Homework 9

Lim Saeyeon (#21102054)

HW9 > C fork.c

```c
#include <stdlib.h>
#define NUM_FORKS 4

char array[NUM_FORKS+2];

int pos = 0;

void work(void* id) {
    char writeMe = '0' + *(int*)id;
    array[pos++] = writeMe;
}

int main() {
    char three = '3';
    int i;
    int pid[NUM_FORKS];

    for (i = 0; i < NUM_FORKS; i++)
    {
        if (!(pid[i] = fork()))
        {
            work((void*)(&i));
            exit(0);
        }
        waitpid(pid[i], NULL, 0); }
        array[pos++] = three;
        array[pos] = '\0';
        printf("%s", array);
        exit(0);
}

```

A. The output is 3.

```
CS21102054@nshcdell ~/HW9 (0.046s)
./fork
3
```

B. Explanation

a. Memory and Process Behavior

- Each time fork() is called, it creates a child process that gets its own copy of the memory space
- However, critically important: The variable `pos` and array `array` in each child process are separate copies, not shared with the parent
- When each child process exits, its memory modifications are lost and don't affect the parent's memory

b. The Loop Execution

- In the main loop, for each iteration:
  - A child process is created via fork()
  - The child calls work() with the current value of i
  - The child writes to its own copy of array[pos]
  - The child exits immediately via exit(0)
  - The parent waits for each child to complete via waitpid()
  - But the parent's array remains unchanged by the child's operations

c. Why Only '3' Appears

- After all forks and waits complete, the parent process:
  - Has pos still equal to 0 (unchanged by children)
  - Writes '3' to array[0]
  - Adds null terminator
  - Prints the array

Although each child process does write a character to its local copy of the array, these writes are isolated to each child's memory space when each child exits, these changes are lost and only the parent's write of '3' persists and gets printed.

This is a classic example demonstrating process isolation in Unix/Linux systems. Each process gets its own copy of memory variables after a fork(), and modifications in child processes don't affect the parent's memory space.