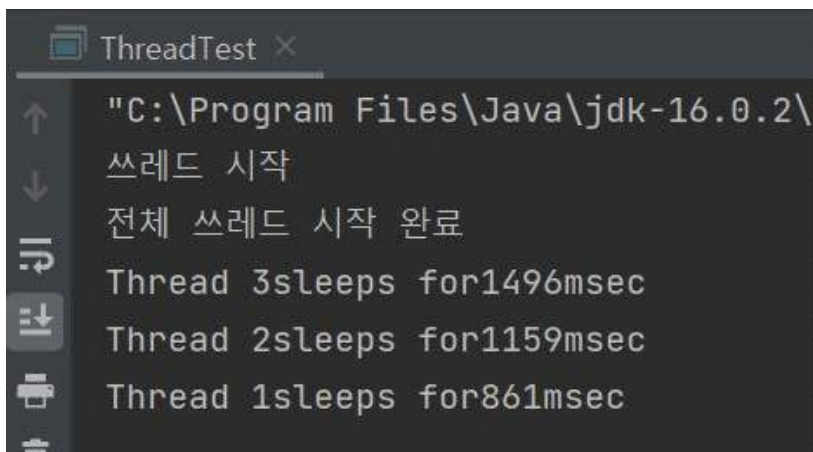
```

## 6. 자바 쓰레드2 코드

```
1 public class ThreadTest
2 {
3     public static void main(String[] args) {
4         // 3개의 쓰레드 객체 - 멀티스레드 프로그래밍
5         PrintThread thread1 = new PrintThread( name: "Thread 1");
6         PrintThread thread2 = new PrintThread( name: "Thread 2");
7         PrintThread thread3 = new PrintThread( name: "Thread 3");
8         System.out.println("쓰레드 시작");
9         thread1.start(); // 순차진행 X , 1-2-3 차례대로 나오지않을수도 있음
10        thread2.start();
11        thread3.start();
12        System.out.println("전체 쓰레드 시작 완료");
13    }
14 }
```

```
1 public class PrintThread extends Thread
2 {
3     private int sleepTime;
4     public PrintThread(String name) {
5         super(name);
6         sleepTime = (int) (Math.random() * 3001);
7     }
8     public void run() { // 쓰레드로 동작
9         try {
10            System.out.println(getName() + "sleeps for" + sleepTime + "msec");
11            Thread.sleep(sleepTime); ;
12        }
13        catch (InterruptedException exception) { // 예외처리
14            exception.printStackTrace();
15        }
16    }
17 }
```

## 자바 쓰레드2 결과



```
ThreadTest x
"C:\Program Files\Java\jdk-16.0.2\
쓰레드 시작
전체 쓰레드 시작 완료
Thread 3sleeps for1496msec
Thread 2sleeps for1159msec
Thread 1sleeps for861msec
```

## 5장. 비동기병행실행

### 7. 생산자-소비자 관계 멀티 쓰레딩 코드(p.7)

```
public interface buffer {
    public void set(int value);
    public int get();
}
```

```
public class UnsynchronizedBuffer implements buffer {
    private int buffer = -1;
    public void set(int value) { // 현재 실행 쓰레드의 저장한 값 출력
        System.err.println(Thread.currentThread().getName() + " 저장한 값 : " + value);
        buffer = value; // 버퍼에 저장한 값 넣어주기
    }
    public int get() { // 현재 실행 쓰레드의 읽은 값 출력
        System.err.println(Thread.currentThread().getName() + " 읽은 값 : " + buffer);
        return buffer;
    }
}
```

```
// 동기화 하지않고 공유객체를 변경하는 쓰레드를 시작하는 클래스
public class SharedBufferTest {
    public static void main(String[] args) {
        buffer sharedLocation = new UnsynchronizedBuffer( );
        Producer1 producer = new Producer1(sharedLocation); // 쓰레드 객체
        Consumer1 consumer = new Consumer1(sharedLocation); // 쓰레드 객체
        producer.start(); // 쓰레드이므로 각각 개별동작함
        consumer.start();
    }
}
```

```
public class Producer1 extends Thread{
    private buffer sharedLocation;
    public Producer1(buffer shared) {
        super( name: "생산자");
        sharedLocation = shared;
    }
    public void run() { // 쓰레드로 동작
        for (int count = 1; count <= 4; count++) { // 4번 반복
            try {
                Thread.sleep((int)(Math.random( )*3001)); // 기다린다
                sharedLocation.set(count); // 공유공간에 값 저장
            }
            catch (InterruptedException exception) {
                exception.printStackTrace( );
            }
        }
        System.out.println("--> " + getName( )+ " 종료!");
    }
}
```



```
public class Consumer1 extends Thread{
    private buffer sharedLocation;
    public Consumer1(buffer shared) {
        super( name: "소비자");
        sharedLocation = shared;
    }
    public void run() {
        int sum = 0;
        for (int count = 1; count <= 4; count++) { // 4번 반복
            try {
                Thread.sleep((int)(Math.random( )*3001)); // 기다린다
                sum += sharedLocation.get( ); // 출력되는 값을 sum에 저장한다
            }
            catch (InterruptedException exception) {
                exception.printStackTrace( );
            }
        }
        System.out.println(getName( ) + " 읽은 값 합계: " + sum);
        System.out.println("--> " + getName( ) + " 종료!"); }
}
```

## 생산자-소비자 관계 멀티 쓰레딩 실행결과

```
"C:\Program Files\Java\jdk
생산자 저장한 값 : 1
생산자 저장한 값 : 2
소비자 읽은 값 : 2
생산자 저장한 값 : 3
생산자 저장한 값 : 4
--> 생산자 종료!
소비자 읽은 값 : 4
소비자 읽은 값 : 4
소비자 읽은 값 : 4
소비자 읽은 값 합계: 14
--> 소비자 종료!
```

```
"C:\Program Files\Java\
소비자 읽은 값 : -1
소비자 읽은 값 : -1
생산자 저장한 값 : 1
소비자 읽은 값 : 1
생산자 저장한 값 : 2
소비자 읽은 값 : 2
소비자 읽은 값 합계: 1
--> 소비자 종료!
생산자 저장한 값 : 3
생산자 저장한 값 : 4
--> 생산자 종료!
```

```
"C:\Program Files\Java\"
생산자 저장한 값 : 1
소비자 읽은 값 : 1
소비자 읽은 값 : 1
생산자 저장한 값 : 2
생산자 저장한 값 : 3
소비자 읽은 값 : 3
소비자 읽은 값 : 3
소비자 읽은 값 합계: 8
--> 소비자 종료!
생산자 저장한 값 : 4
--> 생산자 종료!
```

## 8. POSIX 세마포어 코드(p.40)

```
#include<ctype.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <string.h>
#define MAXMSGLEN 256
sem_t sem1; // 세마포어 변수
sem_t sem2;
char msg1[MAXMSGLEN]; //버퍼 크기만큼 할당
char msg2[MAXMSGLEN];
void* threadFunc1(void *arg);
void toggleCase(char *buf);
```

```

int main()
{
pthread_t thread1;
char argmsg1[]="Thread1: ";
int res;
int thNum;
// 쓰레드 생성
res = sem_init(&sem1, 0, 0);
res = sem_init(&sem2, 0, 0);
res =
pthread_create(&thread1, NULL, threadFunc1,argmsg1);
while(1)
// 입력과 출력을 계속 반복
{
printf("Print message to send:\n");
fgets(msg1, MAXMSGLEN, stdin);
sem_post(&sem1); //1. V 연산 (signal 연산)
sem_wait(&sem2); //4. sem2 -> P 연산 (wait)
printf("Resp message: %s \n",msg2);
}
return 0;
}

```

```

void* threadFunc1(void *arg)
{
printf("I am :%s \n",arg);
while(1)
{
sem_wait(&sem1); //2. 앞에서 V연산 후,sem1-> wait연산
strcpy(msg2,msg1);
toggleCase(msg2);
sem_post(&sem2);// 3. sem2에 V연산
}
}
void toggleCase(char *str)
{
while(*str)
{
if (isupper(*str))
*str = tolower(*str);
else if (islower(*str))
*str = toupper(*str);
str++;
}
}

```

## POSIX 세마포어 실행 결과

```

chahyeona@chahyeona-virtual-machine:~/os_homework$ ./a.out
Print message to send:
I am :Thread1:
hello, there
Resp message: HELLO, THERE

Print message to send:
hi
Resp message: HI

Print message to send:

```

## 9. 자바 세마포어 코드(p.41)

```
public class java_semaphore {
    public static void main(String[] args) {
        System.out.println("Starting...");
        BoundedResource resource = new BoundedResource( nCount: 3); // 3개의 가용자원 할당
        for (int i=0; i<10; i++) {
            new UserThread(resource).start(); // 10개 쓰레드 생성
            // 10개의 Thread가 생성되어 있지만, 동시에 resource를 사용할 수 있는 Thread는 총 3개뿐임
        }
    }
}
```

```
import java.util.Random;
import java.util.concurrent.Semaphore;

public class Log {
    public static void show(String strMessage) {
        // 현재 쓰레드명 출력
        System.out.println(Thread.currentThread().getName() + " : " + strMessage);
    }
}
```

```
class BoundedResource {
    private final Semaphore m_Semaphore; // 세마포어 선언
    private final int m_nPermits;
    private final static Random m_Random = new Random( seed: 10000);
    public BoundedResource(int nCount) {
        this.m_Semaphore = new Semaphore(nCount); // 세마포어 생성
        this.m_nPermits = nCount;
    }
    public void use() throws InterruptedException {
        m_Semaphore.acquire(); // 세마포어 리소스 확보, P 연산
        try { doUse(); }
        finally { m_Semaphore.release(); // 세마포어 리소스 해제, V 연산
        }
    }
    protected void doUse() throws InterruptedException { // 세마포어 가용 값
        // 최대 Resource 개수 - 세마포어에서 이용가능한 Resource 개수 == 현재 사용중인 Resource 개수
        Log.show( strMessage: "Begin : 사용중인 Resource 개수 = " + (m_nPermits - m_Semaphore.availablePermits()));
        Thread.sleep(m_Random.nextInt( bound: 500));
        Log.show( strMessage: "End 사용중인 Resource 개수 = " + (m_nPermits - m_Semaphore.availablePermits()));
    }
}
```

```
class UserThread extends Thread
{
    private final static Random m_Random = new Random( seed: 10000);
    private final BoundedResource m_resource;
    public UserThread(BoundedResource resource) {
        m_resource = resource;
    }
    public void run() {
        try {
            while (true) {
                m_resource.use();
                Thread.sleep(m_Random.nextInt( bound: 3000));
            }
        }
        catch (InterruptedException e) { }
    }
}
```

## 자바 세마포어 실행 결과

```

java_semaphore x
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe"
Starting...
Thread-2 : Begin : 사용중인 Resource 개수 = 3
Thread-1 : Begin : 사용중인 Resource 개수 = 2
Thread-0 : Begin : 사용중인 Resource 개수 = 1
Thread-1 : End 사용중인 Resource 개수 = 3
Thread-3 : Begin : 사용중인 Resource 개수 = 3
Thread-0 : End 사용중인 Resource 개수 = 3
Thread-4 : Begin : 사용중인 Resource 개수 = 3
Thread-4 : End 사용중인 Resource 개수 = 3
Thread-9 : Begin : 사용중인 Resource 개수 = 3
Thread-9 : End 사용중인 Resource 개수 = 3
Thread-8 : Begin : 사용중인 Resource 개수 = 3
Thread-2 : End 사용중인 Resource 개수 = 3
Thread-7 : Begin : 사용중인 Resource 개수 = 3

```

## 10. 자바 모니터 코드(p.49)

```

import java.util.Vector;
public class Producer extends Thread
{
    static final int MAXQUEUE = 5;
    private Vector messages = new Vector();

    public void run( ) {
        try {
            while (true) {
                putMessage( );
                sleep( millis: 1000);
            }
        } catch (InterruptedException e) { }
    }

    private synchronized void putMessage( ) throws InterruptedException { // 모니터처럼 동작
        while (messages.size( ) == MAXQUEUE) {
            wait( ); // 제어권 넘겨주고 대기집합에서 대기상태로 기다림
        }
        messages.addElement(new java.util.Date( ).toString( ));
        System.out.println("put message");
        notify( ); // 진입집합으로 진입
    }

    public synchronized String getMessage( ) throws InterruptedException { // 모니터처럼 동작
        notify( ); // 대기집합에서 진입집합으로 스레드 하나를 이동시킴
        while (messages.size( ) == 0) {
            wait( ); // 잠금 반환 후, 대기집합에 자신을 추가하고 대기상태로 변경
        }
        String message = (String) messages.firstElement( );
        messages.removeElement(message);
        return message;
    }
}

```

```
public class Consumer extends Thread{
    Producer producer;
    Consumer(Producer p) {
        producer = p;
    }
    public void run( ) {
        try {
            while (true) {
                String message = producer.getMessage( );
                System.out.println("Got message: "+message); // 받은 메시지 출력
                sleep( millis: 200);
            }
        } catch (InterruptedException e) { // 예외처리
            e.printStackTrace();
        }
    }
    public static void main(String args[ ]) {
        Producer producer = new Producer( ); // 생산자 객체 생성
        producer.start( );
        new Consumer(producer).start( );
    }
}
```

## 자바 모니터 실행 결과

```
"C:\Program Files\Java\jdk-16.0.2\bin\java.
put message
Got message: Wed Apr 06 15:54:40 KST 2022
put message
Got message: Wed Apr 06 15:54:41 KST 2022
put message
Got message: Wed Apr 06 15:54:42 KST 2022
put message
Got message: Wed Apr 06 15:54:43 KST 2022
put message
Got message: Wed Apr 06 15:54:44 KST 2022
put message
Got message: Wed Apr 06 15:54:45 KST 2022
put message
Got message: Wed Apr 06 15:54:46 KST 2022
```



## 11. Dekker 알고리즘 코드(p.18)

```
#include <iostream>
#include <thread>

using namespace std;

int favoredThread = 1; // 우선순위 쓰레드 변수
int sum=0;
bool t1WantsToEnter = false;
bool t2WantsToEnter = false;

// 쓰레드 T1
void ThreadT1( )
{
    for(int i=0;i<100;i++)
    {
        t1WantsToEnter = true; // T1 쓰레드가 들어가길 원한다고 표현
        while (t2WantsToEnter) // T2 쓰레드도 진입을 원한다면,
        { // 임계구역 진입
            if (favoredThread == 2) { // T2 쓰레드가 임계구역에 진입했는지 확인
                t1WantsToEnter = false; // 진입했다면 T1 쓰레드는 진입하지 않겠다고 선언
                while (favoredThread == 2) ; // favoredThread == 2일 동안 강제로 기다림
                // favoredThread == 1이 되면, 임계구역 진입 선언
                t1WantsToEnter = true;
            }
        }
        // 임계구역에서는 전역변수 증감을 실행
        sum++;

        // 임계구역 출구 -> favoredThread, t1WantsToEnter값을 바꿔주고 다른 쓰레드가 들어올 수 있게 처리
        favoredThread = 2;
        t1WantsToEnter = false; // 임계구역 출구
    }
}
```

```
// 쓰레드 T1과 동일하게 동작
void ThreadT2( )
{
    for(int i=0;i<100;i++) {
        t2WantsToEnter = true;
        while (t1WantsToEnter)
        { // 임계구역 진입
            if (favoredThread == 1)
            {
                t2WantsToEnter = false;
                while (favoredThread == 1) ;
                t2WantsToEnter = true;
            }
        }
        // 임계구역
        sum++;

        // 임계구역 출구
        favoredThread = 1;
        t2WantsToEnter = false;
    }
}
```

```
int main()
{
    thread t1(ThreadT1);
    thread t2(ThreadT2);

    t1.join();
    t2.join();

    cout << " sum :: " << sum << endl;

    return 0;
}
```

## Dekker 알고리즘 실행결과

```
> Executing task: cmd /C c:\Users\user\Desktop\de\dekker <
```

```
sum :: 200
```

터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.

## 12. Peterson 알고리즘 코드(p.19)

```
#include <iostream>
#include <thread>

using namespace std;

int favoredThread = 1;
int sum=0;
bool t1WantsToEnter = false; // 임계구역 진입 변수
bool t2WantsToEnter = false;
```

```
// 쓰레드 T1
void ThreadT1( )
{
    for(int i=0;i<15;i++) {
        t1WantsToEnter = true; // 임계구역에 진입하고싶음을 표현
        favoredThread = 2; // T2 쓰레드가 양보
        // 컨텍스트 스위칭이 되지 않았으면 반복문에 갇힘
        while (t2WantsToEnter && favoredThread == 2) ;

        // 임계구역
        sum++;
        cout << " sum :: " << sum << endl;
        // 작업 종료 후 다른 쓰레드가 이용할 수 있도록 처리
        t1WantsToEnter = false; // 임계구역 출구
    }
}
```

```
int main()
{
    thread t1(ThreadT1);
    thread t2(ThreadT2);

    t1.join();
    t2.join();

    cout << " sum :: " << sum << endl;

    return 0;
}
```

```
// T1 쓰레드와 로직 동일
void ThreadT2( )
{
    for(int i=0;i<15;i++) {
        t2WantsToEnter = true;
        favoredThread = 1;
        // 임계구역 진입
        while (t1WantsToEnter && favoredThread == 1);
        sum++;
        cout << " sum :: " << sum << endl;

        t2WantsToEnter = false; // 임계구역 출구
    }
}
```

## Peterson 알고리즘 실행 결과

```
> Executing task: cmd /C c:\Users\user\Desktop\peterson\peterson <

sum :: 1
sum :: 2
sum :: 3
sum :: 4
sum :: 5
sum :: 6
sum :: 7
sum :: 8
sum :: 9
sum :: 10
sum :: 11
sum :: 12
sum :: 13
sum :: 14
sum :: 15
sum :: 16
sum :: 17
sum :: 18
sum :: 19
sum :: 20
sum :: 21
sum :: 22
sum :: 23
sum :: 24
sum :: 25
sum :: 26
sum :: 27
sum :: 28
sum :: 29
sum :: 30
sum :: 30
```

## 13. Swap 명령어 코드(p.27)

```
#include <iostream>
#include <thread>

using namespace std;

int sum=0;
bool t1MustWait = false; // 임계구역 진입 변수
bool t2MustWait = false;
bool occupied = false;
```

```
// 쓰레드 t1
void ThreadT1( )
{
    bool t1MustWait = true;
    for(int i=0;i<10;i++)
    {
        // 임계구역 진입
        do {
            swap(t1MustWait, occupied); // t1MustWait의 값을 occupied에 할당
            // t1MustWait이 false라면, 반복문을 빠져나와 임계구역으로 진입한다.
            // t1MustWait이 true라면, occupied에 true를 할당하고 반복문을 빠져나가지 못하고 swap을 반복실행
        } while (t1MustWait);

        sum++;

        // 임계구역 빠져나감
        t1MustWait = true;
        occupied = false;
    }
}
```

```
int main()
{
    thread t1(ThreadT1);
    thread t2(ThreadT2);

    t1.join();
    t2.join();

    cout << " sum : " << sum << endl;

    return 0;
}
```

```
// 쓰레드 t1과 동작 로직은 동일하다
void ThreadT2( )
{
    bool t2MustWait = true;
    for(int i=0;i<10;i++)
    {
        // 임계구역 진입
        do {
            swap(t2MustWait, occupied);
        } while (t2MustWait);

        sum++;

        // 임계구역 빠져나감
        t2MustWait = true;
        occupied = false;
    }
}
```

## Swap 명령어 실행결과

```
> Executing task: cmd /C c:\Users\user\Desktop\swap\swap <
```

```
sum : 20
```

터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.

## 14. Reader-Writer 문제 코드(p.39)

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

sem_t wrt;
pthread_mutex_t mutex;
int resource = 1; //공유하고 있는 변수
int readCount = 0; //현재 저장공간을 접근하고 있는 독자의 수

void *writer(void *w)
{
    sem_wait(&wrt); //임계구역 진입을 위한 대기
    resource += 3; //공유변수 값 바꾸기
    printf("Writer %d modified resource to %d\n",*((int *)w),resource); //출력
    sem_post(&wrt); //임계구역 나오기
}

void *reader(void *r)
{
    readCount++; // 독자 수 증가
    if(readCount == 1) {
        sem_wait(&wrt); // 쓰고있는 writer가 없을 때까지 대기
    }
    printf("Reader %d: read resource as %d\n",*((int *)r),resource);
    readCount--;
    if(readCount == 0) {
        sem_post(&wrt); //임계구역 나오기
    }
}

int main()
{
    pthread_t read[8],write[3];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&wrt,0,1); //쓰레드 속성 구조체 생성 및 초기화

    int book[8] = {1,2,3,4,5,6,7,8}; // 쓰레드 생성시 전달 될 파라미터

    for(int i = 0; i < 8; i++) {
        pthread_create(&read[i], NULL, (void *)reader, (void *)&book[i]); //쓰레드 생성
    }
    for(int i = 0; i < 3; i++) {
        pthread_create(&write[i], NULL, (void *)writer, (void *)&book[i]);
    }

    for(int i = 0; i < 8; i++) {
        pthread_join(read[i], NULL); // 해당 쓰레드가 종료할때까지 대기
    }
    for(int i = 0; i < 3; i++) {
        pthread_join(write[i], NULL); //해당 쓰레드가 종료할때까지 대기
    }

    pthread_mutex_destroy(&mutex);
    sem_destroy(&wrt);

    return 0;
}
```

## Reader-Writer 문제 실행 결과

```
chahyeona@chahyeona-virtual-machine:~/os_homework$ ./a.out
Reader 3: read resource as 1
Reader 4: read resource as 1
Reader 5: read resource as 1
Reader 8: read resource as 1
Writer 1 modified resource to 4
Reader 1: read resource as 4
Writer 2 modified resource to 7
Writer 3 modified resource to 10
Reader 7: read resource as 10
Reader 6: read resource as 10
Reader 2: read resource as 10
```

## 15. lamport 알고리즘 코드(p.21)

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

int numbers[5]={0}; // 티켓번호 저장
int choosing[5]={0}; // 티켓 받는중

int counter=0; // 공유변수
int sum=0;
void* thread_body(void*);
void ThreadTx(int);
int main(int argc, char** argv)
{
    pthread_t threads[5];
    for (int i = 0; i < 5; ++i) {
        pthread_create(&threads[i], NULL, &thread_body, (void*)((long)i)); // 스레드 생성
    }

    for (int i = 0; i < 5; ++i) {
        pthread_join(threads[i], NULL);
    }
    printf("\n");

    return 0;
}

void* thread_body(void* arg)
{
    long t_id = (long)arg; // i를 t_id로 넘겨준다.
    ThreadTx(t_id);
    // 임계구역 진입
    sum++;
    printf("sum: %d\n",sum);
    return NULL;
}
```

```
void ThreadTx(int t_id)
{
    choosing[t_id] = 1; // 티켓 받는중

    int max = 0;

    for (int i = 0; i < 5; ++i) {
        max = numbers[i] > max ? numbers[i] : max;
    }
    numbers[t_id] = max + 1; // 현재 발생된 티켓보다 1 큰값

    choosing[t_id] = 0; // 티켓 발급 완료

    for (int j = 0; j < 5; ++j) {

        while (choosing[j]); // 티켓을 받는 중이면 대기

        while (numbers[j] != 0 && (numbers[j] < numbers[t_id]
            || (numbers[j] == numbers[t_id] && j < t_id))) {
        } // 우선순위가 높은것이면 대기, 티켓번호가 같고 식별번호가 작은것이면 대기
    }

    numbers[t_id] = 0; // 임계구역 출구, 티켓 반납
}
```

## lamport 알고리즘 실행결과

```
chahyeona@chahyeona-virtual-machine:~/os_homework$ ./a.out
sum: 1
sum: 3
sum: 4
sum: 2
sum: 5
```



## 16. Test-and-Set 명령어를 사용한 상호배제 구현 코드(p.25)

```
boolean occupied = false;
// 쓰레드 T1
void ThreadT1()
{
    boolean t1MustWait = true; //
    while (!done)
    {
        // 임계구역 진입
        while (t1MustWait) // t1MustWait이 true이면, 무한반복
            testAndSet(t1MustWait, occupied); //occupied를 t1MustWait에 할당하고,
            // occupied를 true로 세팅
        //
        // 임계구역 수행
        //
        t1MustWait = true; // 임계구역 출구
        occupied = false; // 다른쓰레드가 진입할 수 있도록 값 세팅
    }
}
```

```
// 쓰레드 T2_T1 쓰레드와 로직은 동일하다
void ThreadT2()
{
    boolean t2MustWait = true;
    while (!done)
    {
        // 임계구역 진입
        while (t2MustWait)
            testAndSet(t2MustWait, occupied);
        //
        // 임계구역
        //
        t2MustWait = true; // 임계구역 출구
        occupied = false;
    }
}
```

## 17. 한정대기를 보장하는 Test-and-Set 명령어 사용한 임계구역(p.26)

```
boolean waiting[n]; // 전역변수
boolean lock = false; // 전역변수
// i번째 쓰레드의 임계 구역 접근 제어
// 일정 횟수 이내로 대기 가능, 최대 n-1번 양보 후 임계구역 진입
do
{
    waiting[i] = true; // i번째 쓰레드가 임계구역 진입에 대한 의사 표시
    key = true;
    while (key && waiting[i])
        testAndSet(key, lock); // lock값을 key에 복사, lock을 true로 세팅
    waiting[i] = false;
    //
    // 임계 구역 수행
    //
    j = (i + 1) % n; // j는 i바로 다음에 위치한 쓰레드
    while ((j != i) && !waiting[j])
        j = (j + 1) % n; // 대기중인 쓰레드를 검색
    // 대기중인 쓰레드가 없으면, 잠금해제
    if (j == i) // 다른 쓰레드가 진입할 수 있도록 lock 해제
        lock = false;
    else
        // 대기중인 것이 있으면, 해당 쓰레드를 바로 다음에 임계구역으로 진입시킴
        waiting[j] = false;
} while (true);
```

## 18. CAS 명령어를 통한 하드웨어 상호배제 구현(p.28)

```
bool CAS(int *p, int oldVal, int newVal)
{
    // *p와 oldVal의 값이 같지 않으면, false 리턴
    if (*p != oldVal)
    {
        return false;
    }
    // *p와 oldVal의 값이 같으면, *p에 newVal을 넣고 true를 리턴
    *p = newVal;
    return true;
}
```

(p.29)

```
void ThreadT1()
{
    do
    {
        // lock이 0이랑 같지않으면 false리턴
        // 같으면 lock에 1을 넣고 true를 리턴
        while (CAS(&lock, 0, 1) != 0) ;
        // 임계구역
        lock = 0; // 잠금해제
    } while (true);
}
```

## 19. 세마포어를 이용한 생산자/소비자 관계 구현(p.35)

```
Semaphore valueProduced = new Semaphore(0); // 세마포어 변수
Semaphore valueConsumed = new Semaphore(1); // 세마포어 변수
int sharedValue; // 생산자와 소비자가 공유하는 변수

// 생산자 쓰레드
void Producer()
{
    int nextValueProduced;
    while (!done)
    {
        nextValueProduced = generateTheValue();
        // valueConsumed가 1일때, P연산을 통해 0으로 만들어줌
        // 값이 1일때 통과가능 : 값을 쓸 수 있다.
        P(valueConsumed);
        sharedValue = nextValueProduced; // 임계 구역
        // 값을 쓰고, 소비자가 읽어갈 수 있도록 valueProduced를 1으로 세팅
        V(valueProduced);
    }
}
```

```
// 소비자 쓰레드
void Consumer()
{
    int nextValueProduced;
    while (!done)
    {
        // valueProduced가 1일 때 소비가능
        P(valueProduced);
        nextValueConsumed = sharedValue; // 임계 구역
        // 소비 후, 생산자가 값을 쓸 수 있도록 valueConsumed를 1로 세팅
        V(valueConsumed);
        processReceivedValue(nextValueConsumed);
    }
}
```

## 20. 계수 세마포어 구현(p.37)

```
// P연산
wait(semaphore *S)
{
    S->value--; // value값을 빼기
    if (S->value < 0) // 값을 뺐는데도 0보다 작다 -> 대기하는 스레드가 있다.
    {
        S->list; // 리스트에 추가
        block(); // 해당 프로세스를 waiting queue 삽입
    }
}
```

```
// V연산
signal(semaphore *S)
{
    S->value++; // value값 증가
    if (S->value <= 0) // 값을 증가했는데도 0이하-> 대기하는 스레드가 있다.
    {
        remove a process P from S->list; // list에서 하나 선택 후 깨우기
        wakeup(P); // waiting queue의 process 하나를 ready queue로 보냄
    }
}
```

## 21. 제어변수가 있는 모니터(p.45)

```
monitor monitor-name
{
    boolean inUse = false; // 공유 변수
    Condition available; // 조건 변수
    void getResource( ) {
        if (inUse)
            // available에 대한 signal 수신 때까지 대기
            // 현재 스레드를 중단하고 available 대기열로 들어감
            wait(available);
    }
    void returnResource( )
    {
        inUse = false;
        // available의 대기열에 있는 스레드를 선택 실행해서 재개
        // signal을 호출한 스레드는 긴급 대기열에 진입
        signal(available);
    }
}
```