

Nhóm: 8

Thông tin thành viên:

- Dương Thuận Trí- 22521517
- Trương Quốc Vinh- 22521682
- Nguyễn Thị Kim Ngân- 21521174
- Nông Tiến Dũng- 20521213

Lớp: 1

## **HỆ ĐIỀU HÀNH** **BÁO CÁO LAB 3**

### **CHECKLIST**

#### **3.5. BÀI TẬP THỰC HÀNH**

|                             | <b>BT 1</b>              | <b>BT 2</b>              | <b>BT 3</b>              | <b>BT 4</b>              |
|-----------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| <b>Trình bày cách làm</b>   | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <b>Chụp hình minh chứng</b> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <b>Giải thích kết quả</b>   | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

#### **3.6. BÀI TẬP ÔN TẬP**

|                             | <b>BT 1</b>              |
|-----------------------------|--------------------------|
| <b>Trình bày cách làm</b>   | <input type="checkbox"/> |
| <b>Chụp hình minh chứng</b> | <input type="checkbox"/> |
| <b>Giải thích kết quả</b>   | <input type="checkbox"/> |

**Tự chấm điểm:** 10

*\*Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp:*

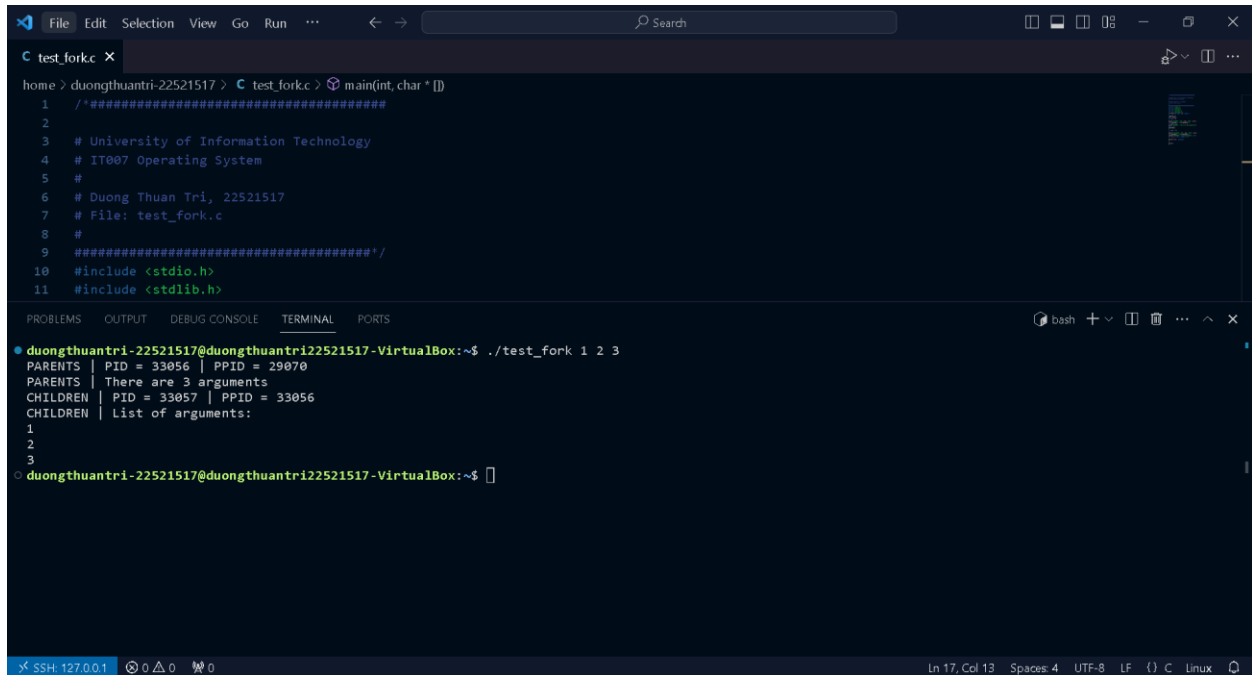
**<MSSV>\_LAB3.pdf**

## 2.5. BÀI TẬP THỰC HÀNH

### 1. Thực hiện Ví dụ 3-1, Ví dụ 3-2, Ví dụ 3-3, Ví dụ 3-4 giải thích code và kết quả nhận được?

#### ❖ Ví dụ 3-1:

##### ➤ Thực hiện trên VSCode:



```
home > duongthuantri-22521517 > C test_fork.c > main(int, char * [])
1  /* *****
2
3  # University of Information Technology
4  # IT007 Operating System
5  #
6  # Duong Thuan Tri, 22521517
7  # File: test_fork.c
8  #
9  ***** */
10 #include <stdio.h>
11 #include <stdlib.h>

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
• duongthuantri-22521517@duongthuantri22521517-VirtualBox:~$ ./test_fork 1 2 3
PARENTS | PID = 33056 | PPID = 29070
PARENTS | There are 3 arguments
CHILDREN | PID = 33057 | PPID = 33056
CHILDREN | List of arguments:
1
2
3
o duongthuantri-22521517@duongthuantri22521517-VirtualBox:~$
```

##### ➤ Code:

```
# University of Information Technology
# IT007 Operating System
#
# <Your name>, <your Student ID>
# File: test_fork.c
#
#####*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
int main(int argc, char *argv[])
{
    __pid_t pid;
    pid = fork();
```

```
if (pid > 0)
{
printf("PARENTS | PID = %ld | PPID = %ld\n",
(long)getpid(), (long)getppid());
if (argc > 2)
printf("PARENTS | There are %d arguments\n",
argc - 1);
wait(NULL);
}

if (pid == 0)
{
printf("CHILDREN | PID = %ld | PPID = %ld\n",
(long)getpid(), (long)getppid());
printf("CHILDREN | List of arguments: \n");
for (int i = 1; i < argc; i++)
{
printf("%s\n", argv[i]);
}
}
exit(0);
}
```

➤ **Giải thích code:**

1. Chương trình bắt đầu bằng cách khai báo và định nghĩa các thư viện cần thiết như stdio.h, stdlib.h, unistd.h, sys/wait.h, sys/types.h.
2. Hàm main nhận vào tham số argc và argv để xử lý các đối số dòng lệnh.
3. Dòng pid = fork(); tạo một bản sao của tiến trình hiện tại. Nếu thành công, giá trị của pid sẽ lớn hơn 0 trong tiến trình cha và bằng 0 trong tiến trình con.
4. Trong tiến trình cha (if (pid > 0)), nó in ra PID (Process ID) và PPID (Parent Process ID) của tiến trình cha, và nếu có thêm đối số dòng lệnh, nó in ra số lượng các đối số đó:

```
printf("PARENTS | There are %d arguments\n", argc - 1);
```

5. Trong tiến trình con (if (pid == 0)), nó in ra PID và PPID của tiến trình con, sau đó in ra các đối số dòng lệnh:

```
printf("CHILDREN | List of arguments: \n");
for (int i = 1; i < argc; i++)
{
printf("%s\n", argv[i]);
}
```

}

6. Cuối cùng, `exit(0)` được gọi để kết thúc cả hai tiến trình.

➤ **Kết quả khi truyền vào 3 tham số: 1, 2, 3**

PARENTS | PID = 33056 | PPID = 29070

PARENTS | There are 3 arguments

CHILDREN | PID = 33057 | PPID = 33056

CHILDREN | List of arguments:

1

2

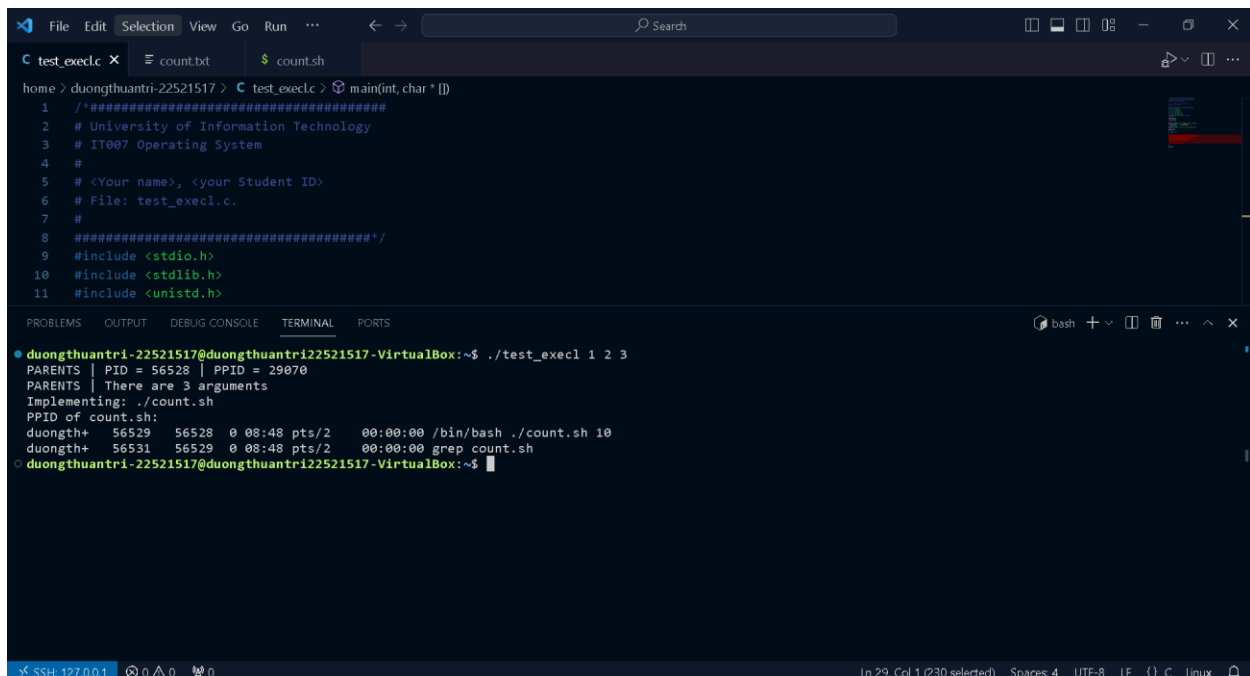
3

➤ **Giải thích kết quả:**

1. Tiến trình cha có PID là 33056 và PPID là 29070.
2. Tiến trình cha thông báo rằng có 3 đối số dòng lệnh vì số đối số dòng lệnh lớn hơn 2
3. Tiến trình con có PID là 33057 và PPID là 33056.
4. Tiến trình con in ra danh sách các đối số dòng lệnh là 1, 2, 3.

❖ **Ví dụ 3-2:**

➤ **Thực hiện trên VSCode:**



```
home > duongthuantri-22521517 > C test_execl.c > main(int, char *[])
1  /*
2  # University of Information Technology
3  # IT007 Operating System
4  #
5  # <Your name>, <your Student ID>
6  # File: test_execl.c.
7  #
8  */
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <unistd.h>

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
duongthuantri-22521517@duongthuantri22521517-VirtualBox:~$ ./test_execl 1 2 3
PARENTS | PID = 56528 | PPID = 29070
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
duongth+ 56529 56528 0 08:48 pts/2 00:00:00 /bin/bash ./count.sh 10
duongth+ 56531 56529 0 08:48 pts/2 00:00:00 grep count.sh
duongthuantri-22521517@duongthuantri22521517-VirtualBox:~$
```

## ➤ Giải thích code:

### 1. Script Bash "count.sh":

- Script này thực hiện đếm từ 1 đến giá trị được truyền vào từ tham số dòng lệnh (\$1).
- In ra thông điệp "Implementing:" cùng với tên của script.
- Sử dụng lệnh `ps -ef | grep count.sh` để hiển thị thông tin về các tiến trình đang chạy với tên "count.sh".
- Sử dụng vòng lặp `while` để đếm và ghi kết quả vào tệp "count.txt".
- Cuối cùng, thoát với mã lỗi 0.

### 2. Chương trình C "test\_execl.c":

- Chương trình sử dụng hàm `fork()` để tạo một tiến trình con.
- Trong tiến trình cha, in ra thông tin PID và PPID của chính nó.
- Nếu có nhiều hơn 2 đối số dòng lệnh, in ra số lượng đối số.
- Tiến trình cha đợi tiến trình con kết thúc bằng hàm `wait(NULL)`.
- Trong tiến trình con, sử dụng hàm `execl()` để thực thi script "count.sh" với tham số "10".
- Nếu hàm `execl()` thành công, các dòng code sau hàm `execl()` sẽ không được thực hiện (đó là lý do những câu lệnh `printf` sau hàm `execl()` không được thực hiện).

➤ **Giải thích kết quả:**

1. PARENTS | PID = 56528 | PPID = 29070:

- PID (Process ID) là số nhận dạng duy nhất cho tiến trình cha và PPID (Parent Process ID) là ID của tiến trình cha.
- Trong trường hợp này, PID của tiến trình cha là 56528 và PPID là 29070.

2. PARENTS | There are 3 arguments:

- In ra thông báo rằng có 3 đối số dòng lệnh được truyền vào chương trình cha.

3. Implementing: ./count.sh:

- In ra thông điệp "Implementing:" kèm theo tên của script được thực thi, ở đây là "count.sh".

4. PPID of count.sh:

- Sử dụng lệnh `ps -ef | grep count.sh` để hiển thị thông tin về các tiến trình đang chạy có tên "count.sh".
- Dòng output này thường sẽ hiển thị thông tin của tiến trình bash đang thực thi script "count.sh". Trong ví dụ này, PID của tiến trình bash là 56529, và PPID là 56528 (PID của tiến trình cha).

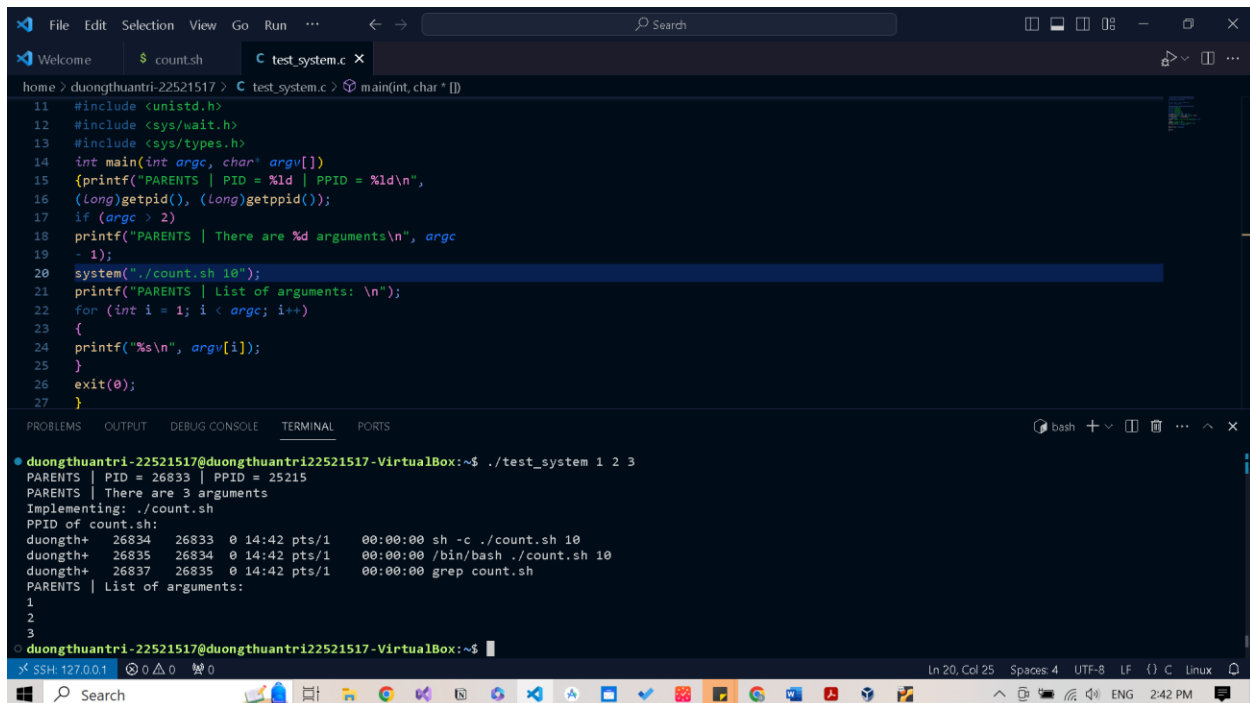
5. duongth+ 56529 56528 0 08:48 pts/2 00:00:00 /bin/bash ./count.sh 10:

- Dòng này hiển thị thông tin về tiến trình bash đang thực thi script "count.sh".
- 56529 là PID của tiến trình bash, 56528 là PPID (PID của tiến trình cha), "08:48" là thời gian bắt đầu của tiến trình, "pts/2" là terminal session, "00:00:00" là thời gian CPU đã sử dụng, và "/bin/bash ./count.sh 10" là lệnh đầy đủ của tiến trình.

6. duongth+ 56531 56529 0 08:48 pts/2 00:00:00 grep count.sh:

- Dòng này hiển thị thông tin về tiến trình grep đang chạy để tìm kiếm các tiến trình có tên "count.sh".
- 56531 là PID của tiến trình grep, 56529 là PPID (PID của tiến trình bash thực thi script "count.sh").

❖ **Ví dụ 3-3:**



The screenshot shows a Visual Studio Code window with a C file named `test_system.c` and a terminal window. The C code is as follows:

```
11 #include <unistd.h>
12 #include <sys/wait.h>
13 #include <sys/types.h>
14 int main(int argc, char* argv[])
15 {printf("PARENTS | PID = %ld | PPID = %ld\n",
16 (long)getpid(), (long)getppid());
17 if (argc > 2)
18 printf("PARENTS | There are %d arguments\n", argc
19 - 1);
20 system("./count.sh 10");
21 printf("PARENTS | List of arguments: \n");
22 for (int i = 1; i < argc; i++)
23 {
24 printf("%s\n", argv[i]);
25 }
26 exit(0);
27 }
```

The terminal output shows the execution of the program with arguments 1 2 3:

```
duongthuantri-22521517@duongthuantri22521517-VirtualBox:~$ ./test_system 1 2 3
PARENTS | PID = 26833 | PPID = 25215
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
duongth+ 26834 26833 0 14:42 pts/1 00:00:00 sh -c ./count.sh 10
duongth+ 26835 26834 0 14:42 pts/1 00:00:00 /bin/bash ./count.sh 10
duongth+ 26837 26835 0 14:42 pts/1 00:00:00 grep count.sh
PARENTS | List of arguments:
1
2
3
```

➤ **Giải thích code:**

1. Chương trình in ra thông tin về PID (Process ID) và PPID (Parent Process ID) của tiến trình cha:

```
printf("PARENTS | PID = %ld | PPID = %ld\n", (long)getpid(), (long)getppid());
```

2. Nếu có nhiều hơn 2 đối số dòng lệnh, in ra số lượng đối số:

```
if (argc > 2)
printf("PARENTS | There are %d arguments\n", argc- 1);
```

3. Sử dụng hàm `system("./count.sh 10")` để thực thi script "count.sh" với tham số "10" -> Một dãy số từ 1 đến 10 lại được in vào file count.txt ( mỗi số 1 dòng)

4. In ra list các đối số dòng lệnh sau khi hàm `system()` đã thực hiện:

```
printf("PARENTS | List of arguments: \n");
for (int i = 1; i < argc; i++)
{
printf("%s\n", argv[i]);
}
```

➤ **Kết quả nhận được sau khi thực thi với các tham số lần lượt là 1 2 3:**

PARENTS | PID = 62929 | PPID = 29070

PARENTS | There are 3 arguments

Implementing: ./count.sh

PPID of count.sh:

```
duongth+ 62930 62929 0 09:34 pts/2 00:00:00 sh -c ./count.sh 10
```

```
duongth+ 62931 62930 0 09:34 pts/2 00:00:00 /bin/bash ./count.sh 10
```

```
duongth+ 62933 62931 0 09:34 pts/2 00:00:00 grep count.sh
```

PARENTS | List of arguments:

1

2

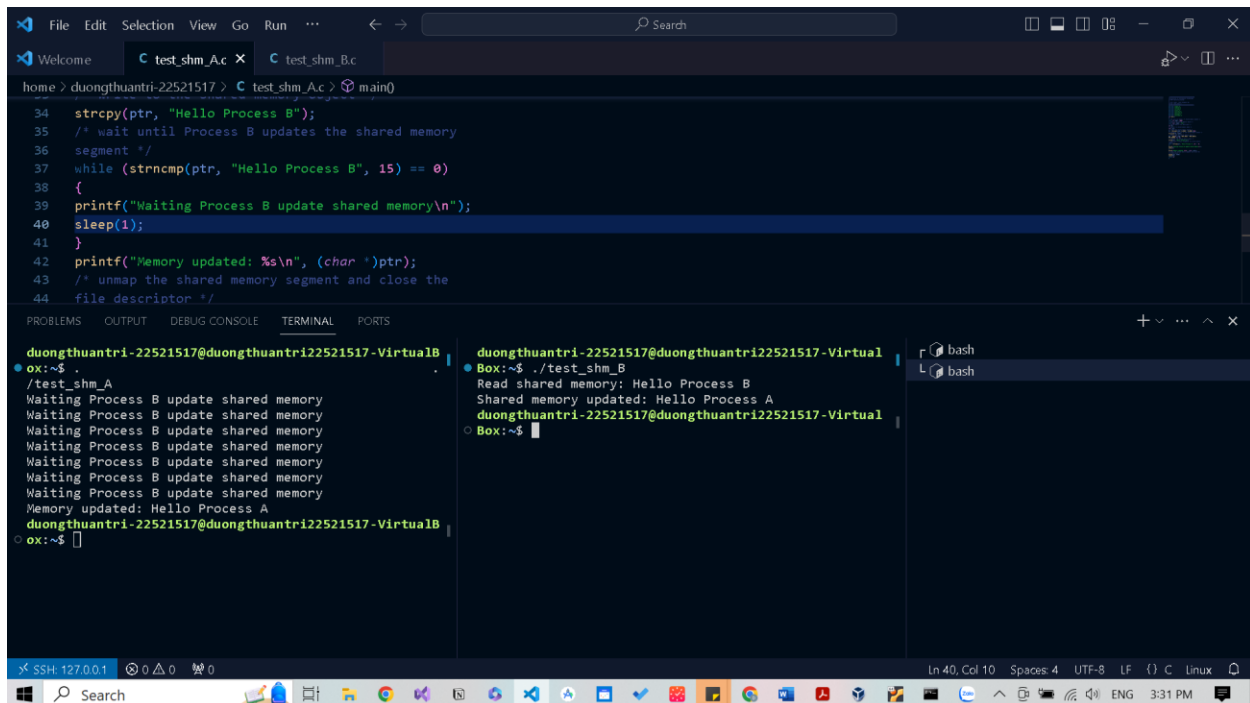
3

➤ **Giải thích kết quả:**

1. Tiến trình cha in ra thông tin về PID và PPID của chính nó: PID = 62929 | PPID = 29070
2. In ra thông báo về số đối số dòng lệnh ( vì có lớn hơn 2 đối số dòng lệnh)
3. Sử dụng hàm system("./count.sh 10") để thực thi script "count.sh" với tham số "10". Hàm này tạo một shell và chạy lệnh được cung cấp trong đó.
4. Kết quả hiển thị thông tin về các tiến trình liên quan đến việc thực thi script "count.sh". Đầu tiên, một shell (PID 62930) được tạo ra, sau đó tiến trình bash (PID 62931) thực thi script, và cuối cùng là tiến trình grep (PID 62933) tìm kiếm các tiến trình có tên "count.sh".
5. Tiến trình cha in ra list các đối số dòng lệnh sau khi hàm system() đã thực hiện.

❖ **Ví dụ 3-4:**





```
34 strcpy(ptr, "Hello Process B");
35 /* wait until Process B updates the shared memory
36 segment */
37 while (strncmp(ptr, "Hello Process B", 15) == 0)
38 {
39     printf("Waiting Process B update shared memory\n");
40     sleep(1);
41 }
42 printf("Memory updated: %s\n", (char *)ptr);
43 /* unmap the shared memory segment and close the
44 file descriptor */
```

```
duongthuantri-22521517@duongthuantri22521517-VirtualB
ox:~$ ./test_shm_A
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Memory updated: Hello Process A
duongthuantri-22521517@duongthuantri22521517-VirtualB
ox:~$
```

```
duongthuantri-22521517@duongthuantri22521517-Virtual
Box:~$ ./test_shm_B
Read shared memory: Hello Process B
Shared memory updated: Hello Process A
duongthuantri-22521517@duongthuantri22521517-Virtual
Box:~$
```

### ➤ Giải thích code:

#### Process A - test\_shm\_A.c:

- Khởi tạo Bộ Nhớ Chia Sẻ:
  - Sử dụng `shm_open` để tạo hoặc mở đối tượng bộ nhớ chia sẻ với tên "OS".
  - Cấu hình kích thước của bộ nhớ chia sẻ bằng `ftruncate`.
  - Ánh xạ bộ nhớ chia sẻ vào không gian địa chỉ của tiến trình với `mmap`.
- Ghi Dữ Liệu và Chờ Cập Nhật từ Process B:
  - Ghi chuỗi "Hello Process B" vào bộ nhớ chia sẻ.
  - Sử dụng vòng lặp `while` để chờ cho đến khi Process B cập nhật bộ nhớ chia sẻ.
  - In ra dữ liệu đã được cập nhật khi Process B đã thay đổi nó.
- Giải phóng Bộ Nhớ:
  - Sử dụng `munmap` để hủy ánh xạ đối tượng bộ nhớ chia sẻ và đóng file descriptor bằng `close`.

#### Process B - test\_shm\_B.c:

- Truy Cập Bộ Nhớ Chia Sẻ:
  - Mở đối tượng bộ nhớ chia sẻ đã tạo bởi Process A bằng `shm_open`.
  - Ánh xạ bộ nhớ chia sẻ vào không gian địa chỉ của tiến trình với `mmap`.

- Đọc và Cập Nhật Bộ Nhớ Chia Sẻ:
  - Đọc dữ liệu từ bộ nhớ chia sẻ và in ra màn hình.
  - Cập nhật bộ nhớ chia sẻ với chuỗi "Hello Process A".
  - Dừng 5 giây để Process A có thể nhận thấy sự thay đổi.
- Giải phóng Bộ Nhớ và Xóa Đối Tượng Bộ Nhớ Chia Sẻ:
  - Sử dụng munmap để hủy ánh xạ đối tượng bộ nhớ chia sẻ và đóng file descriptor bằng close.
  - Xóa đối tượng bộ nhớ chia sẻ bằng shm\_unlink.

➤ **Giải thích kết quả:**

🚦 Chạy Process A:

- In: "Waiting Process B update shared memory" liên tục mỗi giây (nếu Process B chưa cập nhật).
- Sau khi Process B cập nhật: "Memory updated: Hello Process A".

🚦 Chạy Process B:

- In: "Read shared memory: Hello Process B".
- In: "Shared memory updated: Hello Process A".

➔ Kết quả chính là quá trình trao đổi dữ liệu qua bộ nhớ chia sẻ giữa hai tiến trình. Process A khởi tạo, ghi dữ liệu, và chờ cập nhật từ Process B. Process B đọc dữ liệu, cập nhật bộ nhớ, và kết thúc bằng cách xóa đối tượng bộ nhớ chia sẻ.

## 2. Viết chương trình time.c thực hiện đo thời gian thực thi của một lệnh shell.

Chương trình sẽ được chạy với cú pháp `./time <command>` với `<command>` là lệnh shell muốn đo thời gian thực thi.

Ví dụ:

```
$ ./time ls
```

```
time.c
```

```
time
```

```
Thời gian thực thi: 0.25422
```

**Gợi ý:** Tiến trình cha gọi hàm `fork()` tạo ra tiến trình con rồi `wait()`. Tiến trình con gọi hàm `gettimeofday()` để lấy mốc thời gian trước khi thực thi lệnh shell, sau đó sử dụng hàm `execl()` để thực thi lệnh. Sau khi tiến trình con kết thúc, tiến

trình cha tiếp tục gọi hàm `gettimeofday()` một lần nữa để lấy mốc thời gian sau khi thực thi lệnh shell và tính toán.

➤ **Code:**

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/time.h>
4  #include <sys/wait.h>
5  #include <string.h>
6  #include <sys/mman.h>
7  #include <stdlib.h>
8  #include <fcntl.h>
9  #include <sys/shm.h>
10 #include <sys/stat.h>
11
12 const int SIZE = 4096;
13 const char *name = "ms";
14 int main(int argc, char *argv[]) {
15     int pid = fork();
16     struct timeval time;
17     int fd;
18     int *ptr;
19     fd = shm_open(name, O_CREAT | O_RDWR, 0666);
20     ftruncate(fd, SIZE);
21     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
22
23     if(pid > 0) {
24         wait(NULL);
25         gettimeofday(&time, NULL);
26         printf("Thoi gian thuc thi: %f", (double)(time.tv_usec - *ptr) / 1000000);
27         munmap(ptr, SIZE);
28         close(fd);
29         shm_unlink(name);
30         return 0;
31     } else if (pid == 0) {
32         gettimeofday(&time, NULL);
33         *ptr = time.tv_usec;
34         char cmd[] = "/bin/";
35         strcat(cmd, argv[1]);
36         execl(cmd, argv[1], argv[2], argv[3], NULL);
37         exit(0);
38     }
39 }
```

➤ **Kết quả sau khi thực thi:**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

> ./time ls
time time.c
Thoi gian thuc thi: 0.028335%

~/LAB3/LAB03
```

➤ **Giải thích code:**

- Khởi tạo các biến hằng để lưu trữ độ lớn của bộ nhớ chia sẻ và tên của bộ nhớ chia sẻ.
- Đề chương trình có thể nhận các tham số khi thực thi, tiến hành truyền các tham số vào hàm main. argc là tham số thể hiện số tham số được truyền vào còn argv là mảng các tham số.
- Tạo tiến trình mới bằng hàm fork()
- Tạo biến time để lưu trữ giá trị thời gian.
- Khởi tạo biến fd để tạo vùng nhớ chia sẻ
- Khởi tạo biến con trỏ ptr để ánh xạ đến vùng nhớ chia sẻ
- Khởi tạo vùng nhớ được chia sẻ với lời gọi hệ thống shm\_open. Các tham số được truyền vào bao gồm tên vùng nhớ chia sẻ, các chế độ của vùng nhớ.
- Cài đặt độ lớn của vùng nhớ bằng hàm ftruncate.
- Tiến hành ánh xạ vùng nhớ tới biến ptr.
- Hàm if dùng để xác định tiến trình, nếu pid > 0 thì chương trình cha sẽ thực thi còn nếu pid = 0 thì tiến trình con sẽ thực thi.
- **Tiến trình cha:**
  - Tiến trình cha sẽ đợi đến khi nào tiến trình con thực thi xong tiến trình cha mới được thực thi bởi vì hàm wait(NULL)
  - Lấy thời gian hiện tại bằng hàm gettimeofday(), thời gian này là thời gian sau khi tiến trình con đã thực thi xong.
  - In ra màn hình kết quả bằng hàm printf, kết quả là thời gian sau thi tiến trình con thực thi xong trừ đi thời gian khi tiến trình con bắt đầu thực thi.
  - Đóng các vùng nhớ chia sẻ
- **Tiến trình con:**
  - Lấy thời gian hiện tại bằng hàm gettimeofday(), thời gian này là thời gian lúc tiến trình con bắt đầu thực thi.
  - Lưu thời gian vừa lấy được vào vùng nhớ chia sẻ để chia sẻ với tiến trình cha.
  - Tạo biến cmd để lưu tên lệnh shell cần thực thi.

- Sử dụng hàm `strcat` để nối chuỗi `cmd` với lệnh shell cần thực thi được truyền vào thông qua hàm `main`.
- Sử dụng hàm `execl` để thực thi lệnh shell.

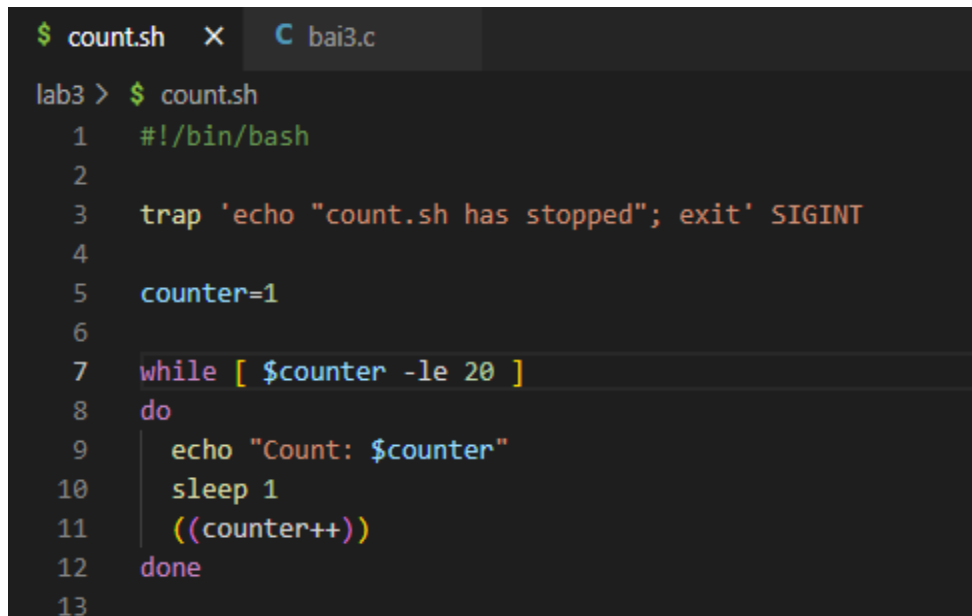
➤ **Giải thích kết quả thực thi:**

- Kết quả thực thi với tham số được truyền vào là lệnh `ls`
- Chương trình đã thực thi lệnh `ls` và liệt kê ra những file có trong thư mục là file `time` và `time.c`
- Thời gian thực thi lệnh `ls` được đo và in ra màn hình

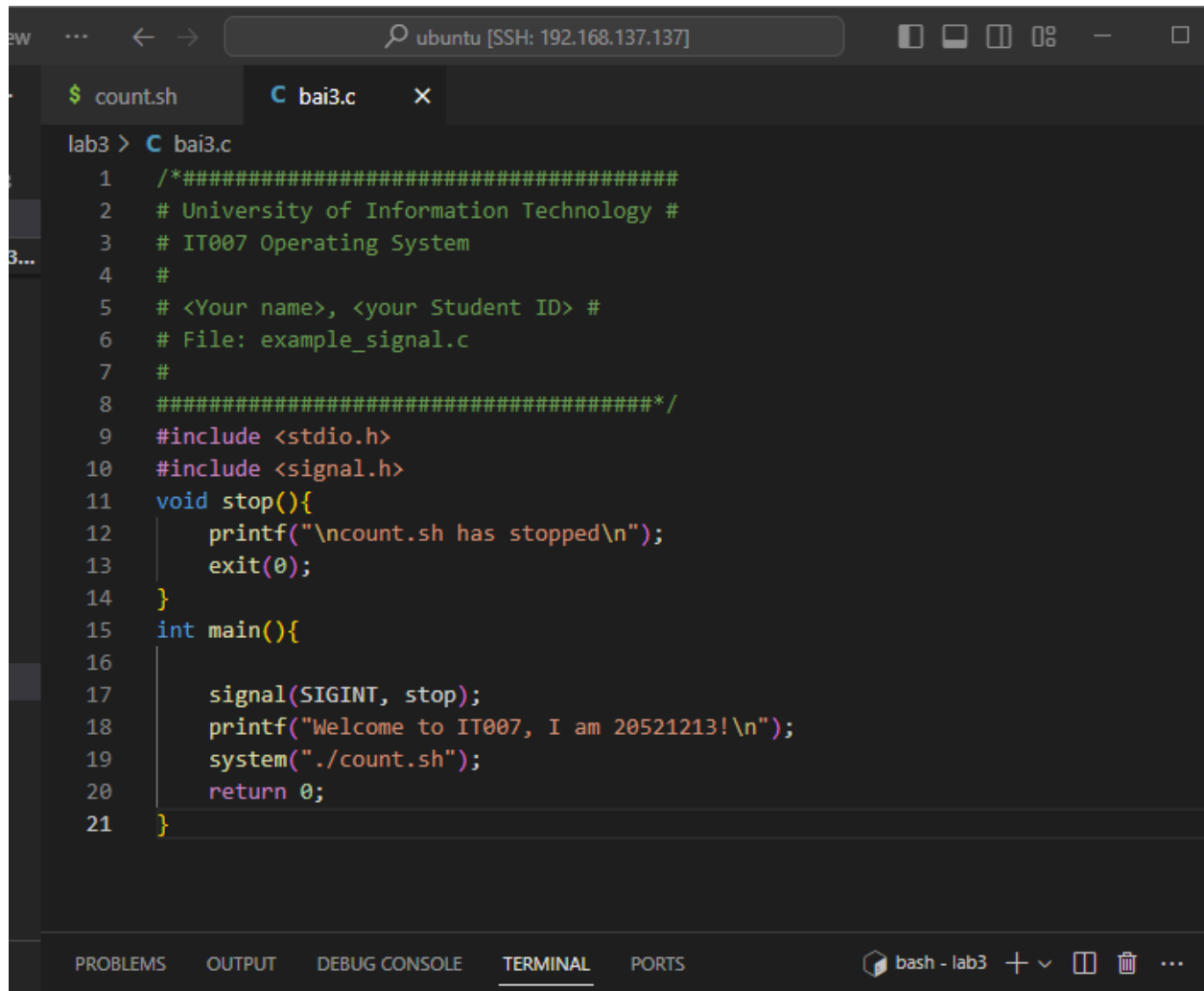
**3. Viết một chương trình làm bốn công việc sau theo thứ tự:**

- In ra dòng chữ: “Welcome to IT007, I am <your\_Student\_ID>!”
- Thực thi file script `count.sh` với số lần đếm là 120
- Trước khi `count.sh` đếm đến 120, bấm **CTRL+C** để dừng tiến trình này
- Khi người dùng nhấn **CTRL+C** thì in ra dòng chữ: “count.sh has stoppped”

➤ **Code:**



```
$ count.sh X C bai3.c
lab3 > $ count.sh
1  #!/bin/bash
2
3  trap 'echo "count.sh has stoppped"; exit' SIGINT
4
5  counter=1
6
7  while [ $counter -le 20 ]
8  do
9      echo "Count: $counter"
10     sleep 1
11     ((counter++))
12 done
13
```



```
lab3 > C bai3.c
1  /*#####
2  # University of Information Technology #
3  # IT007 Operating System
4  #
5  # <Your name>, <your Student ID> #
6  # File: example_signal.c
7  #
8  #####*/
9  #include <stdio.h>
10 #include <signal.h>
11 void stop(){
12     printf("\ncount.sh has stopped\n");
13     exit(0);
14 }
15 int main(){
16
17     signal(SIGINT, stop);
18     printf("Welcome to IT007, I am 20521213!\n");
19     system("./count.sh");
20     return 0;
21 }
```

➤ **Kết quả sau khi thực thi:**

```
● ubuntu@dung-20521213:~/lab3$ ./bai3
Welcome to IT007, I am 20521213!
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
Count: 6
Count: 7
Count: 8
Count: 9
Count: 10
Count: 11
Count: 12
Count: 13
^Ccount.sh has stopped
○ ubuntu@dung-20521213:~/lab3$
```

➤ **Giải thích code:**

- count.sh:
  - Dòng số 3: *trap 'echo "count.sh has stopped"; exit' SIGINT*, ở đây *trap* là 1 lệnh được sử dụng để bắt hay xử lý tín hiệu khi chương trình hoặc tiến trình đang chạy. *echo "count.sh has stopped"; exit* là hành động sẽ được thực hiện khi nhận được tín hiệu (ở đây là *SIGINT*), khi nhận được tín hiệu sẽ in ra thông báo *count.sh has stopped*.
  - Dòng 7 – 12: Thực hiện vòng lặp *while* với điều kiện lặp là biến *\$counter* phải bé hơn số chỉ định (trong đề là 120 nhưng trong quá trình làm thì mình để số bé cho dễ kiểm soát) sau khi in ra màn hình giá trị hiện tại thì chương trình tạm dừng 1 giây để đúng với mục đích là đếm sau đó biến *\$counter* tăng thêm 1 đơn vị để tiếp tục vòng lặp tiếp theo.
- Bai3.c:
  - Dòng 9 - 10: include các thư viện cần thiết.
  - Dòng 11 - 14: Hàm *stop()* được tạo ra với mục đích in ra thông báo "count.sh has stopped" và kết thúc chương trình với *exit(0)*.
  - Dòng 15 - 21: Trong hàm *main()* tại dòng số 17 *signal(SIGINT, stop)* được viết ra để xử lý tín hiệu (ở đây là *SIGINT*) cụ thể là khi có tín hiệu thì hàm *stop()* sẽ được gọi để xử lý.
  - Dòng 18: được viết ra để in thông báo "welcome to IT007, I am 20521213!".

- Dòng 19: `system("./count.sh")` được viết ra với mục đích để thực thi 1 chương trình bên ngoài ( `count.sh` ) bên ngoài chương trình đang chạy ( `bai3.c` ) thông qua lời gọi hệ thống.
- Tiến hành cấp quyền cho các file với lệnh `chmod +x (tên file)`.

➤ **Giải thích kết quả:**

- Khi tiến hành thực thi chương trình ta ngay lập tức nhận được thông báo “welcome to IT007, I am 20521213!”. Vậy là đã hoàn thành mục tiêu đầu tiên của đề bài.
- Tiếp theo câu lệnh `system("./count.sh")` trong hàm `main()` của `bai3.c` sẽ gọi chương trình `count.sh` để thực thi và ta có thể thấy hàng loạt các thông báo Count: số được in ra.
- Có thể thấy trong hình ảnh minh họa trên chương trình đã được cho dừng với lệnh `Ctrl+C` hay `^C` chương trình đếm `count.sh` không còn in ra các thông báo Count: số.
- Với yêu cầu cuối cùng là in ra dòng chữ “count.sh has stoppped” ta cũng có thể thấy dễ dàng ở trong hình ảnh minh họa trên sau khi chương trình được cho dừng với tổ hợp phím `Ctrl+C`.

**4. Viết chương trình mô phỏng bài toán Producer - Consumer như sau:**

- Sử dụng kỹ thuật `shared-memory` để tạo một `bounded-buffer` có độ lớn là 10 bytes.
- Tiến trình cha đóng vai trò là `Producer`, tạo một số ngẫu nhiên trong khoảng `[10, 20]` và ghi dữ liệu vào `buffer`
- Tiến trình con đóng vai trò là `Consumer` đọc dữ liệu từ `buffer`, in ra màn hình và tính tổng
- Khi tổng lớn hơn 100 thì cả 2 dừng lại

➤ **Cách làm ( code và giải thích code):**

- Khởi tạo `shared memory` (name tùy ý, size = 10) như hình:



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <unistd.h>
#include <sys/mman.h>

int main()
{
    // Khởi tạo shared memory theo yêu cầu
    const char* name = "Prod-Cons";
    const int SIZE = 10;
    int fd = shm_open(name, O_CREAT | O_RDWR, 0666);
    int *ptr = (int*) mmap(NULL, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    ftruncate(fd, SIZE);
```

- Dùng lệnh fork() để tạo tiến trình cha con. Trong đó tiến trình cha đóng vai trò Producer, tiến trình con là Consumer:

```
// Tạo tiến trình cha, con
int pid = fork();
```

- Các bước thực hiện như sau:
  - Tiến trình cha:

```
// Tiến trình cha thực hiện ghi số ngẫu nhiên vào shared memory
if (pid > 0) {
    int sum = 0; // Khởi tạo tổng sum
    int i = 0; // Khởi tạo biến đếm kiểm tra kích thước shared memory
    while (1)
    {
        // Kiểm tra điều kiện không được vượt quá memory size
        if (i >= SIZE) break;
        int value = rand()%11 + 10; // Tạo số ngẫu nhiên trong khoảng [10; 20]
        // Ghi dữ liệu vào vùng nhớ
        *ptr = value;
        printf("Producer writes: %d\n", *ptr);
        // Tính tổng, kiểm tra điều kiện
        sum += value;
        if (sum > 100) break;
        i++;
        sleep(1);
    }
}
```

- Tiến trình con:

```
// Tiến trình con thực hiện việc đọc dữ liệu từ shared memory
else {
    int sum = 0;
    int i = 0;
    while (1){
        // Kiểm tra điều kiện kích thước memory
        if (i >= SIZE) {
            printf("Vuot qua gioi han vung nho");
            break;
        }
        // Đọc dữ liệu shared memory
        printf("Consumer reads: %d\n", *ptr);
        sum += *ptr;
        // Kiểm tra điều kiện tổng
        if (sum > 100) {
            printf("Sum = %d\n", sum);
            printf("Tong da vuot qua 100 !\nKet thuc chuong trinh\n");
            break;
        }
        i++;
        sleep(1);
    }
    // Giải phóng shared memory
    munmap((void *)ptr, SIZE);
    exit(0);
}
```

- Giải phóng shared memory sau khi kết thúc chương trình:

```
// Giải phóng shared memory, đóng file vùng nhớ và ngắt kết nối
munmap((void *)ptr, SIZE);
close(fd);
shm_unlink(name);
return 0;
}
```

➤ **Kết quả sau khi thực thi:**

```
● quocvinh_22521682@Ubuntu:~/Lab_03$ gcc procon.c -o procon
● quocvinh_22521682@Ubuntu:~/Lab_03$ ./procon
Producer writes: 16
Consumer reads: 16
Producer writes: 20
Consumer reads: 20
Producer writes: 16
Consumer reads: 16
Producer writes: 12
Consumer reads: 12
Producer writes: 11
Consumer reads: 11
Producer writes: 14
Consumer reads: 14
Producer writes: 10
Consumer reads: 10
Producer writes: 16
Consumer reads: 16
Sum = 115
Tong da vuot qua 100 !
Ket thuc chuong trinh
○ quocvinh_22521682@Ubuntu:~/Lab_03$
```

➤ **Giải thích kết quả:**

- Tiến trình cha và tiến trình con sẽ luân phiên truy cập vào shared memory:
  - Tiến trình cha sẽ lần lượt ghi các số ngẫu nhiên trong khoảng [10; 20] vào vùng nhớ, đồng thời tính tổng sum (chạy ngầm) để xét điều kiện ngừng chương trình.
  - Tiến trình con sẽ lần lượt đọc các số trong vùng nhớ và tính tổng sum (Tổng sum của tiến trình cha và con độc lập với nhau).
  - Đến khi tổng sum > 100 (trên hình là sum = 115) => Tiến trình cha ngừng ghi, tiến trình con cũng ngừng đọc, đồng thời giải phóng vùng nhớ => Kết thúc chương trình

## 2.6. BÀI TẬP ÔN TẬP

1. Phỏng đoán Collatz xem xét chuyện gì sẽ xảy ra nếu ta lấy một số nguyên dương bất kỳ và áp dụng theo thuật toán sau đây:

$$n = \begin{cases} n/2 & \text{nếu } n \text{ là số chẵn} \\ 3 * n + 1 & \text{nếu } n \text{ là số lẻ} \end{cases}$$

Phỏng đoán phát biểu rằng khi thuật toán này được áp dụng liên tục, tất cả số nguyên dương đều sẽ tiến đến 1. Ví dụ, với  $n = 35$ , ta sẽ có chuỗi kết quả như sau:

35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

Viết chương trình C sử dụng hàm fork() để tạo ra chuỗi này trong tiến trình con. Số bắt đầu sẽ được truyền từ dòng lệnh. Ví dụ lệnh thực thi ./collatz 8 sẽ chạy thuật toán trên  $n = 8$  và chuỗi kết quả sẽ ra là 8, 4, 2, 1. Khi thực hiện, tiến trình cha và tiến trình con chia sẻ một buffer, sử dụng phương pháp bộ nhớ chia sẻ, hãy tính toán chuỗi trên tiến trình con, ghi kết quả vào buffer và dùng tiến trình cha để in kết quả ra màn hình. Lưu ý, hãy nhớ thực hiện các thao tác để kiểm tra input là số nguyên dương.

➤ Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <unistd.h>

void collatz(int n, int *sharedBuffer) {
    sharedBuffer[0] = n;
    int i = 1;
    while (n != 1) {
        if (n % 2 == 0)
            n = n / 2;
        else
            n = 3 * n + 1;

        // ghi kết quả vào buffer
        sharedBuffer[i] = n;
    }
}
```

```
        i++;
    }
    // them so 0 vao cuoi buffer de- dat flag
    sharedBuffer[i] = 0;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Tham so khong hop le!\n");
        return 1;
    }

    int startNumber = atoi(argv[1]);

    if (startNumber <= 0) {
        printf("Nhap mot so nguyen duong!\n");
        return 1;
    }

    // tao mot bo nho chia se
    const char *name = "/collatz_shared_buffer";
    const int SIZE = 4096;
    int shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);
    ftruncate(shm_fd, SIZE);
    int *sharedBuffer = (int *)mmap(0, SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);

    // dung lenh fork()
    pid_t pid = fork();

    if (pid < 0) {
        printf("Fork failed.\n");
        return 1;
    } else if (pid == 0) {
        // tien trinh con tinh toan Collatz va ghi vao buffer
        collatz(startNumber, sharedBuffer);
        exit(0);
    } else {
        // tien trinh cha doi tien trinh con ghi xong
        wait(NULL);

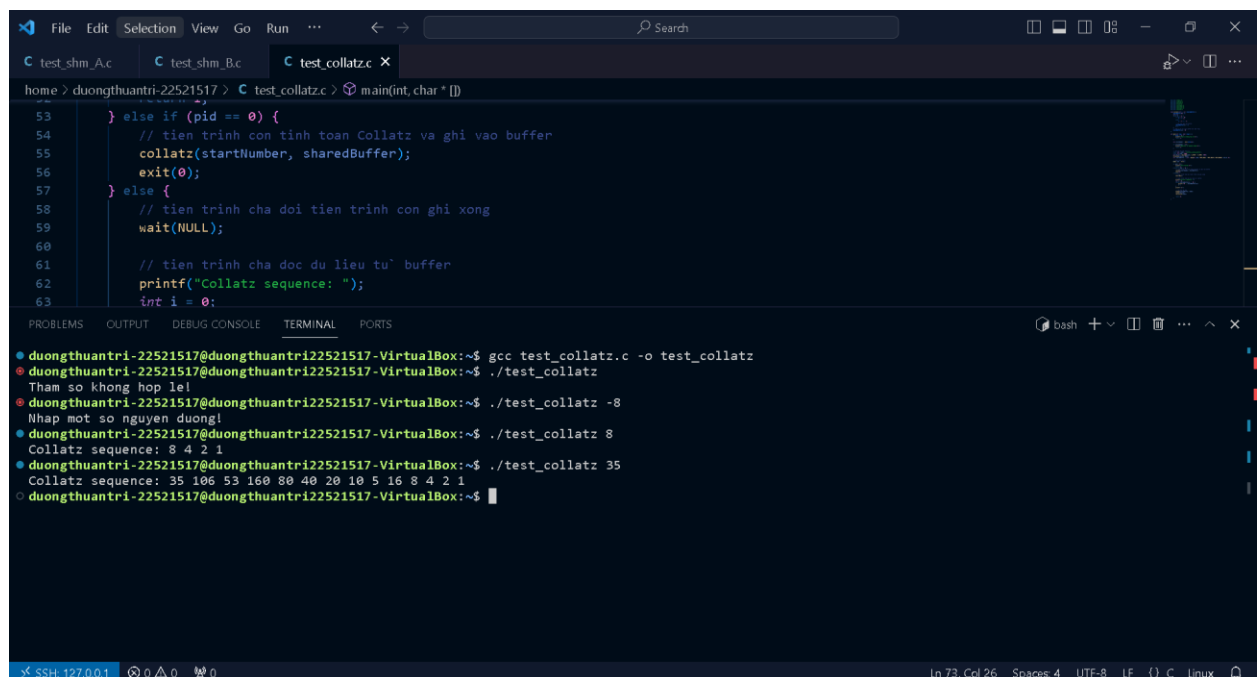
        // tien trinh cha doc du lieu tu` buffer
    }
}
```

```
printf("Collatz sequence: ");
int i = 0;
while (sharedBuffer[i] != 0) {
    printf("%d ", sharedBuffer[i]);
    i++;
}
printf("\n");

// thu hoi bo nho
munmap(sharedBuffer, SIZE);
close(shm_fd);
shm_unlink(name);

return 0;
}
```

### ➤ Thực thi chương trình:



```
File Edit Selection View Go Run ... Search
C test_shm_A.c C test_shm_B.c C test_collatz.c X
home > duongthuantri-22521517 > C test_collatz.c > main(int, char * [])
53 } else if (pid == 0) {
54     // tien trinh con tinh toan Collatz va ghi vao buffer
55     collatz(startNumber, sharedBuffer);
56     exit(0);
57 } else {
58     // tien trinh cha doi tien trinh con ghi xong
59     wait(NULL);
60
61     // tien trinh cha doc du lieu tu' buffer
62     printf("Collatz sequence: ");
63     int i = 0;
64
65     while (sharedBuffer[i] != 0) {
66         printf("%d ", sharedBuffer[i]);
67         i++;
68     }
69     printf("\n");
70
71     // thu hoi bo nho
72     munmap(sharedBuffer, SIZE);
73     close(shm_fd);
74     shm_unlink(name);
75
76     return 0;
77 }
78
79 #endif
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
duongthuantri-22521517@duongthuantri22521517-VirtualBox:~$ gcc test_collatz.c -o test_collatz
duongthuantri-22521517@duongthuantri22521517-VirtualBox:~$ ./test_collatz
Tham so khong hop le!
duongthuantri-22521517@duongthuantri22521517-VirtualBox:~$ ./test_collatz -8
Nhap mot so nguyen duong!
duongthuantri-22521517@duongthuantri22521517-VirtualBox:~$ ./test_collatz 8
Collatz sequence: 8 4 2 1
duongthuantri-22521517@duongthuantri22521517-VirtualBox:~$ ./test_collatz 35
Collatz sequence: 35 106 53 160 80 40 20 10 5 16 8 4 2 1
duongthuantri-22521517@duongthuantri22521517-VirtualBox:~$
```

SSH: 127.0.0.1 0 0 0 0 Ln 73, Col 26 Spaces 4 UTF-8 LF {} C Linux

### ➤ Giải thích code:

- Hàm collatz(int n, int sharedBuffer):
  - Hàm này nhận vào các tham số là một số nguyên n và một con trỏ tới bộ đệm chia sẻ sharedBuffer.

2. Trong vòng lặp while, ta xét  $n$  là số chẵn, thực hiện phép toán  $n = n / 2$ , ngược lại thì  $n = 3 * n + 1$  ( dựa theo thuật toán đề bài cung cấp), thực hiện cho đến khi  $n=1$  thì kết thúc vòng lặp
3. Kết quả của mỗi lần tính toán được ghi vào bộ đệm sharedBuffer:  

```
sharedBuffer[i] = n;
```
4. Chuỗi kết quả kết thúc bằng số 0 để đánh dấu kết thúc của chuỗi (đặt flag).
- Hàm main(int argc, char argv[]):
  1. Ta kiểm tra số lượng đối số dòng lệnh. Nếu không đúng, thông báo lỗi và kết thúc chương trình.
  2. Kiểm tra xem đối số truyền vào có phải là số nguyên dương không. Nếu không, thông báo lỗi và kết thúc chương trình.
  3. Tạo một kích thước bộ nhớ chia sẻ SIZE và tạo một file descriptor cho bộ nhớ chia sẻ (shm\_fd) sử dụng shm\_open().
  4. Cấp phát bộ nhớ chia sẻ sử dụng mmap().
  5. Sử dụng fork() để tạo một tiến trình con.
  6. Trong tiến trình con, gọi hàm collatz() để tính toán chuỗi Collatz và ghi vào bộ nhớ chia sẻ.
  7. Trong tiến trình cha, đợi cho tiến trình con kết thúc, sau đó đọc dữ liệu từ bộ nhớ chia sẻ và in ra màn hình:

```
while (sharedBuffer[i] != 0) {  
    printf("%d ", sharedBuffer[i]);  
    i++;  
}
```

8. Cuối cùng, ta giải phóng bộ nhớ của bộ nhớ chia sẻ.

```
munmap(sharedBuffer, SIZE);  
close(shm_fd);  
shm_unlink(name);
```