For this activity I experimented by removing the sleep and wait functions in the parent and child processes. Furthermore, I called fork() once and anticipated that since the parent was not signaled to wait on the child's completion it would close and leave the child without a parent; thus becoming a zombie process.

```
  GNU nano 5.4                                    NgabaPeter_forkExperiment.c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

int main()
{
        pid_t retVal;

        retVal = fork();
        if(retVal < 0){
                printf("fork() failed\n");
                return 1;
        }
        else if(retVal == 0){
                printf("fork1 retVal == 0 ");
                printf("in child process pid = %d \n", getpid());
                //sleep(5);
                printf(" finished sleeping \n" );
        }
        else{
                printf("parent pid = %d \n", getpid());
                //printf("fork1 in parent process waiting for child ...\n");
                //wait(NULL);
                printf("wait() finished in parent proces \n");
        }
        return 0;
}
```

The actual results were consistent with my anticipation and show that the parent process ended before the child even if the sleep process was removed. Furthermore, it also shows that the fork process prioritizes the parent's execution before handling the child's execution.

```
student@CMSC421Debian:~/Activity$ ./NgabaPeter_forkExperiment
parent pid = 2026
wait() finished in parent proces
fork1 retVal == 0 in child process pid = 2027
 finished sleeping
```