Title: NTU IM Operating-System HW 01
Student ID: R12631070
Name: 林育新

使用方式

```
Make
./hw1_Q1 <int n>
./hw1_Q2 <int n>
./hw1_Q3 <int n>
```

作業處理邏輯

hw1_Q1.c 對應到 3.13 (Q1)

題目: 此程式使用 `fork()` 函數創建子進程，子進程負責計算並印出前 n 項斐波那契數列，父進程則等待子進程完成後印出提示訊息。

1. **子進程邏輯：**
   - 斐波那契數列前兩項為0, 1
   - 後續計算方式為 a[n] = a[n-1] + a[n-2]

```c
// 處理前兩項特殊情況
if (n >= 1)
    printf("%d ", 0); // 第1項：0
if (n >= 2)
    printf("%d ", 1); // 第2項：1

// 初始化變數
int a = 0, b = 1;
// 從第3項開始計算
for (int i = 3; i <= n; i++)
{
    int next = a + b;    // 計算下一項
    printf("%d ", next); // 打印當前項
    a = b;               // 更新前前一項
    b = next;            // 更新前一項
}
```

2. **父進程邏輯：**
   - 只需要`wait(NULL)`等待子進程完成。

```
// 創建子進程 (課本範例見ch03 p26)
pid_t pid = fork(); // 父進程返回子進程的 PID (可以用waitpid()等待子進程結束)，子進程返回 0，失敗返回 -1

if (pid < 0)
{
    // fork() 失敗的情況
    fprintf(stderr, "創建子進程失敗\n");
    return EXIT_FAILURE;
}
else if (pid == 0)
{
    // 子進程執行的程式碼
    generate_fibonacci(n); // 生成斐波那契數列
    exit(EXIT_SUCCESS);    // 子進程正常退出
}
else
{
    // 父進程執行的程式碼
    // wait() 等待任何一個子進程結束，並且返回結束進程的pid，若無執行中子進程，返回-1)。
    // 如果要指定進程結束，可以用waitpid(pid, &status, 0);
    // 如果要等待所有進程結束，可以while(wait(NULL) > 0);。
    wait(NULL);
    printf("父進程：子進程已完成。\n");
}
```

hw1_Q2.c 對應到 3.14 (Q2)

題目: 使用 `fork()` 函數創建子進程，子進程計算並印出 Collatz 序列，父進程僅等待子進程結束，不參與計算，也不會知道結果。

1. **子進程邏輯**：
    - Collatz 序列定義：
      對於任意正整數 n，若 n 為偶數，下一項為 n/2；若 n 為奇數，下一項為 3n+1。
      重複上述步驟，直到數列到達 1 為止。

```
printf("%d", n); // 先印出第一個數字
while (n != 1)
{
    if (n % 2 == 0) // 如果n是偶數 就除2
    {
        n = n / 2;
    }
    else // 如果n是奇數 就乘3加1
    {
        n = 3 * n + 1;
    }
```

```
        printf(", %d", n);
    }
```

2. **父進程邏輯**：
    - 與第一題相同，只需要`wait(NULL)`。

hw1_Q3.c 對應到 3.15 (Q3)

題目: 使用 `fork()` 函數創建子進程，子進程計算 Collatz 序列。透過共享記憶體將結果傳遞給父進程，父進程從共享記憶體讀取結果並印出來。

1. **子進程邏輯**：
    - 計算 Collatz 序列的邏輯與第二題相同，但結果會存入共享記憶體，而非直接輸出。
    - 使用 `snprintf()` 將每個數字以字串形式寫入共享記憶體，並確保不超出記憶體大小。

```c
int n = start;
int offset = 0;
offset += snprintf(shm_ptr + offset, SHM_SIZE - offset, "%d", n);
while (n != 1)
{
    if (n % 2 == 0)
        n = n / 2;
    else
        n = 3 * n + 1;
    if (offset + 1 >= SHM_SIZE - offset)
        break;
    offset += snprintf(shm_ptr + offset, SHM_SIZE - offset, ", %d", n);
}
```

2. **父進程邏輯**：
    - 父進程等待子進程完成後，從共享記憶體讀取結果並輸出。
    - 最後清理共享記憶體，確保不留垃圾資源。

```c
wait(NULL);
printf("Collatz 序列 (起始值 = %d):\n%s\n", start, shm_ptr);
if (munmap(shm_ptr, SHM_SIZE) == -1)
    perror("munmap 失敗");
if (shm_unlink(name) == -1)
    perror("shm_unlink 失敗");
```

3. **共享記憶體的使用**：
    - 使用 `shm_open()` 創建或打開共享記憶體，並使用 `mmap()` 將其映射到進程的虛擬記憶體空間。
    - 使用 `ftruncate()` 設定共享記憶體的大小，確保共享記憶體有足夠的空間存放 Collatz 序列的結果。

- 子進程將計算結果寫入共享記憶體，父進程從中讀取結果。
- 最後使用 `munmap()` 和 `shm_unlink()` 清理共享記憶體。

```
int shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);
if (shm_fd == -1)
{
    perror("shm_open 失敗");
    return EXIT_FAILURE;
}
// 設定共享記憶體大小
if (ftruncate(shm_fd, SHM_SIZE) == -1)
{
    perror("ftruncate 失敗");
    return EXIT_FAILURE;
}
char *shm_ptr = mmap(0, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
if (shm_ptr == MAP_FAILED)
{
    perror("mmap 失敗");
    return EXIT_FAILURE;
}
```