

2022년 2학기 운영체제 & 운영체제 실습

# Assignment 3

**System Software Laboratory**

School of Computer and Information Engineering

Kwangwoon Univ.

# Requirements

- **Ubuntu 16.04.5 Desktop 64bits 환경에서 채점**
- **Copy 발견 시 0점 처리**
- **보고서**
  - **표지**
    - 수업 명, 과제 이름, 담당 교수님, 학번, 이름 필히 명시
      - 과제 이름 → assignment 3
  - **아래의 내용은 보고서에 필히 포함**
    - **Introduction** : 4줄 이상(background 제외) 작성
    - **Conclusion & Analysis** : 수행한 내용을 캡처 및 설명
    - **고찰** : 과제를 수행하면서 느낀점 작성
    - **Reference**
      - 과제를 수행하면서 참고한 내용을 구체적으로 기록
      - 강의자료만 이용한 경우 생략 가능

# Requirements (2/3)

## ■ Source

- 아래 소스파일명과 다르게 작성하여 제출 시 감점처리됩니다.
- assignment #3-1
  - numgen.c // 테스트용 파일 생성 코드
  - fork.c // fork를 이용하여 구현한 코드
  - thread.c // pthread를 이용하여 구현한 코드
  - Makefile // numgen.c, fork.c , thread.c 을 모두 컴파일 하도록 작성
- assignment #3-2
  - filegen.c // 테스트용 파일 생성 코드
  - schedtest.c // 성능 측정 코드
  - Makefile // 두 application을 모두 컴파일 하도록 작성
- assignment #3-3
  - process\_tracer.c // wrapping 코드를 포함한 모든 코드
  - Makefile
  - fork() 호출 횟수를 count 하기위해 수정한 커널 소스 코드
    - fork.c
    - sched.h

## ■ Copy 발견 시 0점 처리

# Requirements (cont'd)

- **Softcopy만 작성(Hardcopy 받지 않음)**
- **제출 파일**
  - 보고서(.pdf) + Source file 하나의 압축파일로 압축하여 제출(**tar.gz**)
  - Tar 압축 및 해제 방법
    - 압축 시 → `tar -zcvf [압축 파일명].tar.gz[폴더 명]`
    - 해제 시 → `tar -zxvf 파일명.tar.gz`

- **보고서 및 압축 파일 명 양식**

- **os\_과제번호\_학번\_수강분류코드**


수강요일	이론_월수	이론_금12	실습
수강분류코드	A	B	C

- e.g.
    - 이론\_월수 수강하는 학생인 경우  
(보고서) **os\_3\_20221234567\_A.pdf** (압축파일) **os\_3\_20221234567\_A.tar.gz**
    - 이론\_금12 수강하는 학생인 경우  
(보고서) **os\_3\_20221234567\_B.pdf** (압축파일) **os\_3\_20221234567\_B.tar.gz**
    - 실습 수강하는 학생인 경우  
(보고서) **os\_3\_20221234567\_C.pdf** (압축파일) **os\_3\_20221234567\_C.tar.gz**

# Requirements (cont'd)

- 실습 수업을 수강하는 학생인 경우

- 실습 과목에 과제를 제출(.tar.gz)
- 이론 과목에 간단한 .txt 파일로 제출

 실습수업때제출했습니다.

2022-08-29 오후 3:58    텍스트 문서

OKB

- 이론 과목에 .txt 파일 미 제출 시 감점

- 과제 제출

- KLAS – 강의 과제 제출
- 2022년 11월 10일 목요일 23:59까지 제출
  - 딜레이 받지 않음
    - 제출 마감 시간 내 미제출시 해당 과제 0점 처리
    - 교내 서버 문제 발생 시, 메일로 과제 제출 허용

# Assignment 3

## 주의 사항

### ▶ 매 실험 전에 아래의 명령어를 수행할 것

- ▶ 캐시 및 버퍼를 비워서 실험에 영향을 주는 요소를 제거

- ▶ **rm -rf tmp\***

- ▶ **\$ sync**

- ▶ Linux command to flush file system buffer

- ▶ **\$ echo 3 | sudo tee /proc/sys/vm/drop\_caches**

- ▶ Linux commands to free pagecache, dentrmries, and inodes

### ▶ 자식프로세스에서 부모프로세스로 값을 넘겨줄 때 (**exit ()**)

반환 값은  $2^8$  이상이면 안되며, 8-bit 만큼 right shift 해주어야 child process가 반환된 값을 정상적으로 확인할 수 있음. (e.g. `a >> 8`)

→ 이유를 보고서에 작성

# Assignment 3-1

- 다음의 작업을 다중 프로세스/스레드로 수행하는 프로그램을 작성
  - 프로그램 1. **numgen** (numgen.c)
    - Step 1. 특정 파일("temp.txt")를 생성
    - Step 2. **i번째 값을 integer형 양수로** 생성할 프로세스 수의 2배만큼 (**MAX\_PROCESSES**) 기록

numgen.c

```
...
#define MAX_PROCESSES 4
...
FILE *f_write = fopen(...)

for(i=0; i<MAX_PROCESSES*2; i++)
{
    fprintf(f_write, "%d", i+1);
    ...
}

fclose(f_write);
...
```

temp.txt

```
1
2
3
4
5
6
7
8
```

# Assignment 3-1

- 다음의 작업을 다중 프로세스/스레드로 수행하는 프로그램을 작성
  - 프로그램 2. fork.c, thread.c
    - 프로세스를 수행하는 파일(fork.c)
    - 스레드로 수행하는 파일(thread.c)
    - 프로그램 명
      - **fork → fork.c**
      - **thread → thread.c**
    - **결과에 대한 분석 내용 작성**
    - Step 1. **MAX\_PROCESSES** 만큼 프로세스 또는 스레드를 생성
      - **MAX\_PROCESSES = 8, 64**
    - Step 2. 최상단 프로세스/스레드마다 2개의 숫자를 읽음.
    - Step 3. 각 프로세스/스레드는 두 개의 숫자를 더한 후,  
부모 프로세스/스레드에게 값을 전달 ( fork → **exit()** 사용)
    - Step 4. 최종적으로 나온 값, 전체 프로그램 수행시간 측정
      - clock\_gettime() 사용



```
void main(void){
    :
    clock_gettime( );
    if(!fork()){
        :
    }
    wait(&result);
    clock_gettime( );
    :
}
```



# Assignment 3-1

- 다음의 작업을 다중 프로세스/쓰레드로 수행하는 프로그램을 작성

- Step 4. 최종적으로 나온 값, 전체 프로그램 수행시간 측정

- **결과에 대한 분석 내용 작성**

< MAX\_PROCESS = 8 >

```
sslab@ubuntu:~/assign_3/3-1$ ./fork
value of fork : 136
0.003074
sslab@ubuntu:~/assign_3/3-1$ ./thread
value of thread : 136
0.001764
```

< MAX\_PROCESS = 64 >

```
sslab@ubuntu:~/assign_3/3-1$ ./fork
value of fork : 64
0.020213
sslab@ubuntu:~/assign_3/3-1$ ./thread
value of thread : 8256
0.015052
```

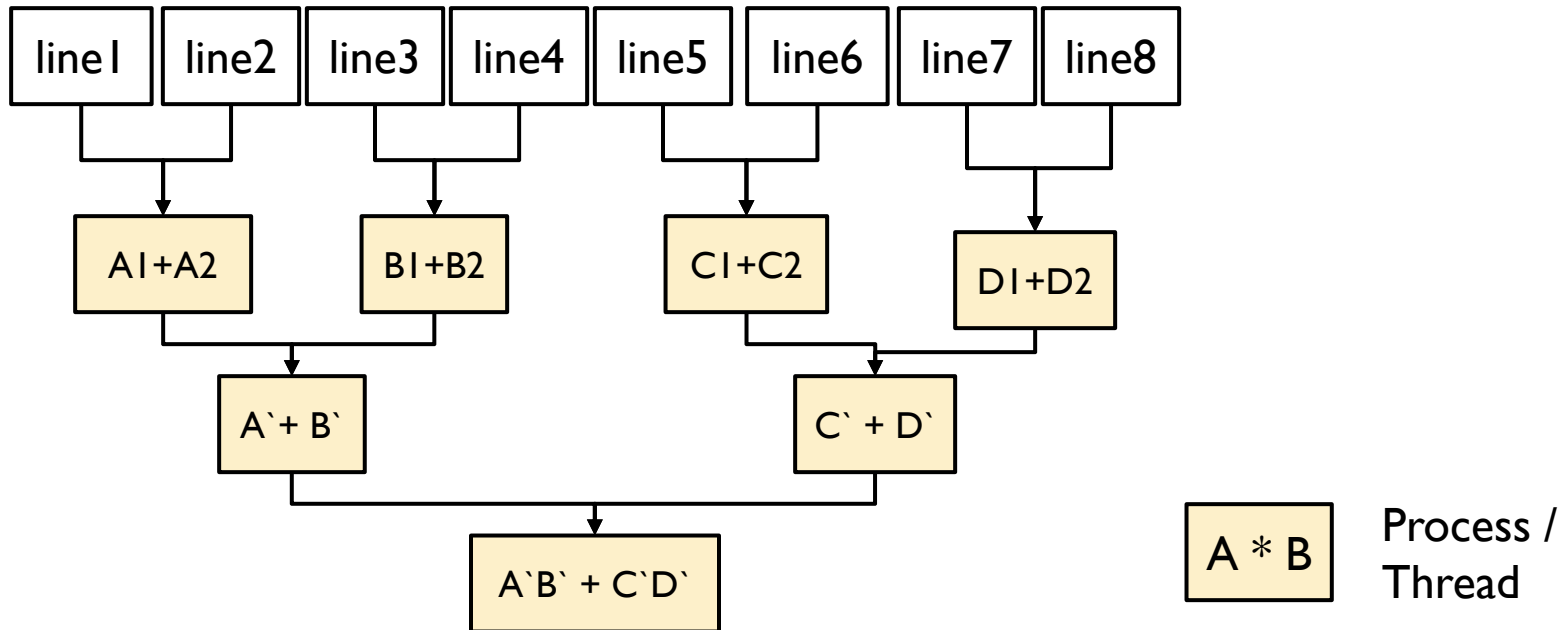
# Assignment 3-1

## <Example>

- numgen.c
  - Ex) MAX\_PROCESS가 4이면, 숫자 8개를 기록

---

## ▪ fork.c / thread.c



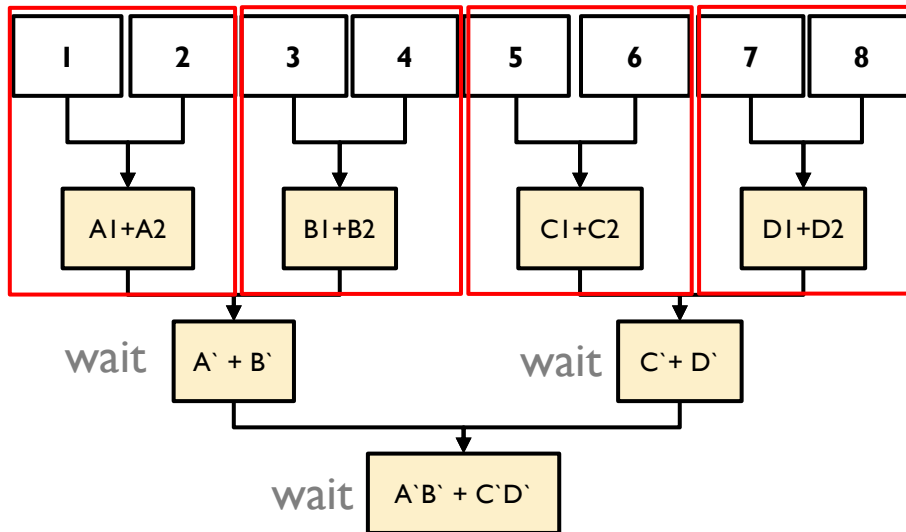
# Assignment 3-1

## Requirements

- 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
  - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

## Hint

- MAX\_PROCESS가 4개 가정



temp.txt

1  
2  
3  
4  
5  
6  
7  
8

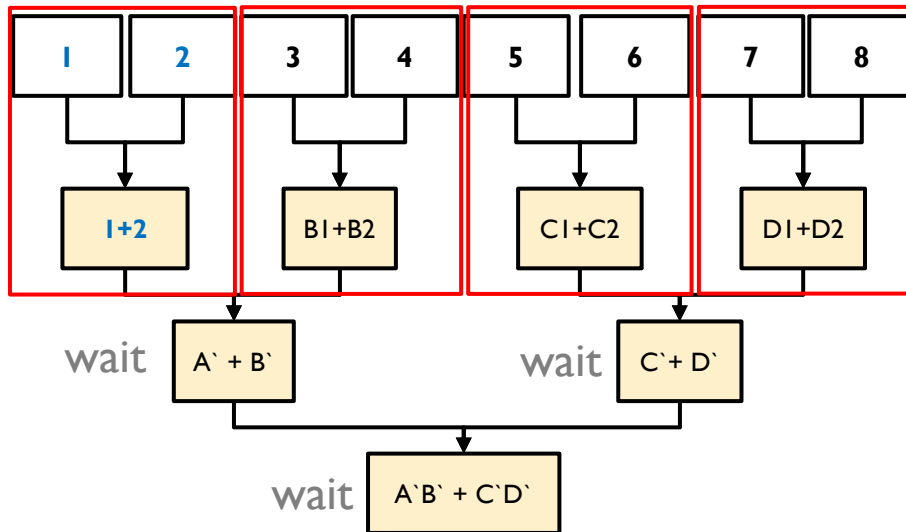
# Assignment 3-1

## Requirements

- 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
  - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

## Hint

- MAX\_PROCESS가 4개 가정**



temp.txt

1  
2  
3  
4  
5  
6  
7  
8  
3

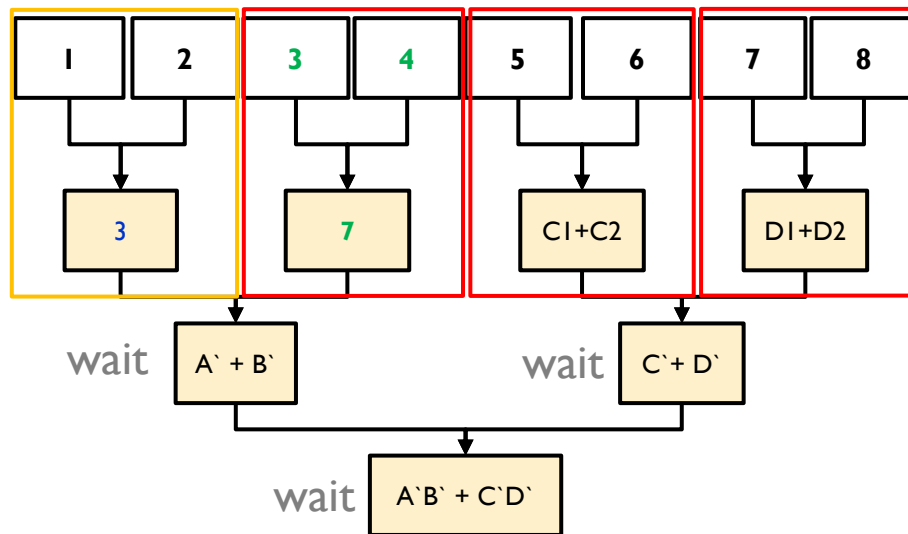
# Assignment 3-1

## Requirements

- 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
  - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

## Hint

- MAX\_PROCESS가 4개 가정



temp.txt

```
1
2
3
4
5
6
7
8
3
7
```

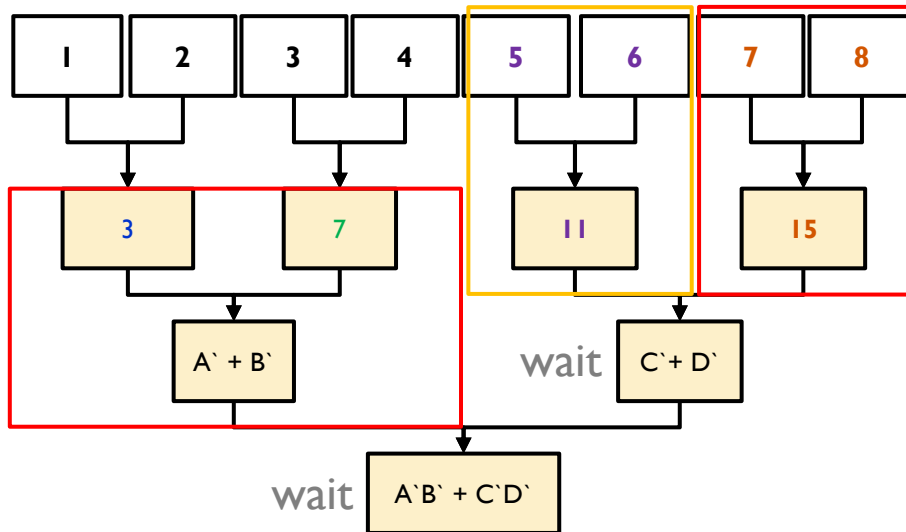
# Assignment 3-1

## Requirements

- 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
  - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

## Hint

- MAX\_PROCESS가 4개 가정



temp.txt

```
1
2
3
4
5
6
7
8
3
7
11
15
```

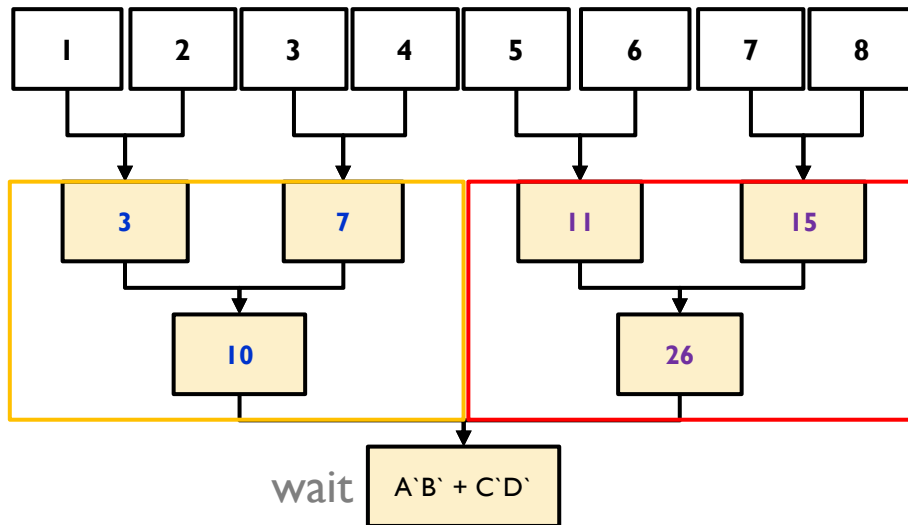
# Assignment 3-1

## Requirements

- 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
  - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

## Hint

- MAX\_PROCESS가 4개 가정



temp.txt

```
1  
2  
3  
4  
5  
6  
7  
8  
3  
7  
11  
15  
10  
26
```

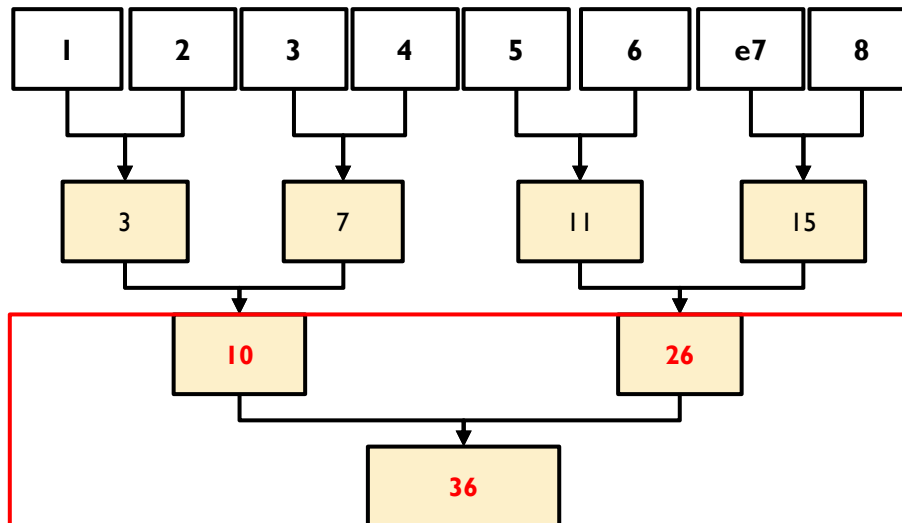
# Assignment 3-1

## Requirements

- 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
  - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

## Hint

- MAX\_PROCESS가 4개 가정



temp.txt

```
1
2
3
4
5
6
7
8
3
7
11
15
10
26
36
```



# Assignment 3-2

- 각 프로세스에서 CPU 스케줄링 정책을 변경
  - 프로그램 1. **filegen** (filegen.c)
    - Step 1. 특정 디렉토리("temp")를 생성
    - Step 2. 이에 **무작위의 integer형 양수( $\leq 9$ )**가 기록되어 있는 파일 (./temp/0, ./temp/1, ./temp/2, ...)을 **생성할 프로세스만큼(MAX\_PROCESSES)** 생성

```
...  
#define MAX_PROCESSES 10000  
...  
    for(i=0; i<MAX_PROCESSES; i++)  
    {  
        FILE *f_write = fopen(...  
        fprintf(f_write, "%d", 1+rand()%9);  
        fclose(f_write);  
        ...  
    }  
...
```

# Assignment 3-2

- 각 프로세스에서 CPU 스케줄링 정책을 변경
  - 프로그램 2. **schedtest** (schedtest.c)
  - 결과에 대한 분석 내용 작성.
    - Step 1. **MAX\_PROCESSES** 만큼 프로세스를 생성 → **fork()** 사용
    - Step 2. 각 프로세스에서 CPU 스케줄링 정책을 변경 (Hint. sched\_setscheduler())
    - Step 3. 각 i(단,  $0 \leq i < \text{MAX\_PROCESSES}$ ) 번째 프로세스에서 미리 만들어져 있는 i 번째 파일(temp/i)에서 integer 데이터 읽음
  - Hint. 3-1의 fork.c에서 파일을 읽는 부분만 수정하여 사용
  - 성능을 비교할 수 있을 정도의 프로세스를 생성하여 실험할 것
    - ex.) **MAX\_PROCESS = 10000**

# Assignment 3-2

- 프로그램 2. schedtest (schedtest.c)
- 매 실험 전에 소스코드에 위치한 디렉토리에 아래의 명령어를 수행할 것
  - 캐시 및 버퍼를 비워서 실험에 영향을 주는 요소를 제거하기 위함
  - `$ rm -rf tmp*`
  - `$ sync`
  - `$ echo 3 | sudo tee /proc/sys/vm/drop_caches`
- Linux에서 지원하는 3가지 CPU 스케줄링과 Priority, Nice의 값을 각각 highest, default, lowest로 설정
  - CPU 스케줄링
    1. The standard round-robin time-sharing policy
    2. A first-in first-out policy
    3. A round-robin policy
  - 각 CPU 스케줄링의 Priority, Nice의 highest, default, lowest 설정 한 값 보고서에 작성

# Assignment 3-3

- **PID를 바탕으로 아래와 같은 프로세스의 정보를 출력하는 Module 작성**
  - (1) 프로세스 이름
  - (2) 현재 프로세스의 상태
    - Running or ready, Wait with ignoring all signals, Wait, Stopped, Zombie process, Dead, etc.
  - (3) 프로세스 그룹 정보
    - PGID, 프로세스 이름
  - (4) 해당 프로세스를 실행하기 위해 수행된 context switch 횟수
  - (5) fork()를 호출한 횟수
  - (6) 부모(parent) 프로세스 정보
    - PID, 프로세스 이름
  - (7) 형제자매(sibling) 프로세스 정보
    - PID, 프로세스 이름, 총 형제자매 프로세스 수
  - (8) 자식(child) 프로세스 정보
    - PID, 프로세스 이름, 총 자식 프로세스 수

# Assignment 3-3

- PID를 바탕으로 아래와 같은 프로세스의 정보를 출력하는 Module 작성
  - Example

```
[49474.036512] ##### TASK INFORMATION of '[1] systemd' ##### (1)
[49474.036513] - task state : Wait with ignoring all signals (2)
[49474.036528] - Process Group Leader : [1] systemd (3)
[49474.036529] - Number of context switches : 7607 (4)
[49474.036529] - Number of calling fork() : 378 (5)
[49474.036530] - it's parent process : [0] swapper/0 (6)
[49474.036530] - it's sibling process(es) : (7)
[49474.036531]   > [385] systemd-journal
[49474.036531]   > [439] vmtoolsd
[49474.036532]   > [437] vmware-vmblock-
[49474.036532]   > [852] acpid
[49474.036533]   > [855] accounts-daemon
```

⋮

```
[49474.036552]   > [22802] systemd-udev (8)
[49474.036552]   > [23193] cupsd
[49474.036552]   > [23194] cups-browsed
[49474.036553]   > [23798] whoopsie
[49474.036553]   > [24093] polkitd
[49474.036553]   > This process has 28 child process(es)
[49474.036554] ##### END OF INFORMATION #####
```

# Assignment 3-3

## ■ Requirements

- 2차 과제에서 작성한 ftrace 시스템 콜(336번)을 다음 함수로 wrapping 하여 사용
  - Hooking 함수 명 : process\_tracer
    - ex. `asmlinkage pid_t process_tracer(pid_t trace_task);`
    - Return value
      - 정상적으로 정보를 출력한 경우; 입력 받은 PID 값
      - 출력을 완료하지 못한 경우; -1
    - Input value
      - `pid_t trace_task` : 정보를 출력할 프로세스의 ID 값

# Assignment 3-3

## ■ Requirements

- 메시지 출력 형식
  - Kernel ring buffer에 본 자료에 포함된 예시와 같은 형태로 출력
  - 프로세스 이름은 축약하지 않은 형태로 출력
- 현재 프로세스의 상태
  - 다음과 같은 7가지의 상태로 구분
    - Running or ready
    - Wait with ignoring all signals
    - Wait
    - Stopped
    - Zombie process
    - Dead
    - etc.

# Assignment 3-3

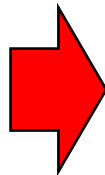
## Requirements

- Context switch 횟수
- fork() 호출한 횟수
  - 참고: 구현 방법
  - task\_struct 구조체에 fork() 호출 횟수를 저장하는 변수를 추가한다.
  - fork()로 자식프로세스를 생성할 때 해당 변수를 초기화시킨다.
  - fork()가 호출될 때 마다 해당 변수의 값을 1씩 증가시킨다.

예시

시스템 콜 wrapping 후

```
1 #include <linux/unistd.h>
2
3 int main(void)
4 {
5     syscall(_NR_ftrace, 1);
6     return 0;
7 }
```



```
[49474.036512] ##### TASK INFORMATION of '[1] systemd' #####
[49474.036513] - task state : Wait with ignoring all signals
[49474.036528] - Process Group Leader : [1] systemd
[49474.036529] - Number of context switches : 7607
[49474.036529] - Number of calling fork() : 378
[49474.036530] - it's parent process : [0] swapper/0
[49474.036530] - it's sibling process(es) :
[49474.036531]   > [385] systemd-journal
[49474.036531]   > [439] vntoolsd
[49474.036532]   > [437] vmware-vmblock-
[49474.036532]   > [852] acpid
[49474.036533]   > [855] accounts-daemon
```

⋮

```
[49474.036552]   > [22802] systemd-udev
[49474.036552]   > [23193] cupsd
[49474.036552]   > [23194] cups-browsed
[49474.036553]   > [23798] whoopsie
[49474.036553]   > [24093] polkitd
[49474.036553]   > This process has 28 child process(es)
[49474.036554] ##### END OF INFORMATION #####
```

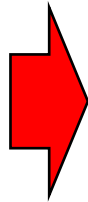


# Assignment 3-3

## ■ Requirements

- 부모(parent) 프로세스 정보
  - 부모 프로세스의 정보를 출력한다.
- 형제자매(sibling) 프로세스 정보
  - 자기 자신을 제외한 형제자매 프로세스의 정보를 출력한다.

```
1 #include <linux/unistd.h>
2 #include <unistd.h>
3
4 int main(void)
5 {
6     syscall(__NR_ftrace, getpid());
7     return 0;
8 }
```



```
[82701.953972] [OSLab.] ##### TASK INFORMATION of '[8138] app' #####
[82701.953976] [OSLab.] - task state : Running or ready
[82701.953977] [OSLab.] - # of context-switch(es) : 2
[82701.953978] [OSLab.] - its parent process : [7934] make
[82701.953980] [OSLab.] - its sibling process(es) :
[82701.953981] [OSLab.] > It has no sibling.
```

- 마지막에 총 형제자매 프로세스 수를 출력한다.

```
[66334.057472] [OSLab.] - its sibling process(es) :
[66334.057473] [OSLab.] > [2] kthreadd
[66334.057475] [OSLab.] > This process has 1 sibling process(es)
```

# Assignment 3-3

## ■ Requirements

- “자식 프로세스 정보”
  - 자식 프로세스의 정보를 출력한다.
  - 없을 경우, 다음과 같이 출력한다.

```
1 #include <linux/unistd.h>
2 #include <unistd.h>
3
4 int main(void)
5 {
6     syscall(__NR_ftrace, getpid());
7     return 0;
8 }
```



```
[82701.953972] [OSLab.] ##### TASK INFORMATION of '[8138] app' #####
[82701.953976] [OSLab.] - task state : Running or ready
[82701.953977] [OSLab.] - # of context-switch(es) : 2
[82701.953978] [OSLab.] - its parent process : [7934] make
[82701.953980] [OSLab.] - its sibling process(es) :
[82701.953981] [OSLab.]   > It has no sibling.
[82701.953982] [OSLab.] - its child process(es) :
[82701.953983] [OSLab.]   > It has no child.
[82701.953984] [OSLab.] ##### END OF INFORMATION #####
```

- 마지막에 총 자식 프로세스 수를 출력한다.

```
[66334.057621] [OSLab.]   > [1652] indicator-application-service
[66334.057623] [OSLab.]   > This process has 45 child process(es)
[66334.057624] [OSLab.] ##### END OF INFORMATION #####
```