```c
/* 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]
 */
#define _POSIX_C_SOURCE 200809L
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <errno.h>

static volatile sig_atomic_t quit_received = 0;
static pid_t child_pid = -1;

static void on_sigquit(int sig) {
    (void)sig;
    quit_received = 1;
    if (child_pid > 0) {
        kill(child_pid, SIGQUIT);
        kill(child_pid, SIGCONT);
    }
}

int main(int argc, char **argv) {
    int status;

    if (argc < 2) {
        printf("Usage: %s <command> [args]\n", argv[0]);
        exit(-1);
    }

    struct sigaction sa_ignore = {0};
    sa_ignore.sa_handler = SIG_IGN;
    sigaction(SIGINT,  &sa_ignore, NULL);
    sigaction(SIGTSTP, &sa_ignore, NULL);

    struct sigaction sa_quit = {0};
    sa_quit.sa_handler = on_sigquit;
    sigemptyset(&sa_quit.sa_mask);
    sa_quit.sa_flags = SA_RESTART;
    sigaction(SIGQUIT, &sa_quit, NULL);

    child_pid = fork();
    if (child_pid == 0) { /* this is child process */
        struct sigaction sa_dfl = {0};
        sa_dfl.sa_handler = SIG_DFL;
        sigaction(SIGINT,  &sa_dfl, NULL);
        sigaction(SIGTSTP, &sa_dfl, NULL);
        sigaction(SIGQUIT, &sa_dfl, NULL);

        execvp(argv[1], &argv[1]);
        perror("execvp");
        exit(-1);
    } else if (child_pid > 0) { /* this is the parent process */
        printf("Wait for the child process to terminate\n");
        for (;;) {
            pid_t r = waitpid(child_pid, &status, 0);
            if (r == -1) {
                if (errno == EINTR) {
                    continue;
                }
                perror("waitpid");
                break;
            }
            if (r == child_pid) {
                if (WIFEXITED(status)) {
                    printf("Child process exited with status = %d\n",
                           WEXITSTATUS(status));
                } else if (WIFSIGNALED(status)) {
                    printf("Child process terminated by signal %d\n",
                           WTERMSIG(status));
                } else if (WIFSTOPPED(status)) {
                    printf("Child process stopped by signal %d\n",
                           WSTOPSIG(status));
                    continue;
                }
                break;
            }
        }

    } 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;
}
```