OS – hw1 學號：1071552 姓名：蔡曜亦

1. 程式的設計理念、程式如何編譯，以及如何操作:

使用 shm_open ,mmap 等函式, 實現 share memory 的方法來傳 parent process 和 child process 計算出來的值。當使用者輸入 n 以後, 會 fork 出一個子行程, 這裡利用 ptr 指標指向一個位址並分配空間利用 ptr[0] ,ptr[1], ...存取這些記憶體裡的數值。 用 while 迴圈和 if-else 來區分奇數和偶數還有 pid == 0 和 pid >0 時該執行的動作，並再設計一個 while 迴圈來判定 share memory 裡存放當下 getpid()的值來使 parent 和 child 可以等待另外一個行程執行完並傳回數值後再讓另一個行程使用改變後的數值繼續做運算。

編譯及操作: (1) g++ 檔名.cpp -o 檔名 -lrt （1071552_01.cpp）(2) ./檔名 (3)輸入測資

```
tsai@tsai-VirtualBox:~$ g++ hw1.cpp -o hw1 -lrt
tsai@tsai-VirtualBox:~$ ./hw1
35
```

2.完成部分:

基本功能:
i. 父行程從命令列讀入整數參數。
ii. 父行程能用 fork()產生一個子行程。
iii. 父行程能用 POSIX shared memory 來傳遞兩個參數給子行程。
iv.父行程與子行程可以成功計算 Collatz conjecture 結果。
v.父行程與子行程能用相同的 shared memory 空間來傳遞正確結果給對方。
vi.父行程與子行程有一個使用 POSIX shared memory 的同步機制，可以讓父行程與子行程的運作正確同步。
vii. 父行程可以正確使用 getpid() 取得自己的 pid。

進階功能:
i. 最後父行程再印出 Collatz conjecture 序列中最大的數字，以及它是第幾次算出的結果。

```
tsai@tsai-VirtualBox:~$ g++ hw1.cpp -o hw1 -lrt
tsai@tsai-VirtualBox:~$ ./hw1
35
[6231 Child]:106
[6230 Parent]:53
[6231 Child]:160
[6230 Parent]:80
[6231 Child]:40
[6230 Parent]:20
[6231 Child]:10
[6230 Parent]:5
[6231 Child]:16
[6230 Parent]:8
[6231 Child]:4
[6230 Parent]:2
[6231 Child]:1
[6230 Parent]:1
Max:160
Turn:3
```

作業.pdf 上的測資

```
tsai@tsai-VirtualBox:~$ ./hw1
96
[6233 Child]:48
[6232 Parent]:24
[6233 Child]:12
[6232 Parent]:6
[6233 Child]:3
[6232 Parent]:10
[6233 Child]:5
[6232 Parent]:16
[6233 Child]:8
[6232 Parent]:4
[6233 Child]:2
[6232 Parent]:1
Max:48
Turn:1
```

<case 1> parent:1 結束

```
tsai@tsai-VirtualBox:~$ ./hw1
100
[6236 Child]:50
[6235 Parent]:25
[6236 Child]:76
[6235 Parent]:38
[6236 Child]:19
[6235 Parent]:58
[6236 Child]:29
[6235 Parent]:88
[6236 Child]:44
[6235 Parent]:22
[6236 Child]:11
[6235 Parent]:34
[6236 Child]:17
[6235 Parent]:52
[6236 Child]:26
[6235 Parent]:13
[6236 Child]:40
[6235 Parent]:20
[6236 Child]:10
[6235 Parent]:5
[6236 Child]:16
[6235 Parent]:8
[6236 Child]:4
[6235 Parent]:2
[6236 Child]:1
[6235 Parent]:1
Max:88
Turn:8
```

<case 2> child:1 需再傳回 parent:1 結束

程式碼解釋： 打在//之後

```cpp
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h> //fork 要使用
#include<time.h>
#include<string.h>
#include<sys/wait.h>
#include<sys/mman.h>
#include<sys/file.h>
#include<fcntl.h>

using namespace std;

void error_and_die(const char *msg){
    perror(msg);
    exit(EXIT_FAILURE);
}

int main(int argc,char *argv[]){

    const char *memname = "pidID";
    int fd = shm_open(memname, O_CREAT | O_TRUNC | O_RDWR, 0666);
    if(fd == -1)error_and_die("shm_open"); //判斷開啟memory有沒有error產生

    int r = ftruncate(fd,sizeof(int)*5);
    if(r!=0)error_and_die("ftruncate");//判斷configure size有沒有error產生

    int *ptr = (int*)mmap(0,sizeof(int),PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    //開啟mmap並指向要存的記憶體位址開始存
    if(ptr == MAP_FAILED)error_and_die("mmap");

    int n = 0;
    cin >> n; //輸入整數n

    if( n < 1 || n > 100) return 0; // 限定輸入範圍 1 <= n <= 100

    //使用剛剛指向的位址（配了5個int)
    ptr[0] = n; //存n
    ptr[1]= getpid(); //存放當下執行的行程的pid（目前為父行程的pid)
    ptr[2] = 0; //max
    ptr[3] = 0; //count
    ptr[4] = 0; //tmp
    pid_t pid = fork();//父行程用fork()產生一個子行程

```

```
    while(ptr[0]!=1 ) { //做到1時結束
        /*使用share memory指向的ptr[1] 存放目前的 pid ,
        若ptr[1] == getpid()讓此行程執行無窮迴圈 等待另一個行程計算出結果並改變ptr[1]之值*/
        while(true) if(ptr[1] != getpid() || ptr[0]==1)break;

        if( ptr[0] % 2 == 1 && ptr[0]!=1 ){ // 奇數
            if(ptr[0]!=1 ){
                ptr[0] = 3* ptr[0] + 1;  // 考拉茲猜想
                ptr[3]++;  // count++
                if(ptr[2] < ptr[0]){
                    ptr[2] = ptr[0];  // replace max
                    ptr[4] = ptr[3];  // replace count
                }
            }
            if (pid == 0) // 若為子行程 印出子行程pid和計算結果
                cout <<"["<<getpid()<<" Child]:"<< ptr[0]<<endl;
            if(pid > 0)  // 若為父行程 印出父行程pid和計算結果
                cout <<"["<<getpid()<<" Parent]:"<< ptr[0]<<endl;
            if (ptr[0]!=1) ptr[1] = getpid(); //存放當下執行的行程的pid
        }
        else if(ptr[0]==1){   // 父行程和子行程進行到最後剩下1要印出時
            if(pid == 0 ){  //若最後為子行程計算出1則不再多印出一次 (計算出後已印出child:1 & parent:1)
                ptr[1] = getpid();
                break;
            }
            if(pid > 0){  //若最後為父行程計算出1則在這裡印出子行程不需要再次印出
                cout <<"["<<getpid()<<" Parent]:"<<ptr[0]<<endl;
                ptr[1] = getpid();
                break;
            }
        }
        else{    // 偶數
            if (ptr[0]!=1){
                ptr[0] = ptr[0] / 2; // 考拉茲猜想
                ptr[3]++;  // count++
                if(ptr[2] < ptr[0]){
                    ptr[2] = ptr[0]; // replace max
                    ptr[4] = ptr[3]; // replace count
                }
            }
            if(pid == 0 )  // 若為子行程 印出子行程pid和計算結果
                cout <<"["<<getpid()<<" Child]:"<<ptr[0]<<endl;
            if(pid > 0)  // 若為父行程 印出父行程pid和計算結果
                cout <<"["<<getpid()<<" Parent]:"<<ptr[0]<<endl;
            if(ptr[0]!=1) ptr[1] = getpid(); //存放當下執行的行程的pid
        }
    }
```

```
    // 印出 max 和 turn
    if(pid > 0 )cout << "Max:"<<ptr[2]<<endl<<"Turn:"<<ptr[4]<<endl;


    close(fd);
    munmap(ptr,sizeof(int)*5);
    shm_unlink(memname);

    return 0;
}
```