QUESTION 9 ANSWERS

Terminal - coder@dce7ae72477b: ~

File Edit View Terminal Tabs Help

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main(){
        int num_children = 3;
        pid_t pid;
        int i;
        printf("Parent process started. PID: %d\n", getpid());
        for (i=0; i < num_children; i++) {
                pid = fork();
                if (pid < 0) {
                        perror("fork failed");
                        exit(1);
                } else if (pid == 0) {
                printf("Child %d started. PID: %d\n", i + 1, getpid());
                sleep(2 + i);
                printf("Child %d exiting. PID: %d\n", i + 1, getpid());
                exit(0);
                }
        }
        for (i = 0; i < num_children; i++) {
        pid_t child_pid = wait(NULL);
        printf("Parent cleaned up child with PID: %d\n", child_pid);
        }
        printf("All children cleaned up. parent exiting.\n");
        return 0; }
~
"zombie_demo.c" 27L, 677C                                1,18          All
```

```
coder@dce7ae72477b:~$ vi zombie_demo.c
coder@dce7ae72477b:~$ gcc -o zombie_demo zombie_demo.c
coder@dce7ae72477b:~$ ./zombie_demo
Parent process started. PID: 870
Child 1 started. PID: 871
Child 2 started. PID: 872
Child 3 started. PID: 873
Child 1 exiting. PID: 871
Parent cleaned up child with PID: 871
Child 2 exiting. PID: 872
Parent cleaned up child with PID: 872
Child 3 exiting. PID: 873
Parent cleaned up child with PID: 873
All children cleaned up. parent exiting.
coder@dce7ae72477b:~$
```

I wrote a C program that creates multiple child processes using `fork()`. Each child terminates after execution, and the parent process uses `wait()` to clean up terminated children and print their PIDs. This ensures that no zombie processes remain.