

New York University
Tandon School of Engineering
Department of Computer Science and
Engineering
Introduction to Operating Systems
Fall 2024
Assignment 4 (10 points)

Problem 1 (2 points)

If you create a `main()` routine that calls `fork()` three times, i.e., if it includes the following code:

```
pid_t x=-11, y=-22, z=-33;  
x = fork();  
if(x==0) y = fork();  
if(y>0) z = fork();
```

Assuming all `fork()` calls succeed, draw a process tree similar to that of Fig. 3.8 (page 116) in your textbook, clearly indicating the values of `x`, `y` and `z` for each process in the tree (i.e., whether 0, -11, -22, -33, or larger than 0). Note that the process tree should only have one node for each process and thus the number of nodes should be equal to the number of processes. The process tree should be a snapshot just after all forks completed but before any process exists. Each line/arrow in the process tree diagram shall represent a creation of a process, or alternatively a parent/child relationship. (Include your process tree diagram here as an image). A correctly labeled diagram should show the parent process with `x = -11`, `y = -22`, `z = -33`. Three child processes with `x=0`, `y=-11`, `z=-33` and `x=0`, `y=0`, `z=-33` and `x=0`, `y=-11`, `z=0` respectively.

Problem 2 (4 points)

Write a program that creates the process tree shown below: (Include your process tree diagram here as an image). (Include your C code here).

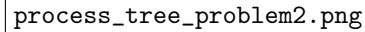


Figure 1: Process Tree for Problem 2

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid1, pid2, pid3, pid4;

    pid1 = fork();
    if (pid1 < 0) {
        perror("fork failed");
        return 1;
    } else if (pid1 == 0) {
        printf("Child_1 (PID: %d)\n", getpid());
        pid3 = fork();
        if (pid3 < 0) {
            perror("fork failed");
            return 1;
        } else if (pid3 == 0) {
            printf("Grandchild_1 (PID: %d)\n", getpid());
        } else {
            wait(NULL);
            printf("Child_1 exiting\n");
        }
    } else {
        pid2 = fork();
        if (pid2 < 0) {
            perror("fork failed");
            return 1;
        } else if (pid2 == 0) {
            printf("Child_2 (PID: %d)\n", getpid());
            pid4 = fork();
            if (pid4 < 0) {
                perror("fork failed");
                return 1;
            } else if (pid4 == 0) {
```

```

        printf("Grandchild_2_(PID:%d)\n", getpid());
    } else {
        wait(NULL);
        printf("Child_2_exiting\n");
    }
} else {
    wait(NULL);
    wait(NULL);
    printf("Parent_exiting\n");
}
}
return 0;
}

```

Problem 3 (4 points)

Write a program whose main routine obtains a parameter n from the user (i.e., passed to your program when it was invoked from the shell, n_2) and creates a child process. The child process shall then create and print a Fibonacci sequence of length n and whose elements are of type **unsigned long long**. You may find more information about Fibonacci numbers at (https://en.wikipedia.org/wiki/Fibonacci_number). The parent waits for the child to exit and then prints two additional Fibonacci elements, i.e., the total number of Fibonacci elements printed by the child and the parent is $n+2$. Do not use IPC in your solution to this problem (i.e., neither shared memory nor message passing). (Include your C code here).

```

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

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <n>\n", argv[0]);
        return 1;
    }

    unsigned long long n = atoll(argv[1]);
    if (n <= 2) {
        fprintf(stderr, "n_must_be_greater_than_2\n");
        return 1;
    }

    pid_t pid = fork();
    if (pid < 0) {
        perror("fork_failed");
        return 1;
    } else if (pid == 0) { // Child process
        unsigned long long a = 0, b = 1, temp;
        printf("Fibonacci_sequence_(child):");
        for (unsigned long long i = 0; i < n; i++) {
            printf("%llu", a);
            temp = a + b;
        }
    }
}

```

```

        a = b;
        b = temp;
    }
    printf("\n");
    exit(0);
} else { // Parent process
    wait(NULL);
    unsigned long long a = 0, b = 1, temp;
    for (int i = 0; i < 2; i++) {
        temp = a + b;
        a = b;
        b = temp;
    }
    printf("Fibonacci sequence (parent): %llu %llu\n", a, b);
}
return 0;
}

```

Problem 4 (1 point)

Explain the concept of a zombie process and how it can be avoided.

Problem 5 (1 point)

Describe the differences between the `exec()` family of functions and the `fork()` system call. When would you use each?

What to hand in (using Brightspace):

Please submit the following files individually:

1. Source file(s) with appropriate comments. The naming should be similar to “lab#_\$.c” (# is replaced with the assignment number and \$ with the question number within the assignment, e.g., lab4_b.c, for lab 4, question b OR lab5_1a for lab 5, question 1a).
2. A single pdf file (for images + report/answers to short-answer questions), named “lab#.pdf” (# is replaced by the assignment number), containing:
 - Screenshot(s) of your terminal window showing the current directory, the command used to compile your program, the command used to run your program and the output of your program.
3. Your Makefile, if any. This is applicable only to kernel modules.

RULES:

- You shall use kernel version 4.x.x or above. You shall not use kernel version 3.x.x.
- You may consult with other students about GENERAL concepts or methods but copying code (or code fragments) or algorithms is NOT ALLOWED and is considered cheating (whether copied from other students, the internet or any other source).
- If you are having trouble, please ask your teaching assistant for help.
- You must submit your assignment prior to the deadline.