

## UNIX 프로그래밍 실습 (2020/10/20)

1. Parent process는 세 개의 child process를 만들고, 1초 간격으로 각 child process에게 SIGUSR1 signal을 보낸 후, child process들이 종료 할 때까지 대기 하였다가, 종료한 child process의 id와 exit status를 출력 후 종료 합니다. 각 child process는 생성 후 parent로부터 signal이 올 때까지 대기 하였다가 (pause() 사용), signal을 받으면 자신의 process id를 세 번 출력 한 후 종료 합니다. 이와 같이 동작하는 프로그램을 아래 코드를 사용하여 작성 하시오.

```
void do_child(int i){

    // SIGUSR1 signal 처리가 가능 하도록 설정

    printf("%d-th child is created...\n", i);
    pause();

    // signal을 받으면, process id 세 번 출력 하도록 설정

    exit(i);
}

int main(void) {
    int i, cid, status;
    pid_t pid[3];

    for (i=0;i<3;i++){
        pid[i]=fork();
        if (pid[i]==0) {
            do_child(i);
        }
    }

    // 1초씩 sleep()하면서, child들에게 SIGUSR1 signal 보내기

    for (i=0;i<3;i++){
        cid=wait(&status);
        printf("child id=%d, exit status=%d\n", cid, WEXITSTATUS(status));
    }

    exit(0);
}
```

2. Parent process는 다섯 개의 child process들을 만들고, 대기하였다가 child process가 모두 종료한 후 종료 합니다. 각 child process는 자신의 순서가 될 때까지 대기 하였다가, 1초씩 쉬면서 (sleep (1); 사용) 자신의 process id(getpid(); 사용)를 5회 출력하는 작업을 한 후 종료 합니다. child process의 id 출력 순서는 생성 순서의 역순이며, 이와 같은 순서 동기화 작업은 signal을 보내서 진행 합니다.

```
void do_child(int i, int *cid){
    // SIGUSR1 signal 처리가 가능 하도록 설정

    // 자신의 출력 순서가 될 때까지 대기

    pid=getpid();
    for (j=0;j<5;j++){
        printf("child %d .... Wn", pid);
        sleep(1);
    }

    // 출력 완료 후 다음 출력 순서의 child에게 출력 시작을 알림

    exit(0);
}

int main(void) {
    int i, status;
    pid_t pid[5];

    for (i=0;i<5;i++){
        pid[i]=fork();
        if (pid[i]==0) {
            do_child(i, pid);
        }
    }

    for (i=0;i<5;i++){
        wait(&status);
    }

    exit(0);
}
```