KWANGWOON UNIVERSITY

2022년 1학기 **시스템프로그래밍실습** 12주차

Synchronize Shared Resource

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

3st Assignment's Descriptions

- **Assignment 3-1**
  - Synchronize the shared resource

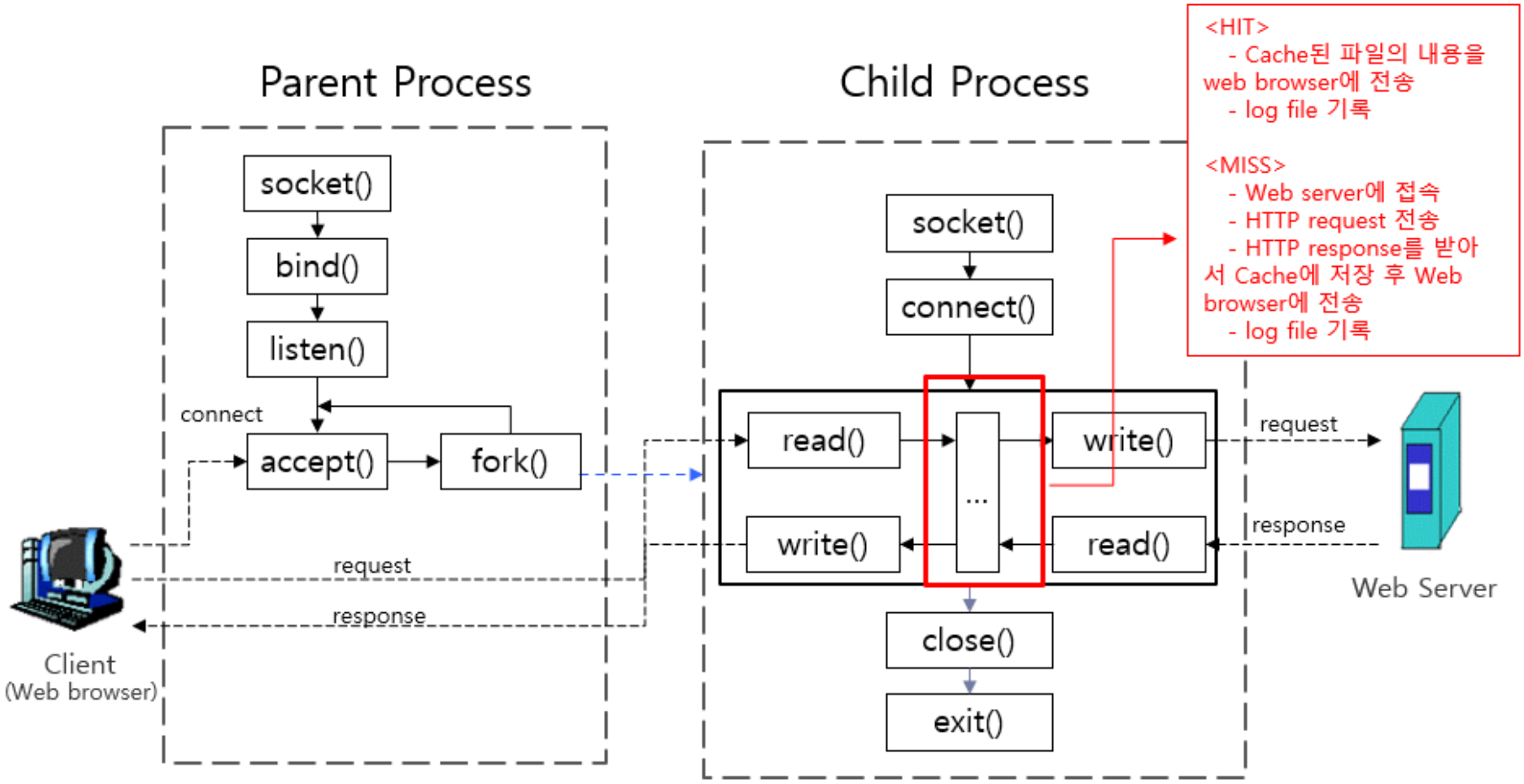
- **Assignment 3-2**
  - Display the status of the server and clients.

Critical-Section Problem

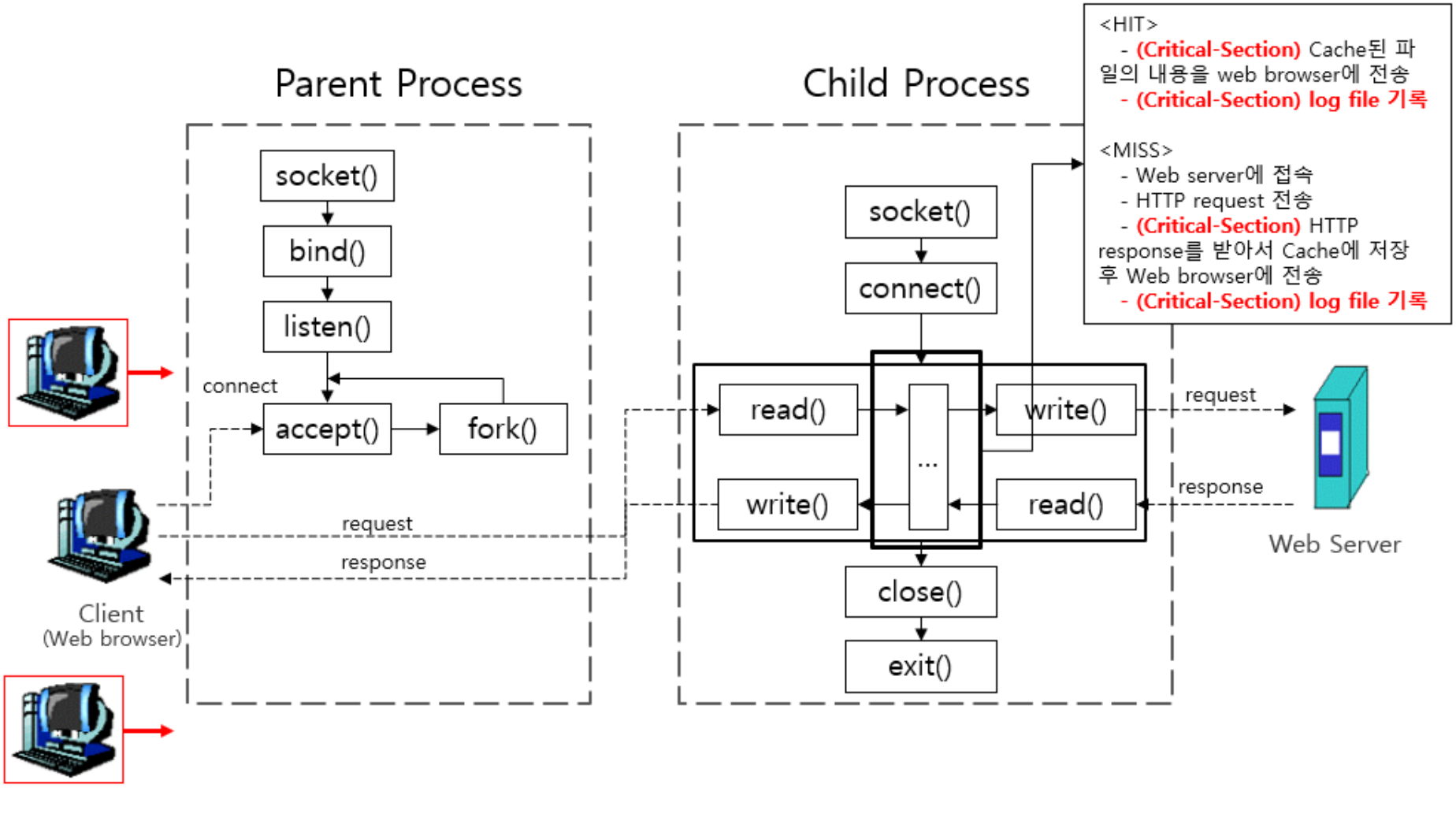
- **Race condition**
  - 여러 프로세스가 공유 데이터를 동시에 액세스하는 상황
  - 공유 데이터에 최종적으로 남는 데이터는 누가 제일 마지막에 기록했는지에 의존

- **한 프로세스가 critical section을 실행 중일 때,**
  - 그 critical section을 실행시키는 다른 프로세스가 없도록 보장

Proxy Server (1/2)

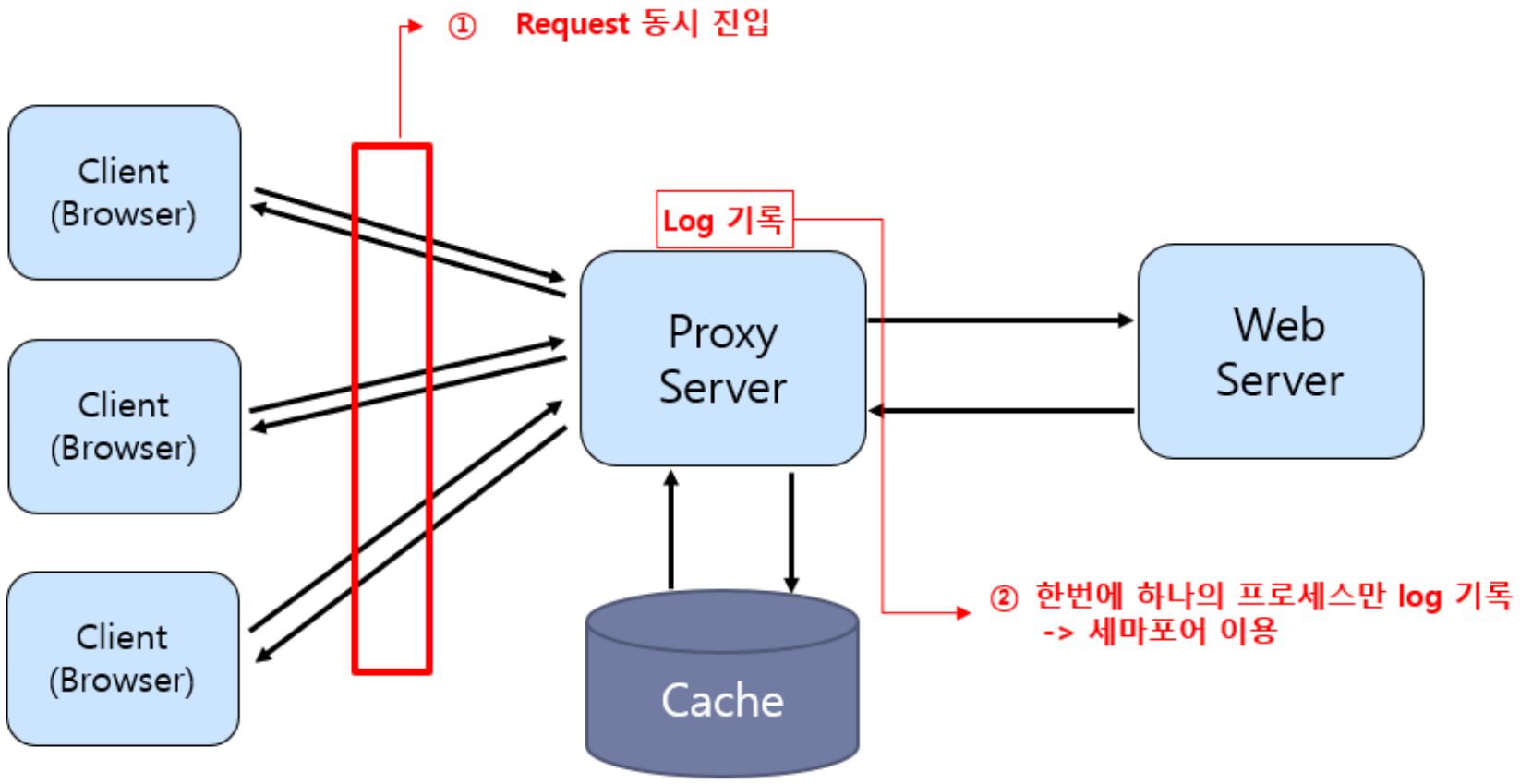


Proxy Server (2/2)



...

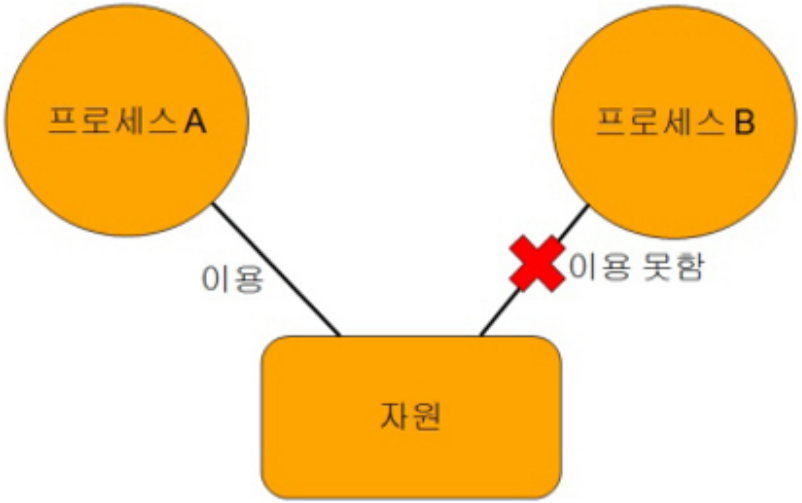
Proxy Server의 동작



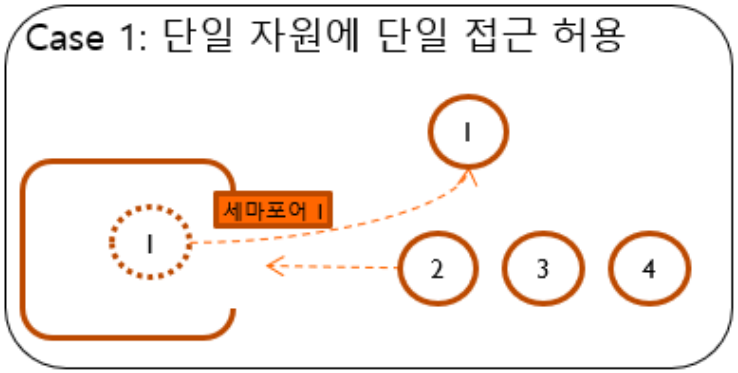
Semaphore

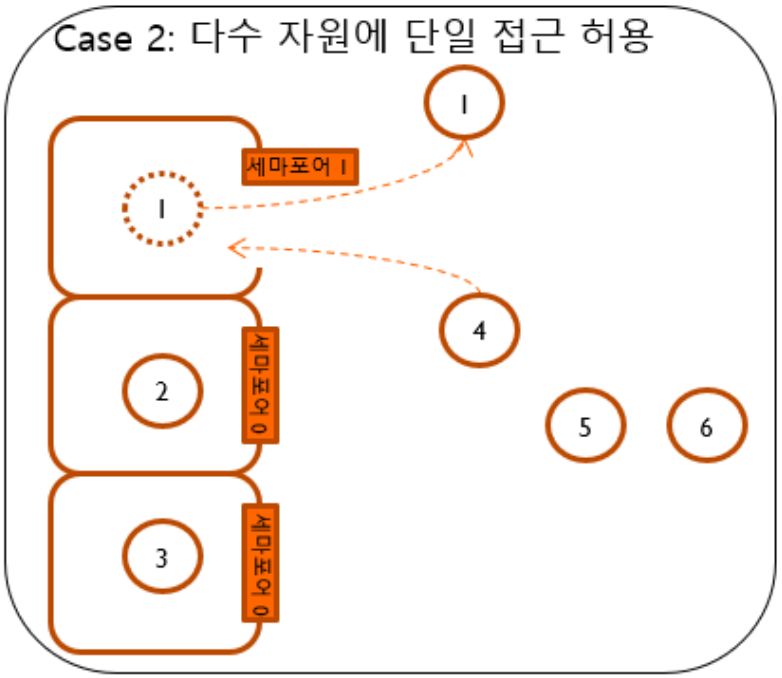
- **정의**
  - 여러 프로세스들이 한정된 수의 자원을 이용할 때, 한정된 수의 프로세스만 이용할 수 있게 하는 방법을 제시하는 개념

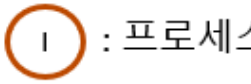
- **Linux의 Semaphore**
  - 다른 프로세스가 이미 가지고 있는 Semaphore를 요청 시 그 프로세스는 휴면(sleep)
  - 오랜 시간 동안 잡게 되는 lock에 적합
    - Context switch 비용 때문
  - 프로세스 문맥에서만 사용
    - Interrupt 문맥에서는 사용 불가
      - 대신, spinlock 사용

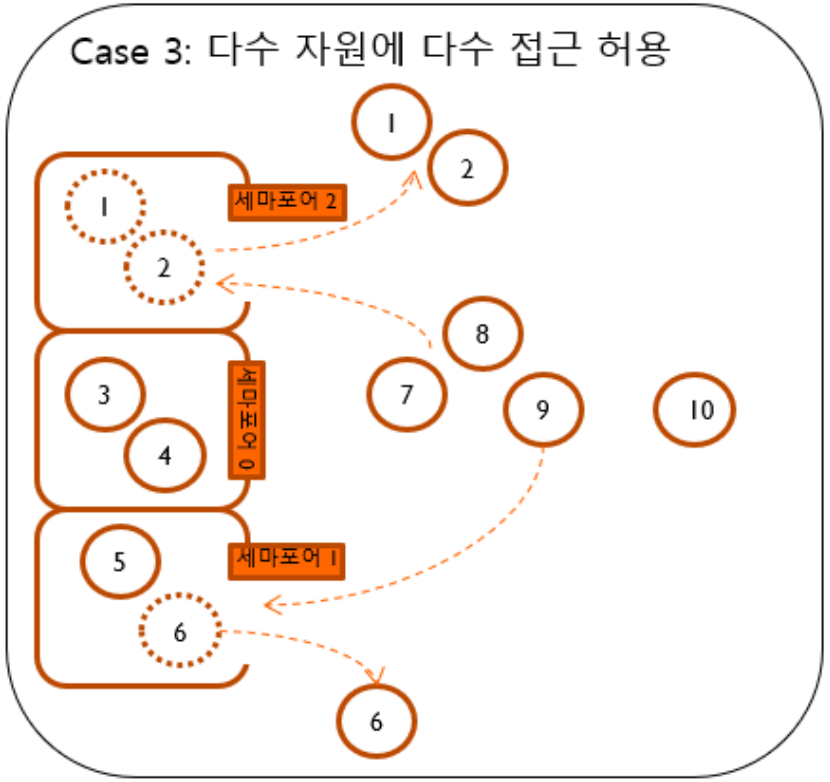


Semaphore Operations









Related Structure (1/2)

- **struct semid_ds**
  - Semaphore의 집합

필드명	내용
`struct ipc_perm sem_perm`	Semaphore의 접근권한
`time_t sem_otime`	최종 semop 호출 시간
`time_t sem_ctime`	최종 수정 시간
`unsigned short sem_nsems`	Semaphore 배열에 있는 semaphore의 수

- **struct sem**
  - Semaphore 집합에 포함되는 각 semaphore

필드명	내용
`unsigned short semval`	Semaphore의 값 (항상 0 이상)
`pid_t sempid`	마지막으로 연산을 수행한 프로세스의 pid

Related Structure (2/2)

- **struct sembuf**
  - semop() 함수는 이 구조체의 array를 사용

필드명	내용
`unsigned short sem_num`	array내에서 세마포어의 번호 (0, 1, …, nsems-1)
`short sem_op`	세마포어 operation (음수, 0, 양수)
`short sem_flg`	세마포어의 플래그를 세팅. 일반적으로 SEM_UNDO로 세팅. 프로세스 종료시 자동으로 세마포어 해제

- **union semun**
  - semctl() 함수 등에서 사용

필드명	내용
`int val`	SETVAL을 위한 값으로 활용
`short semid_ds *buf`	IPC_STAT, IPC_SET을 위한 버퍼로 사용
`unsigned short *array`	GETALL, SETALL 명령을 위한 배열
`struct seminfo *__buf`	IPC_INFO를 위한 버퍼로 사용

Semaphore APIs (1/3)

- **int semop (int sem_id, struct sembuf *ops, unsinged nsops);**
  - 한 Semaphore 집합에 대한 일련의 연산들을 원자적으로 수행
  - **int sem_id**
    - semget에서 얻어온 Semaphore의 ID.
  - **struct sembuf *ops**
    - Semaphore에 대해 수행하고자 하는 연산을 지정한 sembuf 타입의 배열에 대한 포인터.

```
struct sembuf {
        short sem_num;      // Semaphore의 번호.
        short sem_op;
        short sem_flg;      // Semaphore의 flag를 셋팅하는데 사용되는 값.
};
```

➔ Flag는 일반적으로 SEM_UNDO가 사용. SEM_UNDO가 셋팅되면 Semaphore에 대한 변경 들이 관리가 되며 프로세스가 종료할 때 특별히 Semaphore를 제거하지 않아도 자동으로 해제가 된다.

  - **unsigned nops**
    - 연산의 개수

Semaphore APIs (2/3)

- **int semctl (int sem_id, int semnum, int cmd, union semun arg);**
  - Semaphore를 제어하는 함수
    - 특히, Semaphore의 초기값을 설정.
  - **int sem_id**
    - semget을 통해 얻어온 Semaphore의 ID.
  - **int semnum**
    - Semaphore의 멤버 번호

Semaphore APIs (3/3)

- **int semctl (int sem_id, int semnum, int cmd, union semun arg);**
  - **int cmd**
    - Semaphore에 수행할 명령

Name	Function
IPC_STAT	semid_ds 구조체를 arg.buf에 저장하여 semaphore 집합의 정보 획득
IPC_SET	arg.buf의 semid_ds struct가 가진 내용을 바탕으로 semaphore의 접근 권한을 변경
IPC_RMID	semaphore 삭제
GETVAL	semnum으로 지정된 sem.semval 값을 획득
SETVAL	semnum으로 지정된 sem.semval 값을 arg.val로 설정
GETPID	마지막으로 semop() 함수를 실행한 프로세스의 PID 획득
GETALL	모든 semaphore 값을 얻은 뒤, arg.array가 가리키는 배열에 저장
SETALL	모든 semaphore의 값을 arg.array가 가리키는 배열의 값으로 세팅
GETNCNT	semnum으로 지정된 sem.semval이 현재 값보다 커지길 기다리는 프로세스 개수
GETZCNT	semnum으로 지정된 sem.semval이 0이 되길 기다리는 프로세스 개수

실습1. Semaphore – Example (1/3)

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

void repeat(int semid);
void p(int semid);
void v(int semid);


void repeat(int semid)
{
        srand((unsigned int)getpid());
        p(semid);
        printf("%d process is using tool\n",getpid());
        sleep(rand()%5);
        printf("%d process is returning tool\n",getpid());
        v(semid);

        exit(0);
}

void p(int semid){
        struct sembuf pbuf;
        pbuf.sem_num = 0;
        pbuf.sem_op = -1;
        pbuf.sem_flg = SEM_UNDO;
        if((semop(semid, &pbuf,1)) == -1){
                perror("p : semop failed");
                exit(1);
        }
}
```

실습1. Semaphore – Example (2/3)

```
void v(int semid){
        struct sembuf vbuf;
        vbuf.sem_num = 0;
        vbuf.sem_op = 1;
        vbuf.sem_flg = SEM_UNDO;
        if((semop(semid, &vbuf,1)) == -1){
                perror("v : semop failed");
                exit(1);
        }
}

main(){
        int semid, i;
        union semun{
                int val;
                struct semid_ds *buf;
                unsigned short int *array;
        } arg;

        if((semid = semget((key_t)1234,1,IPC_CREAT|0666)) == -1){
                perror("semget failed");
                exit(1);
        }

        arg.val =1;
        if((semctl(semid,0,SETVAL,arg)) == -1){
                perror("semctl failed");
                exit(1);
        }
```

실습1. Semaphore – Example (3/3)

```
        for(i=0;i<3;i++){
                if(!fork())
                repeat(semid);
        }

        sleep(10);

        if((semctl(semid,0,IPC_RMID,arg)) == -1){
                perror("semctl failed");
                exit(1);
        }
        exit(0);
}
```

```
sslab@sslab-VirtualBox:~$ ./sem
3728 process is using tool
3728 process is returning tool
3729 process is using tool
3729 process is returning tool
3730 process is using tool
3730 process is returning tool
sslab@sslab-VirtualBox:~$
```

KWANGWOON UNIVERSITY

2022년 1학기 **시스템프로그래밍실습**

Proxy #3-1

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

Proxy #3-1

- **Requirements**
  - **2-4 과제에서 logfile에 관한 동시접근 제어 추가**
  - **semkey 값은 port number와 같게 할 것**
  - Semaphore를 사용하여 한 번에 하나의 프로세스만 log file에 기록하도록 수정
  - 동시에 여러 프로세스가 접근했을 때를 시뮬레이션하여 서버의 터미널에 상태 출력
    - 시뮬레이션 시나리오 보고서에 작성
    - Critical section 내에 sleep() 등을 사용하면 동시 접근하는 경우를 볼 수 있음
    - 터미널 출력 화면
    - 터미널 양식
    - *PID# getpid() is waiting for the semaphore.
    - *PID# getpid() is in the critical zone.
    - *PID# getpid() exited the critical zone.

```
sslab@ubuntu:~$ ./proxy_cache
*PID# 22496 is waiting for the semaphore.
*PID# 22496 is in the critical zone.
*PID# 22498 is waiting for the semaphore.
*PID# 22515 is waiting for the semaphore.
*PID# 22516 is waiting for the semaphore.
*PID# 22517 is waiting for the semaphore.
*PID# 22520 is waiting for the semaphore.
*PID# 22523 is waiting for the semaphore.
*PID# 22522 is waiting for the semaphore.
*PID# 22524 is waiting for the semaphore.
*PID# 22496 is waiting for the semaphore.
*PID# 22496 exited the critical zone.
*PID# 22498 is in the critical zone.
```