2018 시스템 프로그래밍
- Lab 07 -

제출일자	2018.11.
분 반	01
이 름	이규봉
학 번	201502085

<Shell Lab 2주차 – tsh 5,6,7>

BackGround에서 실행되는 프로세스는 하나 이상일 수 있으므로, 각 작업들을 관리해 줄, 자료구조가 필요함 => job_t로 구현. => 각 프로세스의 PID, 작업의 JID, 프로세스의 상태 (State), 사용자의 입력값을 갖고 있음.

<작업 관리와 관련된 함수들과 그 설명>

```
/* initjobs - Initialize the job list */
void initjobs(struct job_t *jobs) {
    int i;

    for (i = 0; i < MAXJOBS; i++)
        clearjob(&jobs[i]);
}

/* clearjob - Clear the entries in a job struct */
void clearjob(struct job_t *job) {
    job->pid = 0;
    job->jid = 0;
    job->state = UNDEF;
    job->cmdline[0] = '\0';
}
```

```
struct job_t {
    pid_t pid;
    int jid;
    int state;
    char cmdline[MAXLINE];
};
struct job_t jobs[MAXJOBS];
```

initjobs 함수는 job의 구조체 자료구조인 *jobs를 받아 모든 jobs를 clearjob 합니다. 여기서 clearjob은 각 job들의 pid,jid를 0으로 만들고 state를 UNDEF으로 만들고, job의 cmdline[0]을 ₩0으로 만드는 작업입니다.

(또한 여기서 MAXJOBS는 16으로 선언되어 있습니다)

```c
/* addjob - Add a job to the job list */
int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline)
{
    int i;

    if (pid < 1)
        return 0;

    for (i = 0; i < MAXJOBS; i++) {
        if (jobs[i].pid == 0) {
            jobs[i].pid = pid;
            jobs[i].state = state;
            jobs[i].jid = nextjid++;
            if (nextjid > MAXJOBS)
                nextjid = 1;
            strcpy(jobs[i].cmdline, cmdline);
            if(verbose){
                printf("Added job. [%d] %d %s\n", jobs[i].jid, jobs[i].pid, jobs[i].cmdline);
            }
            return 1;
        }
    }
    printf("Tried to create too many jobs\n");
    return 0;
}
```

addjob은 *jobs에 pid, state, cmdline을 넘겨받아 pid==0인 job들에게 넘겨주고, nextjid (기본 값은 1로 설정되어 있음)를 1 증가시킵니다. 만약 증가한 nextjid가 MAXJOBS보다 커지면 (17이 되면) nextjid를 1로 초기화 합니다.

tsh를 v옵션으로 실행시켰을 때, verbose가 1로 셋팅되게 되는데, 이 경우 Added job이란 메시지와 함께, 추가된 job의 jid, pid, cmdline을 출력한 후 1을 리턴해 정상적으로 리턴되었음을 함수 호출자에게 알립니다.

만약, MAXJOBS의 범위 내에 pid가 0인 job이 없다면, Tried to create too many jobs란 메시지를 출력하고 0을 리턴해, 오류가 발생했음을 함수 호출자에게 알려줍니다.

```c
int deletejob(struct job_t *jobs, pid_t pid)
{
    int i;

    if (pid < 1)
        return 0;

    for (i = 0; i < MAXJOBS; i++) {
        if (jobs[i].pid == pid) {
            clearjob(&jobs[i]);
            nextjid = maxjid(jobs)+1;
            return 1;
        }
    }
    return 0;
}
```

deletejob은 *jobs와 pid를 넘겨받아, 넘겨받은 pid와 일치하는 job들을 clearjob 합니다. 그리고 maxjid란 함수를 호출해 제일 값이 가 큰 jid를 넘겨받고, 1을 더한 값을 nextjid로 셋팅한 후, 1을 리턴합니다.

만약 일치하는 pid가 *jobs내에 없다면 0을 리턴해 호출자에게 오류가 발생했음을 알립니다.

```
/* maxjid - Returns largest allocated job ID */
int maxjid(struct job_t *jobs)
{
    int i, max=0;

    for (i = 0; i < MAXJOBS; i++)
        if (jobs[i].jid > max)
            max = jobs[i].jid;
    return max;
}
```

< maxjid 함수 코드 >

```
/* pid2jid - Map process ID to job ID */
int pid2jid(pid_t pid)
{
    int i;

    if (pid < 1)
        return 0;
    for (i = 0; i < MAXJOBS; i++)
        if (jobs[i].pid == pid) {
            return jobs[i].jid;
        }
    return 0;
}
```

pid2jid는 pid를 인자로 받아, pid값이 1보다 작을 때와 MAXJOBS보다 클 땐 0을 리턴하고,
pid값이 정상적인 범위일 땐, pid와 같은 jobs의 jid를 리턴해 줍니다.

tsh.5, tsh6	BackGround 작업 형태로 프로그램 실행, 동시에 foreground 작업 형태와 background 작업 형태로 프로그램을 실행

```
void waitfg(pid_t pid, int output_fd)
{
    pid = waitpid(pid,NULL,output_fd);
    deletejob(jobs,pid);
    return;
}
```

waitfg는 waitpid 함수를 이용하여 foreground의 종료를 대기하고 있다가, 종료되면, deletejob을 호출해 jobs의 해당 pid에 대응하는 fg 프로세스의 job을 jobs에서 제거해 준 후, 리턴합니다.

```
void eval(char *cmdline)
{
    if(feof(stdin)){
        fflush(stdout);
        exit(0);
    }

    char *argv[MAXARGS];
    int bg = parseline(cmdline, argv);
    pid_t pid;

    if(!builtin_cmd(argv)){
        if((pid=fork())==0){
            if((execve(argv[0],argv,environ) < 0)){
                printf("%s: command not found\n", argv[0]);
                exit(0);
            }
        }
        addjob(jobs,pid,(bg ==1? BG:FG), cmdline);

        if(!bg){
            waitfg(pid,0);
        }
        else{
            printf("(%d) (%d) %s",pid2jid(pid),pid,cmdline);
        }
    }

    return;

}
```

bg란 int형 변수를 선언해 준 후, parseline으로 백그라운드인지를 결정할 값을 받아옵니다. 그리고 Foreground라면 (!bg라면) 작성해 두었던 waitfg를 호출하여, fg 프로세스가 종료되기를 기다렸다가 종료 합니다.
bg 라면, 해당 프로세스의 pid, jid, cmdline을 화면에 출력해줍니다. 이 때, jid를 출력하는 일엔, pid2jid 함수를 이용합니다.

tsh.7 Built-in 명령어 jobs 구현

built-in 명령어를 입력받기 위해선, builtin_cmd 함수 내에서 입력받은 argv[0]값이 jobs인지를 검사하는 코드가 필요합니다. 즉, 아래와 같습니다.

```
int builtin_cmd(char **argv)
{
    if(!strcmp(argv[0],"quit")){exit(0);}

    else if(!strcmp(argv[0],"jobs")){
        listjobs(jobs,1);
        return 1;
    }

    return 0;
}
```

strcmp()를 이용해 입력받은 값이 jobs인지를 검사하게 되며, jobs가 맞다면, 해당 명령어의 기능을 구현하기 위해 listjobs라는 함수를 호출합니다. listjobs는 인자로 *jobs와 output_fd를 받는데, 이것들은 각각 잡리스트 jobs를 가리키는 포인터와, 파일 디스크럽터를 가리킵니다.

```
/* listjobs - Print the job list */
void listjobs(struct job_t *jobs, int output_fd)
{
    int i;
    char buf[MAXLINE];

    for (i = 0; i < MAXJOBS; i++) {
        memset(buf, '\0', MAXLINE);
        if (jobs[i].pid != 0) {
            sprintf(buf, "(%d) (%d) ", jobs[i].jid, jobs[i].pid);
            if(write(output_fd, buf, strlen(buf)) < 0) {
                fprintf(stderr, "Error writing to output file\n");
                exit(1);
            }
            memset(buf, '\0', MAXLINE);
            switch (jobs[i].state) {
                case BG:
                    sprintf(buf, "Running    ");
                    break;
                case FG:
                    sprintf(buf, "Foreground ");
                    break;
                case ST:
                    sprintf(buf, "Stopped    ");
                    break;
                default:
                    sprintf(buf, "listjobs: Internal error: job[%d].state=%d ",
                            i, jobs[i].state);
            }
            if(write(output_fd, buf, strlen(buf)) < 0) {
                fprintf(stderr, "Error writing to output file\n");
                exit(1);
            }
            memset(buf, '\0', MAXLINE);
            sprintf(buf, "%s", jobs[i].cmdline);
            if(write(output_fd, buf, strlen(buf)) < 0) {
                fprintf(stderr, "Error writing to output file\n");
                exit(1);
            }
        }
    }
    if(output_fd != STDOUT_FILENO)
        close(output_fd);
}
```

파일 디스크럽터 fd는 여기서 write 함수에 인자로 들어가게 되는데, write은 출력을 위해 사용하는 함수로, 파일 디스크럽터 값으로 1 (리눅스 계열 운영체제에서 표준출력에 해당) 을 전달해야, 정상적인 값 (양수)를 리턴해줍니다. 따라서, builtin_cmd에서 사용한 listjobs의 인자로 넣어준 값은 jobs,1이 됩니다. 그 후 return문이 없으면 if문 바깥의 0을 리턴하게 되어,

다음과 같이 command not found를 함께 출력하게 됩니다. 따라서, return 1을 넣어줘, 함수 호출자에게 정상적인 함수 종료를 알리도록 했습니다.

< tsh 5,6,7 – sdriver 테스트 >

```
b201502085@2018-sp:~/Shelllab/shlab-handout$ ./sdriver -V -t 05 -s ./tsh
Running trace05.txt...
Success: The test and reference outputs for trace05.txt matched!
Test output:
#
# trace05.txt - Run a background job.
#
tsh> ./myspin1 &
(1) (30285) ./myspin1 &
tsh> quit

Reference output:
#
# trace05.txt - Run a background job.
#
tsh> ./myspin1 &
(1) (30293) ./myspin1 &
tsh> quit
```

```
b201502085@2018-sp:~/Shelllab/shlab-handout$ ./sdriver -V -t 06 -s ./tsh
Running trace06.txt...
Success: The test and reference outputs for trace06.txt matched!
Test output:
#
# trace06.txt - Run a foreground job and a background job.
#
tsh> ./myspin1 &
(1) (30324) ./myspin1 &
tsh> ./myspin2 1

Reference output:
#
# trace06.txt - Run a foreground job and a background job.
#
tsh> ./myspin1 &
(1) (30333) ./myspin1 &
tsh> ./myspin2 1
```

```
b201502085@2018-sp:~/Shelllab/shlab-handout$ ./sdriver -V -t 07 -s ./tsh
Running trace07.txt...
Success: The test and reference outputs for trace07.txt matched!
Test output:
#
# trace07.txt - Use the jobs builtin command.
#
tsh> ./myspin1 10 &
(1) (30664) ./myspin1 10 &
tsh> ./myspin2 10 &
(2) (30666) ./myspin2 10 &
tsh> jobs
(1) (30664) Running    ./myspin1 10 &
(2) (30666) Running    ./myspin2 10 &

Reference output:
#
# trace07.txt - Use the jobs builtin command.
#
tsh> ./myspin1 10 &
(1) (30674) ./myspin1 10 &
tsh> ./myspin2 10 &
(2) (30676) ./myspin2 10 &
tsh> jobs
(1) (30674) Running ./myspin1 10 &
(2) (30676) Running ./myspin2 10 &
```

trace 5,6,7에 대해 정상적으로 작동하는 것을 확인했습니다.