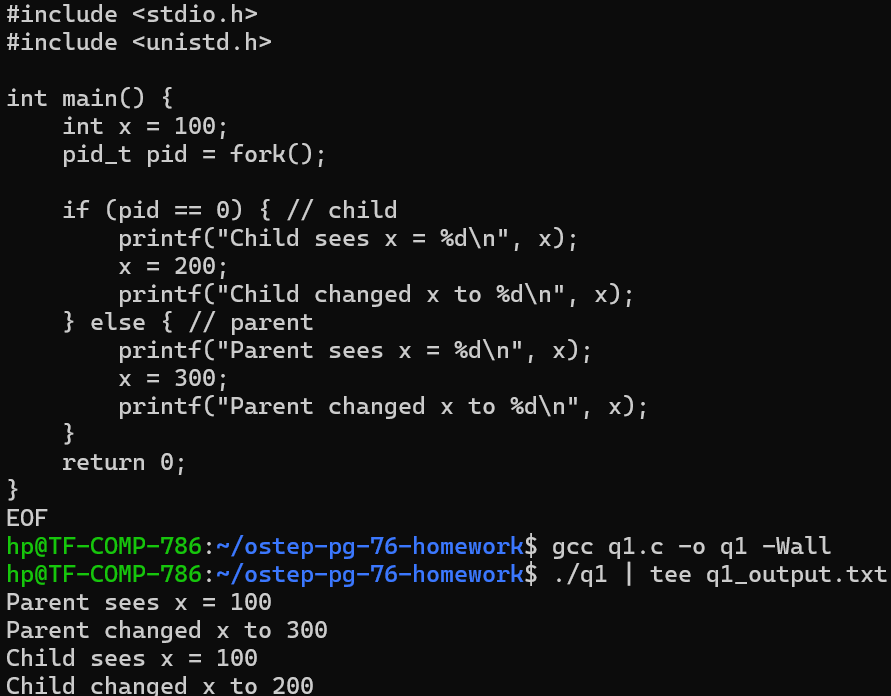
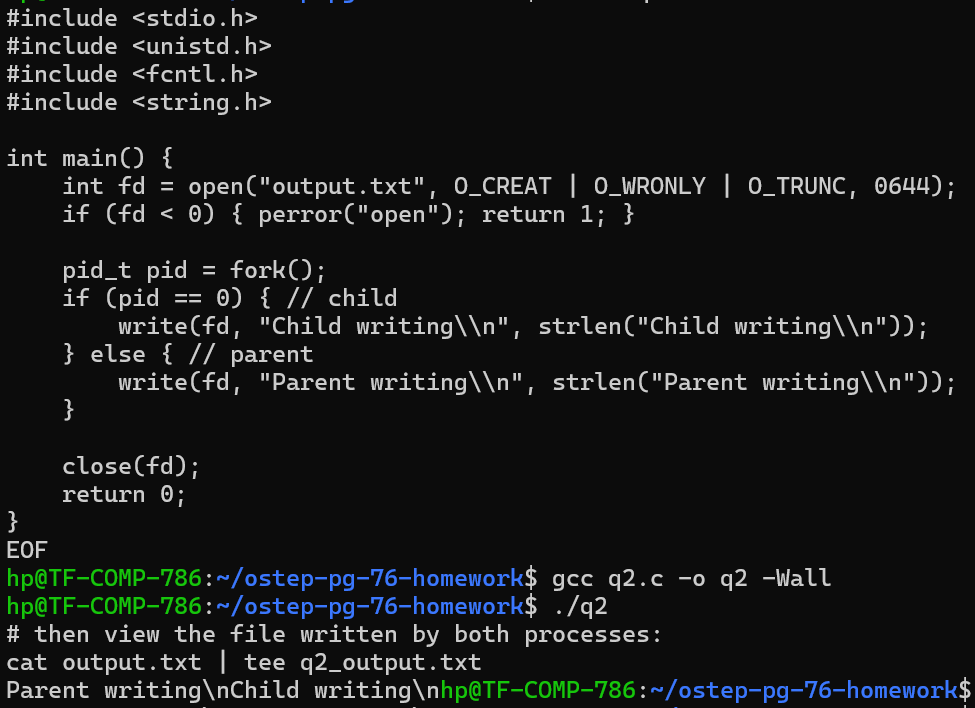
Homework: Process API in C

# Q1. Variable Behavior After fork()



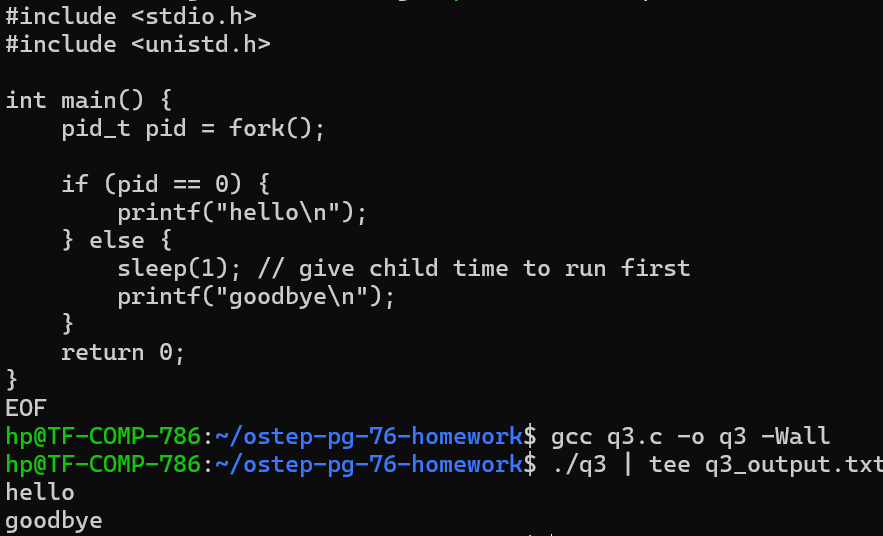
**Explanation:** Both processes initially see x=100. After fork, memory is copied (Copy-on-Write). Changes in one process do not affect the other.

# Q2. File Descriptor Sharing After fork()



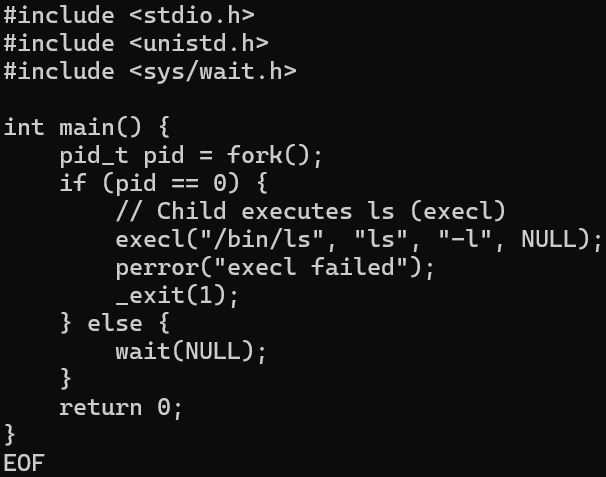
**Explanation:** Both parent and child share the same file descriptor. Writes go to the same file. If concurrent, their outputs may interleave.

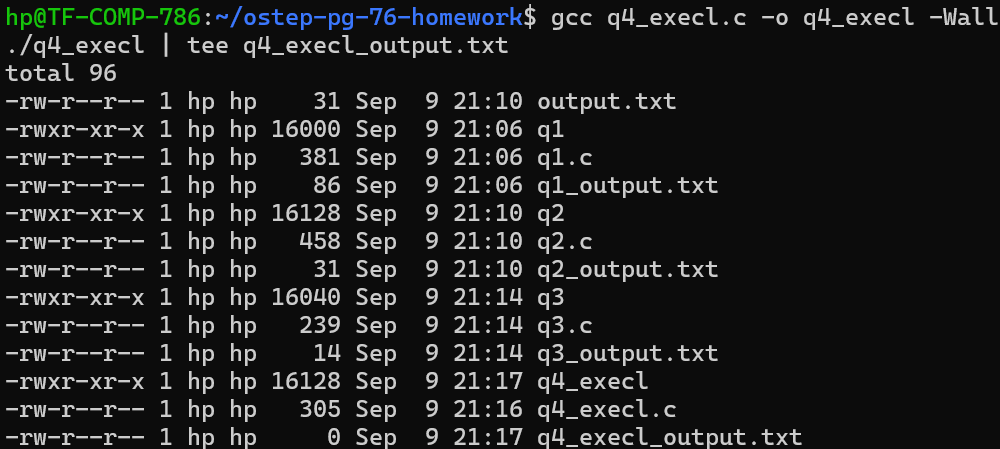
# Q3. Child Prints First Without wait()



**Explanation:** Using sleep() in the parent ensures the child runs and prints 'hello' first. No wait() needed.

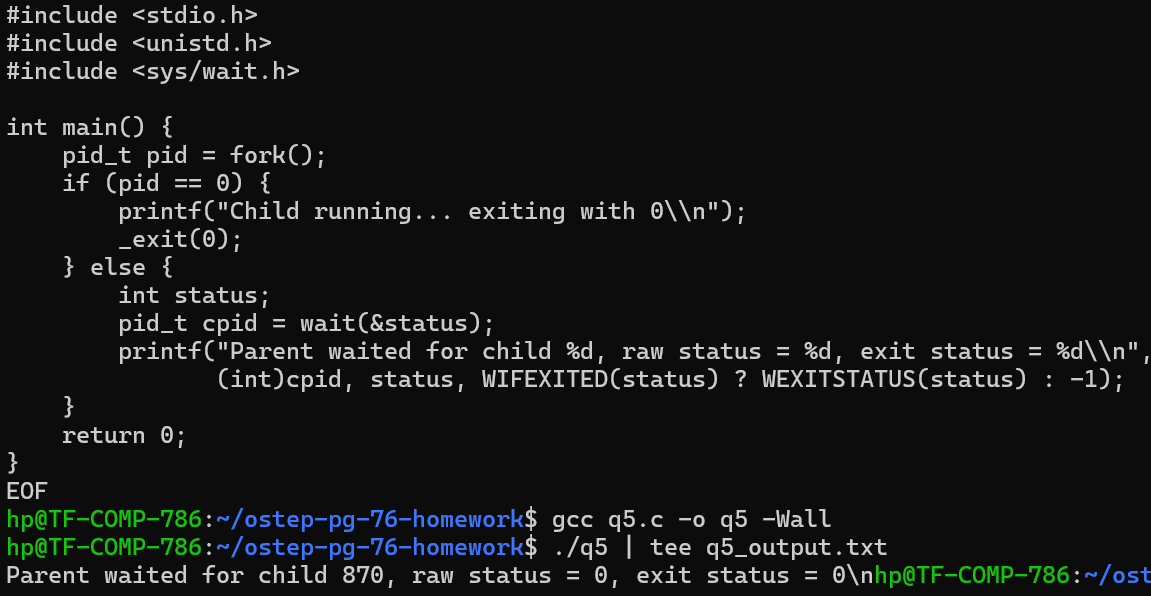
# Q4. Using exec() Variants





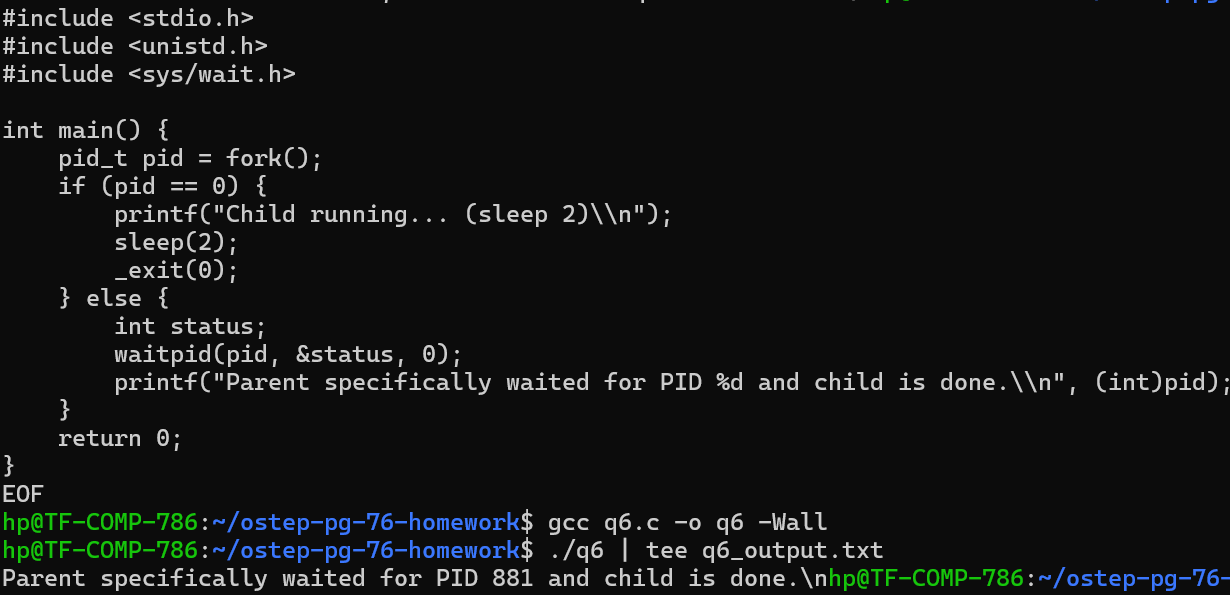
**Explanation:** exec() replaces the child process image with /bin/ls. Variants (execl, execv, execvp, etc.) differ in how arguments and environment are passed.

# Q5. Using wait()



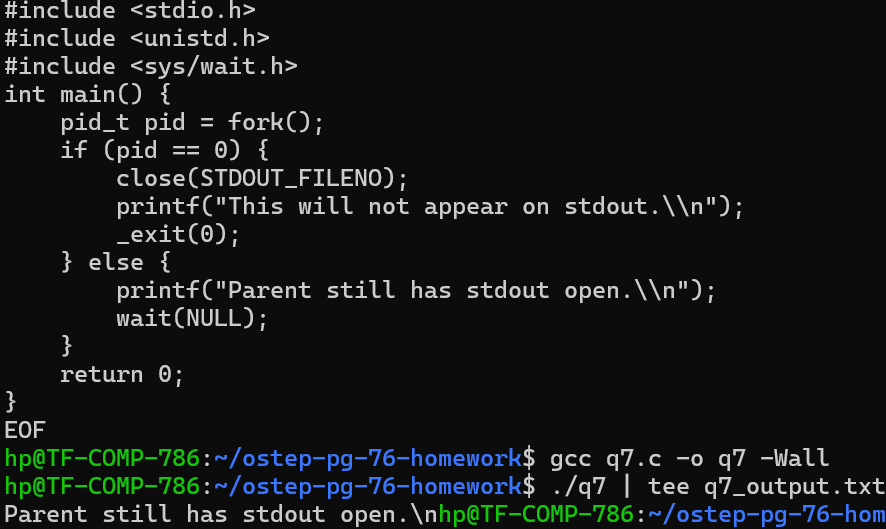
**Explanation:** wait() in the parent returns the PID of the terminated child. In the child, wait() returns -1 with errno=ECHILD (no children).

# Q6. Using waitpid()



**Explanation:** waitpid() allows waiting for a specific child, useful when multiple children exist.

# Q7. Closing STDOUT in Child



**Explanation:** Closing STDOUT in the child means any printf() will fail silently (output discarded).

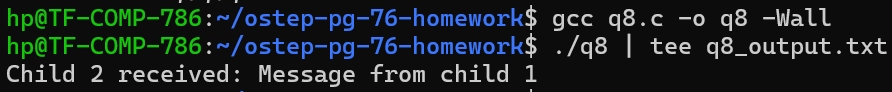
# Q8. Pipe Between Two Children

**Program:**

cat > q8.c <<'EOF'

#include <stdio.h>  
#include <unistd.h>  
#include <string.h>  
#include <sys/wait.h>  
  
int main() {  
 int fd[2];  
 pipe(fd);  
  
 pid\_t pid1 = fork();  
 if (pid1 == 0) { // First child: writer  
 close(fd[0]);  
 write(fd[1], "Message from child 1\n", strlen("Message from child 1\n"));  
 close(fd[1]);  
 return 0;  
 }  
  
 pid\_t pid2 = fork();  
 if (pid2 == 0) { // Second child: reader  
 close(fd[1]);  
 char buf[100];  
 int n = read(fd[0], buf, sizeof(buf)-1);  
 buf[n] = '\0';  
 printf("Child 2 received: %s", buf);  
 close(fd[0]);  
 return 0;  
 }  
  
 close(fd[0]);  
 close(fd[1]);  
 wait(NULL);  
 wait(NULL);  
 return 0;  
}

EOF



**Explanation:** The pipe connects the stdout of the first child to the stdin of the second. Data written by one is read by the other.