283 Project 2 Ruyu Xu

Implementation

1. Read from command and generate the number of process:

```
if(argc!=2){
    printf("No input :P\n");
    exit(1);
}
int num;
// Input number.
num=atoi(argv[1]);
```

2. Create process and print message:
   We can do this in two ways:

A. The first way is simple, we only need to calculate the total amount of process we are going to create, and then loop for that amount of times 2^n. Each time we use fork() to create a child process and let the child process print out a message with the current iterator value then terminate with exit().

Code:

```
#include <sys/wait.h>
#include <math.h>

int main(int argc,char* argv[]){
    if(argc!=2){
        printf("No input :P\n");
        exit(1);
    }
    int num,status,numProcess;
    // Input number.
    num=atoi(argv[1]);
    numProcess=(int)pow(2,(double)num);

    for(int i=0;i<numProcess;i++){
        pid_t fp=fork();
        if(fp<0){
            printf("Fork failed :P\n");
            exit(1);
        }
        // Child process
        else if(fp==0){
            printf("I am process %d\n",i);
            exit(0);
        }
        // Parent process
        else{
            wait(NULL);
        }
    }
    return 0;

}
```

Result:

```
apricot@uuu:~/Desktop$ gcc -o p2m1 p2method1.c -lm
apricot@uuu:~/Desktop$ ./p2m1 3
I am process 0
I am process 1
I am process 2
I am process 3
I am process 4
I am process 5
I am process 6
I am process 7
apricot@uuu:~/Desktop$
```

B. Another way to do this is by operating a loop of n times instead of a straight 2^n times.

I'll show the code and result of it first.

Code:

```c
// This uses the second method.
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc,char* argv[]){
    if(argc!=2){
        printf("No input :P\n");
        exit(1);
    }
    int num;
    // Input number.
    num=atoi(argv[1]);

    // Pid of parent process.
    int parent_pid=getpid();
    printf("I am process %d, no.0.\n",parent_pid);

    for(int i=0;i<num;i++){
        pid_t fp=fork();
        if(fp<0){
            printf("Fork failed :P\n");
            exit(1);
        }
        // Child process
        else if(fp==0){
            printf("I am process %d, no.%d.\n",(int)getpid(),(int)getpid()-parent_pid);
        }
        // Parent process
        else{
            wait(NULL);
        }
    }
    return 0;
}
```

Result:

```
apricot@uuu:~/Desktop$ ./p2 4
I am process 2034, no.0.
I am process 2035, no.1.
I am process 2036, no.2.
I am process 2037, no.3.
I am process 2038, no.4.
I am process 2039, no.5.
I am process 2040, no.6.
I am process 2041, no.7.
I am process 2042, no.8.
I am process 2043, no.9.
I am process 2044, no.10.
I am process 2045, no.11.
I am process 2046, no.12.
I am process 2047, no.13.
I am process 2048, no.14.
I am process 2049, no.15.
```

The reason why this would work is this:

We know that the fork() sys call duplicate an (almost) exactly the same process, then for each fork() in a for loop, it's going to double the number of total processes in general (which means not at the same time). To make it more clear, we can see a loop for n times as a repeat of fork() for n times:

```
for(i=0;i<n;i++){
        fork();
}
```

$=$ n { fork(); fork(); …… fork();

After the 1st fork(), there's 2 processes. Those two processes are all going to run the 2nd fork(), and then there's going to be 4 processes in total (2 old ones and 2 new ones). The processes exist after kth fork() (the fork() in the iteration when i=k) run in every process is two times of that before it.
Repeat the duplication for n times. After the nth fork(), there's going to be 2^n processes(including the parent process).
By doing so, we create 2^n processes.
Before everything starts, we record the pid of the parent processes, then each time we create a new child process, we print out its PID and the number of it as #child_pid - parent_pid

Testing:

1. Input nothing.

```
apricot@uuu:~/Desktop$ ./p2
No input :P
```

2. Input n as some random numbers:3,7,1,0.

n=3

```
apricot@uuu:~/Desktop$ ./p2 3
I am process 2100, no.0.
I am process 2101, no.1.
I am process 2102, no.2.
I am process 2103, no.3.
I am process 2104, no.4.
I am process 2105, no.5.
I am process 2106, no.6.
I am process 2107, no.7.
```

n=7

```
apricot@uuu:~/Desktop$ ./p2 7
I am process 2114, no.0.
I am process 2115, no.1.
I am process 2116, no.2.
I am process 2117, no.3.
I am process 2118, no.4.
I am process 2119, no.5.
I am process 2120, no.6.
I am process 2121, no.7.
I am process 2122, no.8.
I am process 2123, no.9.
I am process 2124, no.10.
I am process 2125, no.11.
I am process 2126, no.12.
I am process 2127, no.13.
I am process 2128, no.14.
I am process 2129, no.15.
I am process 2130, no.16.
I am process 2131, no.17.
```

(fold some lines here cause it's too long)

```
I am process 2218, no.104.
I am process 2219, no.105.
I am process 2220, no.106.
I am process 2221, no.107.
I am process 2222, no.108.
I am process 2223, no.109.
I am process 2224, no.110.
I am process 2225, no.111.
I am process 2226, no.112.
I am process 2227, no.113.
I am process 2228, no.114.
I am process 2229, no.115.
I am process 2230, no.116.
I am process 2231, no.117.
I am process 2232, no.118.
I am process 2233, no.119.
I am process 2234, no.120.
I am process 2235, no.121.
I am process 2236, no.122.
I am process 2237, no.123.
I am process 2238, no.124.
I am process 2239, no.125.
I am process 2240, no.126.
I am process 2241, no.127.
```

n=1

```
apricot@uuu:~/Desktop$ ./p2 1
I am process 2108, no.0.
I am process 2109, no.1.
```

n=0

```
apricot@uuu:~/Desktop$ ./p2 0
I am process 2110, no.0.
```

Compile:

```
apricot@uuu:~/Desktop$ make -f Makefile
gcc -o p2 p2.c
apricot@uuu:~/Desktop$ ./p2 3
I am process 2282, no.0.
I am process 2283, no.1.
I am process 2284, no.2.
I am process 2285, no.3.
I am process 2286, no.4.
I am process 2287, no.5.
I am process 2288, no.6.
I am process 2289, no.7.
apricot@uuu:~/Desktop$
```

Original Codes:

The p2.c implemented this appoarch.

Using the second method mentioned above:

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

int main(int argc,char* argv[]){
    if(argc!=2){
        printf("No input :P\n");
        exit(1);
    }
    int num;
    // Input number.
    num=atoi(argv[1]);

    // Pid of parent process.
    int parent_pid=getpid();
    printf("I am process 0\n");

    for(int i=0;i<num;i++){
        pid_t fp=fork();
        if(fp<0){
            printf("Fork failed :P\n");
            exit(1);
        }
        // Child process
        else if(fp==0){
            printf("I am process %d\n",(int)getpid()-parent_pid);
        }
        // Parent process
        else{
            wait(NULL);
        }
    }
    return 0;
}
```

Using the first method:

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

int main(int argc,char* argv[]){
	if(argc!=2){
		printf("No input :P\n");
		exit(1);
	}
	int num,numProcess;
	// Input number.
	num=atoi(argv[1]);
	numProcess=(int)pow(2,(double)num);

	for(int i=0;i<numProcess;i++){
		pid_t fp=fork();
		if(fp<0){
			printf("Fork failed :P\n");
			exit(1);
		}
		// Child process
		else if(fp==0){
			printf("I am process %d\n",i);
			exit(0);
		}
		// Parent process
		else{
			wait(NULL);
		}
	}
	return 0;

}
```