KWANGWOON UNIVERSITY

2019년 1학기 **시스템프로그래밍실습** 13주차

Process pool management

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

Assignment #4 – description

- **TCP pre-forked server (Assignment 4-1)**
  - Process pre-forking
  - Signal processing

- **Process pool management (Assignment 4-2)**
  - Shared memory, mutex, pthread
  - Process scheduling (Max & Min bound)

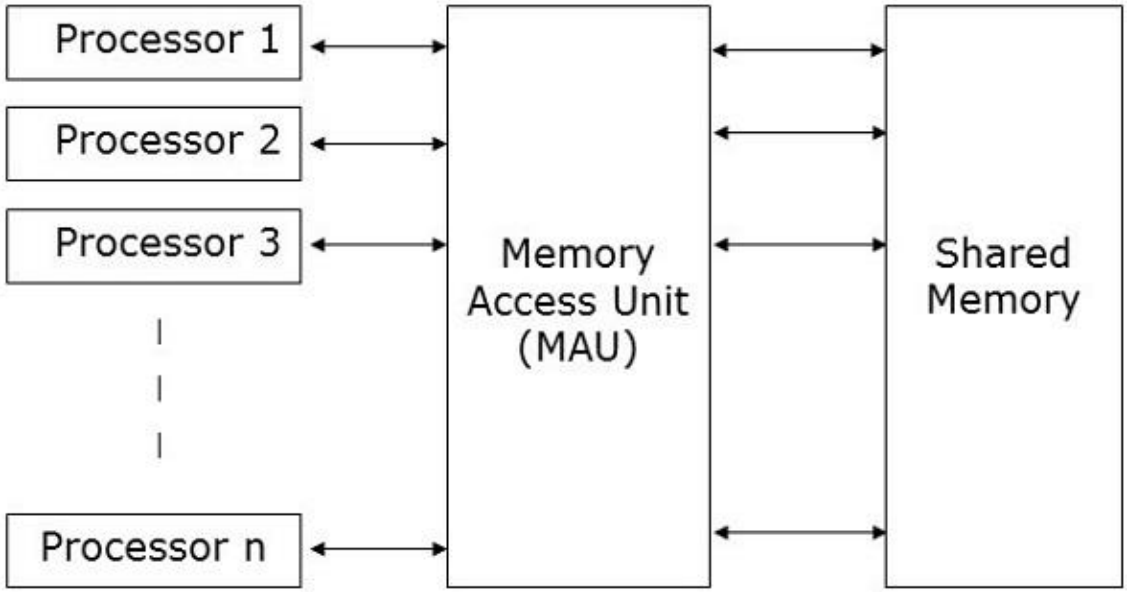
- **Mutual Exclusion (Assignment 4-3)**
  - Log file 작성

IPC (Inter-process communication)

- **두 프로세스 사이에 데이터를 주고 받는 통신 기능**
  - 프로세스 간에는 메모리를 공유하지 않기 때문에 프로세스 사이에서 데이터를 주고 받기 위해 IPC 기능을 이용
  - IPC 기능으로는 pipe, FIFO, **shared memory** 등이 있다.

Shared memory

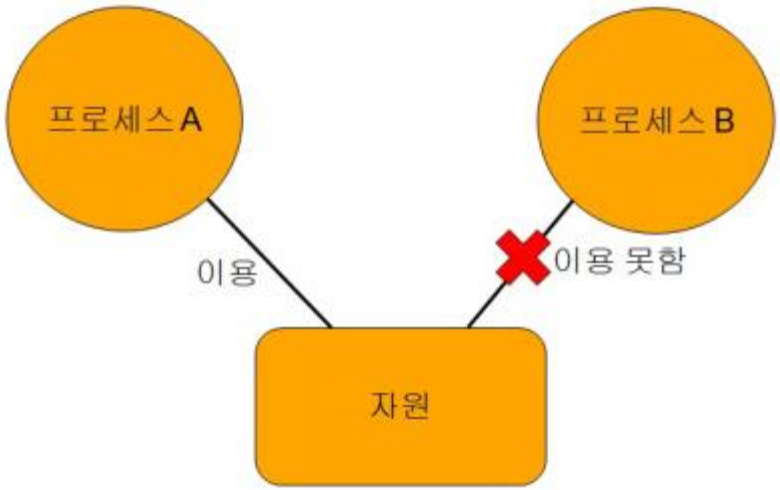
- 여러 개의 프로세스가 특정 메모리 공간을 동시에 접근해야 할 때 사용
- 여러 IPC 중에서 가장 빠른 수행 속도를 보여준다.



Semaphore

- **정의**
  - 여러 프로세스들이 한정된 수의 자원을 이용할 때, 한정된 수의 프로세스만 이용할 수 있게 하는 방법을 제시하는 개념

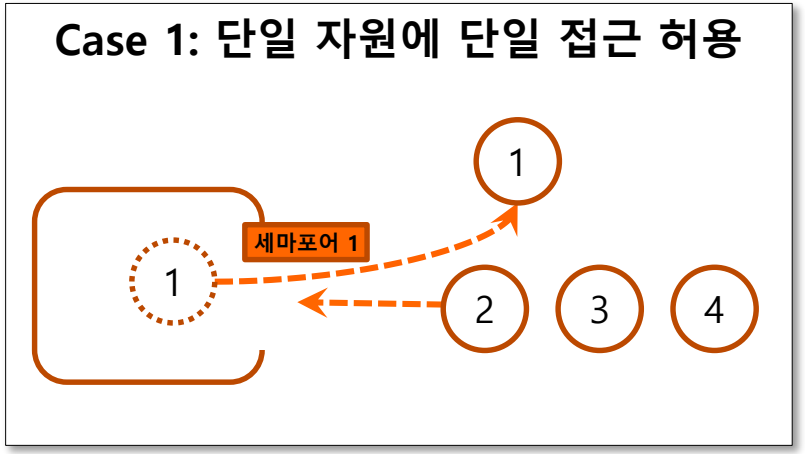
- **Linux의 Semaphore**
  - 다른 프로세스가 이미 가지고 있는 semaphore를 요청 시 그 프로세스는 휴면 (sleep)
  - 오랜 시간 동안 잡게 되는 lock에 적합
    - Context switch 비용 때문
  - 프로세스 문맥에서만 사용
    - Interrupt 문맥에서는 사용 불가
      - 대신, spinlock 사용

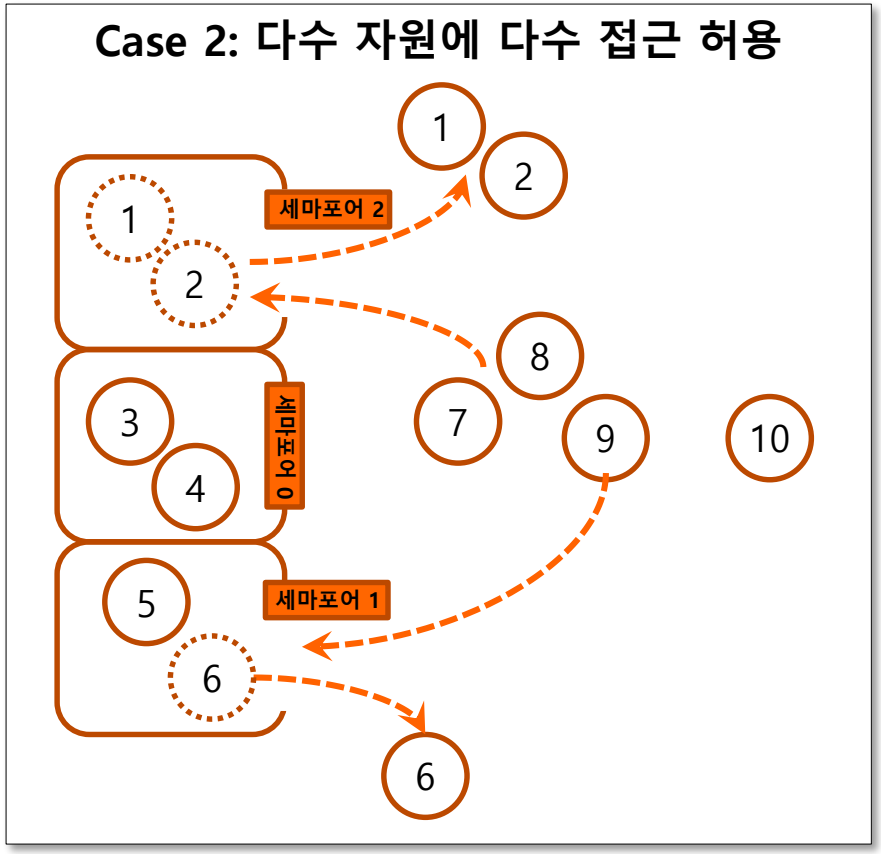


Semaphore (cont'd)

- **Semaphore operations**

1 : 프로세스, 세마포어 1 : 세마포어





shmget()

- **공유 메모리를 생성**

```
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget(key_t key, int size, int shmflg);
```

  - key_t key
    - 공유메모리를 구별하는 식별번호
  - int size
    - Shared memory size
  - int shmflg
    - IPC_CREAT
      - 키에 해당하는 공유 메모리 생성
      - 만약 있다면 무시하여 생성을 위해 권한을 지정해 주어야 한다.
    - IPC_EXCL
      - 공유메모리가 이미 존재하면 -1을 return
  - Return values
    - Success: share memory identifier, Fail: -1

shmat()

- **공유 메모리를 자신의 프로세스에 첨부 (자신의 메모리처럼 사용)**

```
#include <sys/ipc.h>
#include <sys/shm.h>

void *shmat(int shmid, void *shmaddr, int shmflg);
```

  - int shmid
    - 공유메모리를 구별하는 식별번호
  - void *shmaddr
    - 공유 메모리 위치를 지정할 주소
    - 위치를 자동으로 지정할 때에는 NULL 사용
  - int shmflg
    - SHM_RDONLY: 공유메모리를 읽기 전용으로
    - SHM_RND: shmaddr이 NULL이 아닌경우, shmaddr을 메모리 페이지 경계에 맞춤
  - Return values
    - Success: share memory address, Fail: -1

shmdt()

- **공유 메모리를 자신의 프로세스에서 분리**

```
#include <sys/ipc.h>
#include <sys/shm.h>

int shmdt(const void *shmaddr);
```

  - void *shmaddr
    - 공유 메모리 주소
  - Return values
    - Success: 0, Fail: -1

shmctl()

- **공유메모리의 정보를 읽거나 삭제**

```
#include <sys/ipc.h>
#include <sys/shm.h>

int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

  - int shmid
    - 공유메모리를 구별하는 식별번호
  - int cmd
    - 제어명령
      - IPC_RMID: 공유메모리 제거
      - IPC_STAT: 공유메모리 정보 얻기
  - struct shmid_df *buf
    - 공유메모리 정보를 얻기 위한 포인터
  - Return values
    - Success: 0, Fail: -1

Thread

- **Thread**
  - 프로세스 내에서 실행되는 흐름의 단위
  - 일반적으로 한 프로그램은 하나의 thread (main thread)를 가짐
  - 프로그램 환경에 따라 둘 이상의 Thread를 동시에 실행 가능 ➔ multithread

- **Fork와 Thread의 차이**
  - Fork는 process 생성시 모든 자료구조를 다시 생성하고 복사
  - Thread
    - Process의 code, data, heap 영역을 공유
    - Stack은 개별 소유

- **Thread를 사용한 소스 코드를 컴파일 시 lpthread 옵션 사용**
  - ex) gcc thread.c -o thread -lpthread

pthread_create()

- **Thread 생성**

```
#include <pthread.h>

int pthread_create(pthread_t *tid, const pthread_attr_t *attr, void
                                                *(func)(void *), void *arg);
```

  - pthread_t *tid
    - Thread ID
  - const pthread_attr_t *attr
    - Priority, initial stack size, etc.
  - void *(func)(void *)
    - Function pointer, this thread starts by calling this function
  - void *arg
    - Single pointer argument, func function is called with this argument
  - Return values
    - Success: 0, Fail: error code

pthread_exit()

- **Thread 종료**

```
#include <pthread.h>

void pthread_exit(void *ret_value);
```

  - 생성된 thread를 종료, return을 이용해서 종료할 수도 있음.

  - void *ret_value
    - return value.

pthread_join()

- **Thread 종료를 기다림**

```
#include <pthread.h>

int pthread_join(pthread_t *tid, void **status);
```

  - pthread_t *tid
    - Thread ID
  - void **status
    - If non-null, the return value from the thread is stored in the location pointed to by status
  - Return values
    - Success: 0, Fail: error code

pthread_detach()

- **Thread 분리**

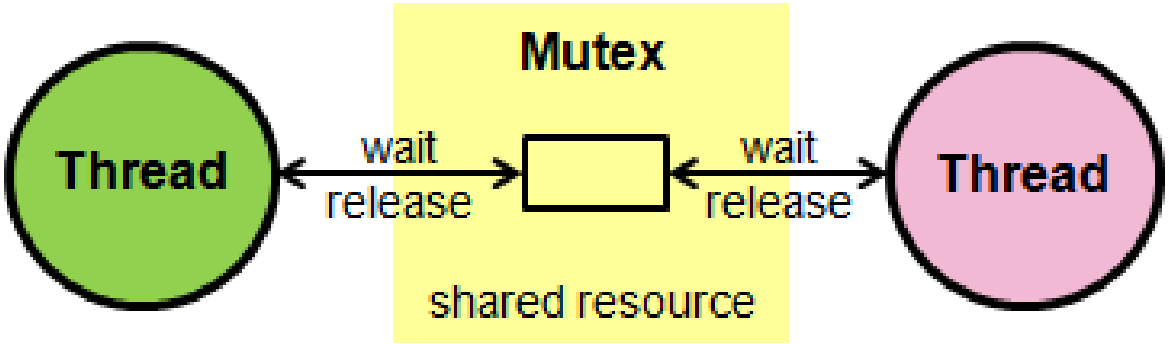
```
#include <pthread.h>

int pthread_detach(pthread_t tid);
```

  - 생성된 thread를 main thread와 분리 시킴

  - pthread_t tid
    - Thread ID
  - Return values
    - Success: 0, Fail: error code

Mutex

- Thread 동기화 문제를 해결하기 위한 방법 중 하나
- 공유 자원에 대해 한 번에 하나만 접근 가능
- Mutex가 걸려있으면 다른 Thread는 공유 자원에 접근 불가



Mutex functions

- **Mutex 초기화**

```
#include <pthread.h>

int pthread_mutex_init(pthread_mutex_t *mptr,
                       const pthread_mutex_attr *attr);
```

  - pthread_mutex_t *mptr
    - Mutex variable
    - It can be also initialized by the constant **PTHREAD_MUTEX_INITIALIZER**.
  - pthread_mutex_attr *attr
    - Set mutex attribute
  - Return values
    - Success: 0, Fail: -1

Mutex functions (cont'd)

- **Mutex 잠금 혹은 해제**

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mptr);
int pthread_mutex_unlock(pthread_mutex_t *mptr);
```

  - pthread_mutex_t *mptr
    - Mutex variable
  - Return values
    - Success: 0, Fail: error code

Mutex functions (cont'd)

- **Mutex 제거**

```
#include <pthread.h>

int pthread_mutex_destroy(pthread_mutex_t *mptr);
```

  - pthread_mutex_t *mptr
    - Mutex variable
  - Return values
    - Success: 0, Fail: -1

Lab.

- **Code**

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>

#define KEY_NUM     9527
#define MEM_SIZE    1024

pthread_mutex_t counter_mutex = PTHREAD_MUTEX_INITIALIZER;

int cnt;

void *doit1(void *);
void *doit2(void *);

int main(int argc, char **argv)
{
    pthread_t tidA, tidB, tidC;
    pthread_create(&tidA, NULL, &doit1, NULL);
    pthread_create(&tidB, NULL, &doit1, NULL);
    pthread_create(&tidC, NULL, &doit2, NULL);

    // wait for both threads to terminate
    pthread_join(tidA, NULL);
    pthread_join(tidB, NULL);
    pthread_join(tidC, NULL);

    return 0;
}
```

Lab. (cont'd)

- **Code** (cont'd)

```c
void *doit1(void *vptr)
{
    int i, val, shm_id;
    void *shm_addr;
    // Each thread fetches, prints, and increments
    // the counter NLOOP times. The value of the counter
    // should increase monotonically.

    if((shm_id = shmget( (key_t)KEY_NUM, MEM_SIZE, IPC_CREAT|0666)) == -1){
        printf("shmget fail\n");
        return NULL;
    }
    if((shm_addr = shmat(shm_id, (void*) 0, 0)) == (void*)-1){
        printf("shmat fail\n");
        return NULL;
    }

    for(i = 0; i < 10; i++){
        pthread_mutex_lock(&counter_mutex);
        val = cnt;
        sleep(1);
        sprintf((char* )shm_addr, "[%d] %d", (int)pthread_self(), val);

        cnt = val + 1;
        pthread_mutex_unlock(&counter_mutex);
        sleep(1);
    }

    return NULL;
}
```

Lab. (cont'd)

- **Code** (cont'd)

```c
void *doit2(void *vptr)
{
    int shm_id, i;
    void *shm_addr;
    char prev[32];

    if((shm_id = shmget((key_t)KEY_NUM, MEM_SIZE, IPC_CREAT|0666)) == -1){
        printf("shmget fail\n");
        return NULL;
    }
    if((shm_addr = shmat(shm_id, (void*)0, 0)) == (void* )-1){
        printf("shmat fail\n");
        return NULL;
    }

    memset((void*)prev, 0, 32);
    while(1){
        if(strcmp(prev, (char*)shm_addr)){
            printf("%s\n", (char*)shm_addr);
            strcpy(prev, (char*)shm_addr);
        }
        if(strstr((char*)shm_addr, " 19") != NULL){
            printf("%s\n", (char*)shm_addr);
            break;
        }
    }

    if(shmctl(shm_id, IPC_RMID, 0) == -1)
        printf("shmctl fail\n");

    return NULL;
}
```

Lab. (cont'd)

- **Results**

```
~/practice$ gcc -o mutex mutex.c -lpthread
~/practice$ ./mutex
[153224960] 0
[144832256] 1
[153224960] 2
[144832256] 3
[153224960] 4
[144832256] 5
[153224960] 6
[144832256] 7
[153224960] 8
[144832256] 9
[153224960] 1
[144832256] 11
[153224960] 12
[144832256] 13
[153224960] 14
[144832256] 15
[153224960] 16
[144832256] 17
[153224960] 18
[144832256] 19
[144832256] 19
```

KWANGWOON UNIVERSITY

2019년 1학기 **시스템프로그래밍실습** 13주차

Assignment 4-2

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

Assignment 4-2

- **Assignment 4-1에 다음의 사항을 변경 및 추가**
  - httpd.conf에 따라 프로세스, history개수 등을 조절

이름	내용
Maxchilds	최대 자식 프로세스 개수
MaxIdleNum	최대 Idel 프로세스 개수
MinIdleNum	최소 Idel 프로세스 개수
StartProcess	프로그램 시작 시 생성될 프로세스 개수
MaxHistory	출력되는 history의 수

  - 10초 마다, 공유메모리에 저장된 history 기록을 최신순서대로 출력
    - 자식 프로세스는 client의 접속이 들어올 때 마다, History정보를 공유메모리에 저장
      - 공유메모리에 저장된 history의 수가 MaxHistory수와 같으면 가장 오래된 history를 제거 후 저장

Assignment 4-2 (cont'd)

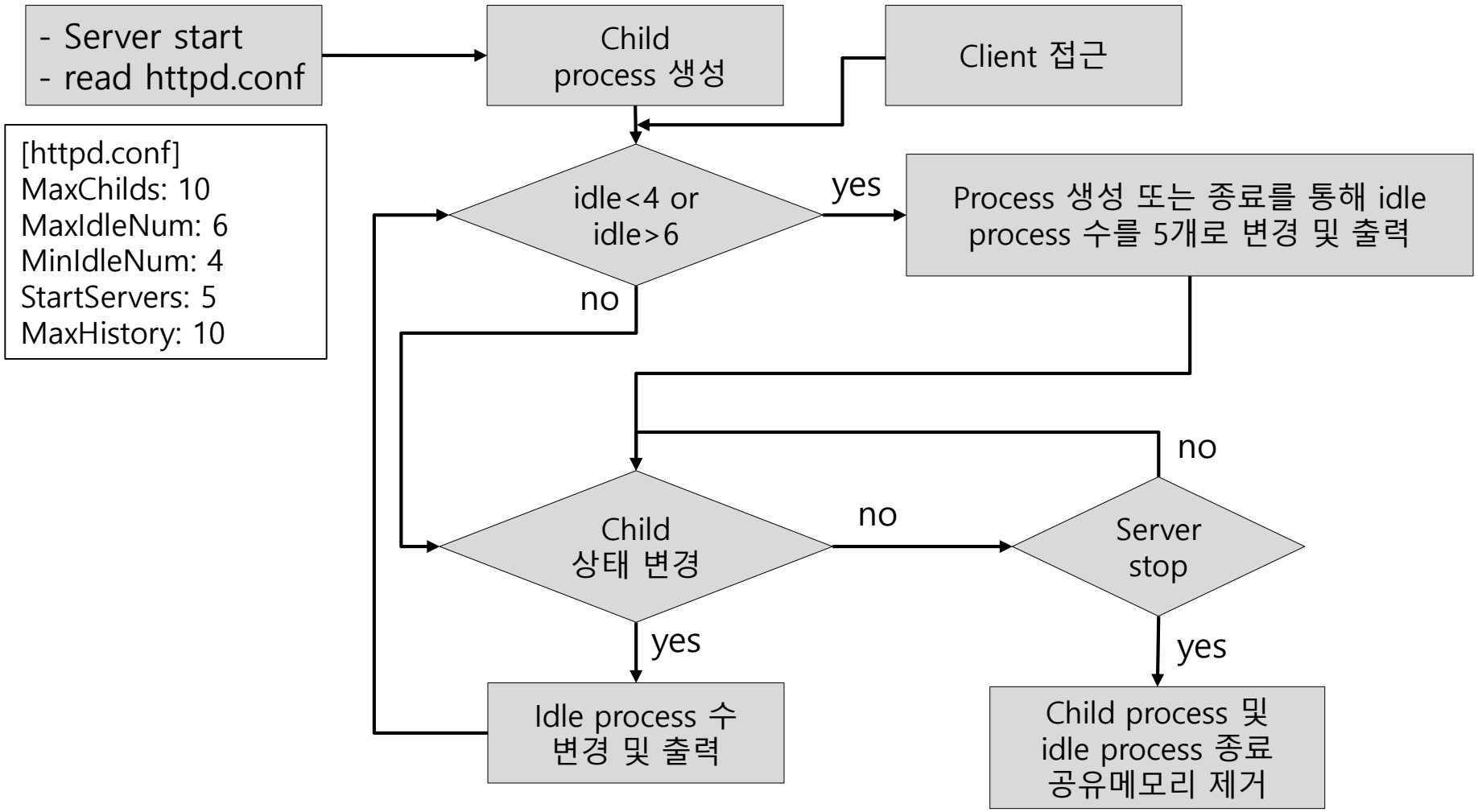
- **Assignment 4-1에 다음의 사항을 변경 및 추가**
  - **Process에서 공유메모리에 접근할 때 thread를 생성한 뒤 이를 활용**
    - fork를 사용하여 (StartProcess)개의 자식 프로세스 생성 (Assignment 4-1의 방식 유지)
    - Shared memory key value는 자신의 포트번호를 사용
    - 공유메모리 동기화 문제를 pthread_mutex_lock, pthread_mutex_unlock로 해결
  - **Idle process 관리**
    - Idle process: client와 연결이 되지 않은 자식 프로세스
    - **Idle process 수를 공유메모리로 관리**
    - Idle process의 수가 변경되는 경우 부모 프로세스에서 출력
    - Idle process의 수가 (MinIdleNum)개 미만이 되거나 (MaxIdleNum)개 초과가 되면 process를 생성 또는 종료하여 5개를 유지
      - Client와 연결된 자식 프로세스의 수와 idle process의 수의 합이 'Maxchilds'를 넘지않도록 조절

Assignment 4-2 (cont'd)

- **Assignment 4-1에 다음의 사항을 변경 및 추가** (cont'd)
  - 프로그램 종료 전 shared memory 제거 확인(SIGINT signal에서도 처리)
  - Client가 연결을 종료하면 자식 프로세스에서 종료 전 5초 sleep (by sleep() function)

  - 기타 내용은 결과 예제 참고

Assignment 4-2 (cont'd)



Assignment 4-2 (cont'd)

- **결과 화면**

```
[Wed May 11 17:49:03 2019] Server is started.

[Wed May 11 17:49:03 2019] 2134 process is forked.
[Wed May 11 17:49:03 2019] IdleProcessCount : 1
[Wed May 11 17:49:03 2019] 2135 process is forked.
[Wed May 11 17:49:03 2019] IdleProcessCount : 2
[Wed May 11 17:49:03 2019] 2136 process is forked.
[Wed May 11 17:49:03 2019] IdleProcessCount : 3
[Wed May 11 17:49:03 2019] 2137 process is forked.
[Wed May 11 17:49:03 2019] IdleProcessCount : 4
[Wed May 11 17:49:03 2019] 2138 process is forked.
[Wed May 11 17:49:03 2019] IdleProcessCount : 5
...
========== New client ==========
[Wed May 11 17:52:08 2019]
IP : 128.134.123.45
Port : 32822
================================

[Wed May 11 17:52:08 2019] IdleProcessCount : 4
...
```

StartProcess 값 만큼 프로세스 생성

Assignment 4-2 (cont'd)

- **결과 화면** (cont'd)

```
========== New client ==========
[Wed May 11 17:52:12 2019]
IP : 128.134.123.45
Port : 32823

[Wed May 11 17:52:12 2019] IdleProcessCount : 3
[Wed May 11 17:52:12 2019] 2139 process is forked.
[Wed May 11 17:52:12 2019] IdleProcessCount : 4
[Wed May 11 17:52:12 2019] 2140 process is forked.
[Wed May 11 17:52:12 2019] IdleProcessCount : 5

...
```

MinIdelNum > IdleProcessCount일 경우

Assignment 4-2 (cont'd)

- **결과 화면** (cont'd)

```
========== Disconnected client ==========
[Wed May 11 17:52:17 2019]
IP : 128.134.123.45
Port : 32823

[Wed May 11 17:52:17 2019] IdleProcessCount : 6
...
========== Disconnected client ==========
[Wed May 11 17:52:19 2019]
IP : 128.134.123.45
Port : 32822

[Wed May 11 17:52:19 2019] IdleProcessCount : 7
[Wed May 11 17:52:19 2019] 2134 process is terminated.
[Wed May 11 17:52:19 2019] IdleProcessCount : 6
[Wed May 11 17:52:19 2019] 2135 process is terminated.
[Wed May 11 17:52:19 2019] IdleProcessCount : 5
...
```

MaxIdelNum < IdleProcessCount일 경우

Assignment 4-2 (cont'd)

- **결과 화면** (cont'd)

```
…
==================== Connection History ========================
NO.       IP               PID      PORT    TIME
1         223.195.33.45    2335     22345   Wed May 11 18:07:34 2019
2         223.195.33.43    2333     22343   Wed May 11 18:05:34 2019
3         223.195.33.39    2332     22340   Wed May 11 18:03:20 2019
================================================================

…
[Wed May 11 18:53:02 2019] 2140 process is terminated.
[Wed May 11 18:53:02 2019] IdleProcessCount : 4
[Wed May 11 18:53:02 2019] 2139 process is terminated.
[Wed May 11 18:53:02 2019] IdleProcessCount : 3
[Wed May 11 18:53:02 2019] 2138 process is terminated.
[Wed May 11 18:53:02 2019] IdleProcessCount : 2
[Wed May 11 18:53:02 2019] 2137 process is terminated.
[Wed May 11 18:53:02 2019] IdleProcessCount : 1
[Wed May 11 18:53:02 2019] 2136 process is terminated.
[Wed May 11 18:53:02 2019] IdleProcessCount : 0
[Wed May 11 18:53:02 2019] Server is terminated.
```

Assignment 4-2 (cont'd)

- **Code Requirements**
  - 이전 과제 부분에 문제가 있는 경우 감점
  - 출력 형식에 맞지 않으면 감점
  - 소스코드의 50% 이상 주석을 달지 않은 경우 감점

- **Makefile Requirements**
  - 실행 파일이 "**ipc_server_####**"로 생성되도록 Makefile 작성
    - ####은 본인의 포트 번호
  - 컴파일 도중 warning 발생 시 감점
  - "$ make" 를 통해 실행 파일이 생성되지 않는 경우, 0점

Assignment 4-2 (cont'd)

- **Report Requirements**
  - 보고서 표지
    - 실습강의시간, 담당 교수님, 학번, 이름 필히 명시
  - 아래의 내용은 보고서에 필히 포함
    - Introduction
      - 5줄 내외 작성
    - Flowchart
      - 본인이 작성한 코드에 대한 flowchart 작성
    - Pseudo code
      - 본인이 작성한 코드에 대한 pseudo code 작성
    - Reference
      - 참고한 자료 명시(웹 페이지, 친구 등)
      - 강의 자료만 활용한 경우 생략 가능
      - **Copy 발생 시 reference를 참고하여 채점**
    - Conclusion

- **과제 질문 관련**
  - 해당 과제 출제 담당 조교에게 이메일로 문의 ➔ 박준택 조교 (juntaek@kw.ac.kr)
  - 과제 제출 마감 당일에는 오후 4시까지 도착한 질문 메일에만 답변

Assignment 4-2 (cont'd)

- **Softcopy Upload**
  - 제출 파일
    - 보고서 (파일명: 실습요일_4-2_학번.**pdf**), ls.c, Makefile
  - 위 파일들을 압축해서 제출 (파일명: 실습 요일_4-2_학번.tar.gz)
    - e.g. 월1,2 ➔ **mon**_4-2_2014722000.**tar.gz**
    - e.g. 화3,4 ➔ **thu**_4-2_2014722000.**tar.gz**
    - E.g. 금5,6 ➔ **fri**_4-2_2014722000.**tar.gz**
  - U-Campus의 과제 제출에 6월 7일(금) 23:59:59까지 제출
    - U-Campus에 올린 후 다시 다운로드 받아서 파일에 문제가 없는지 필히 확인
  - Delay 받지 않음
- **Hardcopy 제출하지 않음**