```
/* Simple program to illustrate the use of fork-exec-wait pattern.
* This version uses execvp and command-line arguments to create a new process.
* To Compile: gcc -Wall forkexecvp.c
* To Run: ./a.out <command> [args]
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

pid_t fork_pid = 0;

static void sig_usr(int signo) {
    switch(signo) {
        case SIGINT:
            printf("received SIGINT signal %d, killing child process...\n", signo);
            printf("  Child ID: %d\n",fork_pid);
            int result1 = 1;
            printf("  result1 before kill: %d\n",result1);
            result1 = kill(fork_pid,SIGINT);
            printf("  result1 after kill: %d\n",result1);
            if(result1==0){
                printf("    Child Process Interrupted!\n");
            }
            else{
                printf("    Something went wrong, could not interrupt child.\n");
            }
            break;
        case SIGTSTP:
            printf("received SIGTSTP signal %d, killing child process...\n", signo);
            int result2 = 1;
            result2 = kill(fork_pid,SIGTSTP);
            if(result2==0){
                printf("    Child Process Killed!\n");
            }
            else{
                printf("    Something went wrong, could not kill child.\n");
            }
            break;
        case SIGQUIT:
            printf("received SIGQUIT signal %d, stopping Parent Process.\n", signo);
            exit(1);
            break;
    }
}

int main(int argc, char **argv) {
    int status;
    if (argc < 2) {
        printf("Usage: %s <command> [args]\n", argv[0]);
        exit(-1);
    }
    fork_pid = fork();
    if (fork_pid == 0) {
        /* this is child process */
        execvp(argv[1], &argv[1]);
        printf("If you see this statement then execl failed ;-(\n");
        perror("execvp");
        exit(-1);
    } else if (fork_pid > 0) {
        printf("PID of Child: %d\n",fork_pid);
        /* this is the parent process */
        printf("Wait for the Ctrl-\\ to terminate\n");
        if (signal(SIGTSTP, sig_usr) == SIG_ERR) {
            printf("can't catch SIGTSTP\n");
            exit(-1);
        }
        if (signal(SIGINT, sig_usr) == SIG_ERR) {
            printf("can't catch SIGINT\n");
```

```
            exit(-1);
        }
        if (signal(SIGQUIT, sig_usr) == SIG_ERR) {
            printf("can't catch SIGINT\n");
            exit(-1);
        }
        while(1){
            signal(SIGINT,sig_usr);
            pause();
        }
    } else { /* we have an error */
        perror("fork"); /* use perror to print the system error message */
        exit(EXIT_FAILURE);
    }

    printf("[%ld]: Exiting program .....\n", (long)getpid());
    return 0;
}
```