Sp_assignment4　20131329 신선우

```
void eval(char *cmdline) //eval function
{
    int parse = parseline(cmdline, argv); // parsing cmdline to argv
    if (argv[0] == NULL) {//blank input return
        return;
    }
    int built = builtin_cmd(argv); //DOING BUILTIN_CMD

    if (built == 0) { //not builtin cmd
        pid = fork(); //forking

        if (pid == 0) {//child process
            setpgid(0,0);// set process group ID
            if(execve(argv[0], argv, environ) == -1) { //executing file
                unix_error("eval execve error");
            }
            exit(0);
        }
        else {
            if(parse == 1) { //=>bg
                if((addjobChck = addjob(jobs, pid, BG, cmdline)) == 0) { //adding job
                    unix_error("eval addjob error");
                }
                printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
```

```
                return;
            }
            else if(parse == 0){ // => fg
                if ((addjobChck = addjob(jobs, pid, FG, cmdline)) == 0) { //adding job
                    unix_error("eval addjob error");
                }
                waitfg(pid); //wait pid when 'fg'
                return;
            }
        }
    }
    return;//built == 1 already done at int built
}

int builtin_cmd(char **argv) //builtin_cmd function
{
    if(strcmp(argv[0], "fg") == 0) { //when the process is 'fg' then do_bgfg
        do_bgfg(argv);
        return 1;
    }
    if(strcmp(argv[0], "bg") == 0) { //when the process is 'bg' then do_bgfg
        do_bgfg(argv);
        return 1;
    }
    if(strcmp(argv[0], "jobs") == 0) { //when type jobs then showing job list
        listjobs(jobs);
```

```
        return 1;
    }
    if(strcmp(argv[0], "quit") == 0) { //when type quit commend then close tsh
        exit(0);
    }
    return 0;       /* not a builtin command */
}

void waitfg(pid_t pid) // waitfg function
{
    struct job_t *temp;
    temp = getjobpid(jobs, pid); //using pid get job state
    if (temp == NULL) { // if there is no job then return
        return;
    }
    else {
        while (temp->state == FG && temp->pid == pid) { //if the state of job is FG and while same pid then sleep It occurred blocking process
            sleep(1);
        }
    }
    return;
}
```