

```
/* Name: Marcos Ondruska
```

```
PantherId: 2685885
```

```
Program description: This program takes a user input integer from the command  
line and applies the Collatz conjecture to it. A first child process is forked  
for this. A second child adds 4 to the userInput and also applies the  
Collatz conjecture
```

```
Task 2: What I've been taught is that these processes are non deterministic,  
and there is no way you can predict in which order the processes will  
finish/complete. The scheduler controls this function. Curiously when I run  
this application on a Mac child 1 always completely executes before child 2  
starts. On Ocelot I ran the application approximately 50 times and there is  
definitely randomness in the outcomes in how the processes concurrently  
run. Child 2 mostly completes after child 1. However child 2 does complete  
before child 1 at times. My guess is that the way that I have structured  
my if.. else with child 1 coded before child 2 biases it this way.
```

```
*/
```

```
#include <stdio.h>
```

```
#include <sys/types.h>
```

```
#include <unistd.h>
```

```
#include <string.h>
```

```
#include <stdlib.h>
```

```
int collatz(int processNumber, int n)
```

```
{
```

```
    while (n > 1)
```

```
    {
```

```
        if (n % 2 == 0)
```

```
        {
```

```
            n = n / 2;
```

```
            printf("From child%d: number = %d\n", processNumber, n);
```

```
        }
```

```
    else
```

```
    {
```

```

        n = 3 * n + 1;

        printf("From child%d: number = %d\n", processNumber, n);
    }
}

return 0;
}

int main(int argc, char *argv[])
{
    pid_t pid1, pid2;
    extern char *optarg;
    extern int optind;
    int userInput = 0;
    int n = 0; // child 1 variable
    int n2 = 0; // child 2 variable

    if (optind < argc)                // these are the arguments after the command-line options
        for (; optind < argc; optind++) // work through the command line options
            userInput = atoi(argv[optind]); // print command line option

    // Project parameters call for a user entered integer between 1 and 39
    if (userInput <= 0 || userInput >= 40)
    {
        printf("Please enter an integer between 1 and 39.\n");
        exit(0);
    }

    printf("\ncollatz%d\n\n", userInput);

    // fork a child process
    pid1 = fork();

    // error occurred
    if (pid1 < 0)
    {
        fprintf(stderr, "Fork failed");
        return 1;
    }
}

```

```

}

// child process 1
else if (pid1 == 0)
{
    printf("From child1, pid=%d, init: number=%d\n", getpid(), userInput);
    n = userInput;

    // apply Collatz conjecture
    collatz(1, userInput);
    printf("From child1, pid=%d I'm done!\n", getpid());
}

// child process 2
else
{
    pid2 = fork();

    if (pid2 < 0)
    {
        fprintf(stderr, "Fork failed");
        return 1;
    }
    else if (pid2 == 0)
    {
        n2 = userInput + 4;
        printf("From child2, pid=%d, init: number=%d\n", getpid(), n2);

        // apply Collatz conjecture
        collatz(2, n2);
        printf("From child2, pid=%d I'm done!\n", getpid());
    }

    // parent process
    else
    {
        wait(NULL);
        wait(NULL);
        printf("All my children Complete\n\n");
    }
}

```

```
}
```

```
return 0;
```

```
}
```