운영체제

2019.6.28

컴퓨터공학과 이병문
가천대학교 Gachon University

강의일정

- 6/24 과목, 강의소개
- 25 운영체제 개요
- 26 Process, Thread
- 27 Concurrent Process
- 28 Concurrent Process
- 7/1 중간고사1

상호배제/동기화
- Monitor

IPC
- Shared Memory
- Message Passing system
- **사례연구**

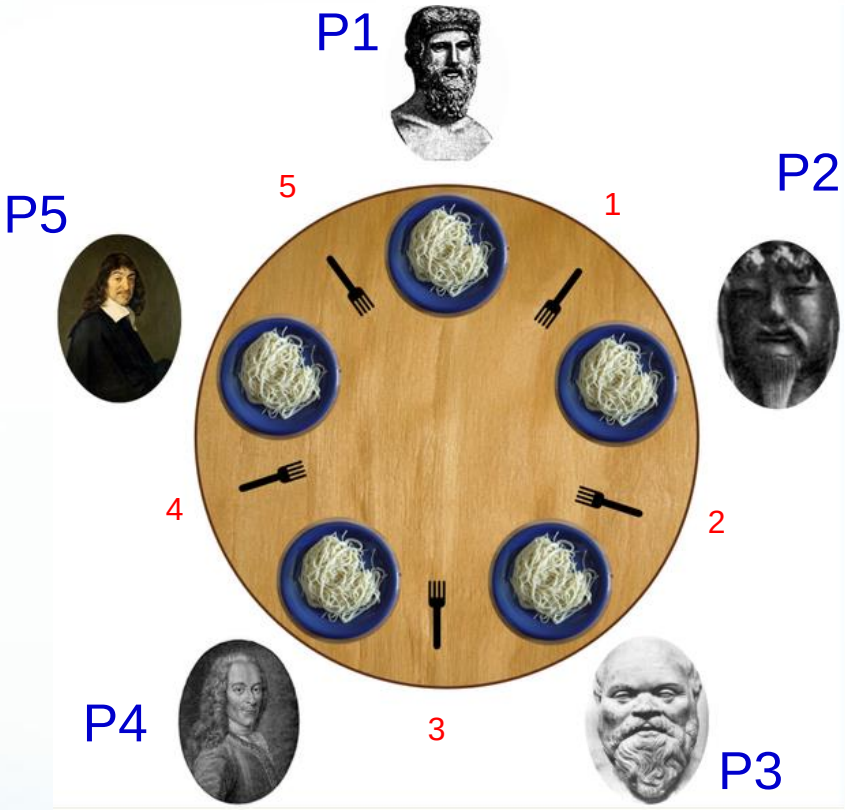
프로세스 사례연구

- fork () **샘플코드**
- thread() **샘플코드**
- pipe () **샘플코드**
- mutex **샘플코드**

상호배제 / 동기화

■ Dining Philosophers Problem

☑ 병행프로세스와 동기화문제의 예



5명 모두가 동시에 왼쪽포크를 집어 들었을 경우 ->

While eating, they are not thinking
While thinking, they are not eating

As spaghetti is difficult to serve and eat with a single fork, it must be assumed that in order for a philosopher to eat, the philosopher must have two forks

```
Semaphore chopStick[5] = { 0 };
```

```
P(chopStick[i])
P(chopStick[ (i+1) % 5])

  eating();

V(chopStick[i])
V(chopStick[ (i+1) % 5])

  thinking();
```

상호배제 / 동기화

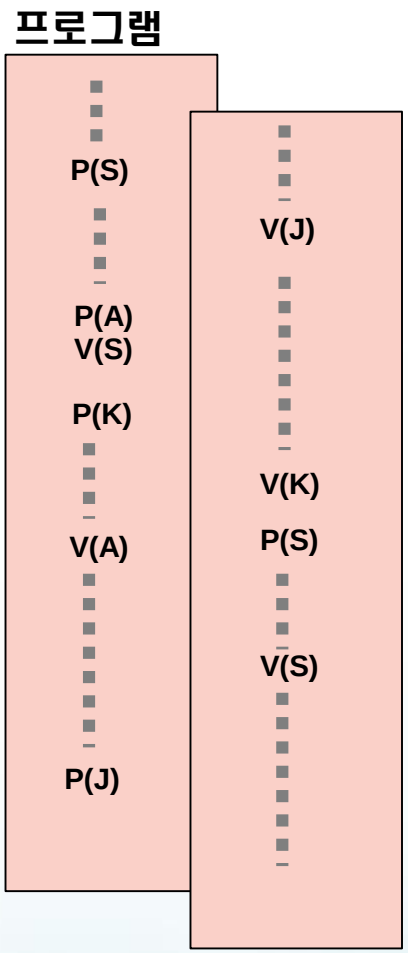
■ 모니터(Monitor) … 프로세스의 동기화 기법

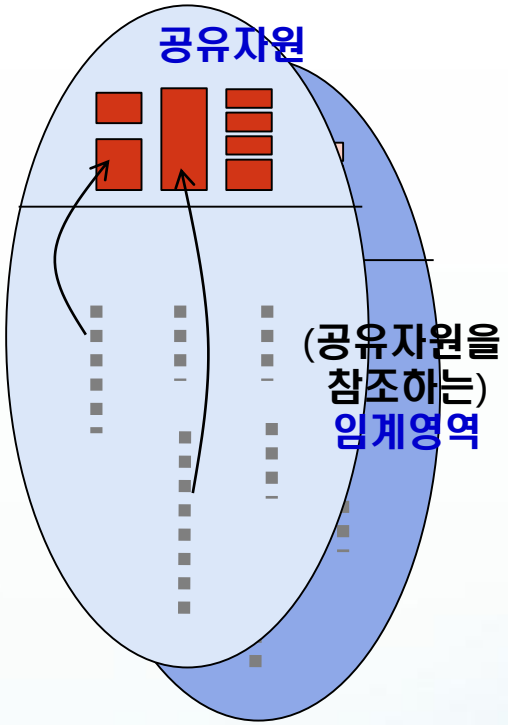
☑ 상호배제 및 프로세스들 사이의 조정을 위한 유연성, 강력한 도구가 필요

- P(wait 동작), V(signal 동작) 연산 :
  프로그램 전체에 널리 퍼짐
- 연산이 각자의 세마포어에
  미치는 영향 파악 어려움
  ⇒ 프로그램 작성 어려움

☑ 조직적인 구성체를 두어
공유자원, 임계영역을
체계적으로 관리

=> 모니터(Monitor)





상호배제 / 동기화

■ 모니터(Monitor) … 프로세스의 동기화 기법

☑ 하나 이상의 프로시저와 초기화 코드, 공유 데이터로 구성된 소프트웨어 모듈 **객체**.

☑ 모니터 경계에서 **한 번에 한 프로세스만 진입**하도록 제어되므로 **상호 배제 원칙**을 지킴.

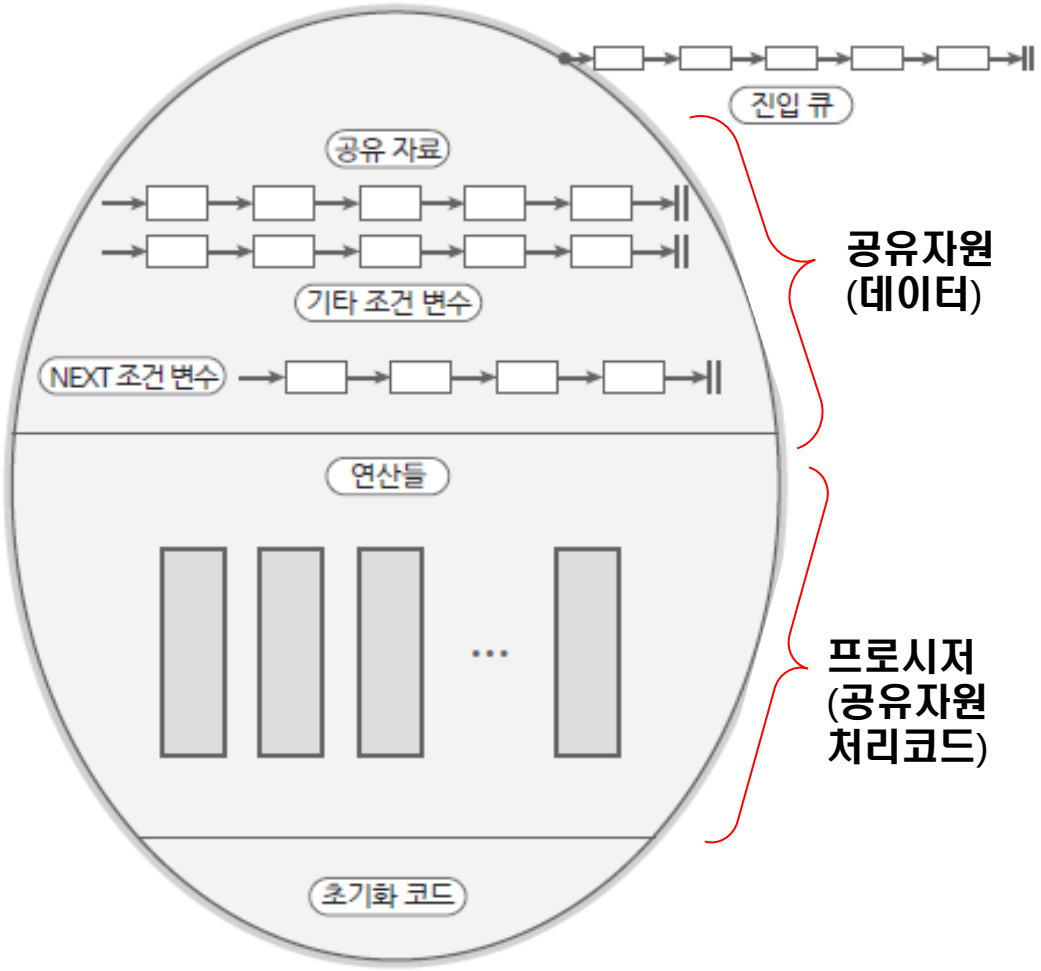
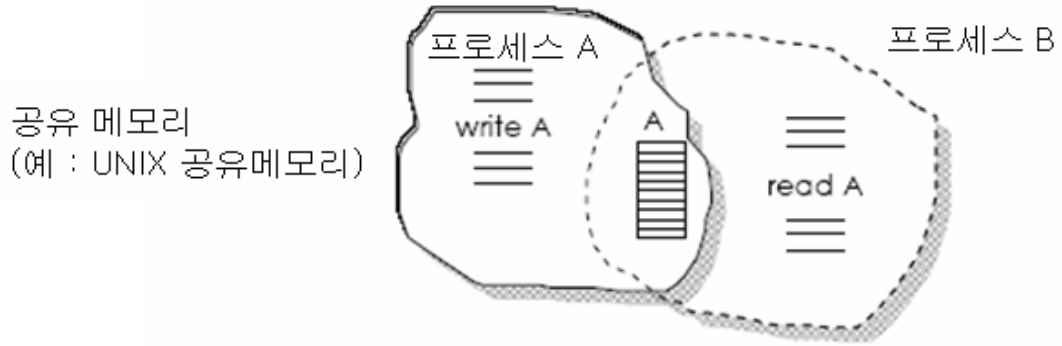


그림 4-1 모니터 구조

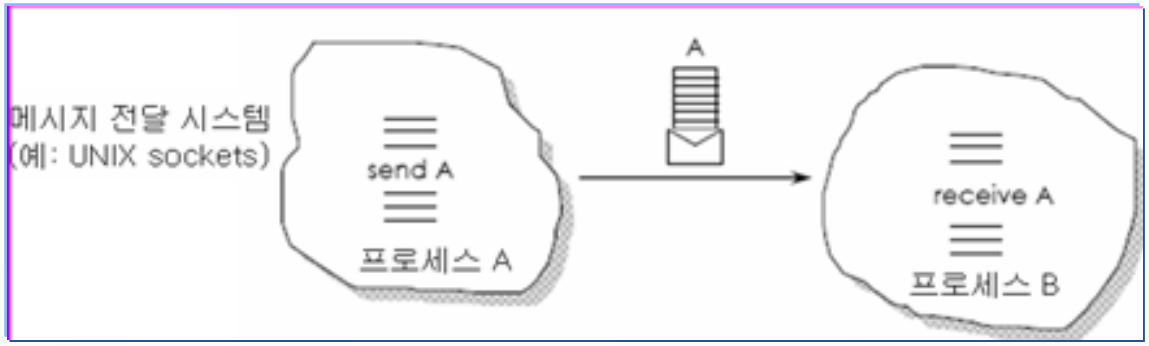
IPC(Inter Process Communication)

■ IPC

☑ Shared Memory



☑ Message Passing System



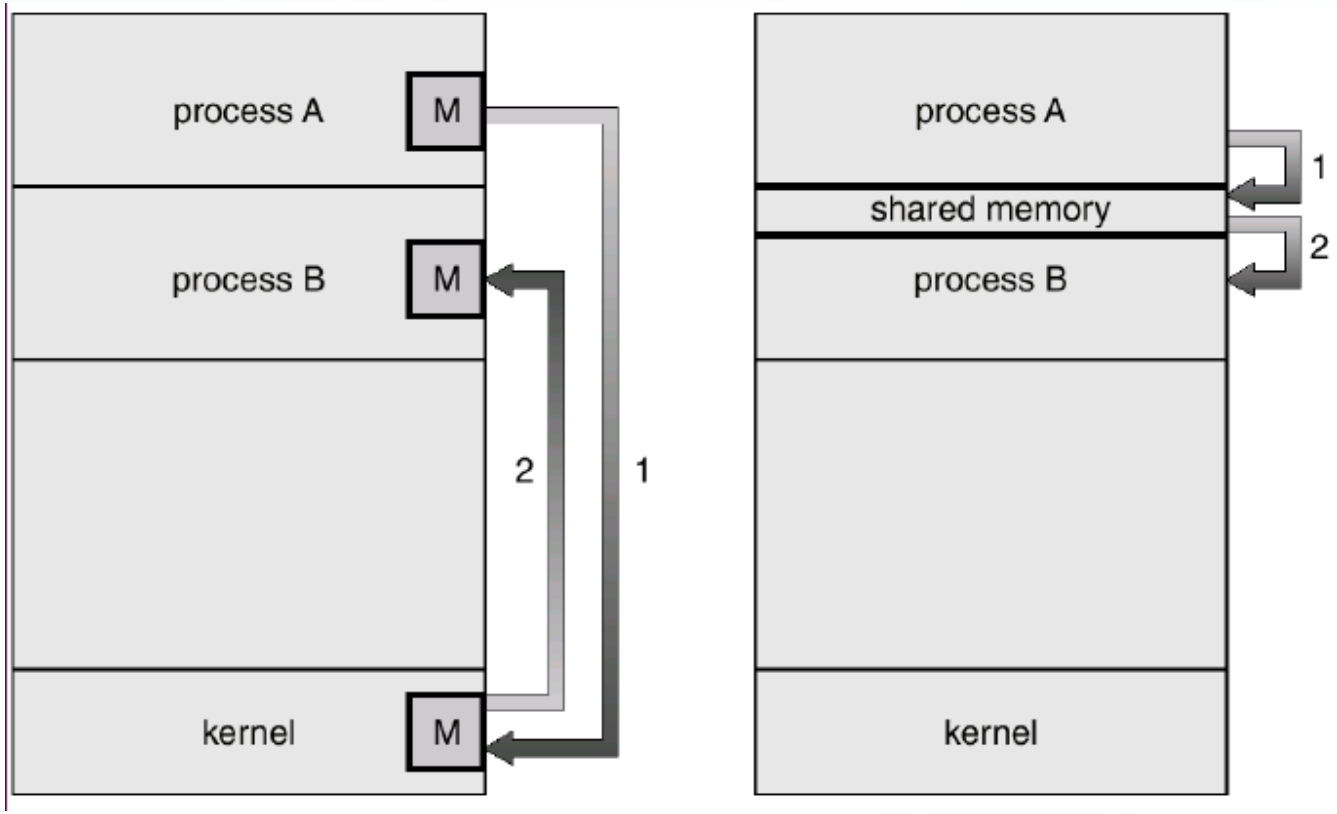
B프로세스 A메시지

send(목적지, message)
receive(발생지, message)

논리적 특성과 구현이슈
- -링크설정 방법/개수
- -송수신 버퍼용량
- -n개 프로세스간의 통신
- -메시지길이(고정, 가변)
- -통신방향(단방향, 양방향)

IPC(Inter Process Communication)

- 2 types of Communication Model



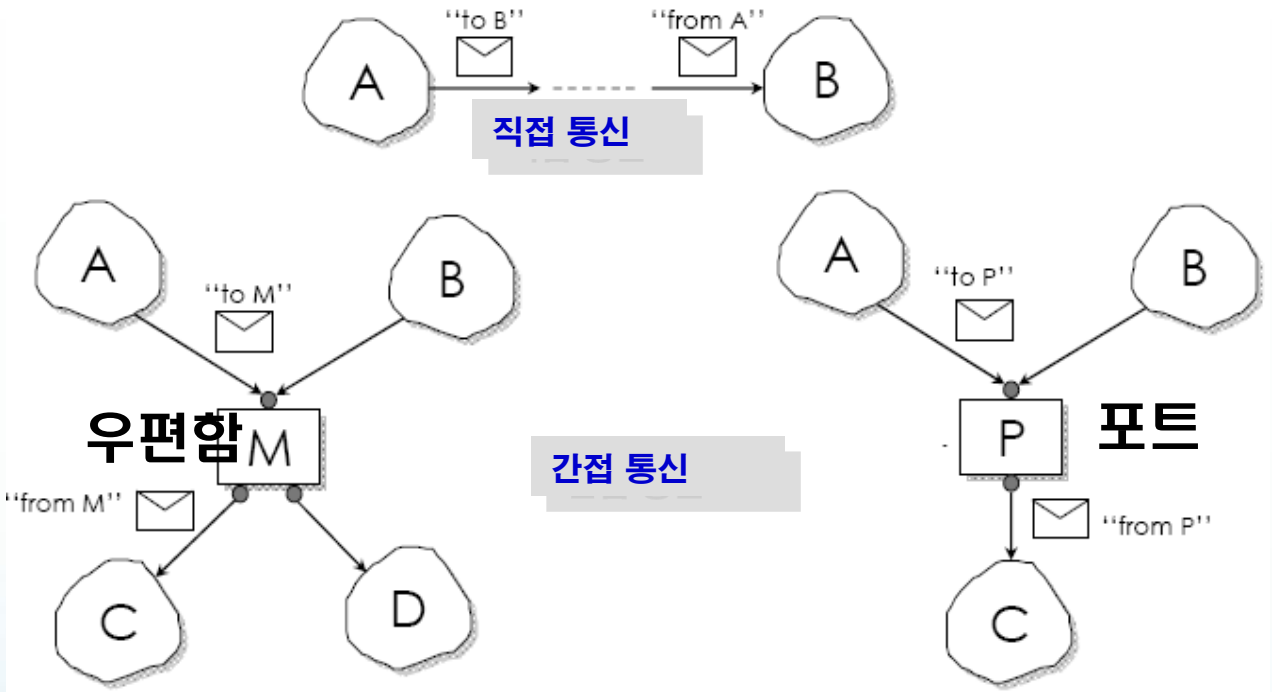
Message Passing **Shared Memory**

IPC(Inter Process Communication)

■ Naming(**명칭부착**)

☑ **상대프로세스를 식별하기 위한 목적으로 명칭이 필요**

☑ **직접통신**(Direct Communication)



직접통신
- -전송자/수신자의 이름명시
- -링크는 명시적으로 지정
- -송수신기간의 하나의 링크

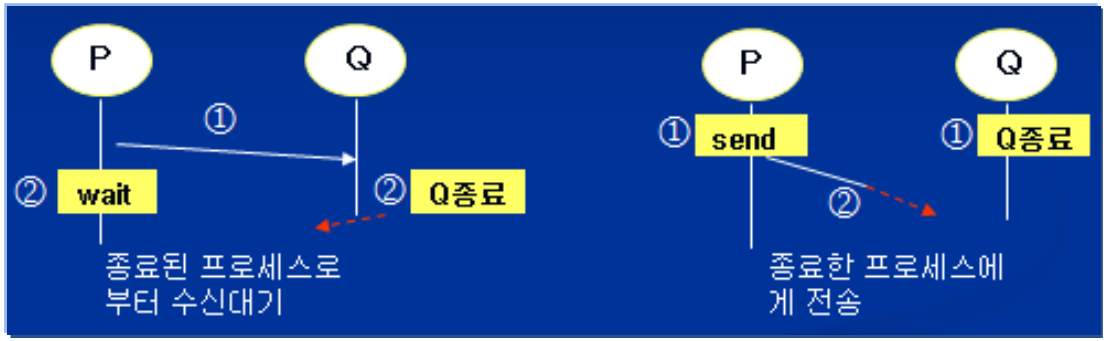
-프로세스 정의의 모듈성을 제한하는 문제점
send(B, message)
receive(A, message)

간접통신
송수신자를 분리하여 메시지사용의 융통성(Flexibility)을 제공

IPC(Inter Process Communication)

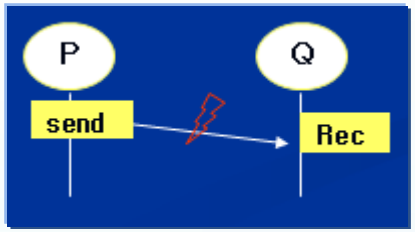
☑ **예외 조건**(Exception Condition)

- 분산처리일 경우 발생할 수 있는 예외조건을 처리하여야 함.
- 프로세스 종료



- 메시지 상실

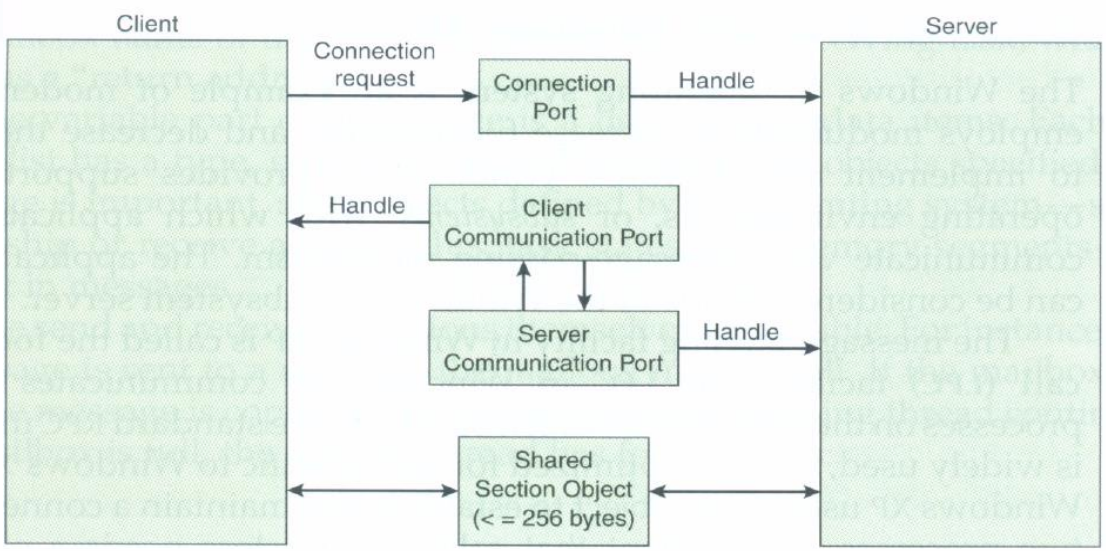
- 훼손메시지

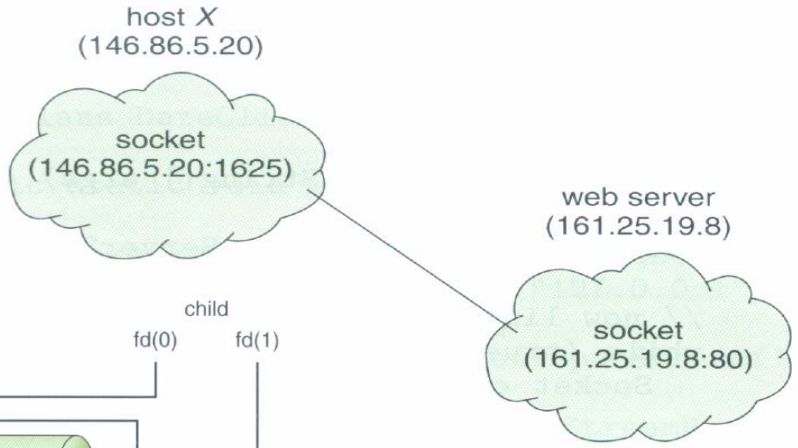


IPC(Inter Process Communication)

☑ 사례연구

- Local Procedure Call(LPC)
  Windows OS
- Remote Procedure Call(RPC)
  Windows OS
  UNIX/LINX
- Socket Communication
  Windows OS
  UNIX/LINUX
- PIPE
  Windows OS
  UNIX/LINUX





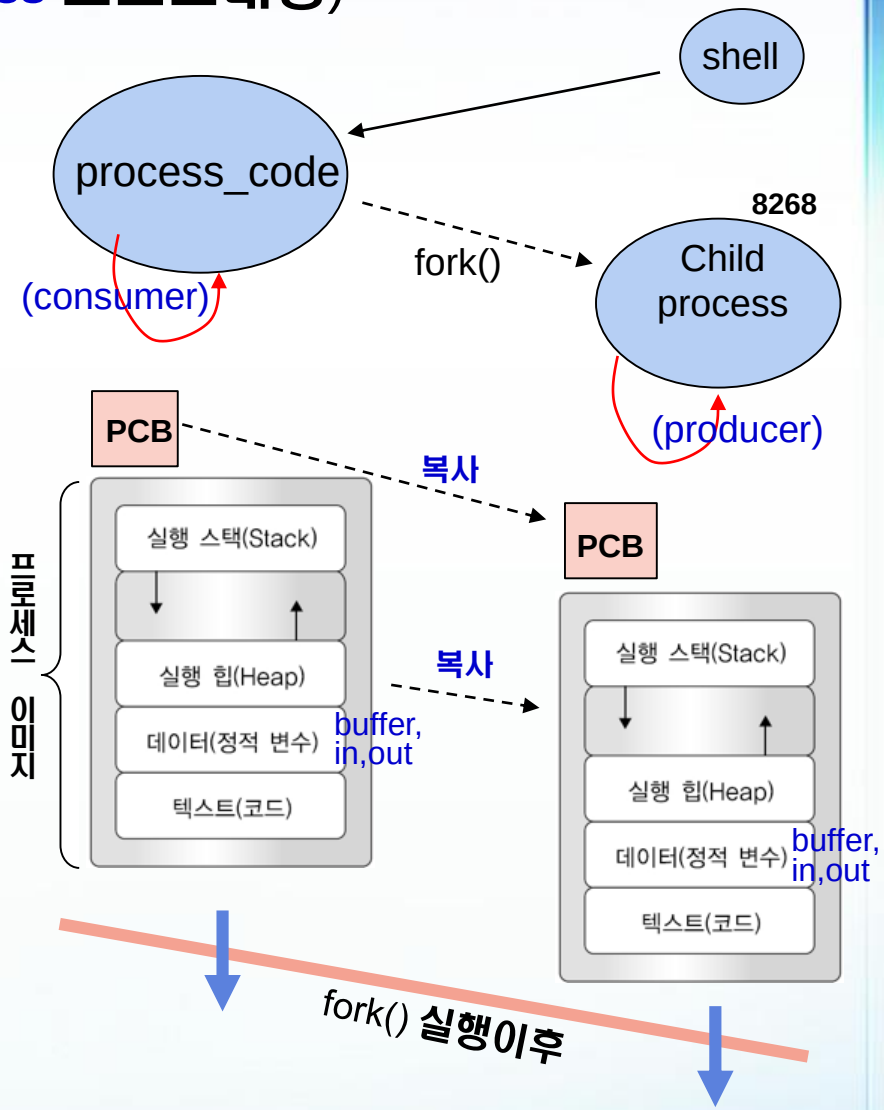
사례연구

■ 병행프로세스 (Linux, C) – (Multi-process 프로그래밍)

컴파일러 실행파일 소스코드

```
$ cc  -o  process_code  process_code.c
$ ./process_code
```

```
파일(F)  편집(E)  보기(V)  검색(S)  터미널(T)  도움말(H)
[leebyungmun@localhost proc]$ ./process_code
parent process!(child: 8268)
consumer: wait...
child process!
producer(): (in, out): data....(0, 0): 3
consumer: wait...
producer(): (in, out): data....(1, 0): 16
consumer: wait...
producer(): (in, out): data....(2, 0): 4
consumer: wait...
producer(): (in, out): data....(3, 0): 4
consumer: wait...
producer(): (in, out): data....(4, 0): 11
consumer: wait...
producer(): (in, out): data....(5, 0): 9
producer: wait...
consumer: wait...
producer: wait...
consumer: wait...
producer: wait...
consumer: wait...
producer: wait...
consumer: wait...
producer: wait...
consumer: wait...
producer: wait...
```



사례연구

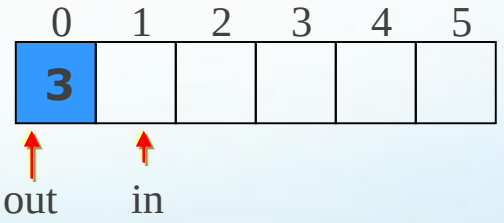
■ **병행프로세스** (Linux, C) – (Multi-process **프로그래밍**)

process_code.c

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

#define       MAXSIZE       6
int           buffer[MAXSIZE];
int           in = 0, out = 0;

int  producer() {
    int  data = 0;

    while (1) {
        data = (random() % 19) + 1;      // 랜덤으로 데이터 임의생성
        printf("producer():  (in,out):data....(%d,%d):%d\n", in, out, data);
        while ( ((in + 1) %  MAXSIZE) == out ) {
              printf("producer: wait...\n");      sleep(1);  /* if full, wait here */
        }
        buffer[in] = data;
        in = (in + 1) %  MAXSIZE;
        sleep(1); 임의로 1밀리초 간격으로 데이터전송
    }
}
```

```
int  consumer() {
    int data = 0;
    while (1) {
        while ( in == out ) {  /* if empty, wait here */
            printf("consumer: wait...\n");        sleep(1);
        }
        data = buffer[out];
        printf("comsumer():  (in,out):data....(%d,%d):%d\n",in,out,data);
        out = (out + 1) %  MAXSIZE;
        printf("consumer:  data[%d]\n", data);
        sleep(1);
    }
}
```

```
producer():   (in, out):data....(1, 0):16
consumer: wait
```

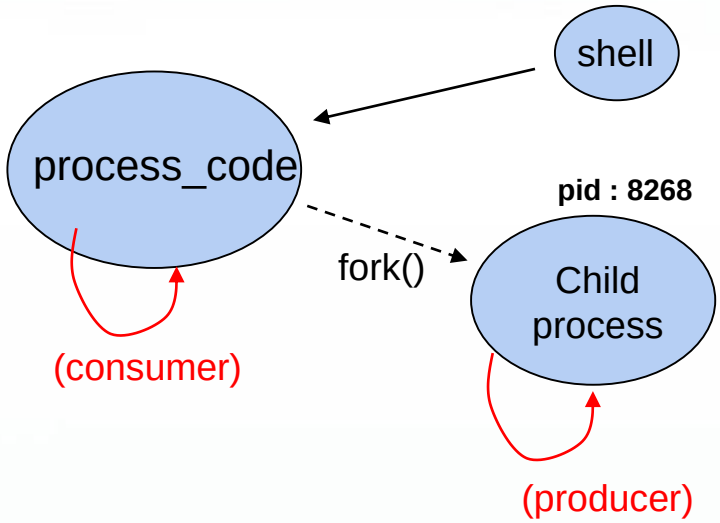


사례연구

■ **병행프로세스** (Linux, C) – (Multi-process **프로그래밍**)

process_code.c

```
void main() {

    int   pid;

    if ((pid = fork()) == 0) { /* child process  */
            printf("child process!\n");
            producer();
    }
    else {  /* parent process */
            printf("parent process!(child:%d)\n", pid);
            consumer();
    }

}
```



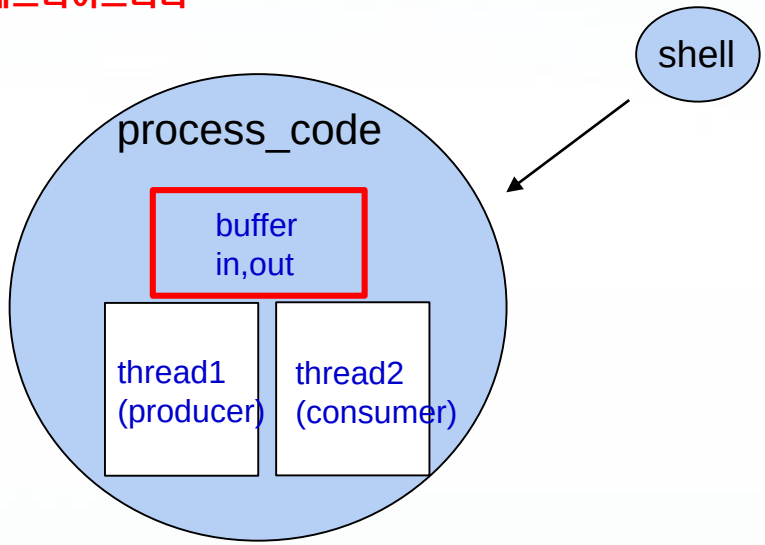
사례연구

■ 병행프로세스 (Linux, C) – (Multi-threads 프로그래밍)

컴파일러 실행파일 소스코드 쓰레드라이브러리

```
$ cc -o thread_code thread_code.c -lpthread
$ ./thread_code
```

```
consumer: wait...
consumer: wait...
consumer: wait...
consumer: wait...
consumer: wait...
consumer: wait...
consumer: wait...
consumer: wait...
consumer: wait...
consumer: wait...
consumer: wait...
producer(): (in, out):data....(0, 0):3
consumer(): (in, out):data....(1, 0):3
producer(): (in, out):data....(1, 1):16
producer(): (in, out):data....(2, 1):4
consumer(): (in, out):data....(3, 1):16
producer(): (in, out):data....(3, 2):4
producer(): (in, out):data....(4, 2):11
producer(): (in, out):data....(5, 2):9
consumer(): (in, out):data....(0, 2):4
producer(): (in, out):data....(0, 3):11
^C
[leebyungmun@localhost proc]$
```



사례연구

■ **병행프로세스** (Linux, C) – (Multi-threads **프로그래밍**)

thread_code.c

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

#define        MAXSIZE        6
int            buffer[MAXSIZE];
int            in = 0, out = 0;

void *producer( ) {
    int data = 0;
    while (1) {
        data = (random() % 19) + 1;     // 랜덤데이터 임의생성
        printf("producer():  (in,out):data....(%d,%d):%d\n",in,out,data);
        while (((in + 1) %  MAXSIZE) == out) {    /* if full, wait here */
                printf("producer: wait...\n");
        }
        buffer[in] = data;
        in = (in + 1) %  MAXSIZE;
        sleep(3);
    }
}
```

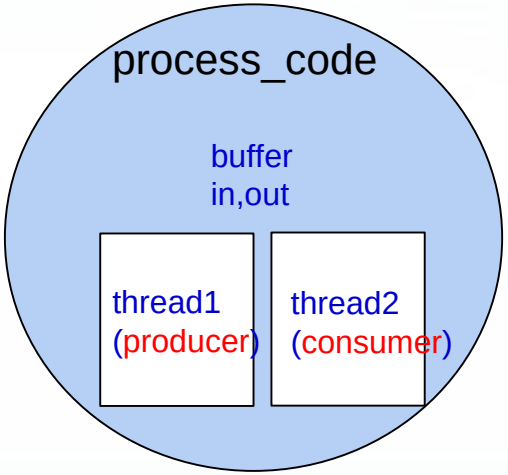
```
void *consumer( ) {
    int data = 0;

    while (1) {
        while (in == out) {     /* if empty, wait here */
            printf("consumer: wait...\n");
        }
        data = buffer[out];
        printf("consumer():  (in,out):data....(%d,%d):%d\n",in,out,data);
        out = (out + 1) %  MAXSIZE;
        sleep(9);
    }
}
```

사례연구

■ **병행프로세스** (Linux, C) – (Multi-threads **프로그래밍**)

thread_code.c

```
int main()
{
    pthread_t    p_thread[2];
    int    thr_id;
    int     status;

    // 쓰레드1 생성
    thr_id = pthread_create(&p_thread[0], NULL, producer, NULL);

    //  쓰레드2 생성
    thr_id = pthread_create(&p_thread[1], NULL, consumer, NULL);

    // 쓰레드 종료를 기다린다
    pthread_join(p_thread[0], (void **)&status);
    printf("thread1: %d\n", status);
    pthread_join(p_thread[1], (void **)&status);
    printf("thread2: %d\n", status);

    return 0;
}
```



사례연구

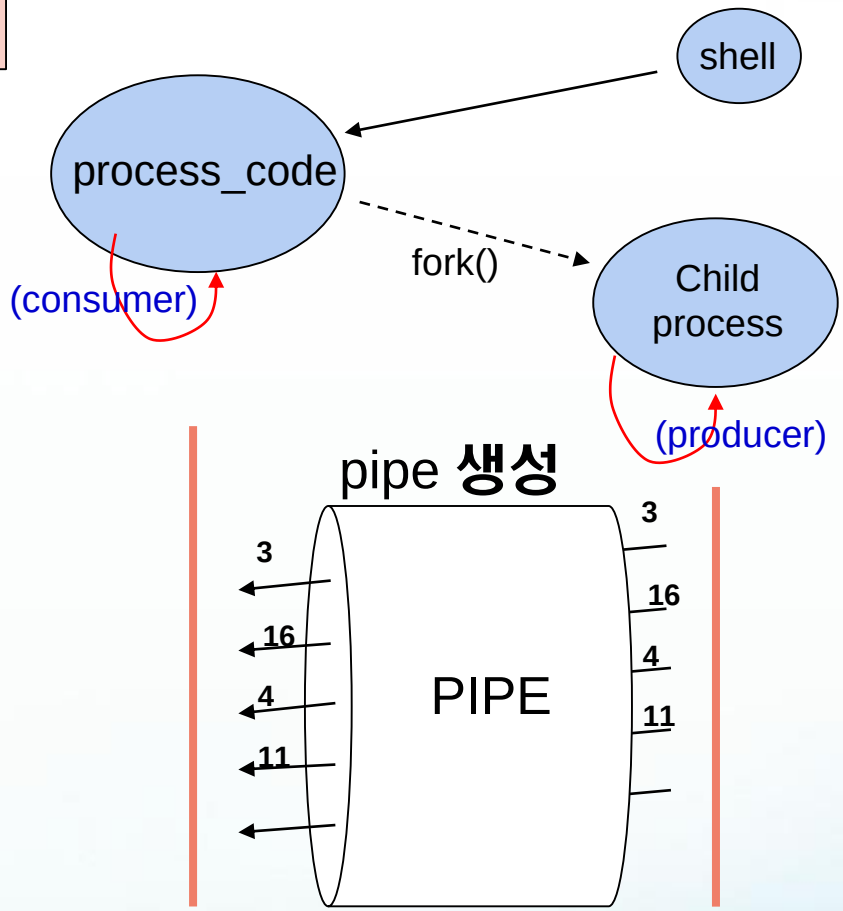
■ **IPC**(PIPE) (Linux, C) – (PIPE**프로그래밍**)

쓰레드라이브러리

컴파일러 실행파일 소스코드

```
$ cc  -o  pipe_code  pipe_code.c  -lpthread
$ ./pipe_code
```

```
파일(F)  편집(E)  보기(V)  검색(S)  터미널(T)  도움말(H
[leebyungmun@localhost proc]$ ./pipe_code
producer: consumer로 송신 : 3
comsumer: producer에서 수신 : 3
producer: consumer로 송신 : 16
comsumer: producer에서 수신 : 16
producer: consumer로 송신 : 4
comsumer: producer에서 수신 : 4
producer: consumer로 송신 : 4
comsumer: producer에서 수신 : 4
producer: consumer로 송신 : 11
comsumer: producer에서 수신 : 11
producer: consumer로 송신 : 9
comsumer: producer에서 수신 : 9
producer: consumer로 송신 : 11
comsumer: producer에서 수신 : 11
producer: consumer로 송신 : 12
comsumer: producer에서 수신 : 12
producer: consumer로 송신 : 3
comsumer: producer에서 수신 : 3
```



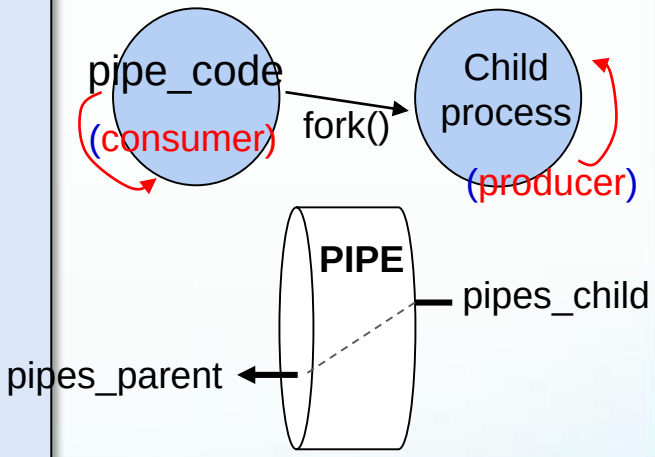
사례연구

■ **IPC**(PIPE) (Linux, C)

pipe_code.c

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

#define  BUFF_SIZE   1024

int   producer(int   pipes_child) {
   char  buff[BUFF_SIZE];
   int   data;

   while(1) {
      data = (random() % 19) + 1;      // 랜덤으로 데이터 임의생성
      memset( buff, 0, BUFF_SIZE);
      sprintf(buff,"%d",data);
      write( pipes_child,  buff,  strlen( buff));
      printf( "producer: consumer로 송신 : %s\n", buff);
      sleep(2);
   }
}
```

```
int   consumer(int   pipes_parent) {
   char  buff[BUFF_SIZE];

   while(1) {
      memset(buff, 0, BUFF_SIZE);
      read(pipes_parent,  buff,  BUFF_SIZE);
      printf( "comsumer: producer에서 수신 : %s\n", buff);
      sleep(2);
   }
}
```

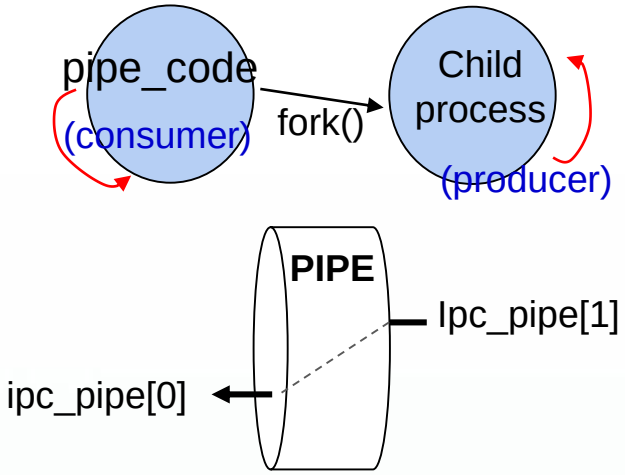


사례연구

■ **IPC**(PIPE) (Linux, C)

pipe_code.c

```
int   main()
{
    int   ipc_pipe[2];
    int   pid;

    if  ( pipe ( ipc_pipe ) < 0 ) {
        perror( "파이프 생성 실패");
        exit(1);
    }

    if  ((pid = fork()) == 0) {
         producer( ipc_pipe[1] );
    }
    else {
         consumer( ipc_pipe[0] );
    }
}
```



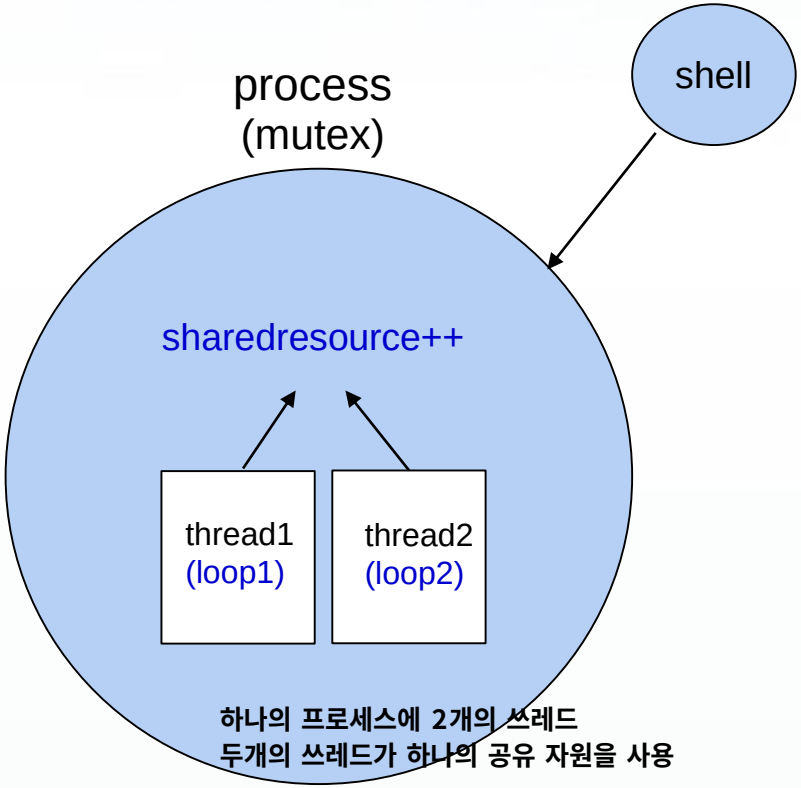
사례연구

■ **상호배제**(Mutex) (Linux, C) – (**병렬프로그래밍에서 상호배제**)

컴파일러 실행파일 소스코드

```
$ cc  -o  mutex  mutex.c  -lpthread
$ ./mutex
```

```
파일(F)  편집(E)  보기(V)  검색(S)  터미널(T)  도움말(H)
[leebyungmun@localhost proc]$ ./mutex
loop1 : 공유자원 : 0
loop1 : 공유자원 : 1
loop2 : 공유자원 : 2
loop1 : 공유자원 : 3
loop2 : 공유자원 : 4
loop1 : 공유자원 : 5
임계영역에서 비정상 리턴함
```



사례연구

■ **상호배제**(Mutex) (Linux, C) – (**병렬프로그래밍에서 상호배제**)

mutex.c

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

 세마포어변수(mutex) 초기화
pthread_mutex_t   mutex = PTHREAD_MUTEX_INITIALIZER;
int               sharedresource;      // 쓰레드간 공유되는 자원

void *do_loop1(void *data)
{
    int i;
    for (i = 0;  i < 5;  i++)   {
        pthread_mutex_lock(&mutex);    // 진입영역
        printf("loop1 :공유자원: %d\n",  sharedresource);
        sharedresource ++;
        if ( i == 3 ) {
           printf("임계영역에서 비정상 리턴함\n");
           return;     // 해제하지 않고 리턴함(의도적)
        }
        pthread_mutex_unlock(&mutex); // 출구영역
        sleep(1);
    }
}
```

```
void *do_loop2(void *data)
{
    int i;

   for (i = 0; i < 5; i++)   {
        pthread_mutex_lock(&mutex);      // 진입영역
        printf("loop2 :공유자원: %d\n", sharedresource);
        sharedresource ++;
        pthread_mutex_unlock(&mutex);  // 출구영역
        sleep(2);
    }
}
```

사례연구

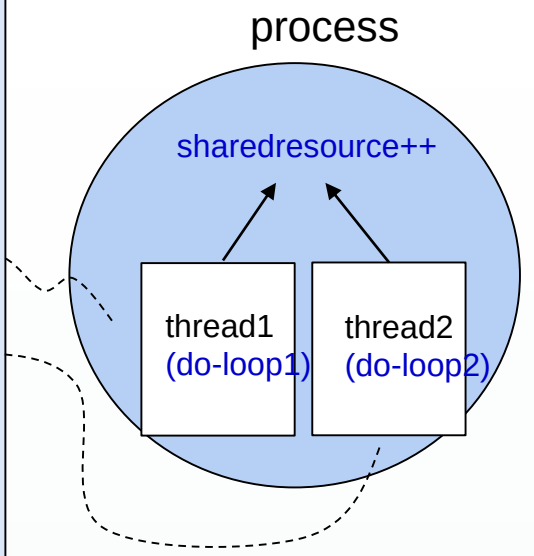
■ **상호배제**(Mutex) (Linux, C) – (**병렬프로그래밍에서 상호배제**)

mutex.c

```
int main()
{
    int       thr_id;
    pthread_t p_thread[2];
    int status;
    int a = 1;

    sharedresource = 0;
    thr_id = pthread_create(&p_thread[0], NULL, do_loop1, (void *)&a);
    sleep(1);
    thr_id = pthread_create(&p_thread[1], NULL, do_loop2, (void *)&a);

    pthread_join(p_thread[0], (void *) &status);
    pthread_join(p_thread[1], (void *) &status);

    status = pthread_mutex_destroy(&mutex);
    printf("code  = %d \n",status);
    return 0;
}
```

쓰레드를 만들고 do_loop1 함수실행

쓰레드를 만들고 do_loop2 함수실행



Q&A

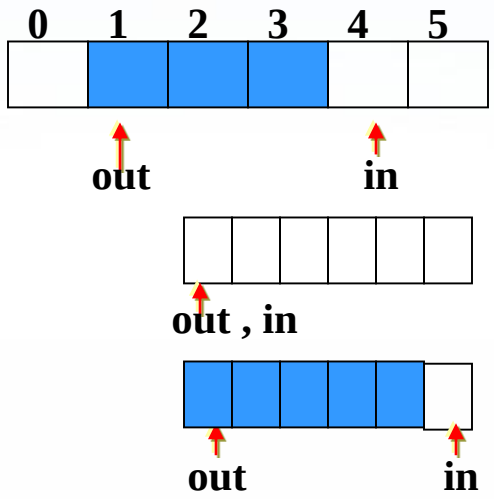
Remind !

■ Example (limited buffer)

☑ Limited buffer

```
char   buffer[n];
int    in,  out,   nextp,    nextc;
       in = 0;     out = 0;
```

buffer[6]



☑ 3가지 조건

1) Empty case ; in == out
2) Full case ; ((in + 1) % n) == out
3) Other case ; in > out

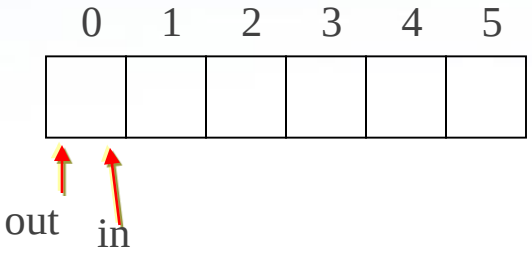
```
parbegin
   생산자 : begin
              repeat
                 …
                nextp 에서 한 항목 생산
                 …
                while (in + 1 ) mod n = out do   no-op ;
                buffer[in] := nextp ;
                in := in + 1 mod n ;
             until false ;
           end ;
```

```
   소비자 : begin
              repeat
                 while in = out do  no-op ;
                 nextc := buffer[out] ;
                 out := out + 1 mod n ;
                       …
                 nextc 에서 한 항목 소비
                       …
              until false ;
           end ;
parend ;
```

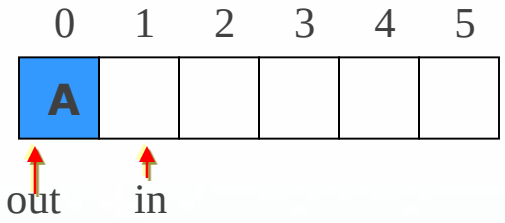
Remind !

- Example

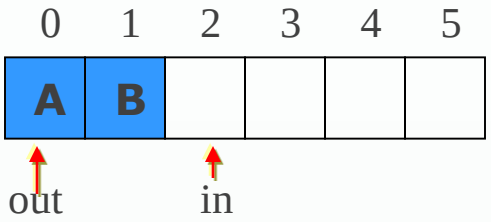
Step 1



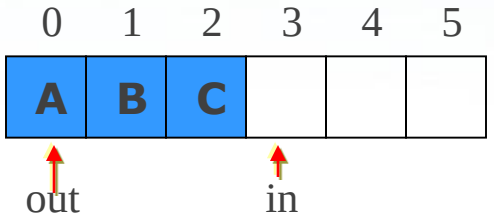
Step 2



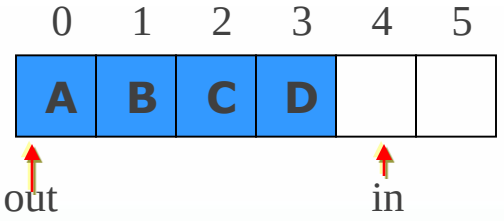
Step 3



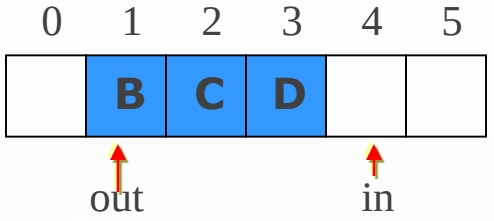
Step 4



Step 5



Step 6



Q&A