System Programming Assignment3

2017047538 양덕관

eval

```
void eval(char *cmdline)
{
    char *argv[MAXARGS];
    int bg;
    pid_t pid;
    sigset_t mask;
```

- 함수에서 필요한 변수들을 선언한다.

argv : argument list 변수
bg : background인지 foreground인지 확인하는 변수
pid : process 식별을 위한 ID
mask : blocking을 위한 변수

```
bg = parseline(cmdline, argv);
if(argv[0] == NULL){
    return;
}
```

- parseline 함수를 통해 cmdline을 분석하고 argv에 저장한다.
- argv[0](filename)이 NULL일 경우 함수를 종료한다.

```
if(!builtin_cmd(argv)){
    sigemptyset(&mask);
    sigaddset(&mask, SIGCHLD);
    sigprocmask(SIG_BLOCK, &mask, NULL);
```

- built in command인지 확인하는 함수(builtin_cmd)를 실행하고 만약 built in command가 아니라면 아래 코드들을 실행한다.
- blocking을 위한 emthy set을 만들고, SIGCHLD에 대한 signal number를 추가한다.
- sigprocmask함수를 통해 blocking을 한다.

```
//child
if((pid = fork()) == 0){
    setpgid(0, 0);
    if(execve(argv[0], argv, environ) < 0){
        _exit(0);
    }
}
```

- fork()함수를 실행해 child process를 생성한다.
- pid가 0과 같을 때(child process가 생성되었다면) setpgid함수를 통해 고유한 하위 프로세스 그룹 ID를 생성한다.
- 프로그램을 실행한다.(execve함수)
- 만약 execve함수의 return 값이 0보다 작을 경우 _exit()함수를 실행한다.

```
//parent
if(!bg){
    addjob(jobs, pid, FG, cmdline);
    sigprocmask(SIG_UNBLOCK, &mask, NULL);
    waitfg(pid);
}
else{
    addjob(jobs, pid, BG, cmdline);
    sigprocmask(SIG_UNBLOCK, &mask, NULL);
    printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
}
```

- parent process이다.

○ foreground
- 만약 background로 실행되지 않는다면 (foreground로 실행), addjob을 통해 child process를 foreground상태로 작업 list에 추가한다.
- sigpromask함수를 통해 mask를 unblocking한다.
- waitfg함수를 통해 child process를 reaping한다.

○ background
- background로 실행되는 경우 addjob을 통해 child process를 background상태로 작업 list에 추가한다.
- sigpromask함수를 통해 mask를 unblocking한다.
- prinf로 job id와 process id, command를 출력한다.

```
    }

    return;
}
```

- 함수를 종료한다.

builtin_cmd

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

- command가 quit일 때 _exit함수를 통해 프로그램을 종료한다.

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

- command가 jobs일 때 listjobs함수를 통해 background 작업 list를 출력한다.
- 1을 return해 built in command가 실행된 것을 넘긴다.

```
else if(!strcmp(argv[0], "bg") || !strcmp(argv[0], "fg")){
    do_bgfg(argv);
    return 1;
}
```

- command가 bg이거나 fg일 때 do_bgfg함수를 통해 background에서 실행 혹은 foreground에서 실행으로 변경한다.
- 1을 return해 built in command가 실행된 것을 넘긴다.

```
    return 0;
}
```

- 위에 조건들에 해당하지 않는 경우 0을 return해 built in command가 실행되지 않았다는 것을 넘긴다.

waitfg

```
void waitfg(pid_t pid)
{
    if(pid == 0){
        return;
    }
```

- pid가 0과 같을 때(child process가 생성되지 않았을 때) 함수를 종료한다.

```
struct job_t *job = getjobpid(jobs, pid);
```

- job에 getjobpid함수로 현재의 jobs struct를 저장한다..

```
if(!job){
    return;
}
else{
    while(pid == fgpid(jobs))
        ;
}
```

- 만약 job이 NULL이면, 함수를 종료한다.
- job이 NULL아 아닐 때, while문을 실행한다.
- while은 pid가 현재 foregrond process id와 같을 때 (fgpid함수로 현재의 foregrond process id를 가져온다.) busy loop를 실행한다.

```
    return;
}
```

- 함수를 종료한다.