```
/* 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>
#include <signal.h>

volatile __sig_atomic_t time_to_quit = 0;

pid_t child_pid;

static void sig_handler(int signo) {
    switch(signo) {
        case SIGINT:
        printf("Ctrl+C caught: passing 'SIGINT' to child\n");
        kill(child_pid, SIGINT);
        break;

        case SIGTSTP:
        printf("Ctrl+Z caught: passing 'SIGTSTP' signal to child\n");
        kill(child_pid, SIGTSTP);
        break;

        case SIGQUIT:
        printf("Ctrl+\\ caught: telling parent to exit\n");
        time_to_quit = 1;
        break;

        case SIGCHLD:
        int status;
        pid_t wpid = waitpid(child_pid, &status, WNOHANG | WUNTRACED);
        if (WIFEXITED(status)) {
            time_to_quit = 1;
        }
        break;
    }
}

int main(int argc, char **argv) {
    pid_t pid;

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

    pid = fork();
    if (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 (pid > 0) { /* this is the parent process */

        //Set the global child pid so signal handler knows where to forward signals
        child_pid = pid;

        printf("Parent is runing, waiting for Ctrl+\\ to exit...\n");

        //Register signal handlers
        signal(SIGINT, sig_handler);
        signal(SIGTSTP, sig_handler);
        signal(SIGQUIT, sig_handler);
```

```
        signal(SIGCHLD, sig_handler);

        //Each time a signal is recieved make sure that it's not time to quit
        while(!time_to_quit) {
            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;
}
```