```
// Jessica Elkins
// CS332 Lab 9
// 3/31/20
// This program is a modification of the forkexecvp.c program.
// When you type Control-C or Control-Z the child process is interrupted
// or suspended and the parent process continues to wait untill it
// receives a quit signal (Control-\).

// TO COMPILE: gcc Lab9.c -o lab9
// TO RUN: ./lab9 <command> [arguments]

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

// making pid global variable
pid_t pid;

/* this function is a signal handler for SIGINT signal
   and causes the parent process to ignore the signal
   and only the child process gets killed */
static void sigint(int signo) {
        signal(signo, SIG_IGN); /* parent process ignore the signal */
        printf("received SIGINT in parent process\n");
        fflush(stdout);
        kill(pid, SIGKILL); /* send signal to child process */
        printf("Child process with pid = %ld killed. \n", (long)pid);
        fflush(stdout);
        signal(signo, sigint); /* reinstall the signal handler */
}

/* this function is a signal handler for SIGTSTP signal
   and causes the parent process to ignore the signal and
   only the child process get stopped */
static void sigtstp(int signo){
        signal(signo, SIG_IGN);
        printf("received SIGTSTP in parent process \n");
        fflush(stdout);
        kill(pid, SIGTSTP);
        printf("Child process with pid = %ld stopped. \n", (long)pid);
        fflush(stdout);
        signal(signo, sigtstp); /* reinstall the signal handler */
}

/* this function intercepts the SIGQUIT signal
   to keep core dump from happening */
static void sigquit(int signo){
        if(pid > 0) {
                printf("received SIGQUIT in parent process \n");
                raise(SIGTERM);
        }
}

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

        int status;

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

```
        /* setting up signal handler to intercept SIGINT */
        if(signal(SIGINT, sigint) == SIG_ERR) {
                printf("Unable to catch SIGINT \n");
        }

        /* setting up signal handler to intercept SIGTSTP */
        if(signal(SIGTSTP, sigtstp) == SIG_ERR) {
                printf("Unable to catch SIGTSTP \n");
        }

        if(signal(SIGQUIT, sigquit) == SIG_ERR) {
                printf("Unable to catch SIGQUIT \n");
        }

        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 parent process */
                printf("Wait for the child process to terminate \n");
                wait(&status);  /* wait for child process to terminate */
                if (WIFEXITED(status)) { /* child process terminated normally */
                        printf("Child process exited with status = %d \n", WEXITSTATUS(
status));
                } else {  /* child process did not terminate normally */
                        printf("Child process did not terminate normally! \n");
                                for( ; ; ) {
                                        pause();
                                }
                }
        } else { /* there is an error */
                perror("fork");
                exit(EXIT_FAILURE);
        }

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

        return 0;
}
```