Problem 3.14

1. 使用command line argument的方式輸入程式參數，ex: ./p1 25，下方會有完整在terminal執行程式的截圖，25代表的是從25開始執行Collatz conjecture演算法。
2. 執行fork() api呼叫system call創造parent及child process，因為parent及child process都可以搶CPU的資源，為了確保child能先於parent執行在parent process執行wait() system call 等待child process完成，child process則是主要執行Collatz conjecture演算法的地方。

```
    /*child process*/
    else if(pid == 0){
        int num = atoi(argv[1]);
        printf("%d ", num);
        while (num != 1){
            if (num % 2 == 0){
                num /= 2;
                printf("%d ", num);
            }
            else{
                num = 3 * num + 1;
                printf("%d ", num);
            }
        }
        printf("\n");
    }
    /*parent process*/
    else{
        printf("Parent is waiting for child to complete...\n");
        wait(NULL);
        printf("Child Complete...\n");
    }
    return 0;
}
```

3. 執行結果的截圖

```
(base) robert@LAPTOP-46RPPSGN:/mnt/c/users/rober/desktop/111-2OS$ gcc p1.c -o p1
(base) robert@LAPTOP-46RPPSGN:/mnt/c/users/rober/desktop/111-2OS$ ./p1 25
Parent is waiting for child to complete...
25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
Child Complete...
```

Problem 3.15

這題要求使用share memory的方式來執行parent及child process間的communication
同上題使用command line argument的方式輸入程式參數，ex: ./p2 25，下方會有完整在terminal執行程式的截圖，25代表的是從25開始執行Collatz conjecture演算法。

a. 首先利用shm_open()l創建share memory的object，而後利用ftruncate()配置空間給此object，最後使用mmap()來獲取共享記憶體的指標，之後我們就可以用這個指標來進行檔案的讀寫。

```
int shm_fd;
void *ptr;

shm_fd = shm_open(name, O_CREAT|O_RDWR, 0666);
ftruncate(shm_fd, SIZE);
ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);
```

b. 執行fork() api呼叫system call創造parent及child process，因為parent及child process都可以搶CPU的資源，為了確保child能先於parent執行在parent process執行wait() system call 等待child process完成，若fork()創建process失敗使用shm_unlink歸還記憶體給OS。

```
pid_t pid;

pid = fork();
if (pid < 0){
    fprintf(stderr, "Fork Failed\n");
    shm_unlink(name);
    return 1;
}
```

c . child process執行Collatz conjecture演算法，並將執行的結果透過sprintf加到mmap()創造的共享記憶體指標當中同時更新下次要加入指標的位置。

```
/*child process*/
else if (pid == 0){
    int num = atoi(argv[1]);
    // printf("%d ", num);
    ptr += sprintf(ptr, "%d ", num);
    while (num != 1){
        if (num % 2 == 0){
            num /= 2;
            // printf("%d ", num);
            ptr += sprintf(ptr, "%d ", num);
        }
        else{
            num = 3 * num + 1;
        //    printf("%d ", num);
            ptr += sprintf(ptr, "%d ", num);
        }
    }
    // printf("\n");
    printf("Child Process completely write the content of the sequence to the share memory object\n");
}
```

d. 最後parent process執行讀取share memory object的動作，將mmap()創造的共享記憶體指標轉型成字元指標印出並使用shm_unlink歸還記憶體給OS。

```c
/*parent process*/
else{
    printf("Parent is waiting for child to complete...\n");
    wait(NULL);
    shm_fd = shm_open(name, O_RDONLY, 0666);
    ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
    printf("%s\n", (char *)ptr);
    shm_unlink(name);
    printf("Child Complete...\n");
}
return 0;
```

執行結果截圖

```
(base) robert@LAPTOP-46RPPSGN:/mnt/c/users/rober/desktop/111-2OS$ gcc p2.c -o p2
(base) robert@LAPTOP-46RPPSGN:/mnt/c/users/rober/desktop/111-2OS$ ./p2 25
Parent is waiting for child to complete...
Child Process completely write the content of the sequence to the share memory object
25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
Child Complete...
```

Problem 3.20

使用command line argument的方式輸入程式參數 ex: ./filecopy input.txt copy.txt

使用pipe() system call 創建ordinary pipe，傳入一個長度為2的integer陣列，0的位置代表read、1的位置代表write。

```c
int file[2];
if (pipe(file) == -1){
    fprintf(stderr, "Pipe failed\n");
    return 1;
}
```

創建要read(input.txt)及write(copy.txt)的檔案指標及傳輸途中暫存的buffer字元指標。

```c
char buffer[size];
FILE *input = fopen(argv[1], "r");
FILE *copy = fopen(argv[2], "w");
```

使用fork() api呼叫system call創造parent及child process，與前2題不同的是parent process先以128個字元為單位讀取input.txt的內容暫存前面提到的buffer字元指標，當

buffer滿了寫入pipe當中再將buffer清空，當input.txt的內容都寫入pipe完畢關閉input.txt檔案指標。

```
/*parent process*/
else {
    close(file[0]);
    while (fgets(buffer, size, input) != NULL) {
        write(f char buffer[128] zeof(buffer));
        memset(buffer, 0, sizeof(buffer));
    }
    close(file[1]);
    fclose(input);
    wait(NULL);
}
```

接著child process從pipe中讀取parent process寫入的內容至buffer中，buffer滿了將其寫入目標檔案中(copy.txt)，寫入完畢關閉copy.txt檔案指標。

```
/*child process*/
if (pid == 0) {
    close(file[1]);
    while (read(file[0], buffer, sizeof(buffer)) > 0) {
        fputs(buffer, copy);
    }
    close(file[0]);
    fclose(copy);
}
```

執行結果截圖

```
(base) robert@LAPTOP-46RPPSGN:/mnt/c/users/rober/desktop/111-2OS$ gcc filecopy.c -o filecopy
(base) robert@LAPTOP-46RPPSGN:/mnt/c/users/rober/desktop/111-2OS$ ls
copy.txt  core  filecopy  filecopy.c  HW1.pdf  input.txt  p1  p1.c  p2  p2.c  tmp  tmp.c
(base) robert@LAPTOP-46RPPSGN:/mnt/c/users/rober/desktop/111-2OS$ ./filecopy input.txt  copy.txt
```