

Họ và tên thành viên nhóm:

1. Hoàng Hồ Quốc Bảo – 22520102
2. Nguyễn Nguyên Ngọc Anh – 22520058
3. Mai Thanh Bách – 22520090
4. Hồ Tiến Vũ Bình – 22520129

Lớp: IT007.O11.1

## HỆ ĐIỀU HÀNH BÁO CÁO LAB 3

### CHECKLIST

#### 3.5. BÀI TẬP THỰC HÀNH

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

#### 3.6. BÀI TẬP ÔN TẬP

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

**Tư chấm điểm:** 10

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Trần Hoàng Lộc.

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

**<MSSV>\_LAB3.pdf**

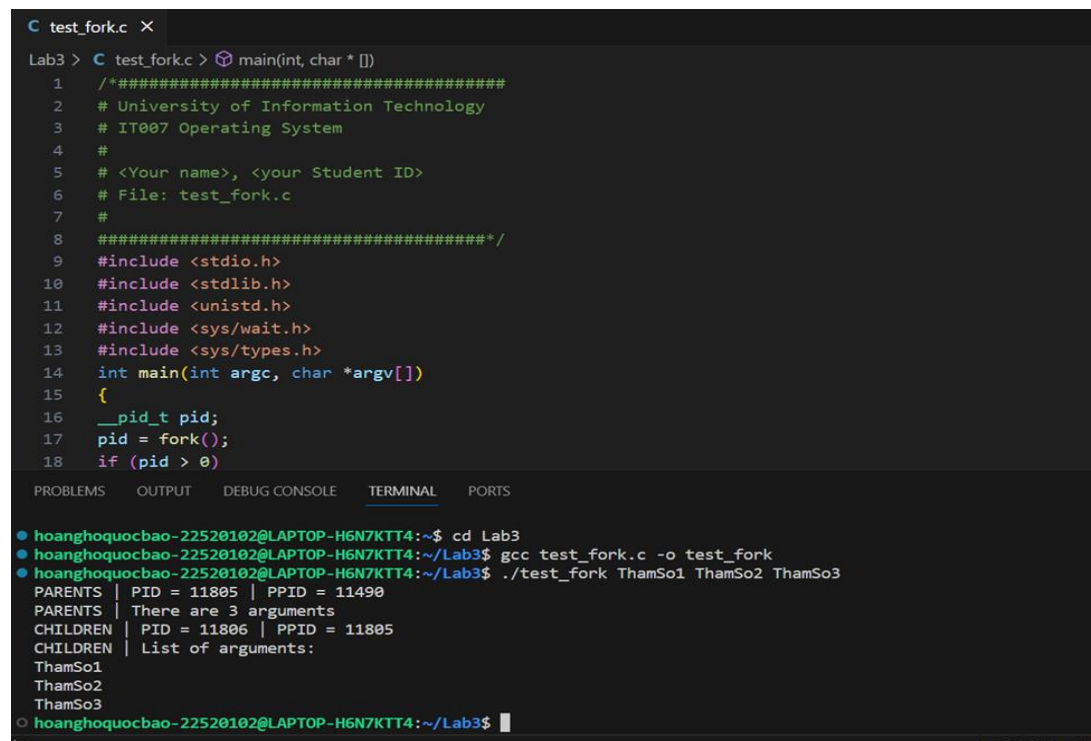
## 2.5. BÀI TẬP THỰC HÀNH

### 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?

Trả lời...

- Ví dụ 3-1:

Ảnh minh chứng :



```
Lab3 > C test_fork.c X
Lab3 > C test_fork.c > main(int, char * [])
1  /*#####*/
2  # University of Information Technology
3  # IT007 Operating System
4  #
5  # <Your name>, <your Student ID>
6  # File: test_fork.c
7  #
8  /*#####*/
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <unistd.h>
12 #include <sys/wait.h>
13 #include <sys/types.h>
14 int main(int argc, char *argv[])
15 {
16     __pid_t pid;
17     pid = fork();
18     if (pid > 0)
19         return 0;
20     if (pid == 0)
21     {
22         if (argc == 4)
23         {
24             printf("Parent: %s\n", argv[0]);
25             printf("Child: %s\n", argv[1]);
26             printf("List of arguments:\n");
27             for (int i = 1; i < argc; i++)
28                 printf("%s\n", argv[i]);
29         }
30         return 0;
31     }
32     return 0;
33 }
```

hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~\$ cd Lab3  
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3\$ gcc test\_fork.c -o test\_fork  
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3\$ ./test\_fork ThamSo1 ThamSo2 ThamSo3  
PARENTS | PID = 11805 | PPID = 11490  
PARENTS | There are 3 arguments  
CHILDREN | PID = 11806 | PPID = 11805  
CHILDREN | List of arguments:  
ThamSo1  
ThamSo2  
ThamSo3  
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3\$

#### - Giải thích code:

Code	Giải thích
#include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <sys/wait.h> #include <sys/types.h>	Những dòng này là để bao gồm các thư viện C cần thiết như stdio.h, stdlib.h, unistd.h, sys/wait.h và sys/types.h. Những thư viện này cung cấp các hàm và định nghĩa được sử dụng trong chương trình.

<pre>int main(int argc, char *argv[]) {</pre>	<p><b>Bắt đầu của hàm main.</b> Nó lấy các đối số argc (số lượng đối số) và argv (mảng đối số) để xử lý các đối số dòng lệnh.</p>
<pre>__pid_t pid; pid = fork();</pre>	<p>Một biến pid kiểu __pid_t được khai báo để lưu trữ ID của quá trình. Hàm fork() được sử dụng để tạo một tiến trình mới. Sau lời gọi này, sẽ có hai tiến trình: tiến trình cha và tiến trình con, và cả hai tiếp tục thực thi từ điểm này. fork() trả về các giá trị khác nhau cho tiến trình cha và con. Trong tiến trình cha, pid chứa ID của tiến trình con, trong khi trong tiến trình con, pid chứa giá trị 0.</p>
<pre>if (pid &gt; 0) {     printf("PARENTS   PID = %ld   PPID = %ld\n",            (long)getpid(), (long)getppid());     if (argc &gt; 2)         printf("PARENTS   There are %d arguments\n",                argc - 1);     wait(NULL); }</pre>	<p>Trong tiến trình cha (được xác định bởi pid &gt; 0), nó in ra ID tiến trình của mình (PID) bằng getpid() và ID của tiến trình cha (PPID) bằng getppid(). Nếu số lượng đối số được truyền vào lớn hơn 2 (loại bỏ tên chương trình), nó in ra một thông báo chỉ ra số lượng đối số. wait(NULL) làm tạm ngừng thực thi của tiến trình cha cho đến khi tiến trình con kết thúc.</p>
<pre>if (pid == 0) {</pre>	<p>Trong tiến trình con (được xác định bởi pid == 0), nó in ra ID tiến trình của mình, ID của tiến trình cha, và hiển thị danh sách</p>

<pre>printf("CHILDREN   PID = %ld   PPID = %ld\n",       (long)getpid(), (long)getppid()); printf("CHILDREN   List of arguments: \n"); for (int i = 1; i &lt; argc; i++) {     printf("%s\n", argv[i]); }</pre>	các đối số được truyền vào chương trình bằng một vòng lặp duyệt qua mảng argv.
<pre>exit(0);</pre>	Dòng này được sử dụng để kết thúc tiến trình và 0 chỉ ra việc kết thúc thành công.

Giải thích kết quả:

- Chương trình này sử dụng hàm fork() để tạo một tiến trình con từ tiến trình gốc (tiền trình cha).
- Tiến trình cha và tiến trình con sẽ chia sẻ cùng một đoạn mã của chương trình, bắt đầu từ hàm main().
- Khi chạy, sau lời gọi fork(), tiến trình cha sẽ nhận được một giá trị pid là ID của tiến trình con, trong khi tiến trình con sẽ nhận giá trị 0.
- Dựa trên giá trị của pid, chương trình in ra các thông tin khác nhau cho mỗi tiến trình (cha hoặc con).
- Ví dụ: Giả sử bạn chạy chương trình với tên là test\_fork và truyền vào các đối số như sau: ./test\_fork arg1 arg2.

Nếu bạn có ít nhất hai đối số (bao gồm tên chương trình), tiến trình cha sẽ in ra PID và PPID của nó cùng với số lượng đối số truyền vào.

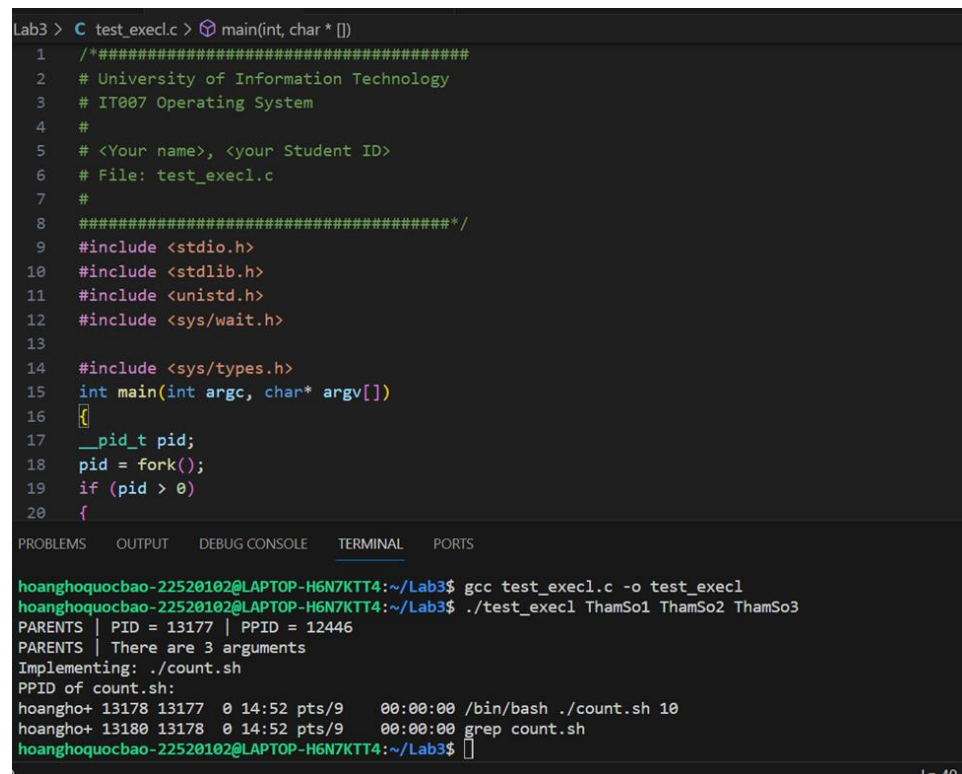
Tiến trình con sẽ in ra PID và PPID của nó, và sau đó in ra danh sách các đối số truyền vào chương trình. Trong trường hợp này, đối số đầu tiên sẽ là "arg1" và đối số thứ hai sẽ là "arg2".

Sau khi tiến trình con hoàn thành việc in thông tin, cả tiến trình cha và tiến trình con đều sẽ kết thúc bằng lệnh `exit(0)`.

- Kết quả cụ thể sẽ phụ thuộc vào cách bạn chạy chương trình và truyền vào những đối số nào. Nếu không có đối số nào được truyền vào, chương trình vẫn chạy nhưng không có thông tin đối số nào được in ra từ tiến trình con.

- Ví dụ 3-2:

Ảnh minh chứng:



```
Lab3 > 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>
12 #include <sys/wait.h>
13
14 #include <sys/types.h>
15 int main(int argc, char* argv[])
16 {
17     __pid_t pid;
18     pid = fork();
19     if (pid > 0)
20     {
21         printf("PARENTS | PID = %d | PPID = %d\n", getpid(), getppid());
22         printf("PARENTS | There are %d arguments\n", argc);
23         printf("Implementing: ./count.sh\n");
24         printf("PPID of count.sh: %d\n", getppid());
25         system("./count.sh 10");
26         printf("hoangho+ %d %d %d %d %d %d %d %d %d %d\n", getpid(), getppid(),
27             argv[1], argv[2], argv[3], argv[4], argv[5], argv[6], argv[7], argv[8]);
28         exit(0);
29     }
30     else if (pid == 0)
31     {
32         printf("CHILD | PID = %d | PPID = %d\n", getpid(), getppid());
33         printf("CHILD | Arguments: %s %s %s\n", argv[1], argv[2], argv[3]);
34         exit(0);
35     }
36     else
37     {
38         printf("Error: fork() failed\n");
39         exit(1);
40     }
41 }
```

hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3\$ gcc test\_execl.c -o test\_execl

hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3\$ ./test\_execl ThamSo1 ThamSo2 ThamSo3

PARENTS | PID = 13177 | PPID = 12446

PARENTS | There are 3 arguments

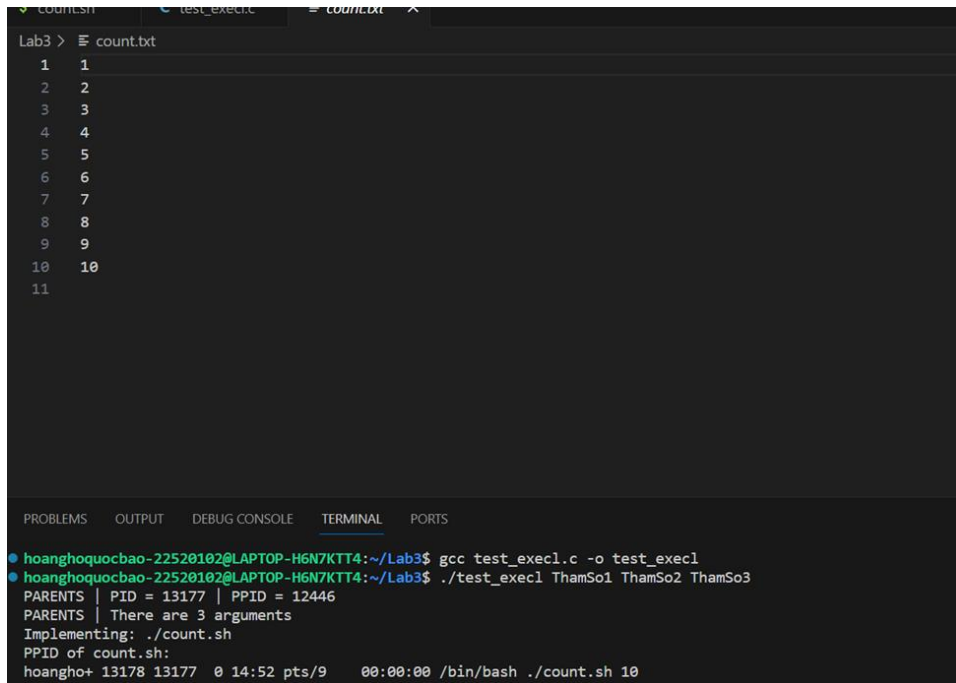
Implementing: ./count.sh

PPID of count.sh:

hoangho+ 13178 13177 0 14:52 pts/9 00:00:00 /bin/bash ./count.sh 10

hoangho+ 13180 13178 0 14:52 pts/9 00:00:00 grep count.sh

hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3\$



```
Lab3 > count.txt
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
10 10
11

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3$ gcc test_execl.c -o test_execl
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3$ ./test_execl ThamSo1 ThamSo2 ThamSo3
PARENTS | PID = 13177 | PPID = 12446
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
hoangho+ 13178 13177 0 14:52 pts/9 00:00:00 /bin/bash ./count.sh 10
```

#### - Giải thích code:

Code	Giải thích
<pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; #include &lt;sys/wait.h&gt; #include &lt;sys/types.h&gt;</pre>	<p>Đây là các khai báo thư viện được sử dụng trong chương trình, bao gồm các thư viện cần thiết cho việc làm việc với hệ thống, quản lý tiến trình, và tiêu chuẩn input/output</p>
<pre>int main(int argc, char* argv[]) {     __pid_t pid;     pid = fork();     if (pid &gt; 0)     {         printf("PARENTS   PID = %ld   PPID = %ld\n",             (long)getpid(), (long)getppid());</pre>	<p>Hàm main() bắt đầu chương trình. Đầu tiên, nó khai báo một biến pid kiểu <code>__pid_t</code> để lưu trữ PID (Process ID) của tiến trình.</p> <p>Sau đó, chương trình gọi hàm <code>fork()</code> để tạo một tiến trình con. Nếu giá trị của pid trả về lớn hơn 0, đây là quá trình cha. Chương trình in ra thông tin về quá trình cha, bao gồm PID và PPID (Parent</p>

<pre> if (argc &gt; 2)     printf("PARENTS   There are %d arguments\n",         argc - 1);     wait(NULL); } </pre>	<p>Process ID). Nếu số lượng đối số đầu vào (arguments) lớn hơn 2, nó sẽ in ra số lượng đối số đó.</p>
<pre> if (pid == 0) {     execl("./count.sh", "./count.sh", "10", NULL);     printf("CHILDREN   PID = %ld   PPID = %ld\n",         (long)getpid(), (long)getppid());     printf("CHILDREN   List of arguments: \n");     for (int i = 1; i &lt; argc; i++)     {         printf("%s\n", argv[i]);     } } exit(0); } </pre>	<p>Nếu giá trị của pid là 0, đây là quá trình con. Quá trình con sẽ thực hiện execl() để thực thi tập lệnh shell có tên là count.sh với các đối số là ./count.sh và "10". Nếu lệnh execl() thành công, các lệnh printf() phía sau sẽ không được thực thi.</p> <p>Nếu lệnh execl() không thành công (ví dụ: không thể tìm thấy tập lệnh count.sh), chương trình sẽ in ra thông tin về quá trình con và liệt kê các đối số đầu vào từ dòng lệnh khi chạy chương trình.</p> <p>Cuối cùng, exit(0); được gọi để thoát khỏi chương trình với mã thoát là 0 (tức là chương trình thoát thành công).</p>

Giải thích kết quả:

- Chương trình tạo một tiến trình con bằng hàm fork().
- Nếu fork() thành công, tiến trình cha và con chia sẻ code của chương trình cha.
- Tiến trình cha in thông tin về chính nó và đợi tiến trình con hoàn thành bằng hàm wait(NULL).
- Tiến trình con sẽ gọi execl() để thực thi một tập lệnh shell có tên là count.sh với hai đối số: ./count.sh và "10".



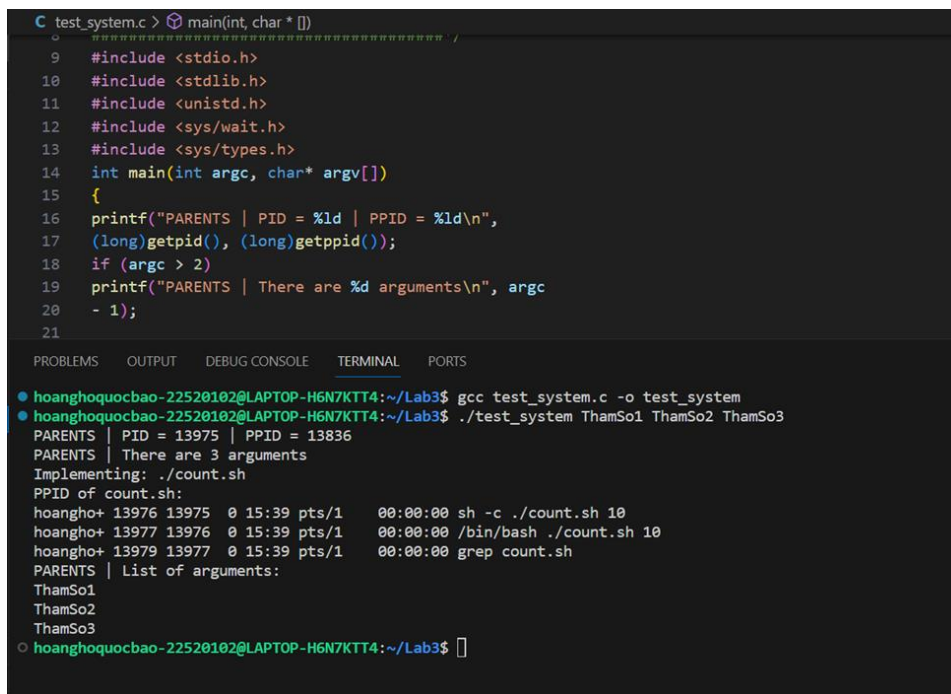
Nếu tập lệnh count.sh tồn tại và thực thi thành công, chương trình sẽ không in ra thông tin nào từ tiến trình con vì sau lệnh `execl()`, nó sẽ thực hiện chạy tập lệnh shell đó và không thực hiện bất kỳ lệnh `printf()` nào sau đó.

Tuy nhiên, nếu tập lệnh count.sh không tồn tại hoặc không thể thực thi, chương trình sẽ tiếp tục thực hiện các dòng lệnh trong khối `if (pid == 0)`. Nó sẽ in ra thông tin của tiến trình con và liệt kê các đối số đầu vào từ dòng lệnh khi chạy chương trình.

- Kết quả cụ thể sẽ phụ thuộc vào việc tập lệnh count.sh có tồn tại và thực thi thành công hay không. Nếu tập lệnh đó được thực thi thành công, thông tin từ `printf()` trong khối `if (pid == 0)` sẽ không xuất hiện và chương trình sẽ không in ra gì từ tiến trình con. Nếu không, thông tin về tiến trình con và các đối số đầu vào sẽ được in ra.

- Ví dụ 3-3:

Ảnh minh chứng:

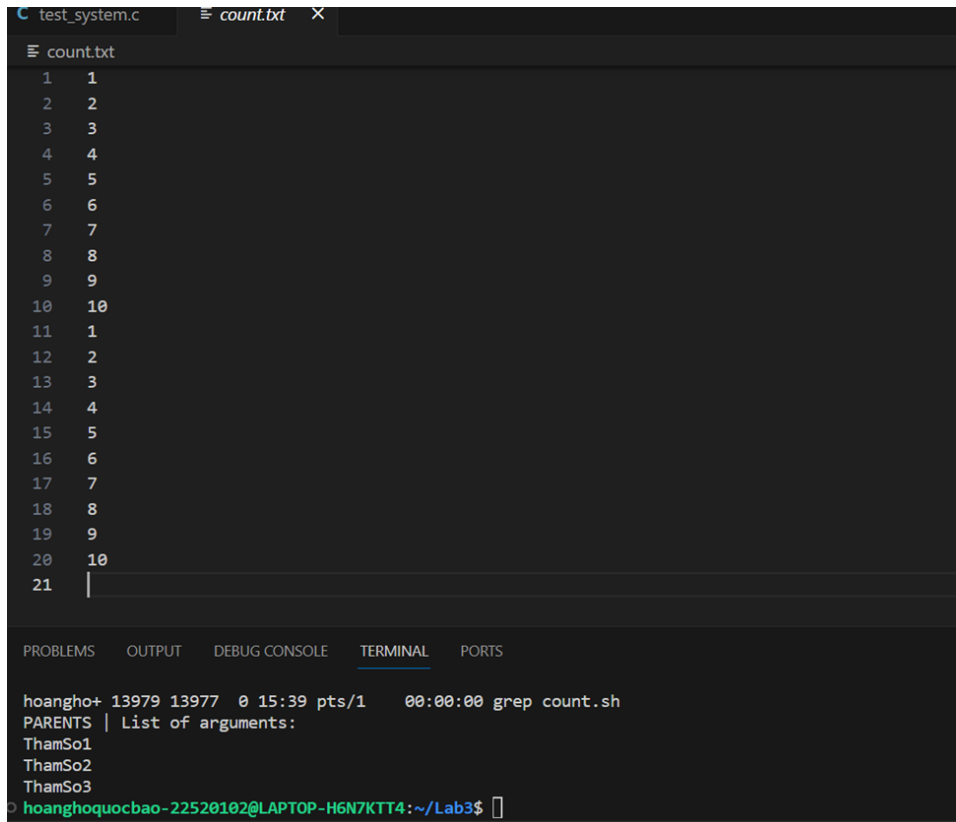


The screenshot shows a C program named `test_system.c` and its execution output in a terminal window. The program is designed to demonstrate process creation and argument passing.

```
C test_system.c > main(int, char * [])
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <unistd.h>
12 #include <sys/wait.h>
13 #include <sys/types.h>
14 int main(int argc, char* argv[])
15 {
16     printf("PARENTS | PID = %ld | PPID = %ld\n",
17         (long)getpid(), (long)getppid());
18     if (argc > 2)
19         printf("PARENTS | There are %d arguments\n", argc
20             - 1);
21 }
```

The terminal output shows the program being compiled and executed with three arguments: `ThamSo1 ThamSo2 ThamSo3`. The output displays the parent and child process information, the number of arguments, and the arguments themselves.

```
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3$ gcc test_system.c -o test_system
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3$ ./test_system ThamSo1 ThamSo2 ThamSo3
PARENTS | PID = 13975 | PPID = 13836
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
hoangho+ 13976 13975  0 15:39 pts/1    00:00:00 sh -c ./count.sh 10
hoangho+ 13977 13976  0 15:39 pts/1    00:00:00 /bin/bash ./count.sh 10
hoangho+ 13979 13977  0 15:39 pts/1    00:00:00 grep count.sh
PARENTS | List of arguments:
ThamSo1
ThamSo2
ThamSo3
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3$
```



```
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
10 10
11 1
12 2
13 3
14 4
15 5
16 6
17 7
18 8
19 9
20 10
21 |
```

```
hoangho+ 13979 13977 0 15:39 pts/1 00:00:00 grep count.sh
PARENTS | List of arguments:
ThamSo1
ThamSo2
ThamSo3
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3$
```

**- Giải thích code:**

Code	Giải thích
<pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; #include &lt;sys/wait.h&gt; #include &lt;sys/types.h&gt;</pre>	<p>Đây là các khai báo thư viện được sử dụng trong chương trình, bao gồm các thư viện cần thiết cho việc làm việc với hệ thống, quản lý tiến trình và tiêu chuẩn input/output.</p>
<pre>int main(int argc, char* argv[]) {     printf("PARENTS   PID = %ld   PPID = %ld\n",            (long)getpid(), (long)getppid());     if (argc &gt; 2)</pre>	<p>Hàm main() bắt đầu chương trình. Đầu tiên, nó in ra thông tin về quá trình gọi nó, bao gồm PID (Process ID) và PPID (Parent Process ID). Sau đó, nó kiểm tra số lượng đối số đầu vào từ dòng lệnh và in ra số lượng đối số đó nếu lớn hơn 2.</p>

<pre>printf("PARENTS   There are %d arguments\n", argc - 1);</pre>	
<pre>system("./count.sh 10");</pre>	Hàm system() được sử dụng để thực hiện lệnh shell. Trong trường hợp này, nó sẽ thực thi lệnh ./count.sh 10. Hàm system() sẽ tạo một tiến trình con để chạy lệnh này và chương trình sẽ tiếp tục thực hiện khi lệnh đó hoàn thành.
<pre>printf("PARENTS   List of arguments: \n"); for (int i = 1; i &lt; argc; i++) {     printf("%s\n", argv[i]); } exit(0); }</pre>	Sau khi hàm system() kết thúc, chương trình in ra danh sách các đối số đầu vào từ dòng lệnh khi chạy chương trình. Nó duyệt qua các đối số này và in ra từng đối số trong một vòng lặp for. Cuối cùng, exit(0); được gọi để thoát khỏi chương trình với mã thoát là 0 (tức là chương trình thoát thành công).

Giải thích kết quả:

- Đoạn code trước khi chạy sẽ in thông tin về quá trình cha, bao gồm PID và PPID của nó. Sau đó, nó kiểm tra số lượng đối số được truyền vào từ dòng lệnh.
- Đoạn code sử dụng hàm system() để thực thi lệnh shell ./count.sh 10. Lệnh này yêu cầu hệ thống thực thi tập lệnh shell có tên là count.sh và truyền vào đối số là "10".
- Nếu tập lệnh count.sh tồn tại và thực thi thành công, thì nó sẽ tiếp tục in ra thông tin của quá trình cha, nhưng không in ra thông tin của các đối số từ dòng lệnh (vì sau khi gọi system(), chương trình sẽ tiếp tục in ra thông tin mà không chờ lệnh shell kết thúc).

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Trần Hoàng Lộc.

- Nếu tập lệnh `count.sh` không tồn tại hoặc có vấn đề trong quá trình thực thi, chương trình sẽ in ra thông tin về quá trình cha và cũng liệt kê các đối số từ dòng lệnh khi chạy chương trình.
- Kết quả cụ thể sẽ phụ thuộc vào việc tập lệnh `count.sh` có tồn tại và thực thi thành công hay không. Nếu thành công, thông tin từ lệnh `printf()` sau `system()` không được in ra, nếu không, thông tin về quá trình cha và các đối số từ dòng lệnh khi chạy chương trình sẽ được in ra.

- Ví du 3-4:

Ảnh minh chứng:

```
C test_shm_A.c x C test_shm_B.c ... bash x
C test_shm_A.c > ...
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14 #include <fcntl.h>
15 #include <sys/shm.h>
16 #include <sys/stat.h>
17 #include <unistd.h>
18 #include <sys/mman.h>
19 int main()
20 {
21     /* the size (in bytes) of shared memory object */
22     const int SIZE = 4096;
23     /* name of the shared memory object */
24     const char *name = "OS";
25     /* shared memory file descriptor */
26     int fd;
27     /* pointer to shared memory object */
28     char *ptr;
29     /* create the shared memory object */
30     fd = shm_open(name, O_CREAT | O_RDWR, 0666);
31     if (fd == -1) {
32         perror("shm_open");
33         return 1;
34     }
35     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
36     if (ptr == MAP_FAILED) {
37         perror("mmap");
38         return 1;
39     }
40     printf("Hello Process A\n");
41     return 0;
42 }
```

```
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3$ gcc test_shm_A.c -lrt -o test_shm_A
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3$ ./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
^[[AWaiting 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
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3$ ./test_shm_A
```

```
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3$ gcc test_shm_B.c -lrt -o test_shm_B
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3$ ./test_shm_B
Segmentation fault
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3$ ./test_shm_B
Read shared memory: Hello Process B
Shared memory updated: Hello Process A
hoanghoquocbao-22520102@LAPTOP-H6N7KTT4:~/Lab3$
```

- **Giải thích code:**

➤ Process A:

Code	Giải thích
<pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;string.h&gt;</pre>	<p>Đây là các khai báo thư viện được sử dụng trong chương trình, bao gồm các thư viện cần thiết cho việc làm việc với</p>

<pre>#include &lt;fcntl.h&gt; #include &lt;sys/shm.h&gt; #include &lt;sys/stat.h&gt; #include &lt;unistd.h&gt; #include &lt;sys/mman.h&gt;</pre>	<p>shared memory, file descriptors và các hàm hệ thống.</p>
<pre>int main() {     const int SIZE = 4096; // Kích thước     (bytes) của đối tượng shared memory     const char *name = "OS"; // Tên của     đối tượng shared memory     int fd; // File descriptor của shared     memory     char *ptr; // Con trỏ tới đối tượng     shared memory</pre>	<p>Định nghĩa các biến cần thiết cho chương trình: SIZE để xác định kích thước của đối tượng shared memory (ở đây là 4096 bytes), name là tên của đối tượng shared memory, fd là file descriptor của đối tượng shared memory, và ptr là con trỏ tới vùng nhớ của đối tượng shared memory.</p>
<pre>fd = shm_open(name, O_CREAT   O_RDWR, 0666);</pre>	<p>Dùng shm_open() để tạo một đối tượng shared memory. Nó tạo hoặc mở một đối tượng shared memory với tên là "OS", sử dụng cờ O_CREAT   O_RDWR để tạo mới nếu đối tượng chưa tồn tại và cho phép đọc và ghi, cấp quyền truy cập 0666 (rw-rw-rw-).</p>
<pre>ftruncate(fd, SIZE);</pre>	<p>Dùng ftruncate() để cấu hình kích thước của đối tượng shared memory đã tạo trước đó. Kích thước được đặt là 4096 bytes.</p>
<pre>ptr = mmap(0, SIZE, PROT_READ   PROT_WRITE, MAP_SHARED, fd, 0);</pre>	<p>Sử dụng mmap() để ánh xạ đối tượng shared memory vào không gian địa chỉ của tiến trình. Nó trả về một con trỏ ptr</p>

	trở tới vùng nhớ của đối tượng shared memory, cho phép đọc và ghi (PROT_READ   PROT_WRITE) và được chia sẻ giữa các tiến trình (MAP_SHARED).
<code>strcpy(ptr, "Hello Process B");</code>	Sao chép chuỗi "Hello Process B" vào vùng shared memory thông qua hàm <code>strcpy()</code> . Điều này có nghĩa là tiến trình hiện tại đang ghi dữ liệu này vào đối tượng shared memory.
<pre>while (strncmp(ptr, "Hello Process B", 15) == 0) {     printf("Waiting for Process B to update shared memory\n");     sleep(1); }</pre>	Tiến trình chờ cho Process B cập nhật đối tượng shared memory. Nó kiểm tra xem nội dung trong vùng shared memory có giống chuỗi "Hello Process B" không. Nếu giống, tiến trình sẽ tiếp tục chờ.
<code>printf("Memory updated: %s\n", (char *)ptr);</code>	Sau khi Process B cập nhật và thay đổi dữ liệu trong shared memory, tiến trình in ra thông tin đã được cập nhật trong shared memory.
<pre>munmap(ptr, SIZE); close(fd);</pre>	Tiến trình hủy ánh xạ đối tượng shared memory ( <code>munmap()</code> ) và đóng file descriptor của nó ( <code>close()</code> ). Điều này giải phóng tài nguyên đã được sử dụng và kết thúc việc sử dụng shared memory.
<code>Return 0;</code>	Đánh dấu sự kết thúc của hàm <code>main()</code> và đồng thời thông báo rằng chương trình đã hoàn thành mà không có lỗi nào xảy ra. Trong ngữ cảnh của hàm <code>main()</code> , giá

	trị trả về 0 thường được sử dụng để biểu thị cho việc thực hiện chương trình thành công.
--	--

➤ Process B:

Code	Giải thích
<pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;string.h&gt; #include &lt;fcntl.h&gt; #include &lt;sys/shm.h&gt; #include &lt;sys/stat.h&gt; #include &lt;unistd.h&gt; #include &lt;sys/mman.h&gt;</pre>	Các lệnh #include là để import các thư viện cần thiết để sử dụng các hàm liên quan đến shared memory, file descriptors và các hàm hệ thống.
<pre>int main() {     const int SIZE = 4096; // Kích     thước (bytes) của đối tượng shared     memory     const char *name = "OS"; // Tên     của đối tượng shared memory     int fd; // File descriptor của     shared memory     char *ptr; // Con trỏ tới đối tượng shared     memory</pre>	Chương trình bắt đầu bằng việc khai báo các biến cần thiết cho chương trình, bao gồm SIZE (kích thước của shared memory), name (tên của shared memory), fd (file descriptor của shared memory), và ptr (con trỏ tới vùng nhớ của shared memory).
<pre>fd = shm_open(name, O_RDWR, 0666);</pre>	Sử dụng shm_open() để mở một đối tượng shared memory có tên là "OS" với quyền đọc và ghi (O_RDWR) và quyền truy cập 0666 (rw-rw-rw-).
<pre>ptr = mmap(0, SIZE, PROT_READ   PROT_WRITE, MAP_SHARED, fd, 0);</pre>	Sử dụng mmap() để ánh xạ đối tượng shared memory vào không gian địa chỉ của tiến trình. Nó trả về con trỏ ptr

	trở tới vùng nhớ của shared memory, cho phép đọc và ghi (PROT_READ   PROT_WRITE) và được chia sẻ giữa các tiến trình (MAP_SHARED).
<pre>printf("Read shared memory: "); printf("%s\n", (char *)ptr);</pre>	Chương trình đọc dữ liệu từ shared memory và in ra màn hình.
<pre>strcpy(ptr, "Hello Process A"); printf("Shared memory updated: %s\n", ptr); sleep(5);</pre>	Sau đó, chương trình cập nhật dữ liệu trong shared memory bằng cách ghi chuỗi "Hello Process A" vào vùng nhớ shared memory. Sau khi cập nhật, nó in ra thông tin về dữ liệu đã được cập nhật và sau đó tạm ngừng chương trình trong 5 giây (sleep(5)) để có thể quan sát dữ liệu trong shared memory.
<pre>munmap(ptr, SIZE); close(fd); shm_unlink(name); return 0; }</pre>	chương trình thực hiện việc huỷ ánh xạ đối tượng shared memory (munmap()), đóng file descriptor của shared memory (close()), và gỡ bỏ shared memory (shm_unlink()). return 0; đánh dấu kết thúc chương trình với việc thực thi thành công (0).

**- Giải thích kết quả:**

Khi chạy lần lượt Process A trước, sau đó chạy Process B, chương trình sẽ diễn ra như sau:

- Process A:

Tạo và mở một shared memory object có tên là "OS".

Ghi chuỗi "Hello Process A" vào vùng shared memory và in ra thông tin đã được cập nhật.



Sau đó, chương trình tạm ngừng trong 5 giây để ta có thể quan sát dữ liệu trong shared memory.

Kết thúc chương trình, đóng file descriptor và gỡ bỏ shared memory.

- Process B:

Process B cũng mở và sử dụng shared memory object có tên "OS".

Nếu Process B bắt đầu chạy sau khi Process A đã ghi dữ liệu, Process B sẽ đọc dữ liệu mới nhất được cập nhật bởi Process A ("Hello Process A").

Process B sẽ đọc thông tin từ shared memory và in ra màn hình.

Do Process A thực hiện việc ghi dữ liệu trước khi Process B đọc dữ liệu từ shared memory, nên khi Process B chạy sau đó, nó sẽ đọc dữ liệu đã được cập nhật bởi Process A ("Hello Process A").

Sau khi Process A ghi dữ liệu, chương trình chờ 5 giây trước khi kết thúc. Process B khi chạy sau đó sẽ đọc dữ liệu đã được cập nhật từ shared memory và in ra màn hình.

## 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.

Trả lời...

**Cách làm:**

- Tạo file code time.c với nội dung sau:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    struct timeval start, end;
    gettimeofday(&start, NULL);
    pid_t pid = fork();
    if (pid == 0) {
        execl("/bin/sh", "sh", "-c", argv[1], NULL);
        perror("execl");
        return EXIT_FAILURE;
    } else if (pid > 0) {
        wait(NULL);
        gettimeofday(&end, NULL);
```

```
double elapsed = (end.tv_sec - start.tv_sec) +  
                 (end.tv_usec - start.tv_usec) / 1000000.0;  
  
printf("Thời gian thực thi: %.5f giây\n", elapsed);  
} else {  
    perror("fork");  
    return EXIT_FAILURE;  
}  
return EXIT_SUCCESS;  
}
```

- Ảnh minh chứng :

```
2  #include <stdlib.h>
3  #include <sys/time.h>
4  #include <sys/types.h>
5  #include <sys/wait.h>
6  #include <unistd.h>
7
8  int main(int argc, char *argv[]) {
9
10     struct timeval start, end;
11     gettimeofday(&start, NULL);
12
13     pid_t pid = fork();
14     if (pid == 0) {
15         execl("/bin/sh", "sh", "-c", argv[1], NULL);
16         perror("execl");
17         return EXIT_FAILURE;
18     } else if (pid > 0) {
19         wait(NULL);
20         gettimeofday(&end, NULL);
21
22         double elapsed = (end.tv_sec - start.tv_sec) +
23             (end.tv_usec - start.tv_usec) / 1000000.0;
24
25         printf("Thời gian thực thi: %.5f giây\n", elapsed);
26     } else {
27         perror("fork");
28         return EXIT_FAILURE;
29     }
30 }
31
32 return EXIT_SUCCESS;
33 }
34
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
nguyennnguyenngocanh-22520058@DESKTOP-7R66M1N:~/Lab3$ ./time ls
HelloWorld count.txt hello.c main.c test_execl.c test_fork.c test_fork_wait.c test_shm_A.c test_shm_B.c test_system.c time.c
count.sh hello hello.h test_excel test_fork test_fork_wait test_shm_A test_shm_B test_system time
Thời gian thực thi: 0.00263 giây
nguyennnguyenngocanh-22520058@DESKTOP-7R66M1N:~/Lab3$
```

- Giải thích code:

Code	Giải thích
<pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;sys/time.h&gt; #include &lt;sys/types.h&gt; #include &lt;sys/wait.h&gt; #include &lt;unistd.h&gt;</pre>	Dòng này sử dụng để include các thư viện cần thiết cho chương trình.

<code>int main(int argc, char *argv[]) { ... }</code>	Hàm main là điểm khởi đầu của chương trình. Nó nhận tham số dòng lệnh là argc (số lượng đối số) và argv (mảng các đối số).
<code>struct timeval start, end;</code>	Khai báo hai biến kiểu struct timeval để lưu thời gian bắt đầu và kết thúc.
<code>gettimeofday(&amp;start, NULL)</code>	Lấy thời gian hiện tại và lưu vào biến start.
<code>pid_t pid = fork();</code>	Tạo một tiến trình con và lấy PID của tiến trình đó.
<code>if (pid == 0) { ... } else if (pid &gt; 0) { ... } else { ... }</code>	Kiểm tra giá trị của pid để xác định xem chương trình đang chạy ở tiến trình con, cha hay có lỗi xảy ra khi fork.
<code>execl("/bin/sh", "sh", "-c", argv[1], NULL);</code>	Trong tiến trình con, thực thi lệnh shell được truyền qua dòng lệnh.
<code>perror("execl"); return EXIT_FAILURE;</code>	In ra thông báo lỗi nếu execl thất bại và thoát với mã lỗi.
<code>wait(NULL);</code>	Tiến trình cha đợi cho tiến trình con kết thúc.
<code>gettimeofday(&amp;end, NULL);</code>	Lấy thời gian sau khi tiến trình con kết thúc và lưu vào biến end.
<code>double elapsed = ...</code>	Tính toán thời gian thực thi bằng cách tính sự chênh lệch giữa thời gian bắt đầu và kết thúc.
<code>printf("Thời gian thực thi: %.5f giây\n", elapsed);</code>	In ra thời gian thực thi của lệnh shell dưới dạng giây với độ chính xác 5 chữ số thập phân.

<code>perror("fork"); return EXIT_FAILURE;</code>	Xử lý lỗi nếu fork thất bại và thoát với mã lỗi.
---	--

- **Giải thích kết quả:**

- Khởi tạo và bắt đầu thời gian (`gettimeofday(&start, NULL);`): Chương trình lấy thời gian bắt đầu trước khi thực hiện lệnh shell.
- Tạo tiến trình con (`fork()`): Chương trình tạo một tiến trình con để thực thi lệnh shell.
- Thực thi lệnh shell trong tiến trình con (`execl("/bin/sh", "sh", "-c", argv[1], NULL);`): Trong tiến trình con, lệnh shell được thực thi bằng cách sử dụng `execl`. Lệnh này được truyền từ đối số dòng lệnh.
- Chờ tiến trình con kết thúc (`wait(NULL);`): Tiến trình cha đợi tiến trình con kết thúc.
- Lấy thời gian kết thúc (`gettimeofday(&end, NULL);`): Sau khi tiến trình con kết thúc, chương trình lấy thời gian hiện tại để đo thời gian kết thúc của lệnh shell.
- Tính toán và in ra thời gian thực thi (`printf("Thời gian thực thi: %.5f giây\n", elapsed);`): Chương trình tính toán thời gian mà lệnh shell đã mất để thực thi bằng cách tính sự chênh lệch giữa thời gian bắt đầu và thời gian kết thúc. Kết quả này được in ra màn hình với độ chính xác đến 5 chữ số thập phân.

### 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”**

Trả lời...

#### Cách làm:

- Tạo file `Bai3.5_3.c` với code như sau:

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
void ctrl_c_handler(int signal)
```

```
{
    if(signal == SIGINT)
    {
        printf("\ncount.sh has stopped\n");
    }
}

int main()
{
    signal(SIGINT, ctrl_c_handler);
    printf("Welcome to IT007, I am 22520129\n");
    system("./count.sh");
    return 0;
}
```

- Tạo file count.sh với code như sau:

```
#!/bin/bash
count=1
while [ $count -le 120 ];
do
    echo "Counting: $count"
    sleep 1
    count=$((count+1))
done
```

- Tạo Split Terminal, tại đó, chạy 2 câu lệnh sau để chuyển mã nguồn từ file thành chương trình thực thi : gcc Bai3.5\_3.c -o Bai3.5\_3
- Cấp quyền thực thi cho file count.sh: chmod +x count.sh
- Để thực thi chương trình từ file Bai3.5\_3: ./Bai3.5\_3

Ảnh minh chứng:

```
C Bai3.c > ctrl_c_handler(int)
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <signal.h>
4  void ctrl_c_handler(int signal)
5  {
6      if(signal == SIGINT)
7      {
8          printf("\ncount.sh has stopped\n");
9      }
10 }
11
12 int main()
13 {
14     signal(SIGINT, ctrl_c_handler);
15     printf("Welcome to IT007, I am 22520129\n");
16     system("./count.sh");
17     return 0;
18 }
```

### Giải thích code cho file Bai3.5\_3.c

#include <stdio.h> #include <stdlib.h> #include <signal.h> #include <unistd.h>	Include vào các thư viện cần thiết
void ctrl_c_handler(int signal)	Tạo hàm kiểm tra tín hiệu với tham số đầu vào là sign và hàm này không có giá trị trả về
if (sign == SIGINT) { printf("\ncount.sh has stopped\n"); }	Trong hàm này , nếu tín hiệu vào là SIGINT thì chương trình sẽ in ra thông điệp count.sh has stopped
signal(SIGINT, ctrl_c_handler);	Đăng ký hàm ctrl_c_handler để xử lý tín hiệu SIGINT (tín hiệu gửi khi nhấn <b>CTRL+C</b> ).
printf("Welcome to IT007, I am 22520129\n");	In ra chuỗi Welcome to IT007, I am 22520129



system("./count.sh");	Thực thi script <b>count.sh</b>
return 0	Kết thúc hàm main và trả về giá trị 0, thông báo chương trình thành công

Ảnh chụp minh chứng:

```

$ count.sh
1  #!/bin/bash
2  count=1
3  while [ $count -le 120 ];
4  do
5      echo "Counting: $count"
6      sleep 1
7      count=$((count+1))
8  done
9

```

Giải thích code cho file count.sh

#!/bin/bash	Dùng shell để thi file này
count = 1	Gán biến count = 1 để đếm
while [ \$count -le 120 ];	Vòng lặp while với điều kiện dừng là count > 120
do	Bắt đầu vòng lặp while
echo "Counting: \$count":	In ra dòng "Counting: " kèm theo giá trị của biến count.
sleep 1	Đợi 1 giây trước khi thực hiện lệnh tiếp theo trong vòng lặp.
count=\$((count+1))	Tăng giá trị của biến count lên 1.

done	Kết thúc khối lệnh của vòng lặp while.
------	--

Ảnh minh chứng:

```
● hotienvubinh-22520129@LAPTOP-J70ECDJF:~/Lab3$ gcc Bai3.c -o Bai3
● hotienvubinh-22520129@LAPTOP-J70ECDJF:~/Lab3$ chmod +x count.sh
● hotienvubinh-22520129@LAPTOP-J70ECDJF:~/Lab3$ ./Bai3
Welcome to IT007, I am 22520129
Counting: 1
Counting: 2
Counting: 3
Counting: 4
^Ccount.sh has stopped
```

+ `chmod +x count.sh`: Dùng để cấp quyền thực thi cho file **count.sh**

+ `gcc Bai3.c -o Bai3`: Dùng để biên dịch mã nguồn từ file **Bai3.c** thành một chương trình thực thi có tên là **Bai3**.

+ `./Bai3`: Dùng để thực thi chương trình từ file **Bai3**.

Khi chương trình chạy thì đầu tiên sẽ in ra dòng lệnh **Welcome to IT007, I am 22520129**.

Sau đó thực hiện counting từ 1 đến 120. Nhưng khi đếm đến 3 thì gặp tín hiệu **Ctrl+C** thì nó ngưng lại và in ra thông báo **count.sh has stopped**.

### Giải thích kết quả:

Khi chạy, chương trình sẽ đăng kí hàm `ctrl_c_handler` để xử lý tín hiệu **SIGINT** (tín hiệu gửi khi nhấn **CTRL+C**) và in ra màn hình **"Welcome to IT007, I am 22520129"** và sau đó thực thi file `count.sh`. Trong file `count.sh` sẽ thực hiện 1 vòng lặp đếm từ 1 đến 120.

Trong quá trình chạy file `count.sh` mà gặp tín hiệu **Ctrl+C** thì chương trình sẽ dừng và in ra thông báo **"count.sh has stopped"**.

#### 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

Trả lời...

Cách làm:

- Tạo file producer.c với code như sau:

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

int main()
{
    const int SIZE = 10;
    const char *name = "BoundedBuffer";

    int fd;
    char *ptr;

    fd = shm_open(name, O_CREAT | O_RDWR, 0666);
    ftruncate(fd, SIZE);
```

```
ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

strcpy(ptr, "Waiting for consumer...");
while (strncmp(ptr, "Waiting for consumer...", 23) == 0)
{
    sleep(1);
}
printf("Consumer has conneted, start producing random number\n");

srand(time(NULL));
while (*ptr != 100)
{
    int random_number = rand() % 11 + 10;
    sprintf(ptr, "%d", random_number);

    printf("Produced: %d\n", random_number);
    sleep(1);
}
printf("Total is greater than 100. Exiting...\n");

munmap(ptr, SIZE);
close(fd);

return 0;
}
```

- Tiếp tục tạo file consumer.c với code sau:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
```

```
#include <sys/stat.h>
#include <unistd.h>
#include <sys/mman.h>

int main()
{
    const int SIZE = 10;
    const char *name = "BoundedBuffer";

    int fd;
    char *ptr;

    fd = shm_open(name, O_RDWR, 0666);
    ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

    printf("Connection has established\n");
    strcpy(ptr, "Connect");

    int total = 0;
    while (1)
    {
        sleep(1);
        int data = atoi(ptr);
        printf("Consumed: %d\n", data);

        total += data;
        printf("Total: %d\n", total);

        if (total > 100)
        {
            printf("Total is greater than 100. Exiting...\n");
            *ptr = 100;
        }
    }
}
```

```
        break;
    }
}

munmap(ptr, SIZE);
close(fd);

return 0;
}
```

- Tạo Split Terminal, tại đó, chạy 2 câu lệnh sau để chuyển mã nguồn từ file thành chương trình thực thi

```
gcc producer.c -lrt -o producer
gcc consumer.c -lrt -o consumer
```

- Tại mỗi Terminal, lần lượt chạy 2 câu lệnh sau

```
./producer
```

```
./consumer
```

- Sau khi chạy lệnh ./producer, chương trình sẽ bị kẹt trong vòng lặp  
while (strncmp(ptr, "Waiting for consumer...", 23) == 0) { sleep(1); }  
cho tới khi lệnh ./consumer được nhập

Hình ảnh:

```
1  /*
2  * University of Information Technology
3  * IT007 Operating System
4  *
5  * Nguyễn Nguyễn Ngọc Anh, 22520058
6  * Mai Thanh Bách, 22520090
7  * Huỳnh Lê Minh Đại, 22520102
8  * Hồ Tiến Vũ Bình, 22520129
9  *
10 # File: producer.c
11 #
12 #******/
13
14 #include <stdio.h>
15 #include <stdlib.h>
16 #include <string.h>
17 #include <fcntl.h>
18 #include <sys/shm.h>
19 #include <sys/stat.h>
20 #include <unistd.h>
21 #include <sys/mman.h>
22 #include <time.h>
23
24 int main()
25 {
26     const int SIZE = 10;
27     const char *name = "BoundedBuffer";
28
29     int fd;
30     char *ptr;
31
32     fd = shm_open(name, O_CREAT | O_RDWR, 0666);
33     ftruncate(fd, SIZE);
34     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
35
36     strcpy(ptr, "Waiting for consumer...");
37     while (strcmp(ptr, "Waiting for consumer...", 23) == 0)
38     {
39         sleep(1);
40     }
41 }
```

```
maithanhbach-22520090@DESKTOP-3P58CQ9:~/lab3/4$ gcc producer.c -lrt -o producer
maithanhbach-22520090@DESKTOP-3P58CQ9:~/lab3/4$ ./producer
Consumer has conneted, start producing random number
Produced: 15
Produced: 19
Produced: 10
Produced: 10
Produced: 13
Produced: 10
Produced: 17
Produced: 13
Total is greater than 100. Exiting...
maithanhbach-22520090@DESKTOP-3P58CQ9:~/lab3/4$
```

```
maithanhbach-22520090@DESKTOP-3P58CQ9:~/lab3/4$ gcc consumer.c -lrt -o consumer
maithanhbach-22520090@DESKTOP-3P58CQ9:~/lab3/4$ ./consumer
Connection has established
Consumed: 15
Total: 15
Consumed: 19
Total: 34
Consumed: 10
Total: 44
Consumed: 10
Total: 54
Consumed: 13
Total: 67
Consumed: 10
Total: 77
Consumed: 17
Total: 94
Consumed: 13
Total: 107
Total is greater than 100. Exiting...
maithanhbach-22520090@DESKTOP-3P58CQ9:~/lab3/4$
```

```
1  /*
2  * University of Information Technology
3  * IT007 Operating System
4  *
5  * Nguyễn Nguyễn Ngọc Anh, 22520058
6  * Mai Thanh Bách, 22520090
7  * Huỳnh Lê Minh Đại, 22520102
8  * Hồ Tiến Vũ Bình, 22520129
9  *
10 # File: consumer.c
11 #
12 #******/
13
14 #include <stdio.h>
15 #include <stdlib.h>
16 #include <string.h>
17 #include <fcntl.h>
18 #include <sys/shm.h>
19 #include <sys/stat.h>
20 #include <unistd.h>
21 #include <sys/mman.h>
22
23 int main()
24 {
25     const int SIZE = 10;
26     const char *name = "BoundedBuffer";
27
28     int fd;
29     char *ptr;
30
31     fd = shm_open(name, O_RDONLY, 0666);
32     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
33
34     printf("Connection has established\n");
35     strcpy(ptr, "connect");
36
37     int total = 0;
38     while (1)
39     {
40         sleep(1);
41     }
42 }
```

```
maithanhbach-22520090@DESKTOP-3P58CQ9:~/lab3/4$ gcc producer.c -lrt -o producer
maithanhbach-22520090@DESKTOP-3P58CQ9:~/lab3/4$ ./producer
Consumer has conneted, start producing random number
Produced: 15
Produced: 19
Produced: 10
Produced: 10
Produced: 13
Produced: 10
Produced: 17
Produced: 13
Total is greater than 100. Exiting...
maithanhbach-22520090@DESKTOP-3P58CQ9:~/lab3/4$
```

```
maithanhbach-22520090@DESKTOP-3P58CQ9:~/lab3/4$ gcc consumer.c -lrt -o consumer
maithanhbach-22520090@DESKTOP-3P58CQ9:~/lab3/4$ ./consumer
Connection has established
Consumed: 15
Total: 15
Consumed: 19
Total: 34
Consumed: 10
Total: 44
Consumed: 10
Total: 54
Consumed: 13
Total: 67
Consumed: 10
Total: 77
Consumed: 17
Total: 94
Consumed: 13
Total: 107
Total is greater than 100. Exiting...
maithanhbach-22520090@DESKTOP-3P58CQ9:~/lab3/4$
```

### Giải thích kết quả:

- Sau khi thực hiện các bước trên, “Connection has established” sẽ được hiện ra, sau đó, “Comsumer has conneted, start producing random number” cũng sẽ được in
- Rồi producer sẽ tạo ra các số ngẫu nhiên và đưa vào trong buffer (ptr). Mỗi lần như thế, producer sẽ in ra “Produced: <số được tạo>”

- Sau mỗi lần trên, consumer sẽ lấy số ngẫu nhiên đó từ trong buffer và đưa vào tổng. Mỗi lần như thế, consumer sẽ in ra “Total: <số nhận được>  
Consumed: <tổng>”
- Khi tổng đã lớn hơn 100, consumer sẽ gán ‘ptr’ bằng 100 (để cho producer kiểm tra) và dừng vòng lặp while, thu hồi bộ nhớ và kết thúc chương trình
- Producer sẽ kiểm tra ‘ptr’ có bằng 100 hay không. Khi ‘ptr’ đã bằng 100 (do consumer gán), bộ nhớ sẽ được thu hồi và 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.**

Trả lời...

**Cách làm:**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
```



```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>

int main(int argc, char * argv [])
{
    if (argc !=2)
    {
        printf("Su dung: %s <so nguyen duong>\n", argv[0]);
        return 1;
    }

    int n = atoi(argv[1]);

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

    //Tạo bộ nhớ chia sẻ
    int shmid = shmget(IPC_PRIVATE, 496, IPC_CREAT | 0666);
    if (shmid == -1)
    {
        perror("Loi khi tao bo nho chia se");
        return 1;
    }
}
```

```
int *share_memory = (int *)shmat(shmid,NULL,0);

pid_t pid = fork();
if(pid == 0)
{
    //Tiến trình con
    int i=0;
    while (n!=1)
    {
        share_memory[i++] = n;
        if(n%2 == 0)
        {
            n = n/2;
        }
        else
            n = 3 * n + 1;
    }
    share_memory[i] = 1;
}
else if( pid > 0)
{
    //Tiến trình cha
    wait(NULL); //chờ tiến trình con kthuc
    int i = 0;
    while (share_memory[i] != 1)
    {
        printf("%d ", share_memory[i]);
        i++;
    }
}
```

```
    }
    printf("1\n");
}
else
{
    perror("Loi khi tao tien trinh con");
    return 1;
}

//Giải phóng bộ nhớ chia sẻ
shmdt(share_memory);
shmctl(shmid, IPC_RMID, NULL);
return 0;
}
```

### Ảnh minh chứng:

```
C Bai2.6.c > main(int, char * [])
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/ipc.h>
6  #include <sys/shm.h>
7  #include <sys/wait.h>
8
9  int main(int argc, char * argv [])
10 {
11     if (argc !=2)
12     {
13         printf("Su dung: %s <so nguyen duong>\n", argv[0]);
14         return 1;
15     }
16
17     int n = atoi(argv[1]);
18
19     if(n<=0)
20     {
21         printf("Nhap so nguyen duong.\n");
22         return 1;
23     }
24 }
```

```
20     {
21         printf("Nhap so nguyen duong.\n");
22         return 1;
23     }
24
25     //Tạo bộ nhớ chia sẻ
26     int shmid = shmget(IPC_PRIVATE, 496, IPC_CREAT | 0666);
27     if (shmid == -1)
28     {
29         perror("Loi khi tao bo nho chia se");
30         return 1;
31     }
32
33     int *share_memory = (int *)shmat(shmid, NULL, 0);
34
35     pid_t pid = fork();
36     if (pid == 0)
37     {
38         //Tiến trình con
39         int i=0;
40         while (n!=1)
41         {
42             share_memory[i++] = n;
43             if (n%2 == 0)
44             {
45                 n = n/2;
46             }
```

```
45             n = n/2;
46             }
47             else
48             {
49                 n = 3 * n + 1;
50             }
51             share_memory[i] = 1;
52         }
53         else if (pid > 0)
54         {
55             //Tiến trình cha
56             wait(NULL); //chờ tiến trình con kthuc
57             int i = 0;
58             while (share_memory[i] != 1)
59             {
60                 printf("%d ", share_memory[i]);
61                 i++;
62             }
63             printf("1\n");
64         }
65         else
66         {
67             perror("Loi khi tao tien trinh con");
68             return 1;
69         }
70     }
71     //Giải phóng bộ nhớ chia sẻ
72     shmdt(share_memory);
73     shmctl(shmid, IPC_RMID, NULL);
```

```

70     shmctl(shmid, IPC_RMID, NULL);
71     shmctl(shmid, IPC_RMID, NULL);
72     return 0;
73 }
74

```

```

● hotienvubinh-22520129@LAPTOP-J70ECDJF:~/Lab3$ gcc Bai2.6.c -o collatz
● hotienvubinh-22520129@LAPTOP-J70ECDJF:~/Lab3$ ./collatz 8
8 4 2 1
● hotienvubinh-22520129@LAPTOP-J70ECDJF:~/Lab3$ ./collatz 35
35 106 53 160 80 40 20 10 5 16 8 4 2 1
● hotienvubinh-22520129@LAPTOP-J70ECDJF:~/Lab3$ ./collatz 100
100 50 25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

```

### Giải thích code:

<pre> #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; #include &lt;sys/types.h&gt; #include &lt;sys/ipc.h&gt; #include &lt;sys/shm.h&gt; #include &lt;sys/wait.h&gt; </pre>	Include các thư viện cần thiết
<pre> int main(int argc, char * argv []) </pre>	Tạo hàm main với 2 đối số là 'argc' và argv'
<pre> if (argc !=2) {     printf("Su dung: %s &lt;so nguyen duong&gt;\n", argv[0]);     return 1; } </pre>	Kiểm tra chương trình xem có đúng một đối số truyền vào hay không, nếu không thì thông báo lỗi và kết thúc trả về 1
<pre> int n = atoi(argv[1]); </pre>	Chuyển đổi số thứ 2 thành 1 số nguyên và gán vào biến 'n'.
<pre> if(n&lt;=0) {     printf("Nhap so nguyen duong.\n");     return 1; } </pre>	Kiểm tra n có phải số nguyên dương không. Nếu không phải thì kết thúc chương trình trả về 1

<code>int shmid = shmget(IPC_PRIVATE, 496, IPC_CREAT   0666);</code>	Tạo bộ nhớ chia sẻ mới sử dụng hàm 'shmget'. Biến 'shmid' lưu trữ ID của bộ nhớ chia sẻ.
<code>if (shmid == -1) {     perror("Lỗi khi tạo bộ nhớ chia sẻ");     return 1; }</code>	Nếu shmid = -1; thì tạo bộ nhớ chia sẻ thất bại in ra thông báo và trả về 1
<code>int *share_memory = (int *)shmat(shmid, NULL, 0);</code>	Dùng hàm 'shmat' để kết nối với tiến trình hiện tại tới bộ nhớ chia sẻ. Biến 'shared_memory' trỏ tới vùng nhớ chia sẻ.
<code>pid_t pid = fork();</code>	Tạo tiến trình con bằng cách sử dụng hàm 'fork'. Biến 'pid' lưu trữ ID cho tiến trình con (0 cho tiến trình con, và một giá trị dương cho tiến trình cha).
<code>if (pid == 0)</code>	Kiểm tra xem nó có phải tiến trình con
<code>int i=0; while (n!=1) {     share_memory[i++] = n;     if(n%2 == 0)     {         n = n/2;     }     else         n = 3 * n + 1; } share_memory[i] = 1;</code>	Một vòng lặp trong tiến trình con để tính toán chuỗi Collatz. Chuỗi Collatz được lưu trữ trong bộ nhớ chia sẻ, và biến i được sử dụng để theo dõi vị trí trong bộ nhớ chia sẻ. Chuỗi dừng lại khi n đạt giá trị 1 và được đánh dấu bằng share_memory[i] = 1.
<code>else if (pid &gt; 0)</code>	Kiểm tra xem nó có phải tiến trình cha
<code>wait(NULL); int i = 0; while (share_memory[i] != 1)</code>	+ Chờ tiến trình con kết thúc + Một vòng lặp trong tiến trình cha để đọc chuỗi Collatz từ bộ nhớ chia sẻ và in nó ra

<pre>{     printf("%d ", share_memory[i]);     i++; } printf("1\n");</pre>	màn hình. Chuỗi Collatz kết thúc khi giá trị trong bộ nhớ chia sẻ bằng 1.
<pre>else {     perror("Loi khi tao tien trinh con");     return 1; }</pre>	Thông báo lỗi tạo tiến trình con và trả về 1
<pre>shmdt(share_memory); shmctl(shmid, IPC_RMID, NULL);</pre>	Giải phóng bộ nhớ chia sẻ khi hoàn thành chương trình.
<pre>return 0;</pre>	Kết thúc chương trình

-Gõ lệnh **gcc -o Bai2.6.c collatz.c** để biên dịch chương trình

- **./collatz 8** : thực thi chương trình với đối số là 8

- Chương trình in ra chuỗi 8 4 2 1

=> Kết thúc bằng 1 vậy phỏng đoán collatz chúng ta đã đúng

```
● hotienvubinh-22520129@LAPTOP-J70ECDJF:~/Lab3$ gcc Bai2.6.c -o collatz  
● hotienvubinh-22520129@LAPTOP-J70ECDJF:~/Lab3$ ./collatz 8  
8 4 2 1
```

-Thử với các đối số khác:

```
8 4 2 1  
● hotienvubinh-22520129@LAPTOP-J70ECDJF:~/Lab3$ ./collatz 35  
35 106 53 160 80 40 20 10 5 16 8 4 2 1  
● hotienvubinh-22520129@LAPTOP-J70ECDJF:~/Lab3$ ./collatz 100  
100 50 25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```