PARENT/CHILD AND PARALLEL COMPUTING

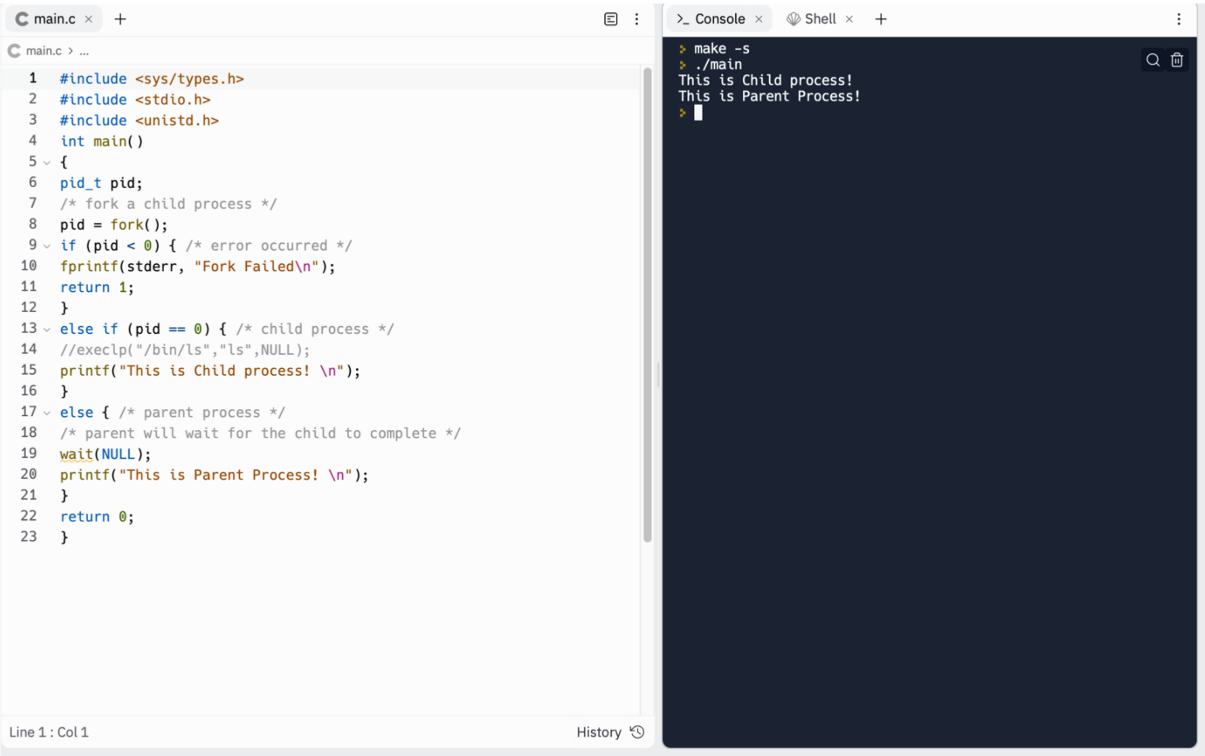
1. INTRODUCTION

In this report, I will focus on the “CREATING A NEW PROCESS USING FORK FUNCTION” and “SPLITTING A TASK BETWEEN TWO PROCESSES” code. I explain how it works, why it written that way and what it does.

2. “CREATING A NEW PROCESS USING FORK FUNCTION”

This program first creates a child program with fork() function. It returns an error message if pid (Process ID) is 0 or less, and if pid is 0, returns "This is Child Process" message. Otherwise (pid greater than 0), it returns "This is Parent Process".

The picture below shows how this program was executed.



```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
pid_t pid;
/* fork a child process */
pid = fork();
if (pid < 0) { /* error occurred */
fprintf(stderr, "Fork Failed\n");
return 1;
}
else if (pid == 0) { /* child process */
//execlp("/bin/ls","ls",NULL);
printf("This is Child process! \n");
}
else { /* parent process */
/* parent will wait for the child to complete */
wait(NULL);
printf("This is Parent Process! \n");
}
return 0;
}
```

```
> make -s
> ./main
This is Child process!
This is Parent Process!
>
```

3. “SPLITTING A TASK BETWEEN TWO PROCESSES”

This program adds up all the numbers from 1 to 400000 by dividing the process into four child programs.

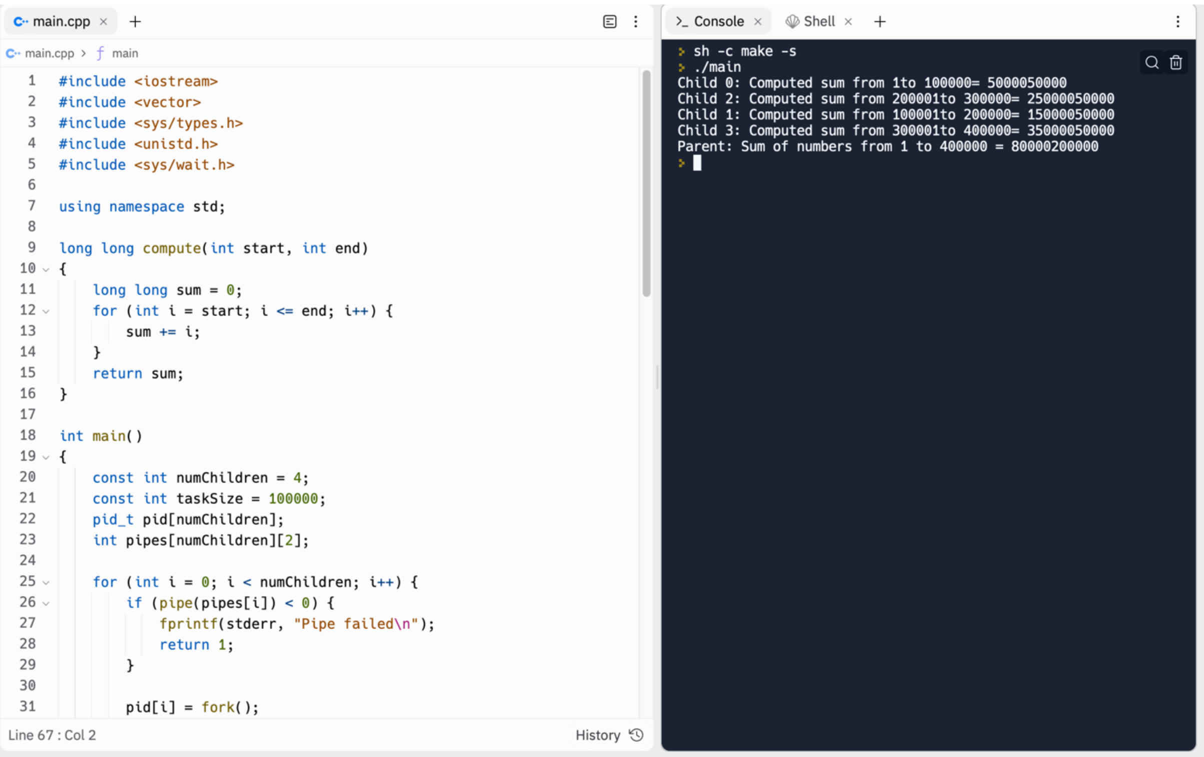
First, “compute” method is defined. This method adds up the numbers in the specified range in order.

Next, “main” method is defined. This method says the number of child programs is 4 and each program has responsible for 100000 calculations. In addition, this method order to obtain the PID of the child program and link it to the parent program.

If the PID is 0, in other words, it was a child program, it adds up all 1-100000 and returns that value to the parent program. The next child program is 100001-200000, the next child program is 200001-300000, and so on.

When all four child programs have been done, the parent program adds up the four received values and displays the result.

The picture below shows how this program was executed.



```cpp
#include <iostream>
#include <vector>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

using namespace std;

long long compute(int start, int end)
{
    long long sum = 0;
    for (int i = start; i <= end; i++) {
        sum += i;
    }
    return sum;
}

int main()
{
    const int numChildren = 4;
    const int taskSize = 100000;
    pid_t pid[numChildren];
    int pipes[numChildren][2];

    for (int i = 0; i < numChildren; i++) {
        if (pipe(pipes[i]) < 0) {
            fprintf(stderr, "Pipe failed\n");
            return 1;
        }

        pid[i] = fork();
```

```
> sh -c make -s
> ./main
Child 0: Computed sum from 1to 100000= 5000050000
Child 2: Computed sum from 200001to 300000= 25000050000
Child 1: Computed sum from 100001to 200000= 15000050000
Child 3: Computed sum from 300001to 400000= 35000050000
Parent: Sum of numbers from 1 to 400000 = 80000200000
> 
```

4. WHAT I LEARNED

We found that the PID of a child program is always 0. This means that the operating system distinguishes between the parent program and the small program based on the PID. We also found that the processing time is reduced by having the child program and sharing the processing.