CS 331 Homework 2

Ararat Saribekyan

September 14, 2025

https://github.com/ararat-aua/Operating-systems-HW/tree/main/fork_exec

Assignment 0: Multiple Fork calls

Objective: Understand the process hierarchy when multiple sequential fork syscalls are invoked.

Source code:

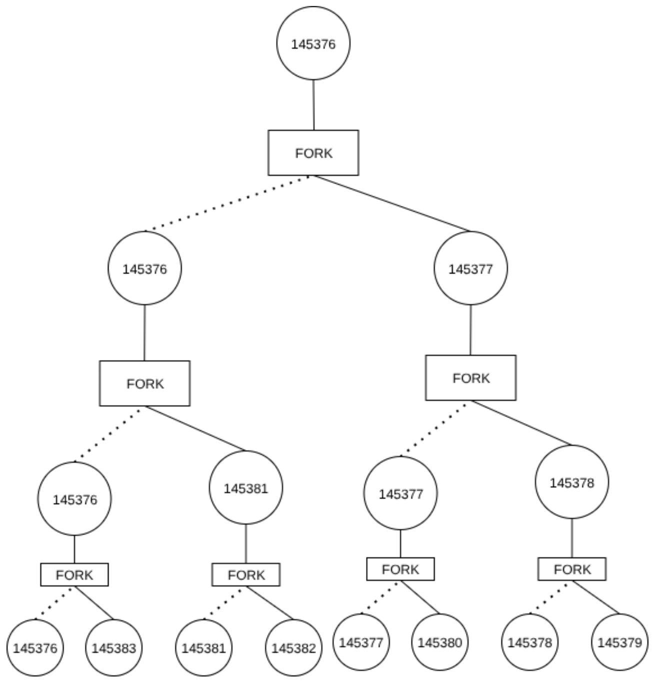
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

void call_fork(int n){
    int space_size = n*8+1;
    char space[space_size];
    int i=0;
    for(; i<space_size-1; i++){
        space[i] = ' ';
    }
    space[i] = '\0';


    pid_t pid = fork();
    if(pid == -1){
        exit(0);
    }
    else if (pid != 0){
        wait(NULL);
    }
    else{
        printf("%s%d->%d\n", space, getppid(), getpid());
    }
}

int main(){

    int n = 3;
    for (int i=0; i<n; i++){
        call_fork(i);
    }
    return 0;
}
```

Output:

```
145376->145377
        145377->145378
                145378->145379
                145377->145380
        145376->145381
                145381->145382
                145376->145383
```

Analysis:

- The following program creates a process hierarchy with 3 levels of fork calls.
- As every parent process waits for its child to finish, the tree is traversed in a in-order manner from right to left.
- The output is printed in a manner where each iteration's fork call prints with tabulation proportional to iteration number. So we can see the parent-child relationship in the output on each iteration.
- The diagram below shows the process hierarchy; as it can be seen, there are 8 processes created in total, corresponding to leaves of the tree.



Assignment 1: Simple Fork and Exec

Objective: Understand the creation of a child process using fork and how to replace it with a new program using execl.

Source code:

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

int main(){
    int ret;
    printf("Before fork\n");
    pid_t pid = fork();
    if (pid==-1){
        exit(1);
    }
    if (pid == 0){
        ret = execl("/usr/bin/ls", "ls", NULL);
    }
    else {
        wait(NULL);
        printf("Parent process done!\n");
    }
    return 0;
}
```

Output:

```
Before fork
0  0.c 1  1.c 2.c  3.c  4.c  test.txt
Parent process done!
```

Analysis:

- The following program creates a child process using fork and replaces it with a new program using execl.
- Parent process waits the child process to finish and prints a message.

Assignment 2: Multiple Forks and Execs

Objective: Work with multiple child processes created using fork and run different commands using execl.

Source code:

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

int main(){
    pid_t cur_pid = getpid();
    pid_t pid = fork();
    if (pid==0){
        execl("/usr/bin/ls", "ls", NULL);
    }
    else{
        wait(NULL);
        pid = fork();
        if (pid==0){
            execl("/usr/bin/date", "date", NULL);
        }
        else{
            wait(NULL);
            printf("Parent process done!\n");
        }
    }

    return 0;
}
```

Output:

```
0  0.c 1  1.c 2  2.c 3.c  4.c  test.txt
Sun Sep 14 13:11:48 UTC 2025
Parent process done!
```

Analysis:

- The following program creates a child process using fork where execl is used to run ls command first
- Then in the parent process another fork is used to run a new child process where execl is used to run date command.

- Parent process waits the child processes to finish and prints a message.
- The second fork is called in the parent process instead of just being called after the first Operating to have exactly two child processes belonging to the parent process.
- Also by using wait in the parent process, we can ensure the order of execution

Assignment 3: Fork and Exec with Arguments

Objective: Understand how to pass arguments to programs executed with execl.

Source code:

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

int main(){
    pid_t pid = fork();

    if (pid == 0){
        execl("/usr/bin/echo", "echo", "Hello from the child process!", NULL);
    }
    else {
        wait(NULL);
        printf("Parent process done!\n");
    }

    return 0;
}
```

Output:

```
Hello from the child process!
Parent process done!
```

Analysis:

- The following program creates a child process using fork where execl is used to run echo command
- execl takes the path to the command, command name and the arguments to the command as arguments

Assignment 4: Fork and Exec with Command-Line Arguments

Objective: Use fork and execl to run a command with multiple arguments.

Source code:

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

int main (){
    pid_t pid = fork();
    if (pid==0){
        execl("/usr/bin/grep", "grep", "main", "test.txt", NULL);
    }
    else {
        wait(NULL);
        printf("Parent process completed!\n");
    }
    return 0;
}
```

Output:

```
int main(){
Parent process completed!
```