2023 시스템 프로그래밍

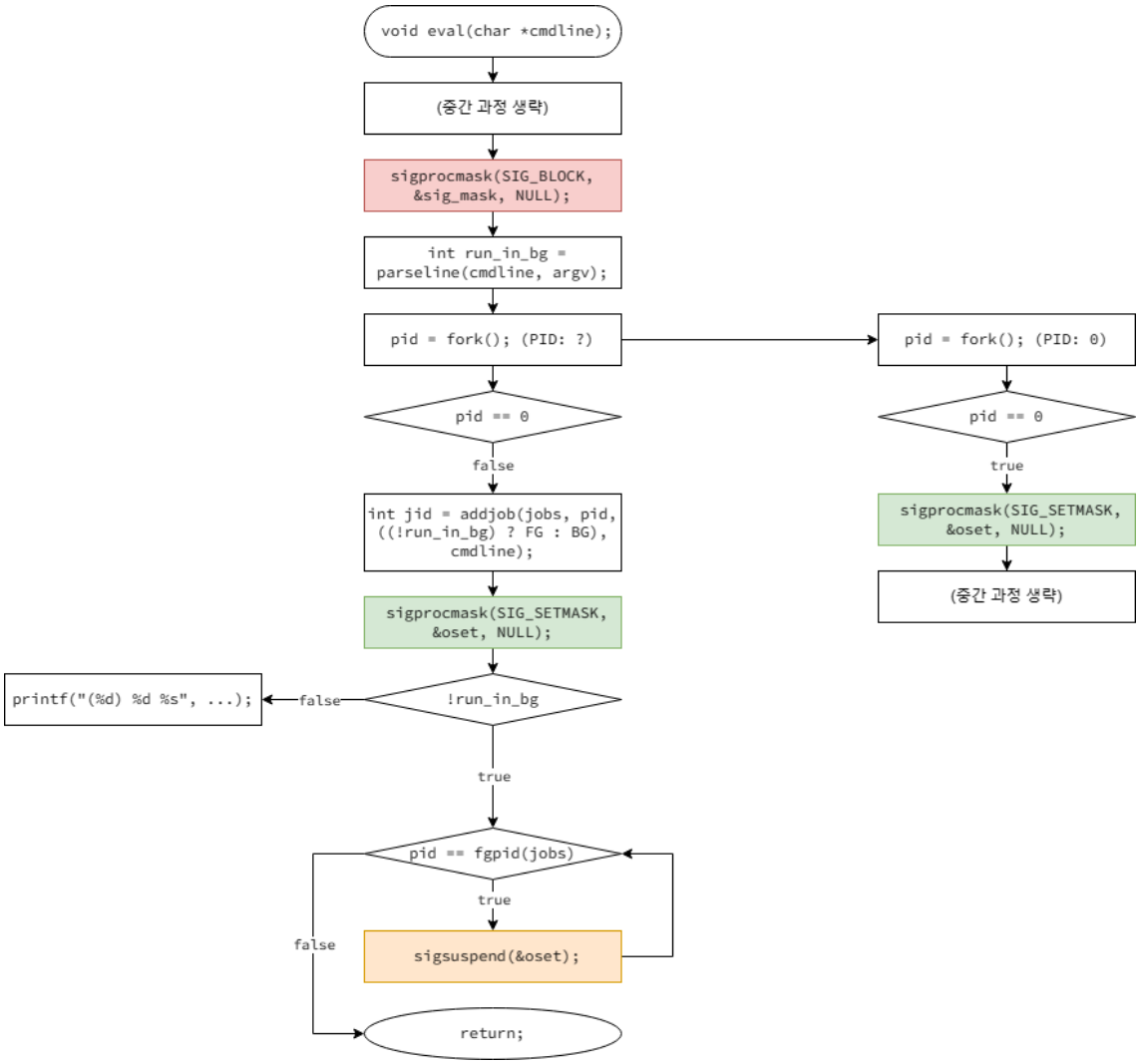
- Lab 03 -

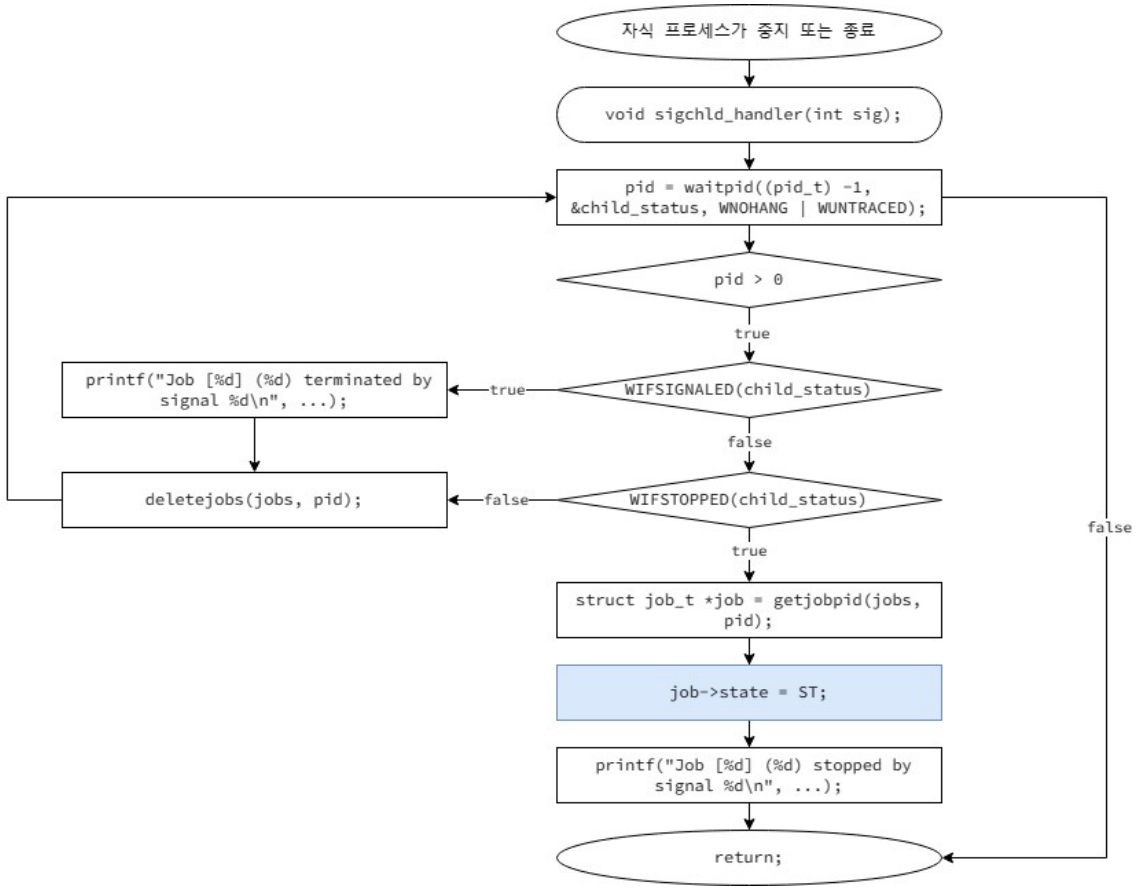
제출일자	2023. 11. 20.
분 반	00
이 름	김재덕
학 번	202104340

Trace 번호 (08, 11)

```
~/shlab-handout $ ./sdriver -U -t 08 -s ./tsh
Running trace08.txt...
Success: The test and reference outputs for trace08.txt matched!
Test output:
#
# trace08.txt - Send fatal SIGINT to foreground job.
#
tsh> ./myintp
Job [1] (1562610) terminated by signal 2
tsh> quit

Reference output:
#
# trace08.txt - Send fatal SIGINT to foreground job.
#
tsh> ./myintp
Job [1] (1562620) terminated by signal 2
tsh> quit

~/shlab-handout $ ./sdriver -U -t 11 -s ./tsh
Running trace11.txt...
Success: The test and reference outputs for trace11.txt matched!
Test output:
#
# trace11.txt - Child sends SIGINT to itself
#
tsh> ./myints
Job [1] (1562646) terminated by signal 2
tsh> quit

Reference output:
#
# trace11.txt - Child sends SIGINT to itself
#
tsh> ./myints
Job [1] (1562664) terminated by signal 2
tsh> quit
```

각 trace 별 플로우 차트





trace 해결 방법 설명

```c
void eval(char *cmdline)
{
    // 명령어 인자 배열을 선언한다.
    char *argv[MAXARGS];

    // 시그널 블록에 사용되는 비트 마스크를 생성한다.
    sigset_t set, oset;

    sigemptyset(&set), sigemptyset(&oset);

    // 비트 마스크에 시그널을 추가한다.
    sigaddset(&set, SIGCHLD);
    sigaddset(&set, SIGINT);
    sigaddset(&set, SIGTSTP);

    // `set`에 설정된 비트에 대응하는 시그널을 블록한다.
    sigprocmask(SIG_BLOCK, &set, &oset);
```

- eval()에는 전역 변수인 jobs를 수정하는 임계 영역 (critical section) 코드가 포함되어 있기 때문에, 자식 프로세스를 생성하고 작업 목록에 새로운 작업을 추가하기 전까지는 시그널 블록을

통해 해당 코드를 원자적 (atomic)으로 실행하도록 한다.[1]

```
    // `parseline()`의 반환값이 1이라면, 이 작업을
    // 백그라운드에서 실행한다.
    int run_in_bg = parseline(cmdline, argv);

    // 먼저 빌트-인 명령어인지 확인한다.
    if (builtin_cmd(argv)) return;

    pid_t pid;

    // `fork()` 수행 후, 자식 프로세스인지 확인한다.
    if ((pid = fork()) == 0) {
        sigprocmask(SIG_SETMASK, &oset, NULL);

        // NOTE: https://pubs.opengroup.org/onlinepubs/007904875/functions/exec.html
        if (execve(argv[0], argv, environ) < 0)
            printf("%s: command not found\n", argv[0]), exit(0);
    }

    // 작업 목록에 새로운 작업을 추가한다.
    int jid = addjob(jobs, pid, ((!run_in_bg) ? FG : BG), cmdline);

    sigprocmask(SIG_SETMASK, &oset, NULL);

    // 이 작업을 포그라운드로 실행해야 하는가?
    if (!run_in_bg) {
        while (pid == fgpid(jobs))
            sigsuspend(&oset);
    } else {
        // 작업 ID, 프로세스 ID와 명령 인수 등을 출력한다.
        printf("(%d) (%d) %s", jid, (int) pid, cmdline);
    }
}
```

- 전역 변수 `jobs`의 수정이 끝나면, `sigprocmask(SIG_SETMASK, &oset, NULL)`를 통해 시그널 블록을 해제하고 셸 프로세스의 시그널 마스크를 복원한다.

```
/*
 * sigint_handler - The kernel sends a SIGINT to the shell whenver the
 *    user types ctrl-c at the keyboard.  Catch it and send it along
 *    to the foreground job.
 */
void sigint_handler(int sig)
{
    // 포그라운드 작업의 프로세스 ID를 가져온다.
    pid_t pid = fgpid(jobs);

    // 프로세스에 `SIGINT` 시그널을 전달한다.
    if (pid > 0) kill(pid, SIGINT);
}
```

---

[1] https://www.gnu.org/software/libc/manual/html_node/Why-Block.html

- SIGINT 시그널 핸들러에서는 포그라운드 작업의 프로세스 ID를 가져온 다음, 그 프로세스를 종료한다.

```c
void sigchld_handler(int sig)
{
    pid_t pid;

    int child_status = 0;

    // 일시 정지 또는 종료된 자식 프로세스가 있는지 확인한다.
    while ((pid = waitpid((pid_t) -1, &child_status, WNOHANG | WUNTRACED)) > 0) {
        /*
         * NOTE: `WIFSIGNALED(stat_val)` evaluates to a non-zero value
         * if status was returned for a child process that terminated
         * due to the receipt of a signal that was not caught.
         */

        if (WIFSIGNALED(child_status)) {
            printf(
                "Job [%d] (%d) terminated by signal %d\n",
                pid2jid(pid), pid, WTERMSIG(child_status)
            );
        } else if (WIFSTOPPED(child_status)) {
            // 일시 정지된 자식 프로세스에 해당하는 작업을 찾는다.
            struct job_t *job = getjobpid(jobs, pid);

            // 작업의 상태를 변경한다.
            job->state = ST;

            printf(
                "Job [%d] (%d) stopped by signal %d\n",
                pid2jid(pid), pid, WSTOPSIG(child_status)
            );

            return;
        }

        deletejob(jobs, pid);
    }
}
```

- SIGCHLD 시그널 핸들러에서는 waitpid((pid_t) -1, &child_status, WNOHANG | WNOTRACED)를 통해 정지 (stopped) 또는 종료 (terminated)된 자식 프로세스가 있는지 확인한다.

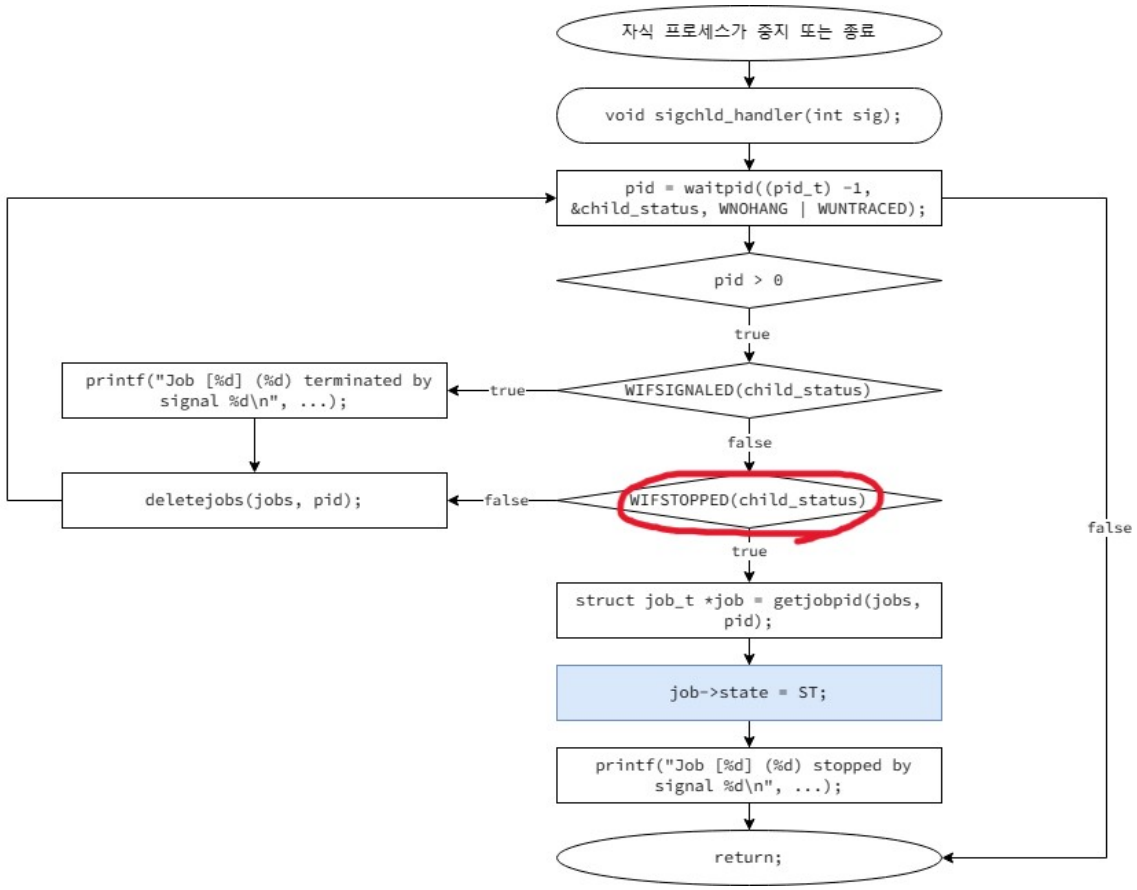
- 자식 프로세스가 시그널에 의해 종료되었다면 WIFSIGNALED(child_status)는 0이 아닌 값을 반환하므로, 작업 목록에서 자식 프로세스에 해당하는 작업을 제거한다. 또한, 자식 프로세스가 단순히 정지 상태라면 작업 목록에서 자식 프로세스에 해당하는 작업을 찾아 그 작업의 상태를 변경하고 반복문에서 빠져나온다.

- Trace 11에서 자식 프로세스가 자기 자신에게 SIGINT를 보낸다면, 부모 프로세스는 SIGCHLD를 받고 시그널 핸들러가 자식 프로세스의 상태를 WIFSIGNALED(child_status)로 확인하여 작업을 제거해주기 때문에, Trace 08 관련 코드를 제대로 구현했다면 바로 통과할 수 있다.

Trace 번호 (09, 12)

```
~/shlab-handout $ ./sdriver -V -t 09 -s ./tsh
Running trace09.txt...
Success: The test and reference outputs for trace09.txt matched!
Test output:
#
# trace09.txt - Send SIGTSTP to foreground job.
#
tsh> ./mytstpp
Job [1] (1562695) stopped by signal 20
tsh> jobs
(1) (1562695) Stopped    ./mytstpp

Reference output:
#
# trace09.txt - Send SIGTSTP to foreground job.
#
tsh> ./mytstpp
Job [1] (1562707) stopped by signal 20
tsh> jobs
(1) (1562707) Stopped ./mytstpp

~/shlab-handout $ ./sdriver -V -t 12 -s ./tsh
Running trace12.txt...
Success: The test and reference outputs for trace12.txt matched!
Test output:
#
# trace12.txt - Child sends SIGTSTP to itself
#
tsh> ./mytstps
Job [1] (1562744) stopped by signal 20
tsh> jobs
(1) (1562744) Stopped    ./mytstps

Reference output:
#
# trace12.txt - Child sends SIGTSTP to itself
#
tsh> ./mytstps
Job [1] (1562754) stopped by signal 20
tsh> jobs
(1) (1562754) Stopped ./mytstps
```

각 trace 별 플로우 차트



trace 해결 방법 설명

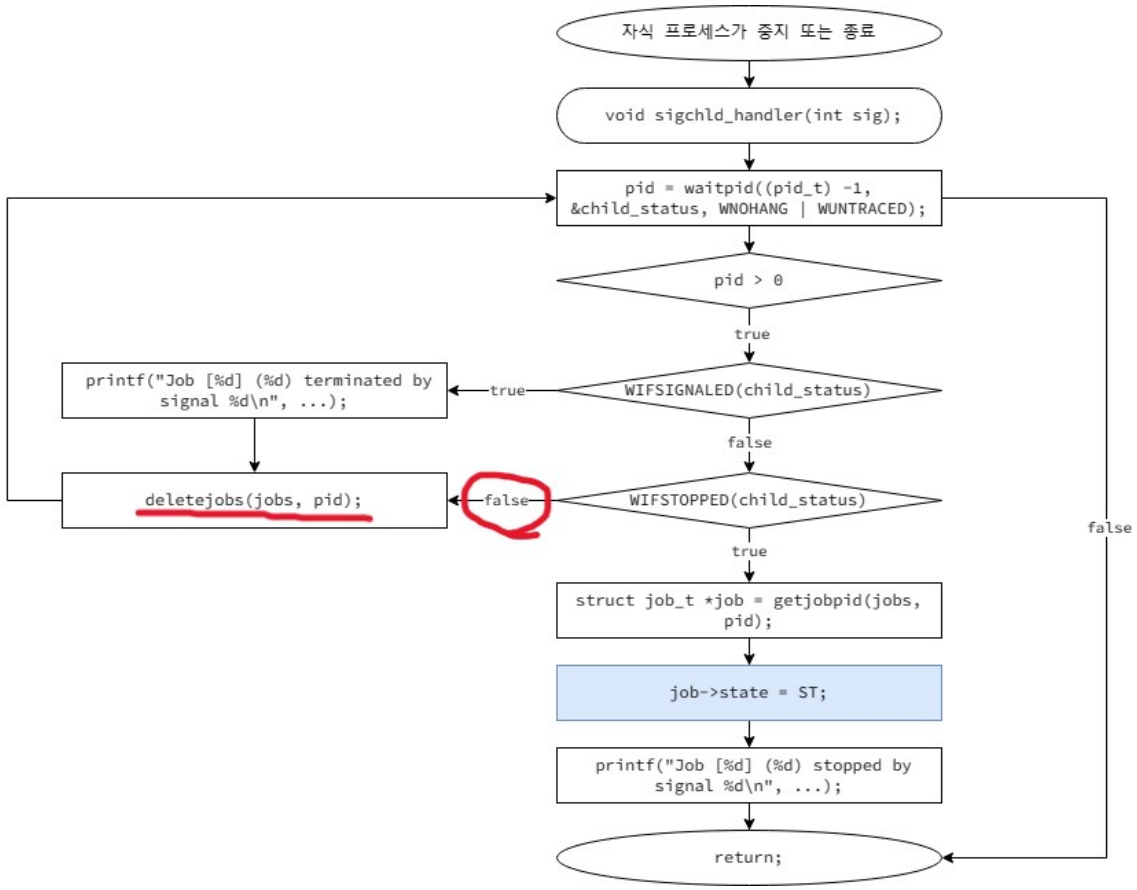
```c
/*
 * sigtstp_handler - The kernel sends a SIGTSTP to the shell whenever
 *     the user types ctrl-z at the keyboard. Catch it and suspend the
 *     foreground job by sending it a SIGTSTP.
 */
void sigtstp_handler(int sig)
{
    // 포그라운드 작업의 프로세스 ID를 가져온다.
    pid_t pid = fgpid(jobs);

    // 프로세스에 `SIGTSTP` 시그널을 전달한다.
    if (pid > 0) kill(pid, SIGTSTP);
}
```

- sigtstp_handler()에서 프로세스에 시그널을 전달하는 부분과 sigchld_handler()에서 정지 상태인 자식 프로세스에 해당하는 작업 상태를 변경하는 코드를 구현하면 통과할 수 있다.

Trace 번호 (10)

```
~/shlab-handout $ ./sdriver -V -t 10 -s ./tsh
Running trace10.txt...
Success: The test and reference outputs for trace10.txt matched!
Test output:
#
# trace10.txt - Send fatal SIGTERM (15) to a background job.
#
tsh> ./myspin1 5 &
(1) (1562786) ./myspin1 5 &
tsh> /bin/kill myspin1
kill: failed to parse argument: 'myspin1'
tsh> quit

Reference output:
#
# trace10.txt - Send fatal SIGTERM (15) to a background job.
#
tsh> ./myspin1 5 &
(1) (1562797) ./myspin1 5 &
tsh> /bin/kill myspin1
kill: failed to parse argument: 'myspin1'
tsh> quit
```

각 trace 별 플로우 차트



trace 해결 방법 설명

- SIGTERM 시그널을 보내 자식 프로세스를 종료시키면 WIFEXITED(child_status)는 0이 아닌 값을 반환하는데, sigchld_handler()에서는 WIFSIGNALED(child_status)와 WIFSTOPPED(child_status)를 제외한 모든 경우에 대해 deletejobs(jobs, pid)를 수행하므로, WIFEXITED(child_status)를 정상적으로 처리할 수 있다.