MutEx & Semaphore

Ways to control programme concurrently

Contents

- MutEx
- Semaphore
- Ex code
- Todo

MutEx?

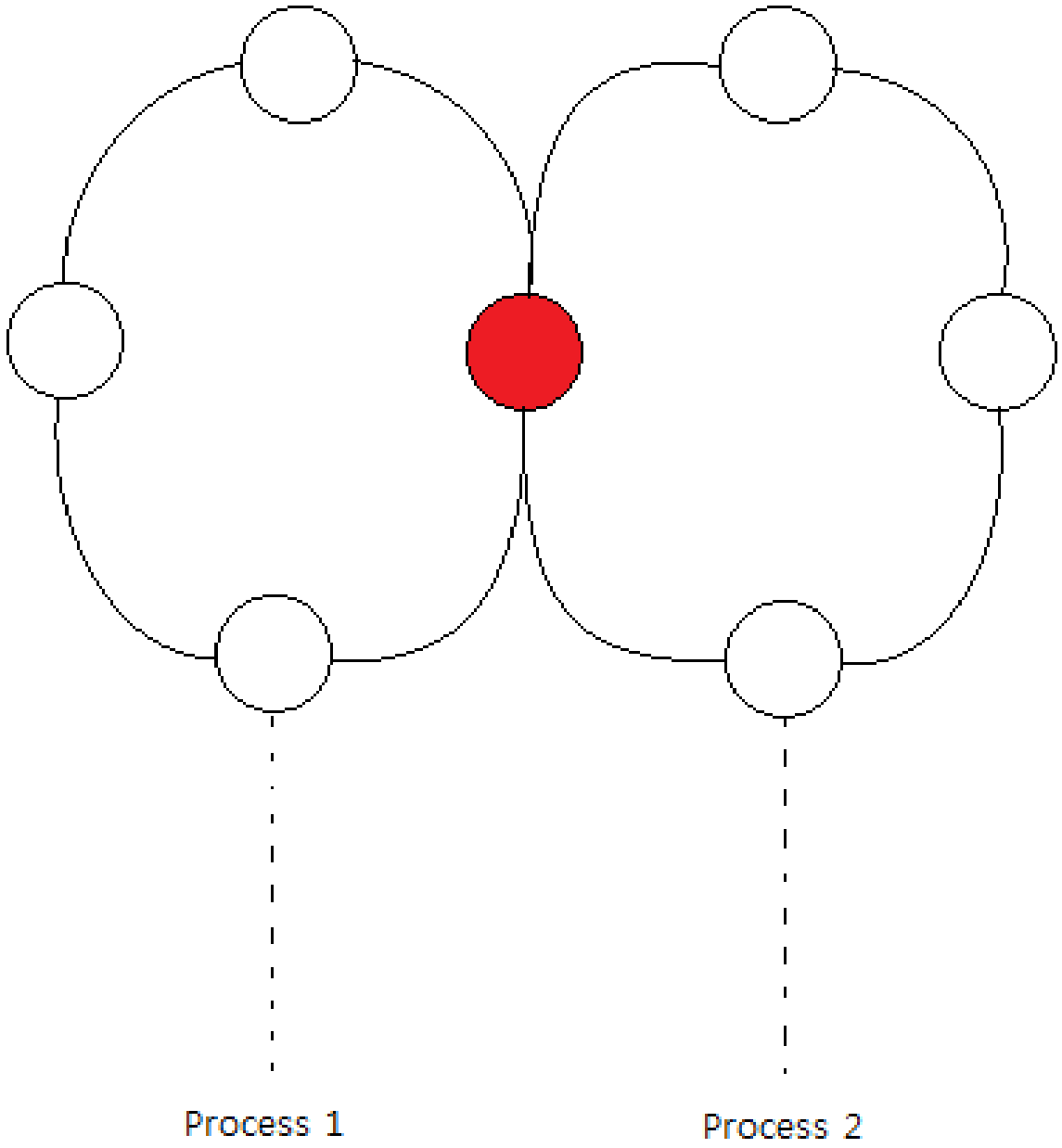
- Short form of “Mutual Exclusion”
- Kind of locking mechanism, monopolising resource
- To prevent **race condition**

Race condition?

- Occurs when…
  - 1. Two or more threads can access shared data
  - 2. They try to change it at the same time
- These shared data needs to be in **critical section**

Critical section?

- Parts of the program where the shared resource is accessed
- Cannot be executed by more than one process
- Ex) When different codes/processes needs to access same variable



Ex code for MutEx(Dekker's Algorithm)

```
void nmp(int pid)
{
    std::cout << "Launched from thread " << pid << std::endl;
}

int main()
{
    for (int i = 0; i < 10; i++)
    {
        std::thread t0(nmp, 1);
        std::thread t1(nmp, 2);

        t0.join();
        t1.join();

        std::cout << "Loop " << i + 1 << std::endl;
    }

    return 0;
}
```

Ex code for MutEx(Dekker’s Algorithm)

```
Launched from thread Launched from thread 2
1
Loop 1
Launched from thread 1
Launched from thread 2
Loop 2
Launched from thread 1Launched from thread 2

Loop 3
Launched from thread 1
Launched from thread 2
Loop 4
Launched from thread 1
Launched from thread 2
Loop 5
Launched from thread 1
Launched from thread 2
Loop 6
Launched from thread 1
Launched from thread 2
Loop 7
Launched from thread 1Launched from thread 2

Loop 8
Launched from thread 1
Launched from thread 2
Loop 9
Launched from thread 1
Launched from thread 2
Loop 10
계속하려면 아무 키나 누르십시오 . . .
```

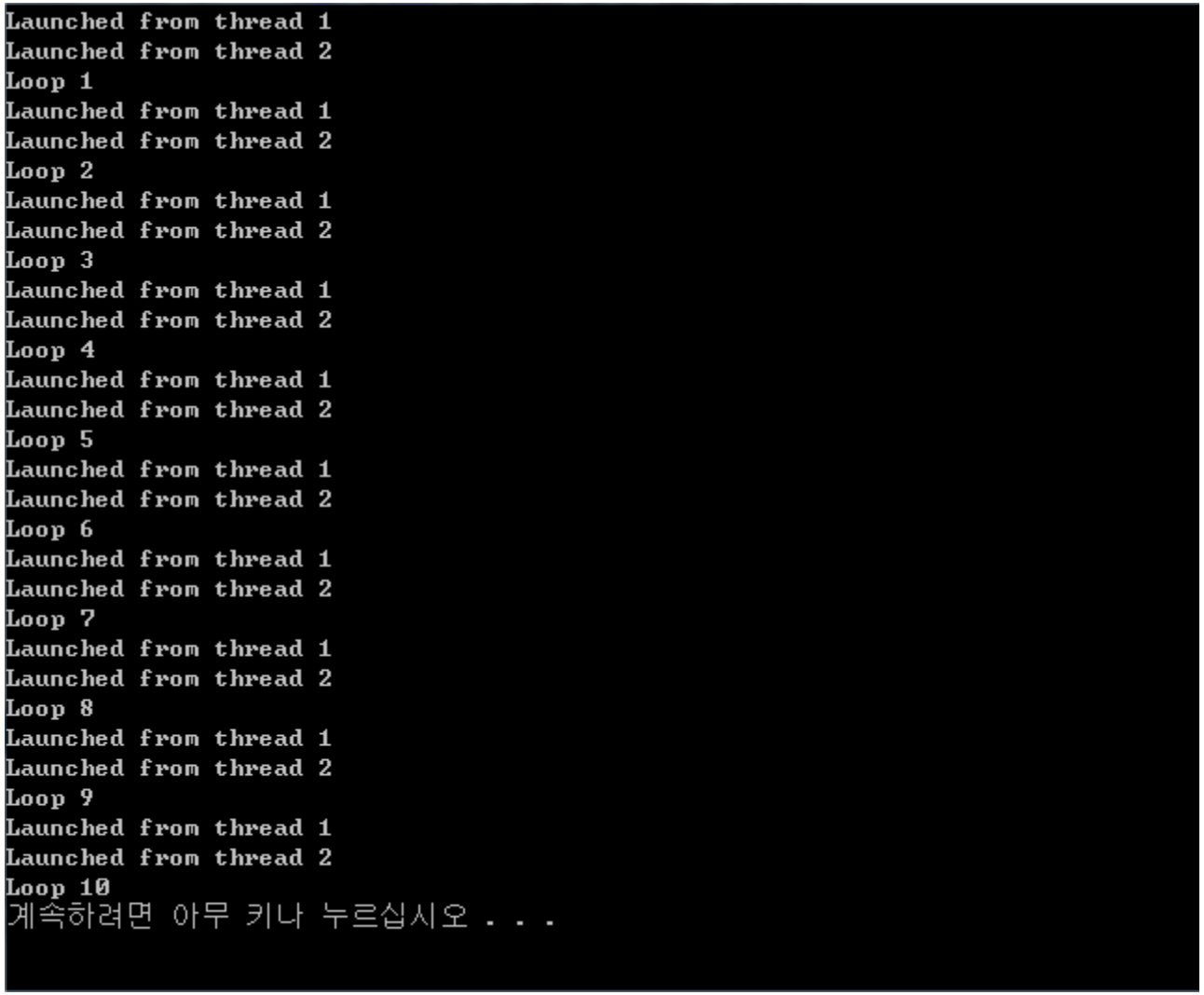
Ex code for MutEx(Dekker's Algorithm)

```
bool flag[2] = { false, false };
int turn = 0;
```

```
void p0(int pid)
{
    flag[0] = true;

    while (flag[1])
    {
        if (turn != 0)
        {
            flag[0] = false;
            while (turn != 0) {}
            flag[0] = true;
        }
    }

    std::cout << "Launched from thread " << pid << std::endl;

    turn = 1;
    flag[0] = false;
}
```

```
void p1(int pid)
{
    flag[1] = true;

    while (flag[0])
    {
        if (turn != 1)
        {
            flag[1] = false;
            while (turn != 1) {}
            flag[1] = true;
        }
    }

    std::cout << "Launched from thread " << pid << std::endl;

    turn = 0;
    flag[1] = false;
}
```

Ex code for MutEx(Dekker’s Algorithm)



Semaphore?

- A variable or abstract data type
- Kind of signalling mechanism, sharing resource
- To control access to a common resource by multiple processes
- MutEx is-a Semaphore(MutEx ≈ Binary Semaphore)

Ex code for Semaphore

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

void p(int tid)
{
    std::cout << "Launched from thread " << tid << std::endl;
}

int main()
{
    std::thread t[5];

    for (int i = 0; i < 5; i++) t[i] = std::thread(p, i);

    for (int i = 0; i < 5; i++) t[i].join();

    return 0;
}
```

Ex code for Semaphore

```
Launched from thread Launched from thread 1Launched from thread 3
0
Launched from thread 2
Launched from thread 4

계속하려면 아무 키나 누르십시오 . . .
```

Ex code for Semaphore

```cpp
#pragma once

#include <mutex>

class Semaphore
{
    int m_Cnt;
    const int m_Limit;
    std::mutex m_Mutex;

public:
    Semaphore() : m_Cnt(0), m_Limit(5) {}
    Semaphore(unsigned int i) : m_Cnt(0), m_Limit(i) {}
    ~Semaphore() {}

    void Signal();
    void Wait();
};
```

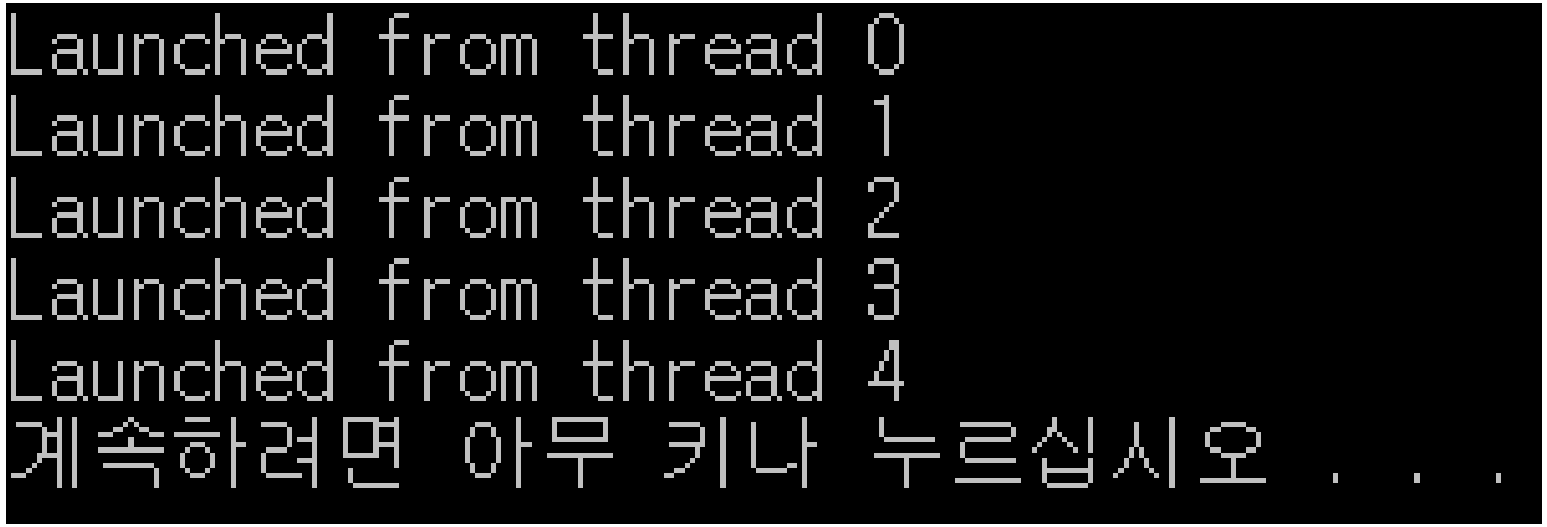
Ex code for Semaphore

```
#include "Semaphore.h"

void Semaphore::Signal()
{
    if (m_Cnt < m_Limit)
    {
        m_Cnt++;

        m_Mutex.unlock();
    }
}

void Semaphore::Wait()
{
    if (m_Cnt >= 0) m_Cnt--;

    if (m_Cnt < 0) m_Mutex.lock();
}
```

Ex code for Semaphore

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

#include "Semaphore.h"

Semaphore s;

void p(int tid)
{
    s.Wait();
    std::cout << "Launched from thread " << tid << std::endl;
    s.Signal();
}

int main()
{
    std::thread t[5];

    for (int i = 0; i < 5; i++) t[i] = std::thread(p, i);

    for (int i = 0; i < 5; i++) t[i].join();

    return 0;
}
```

Ex code for Semaphore



Todo

- Application of multithread programming to computer graphics
- Especially on physic simulation(optics, dynamics and so on)