

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

Họ và tên: Đoàn Phương Nam - Trương Thiên Lộc - Phạm Duy Long - Trần Quỳnh Thy

Mã số sinh viên: 22520908 – 21520330 – 20521573 – 22521461

Lớp: IT007.O11.2

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: 10/10

**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?

Trả lời...

- Ví dụ 3-1

+ Hình minh chứng:

```
#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);
}
```

```
● thy-22521461@thy-22521461:~/Desktop/Lab3$ ./test_fork 1 2 3
PARENTS | PID = 2647 | PPID = 2382
PARENTS | There are 3 arguments
CHILDREN | PID = 2648 | PPID = 2647
CHILDREN | List of arguments:
1
2
3
```

+ Giải thích:

- . `__pid_t`: Là kiểu dữ liệu của số hiệu tiến trình.
- . `fork()`: Tạo tiến trình mới là tiến trình con của tiến trình hiện tại
- . `if (pid > 0)`: Nếu tiến trình là tiến trình cha.
- . `if (pid == 0)`: Nếu tiến trình là tiến trình con.

- . getpid(): Lấy số hiệu tiến trình hiện tại.
- . getppid(): Lấy số hiệu tiến trình cha của tiến trình con.
- . argc: Số lượng tham số
- . argv[]: Mảng chứa các tham số
- . wait(NULL): đợi tiến trình con kết thúc

Giải thích kết quả:

Vì ta nhập vào 3 tham số nên ở tiến trình cha $argc = 4 > 2$ nên sẽ in ra dòng “PARENTS | There are %d arguments\n”, ở tiến trình con thì sẽ in ra giá trị của 3 tham số nhập vào.

- Ví dụ 3-2

+ Hình minh chứng:

```
#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();

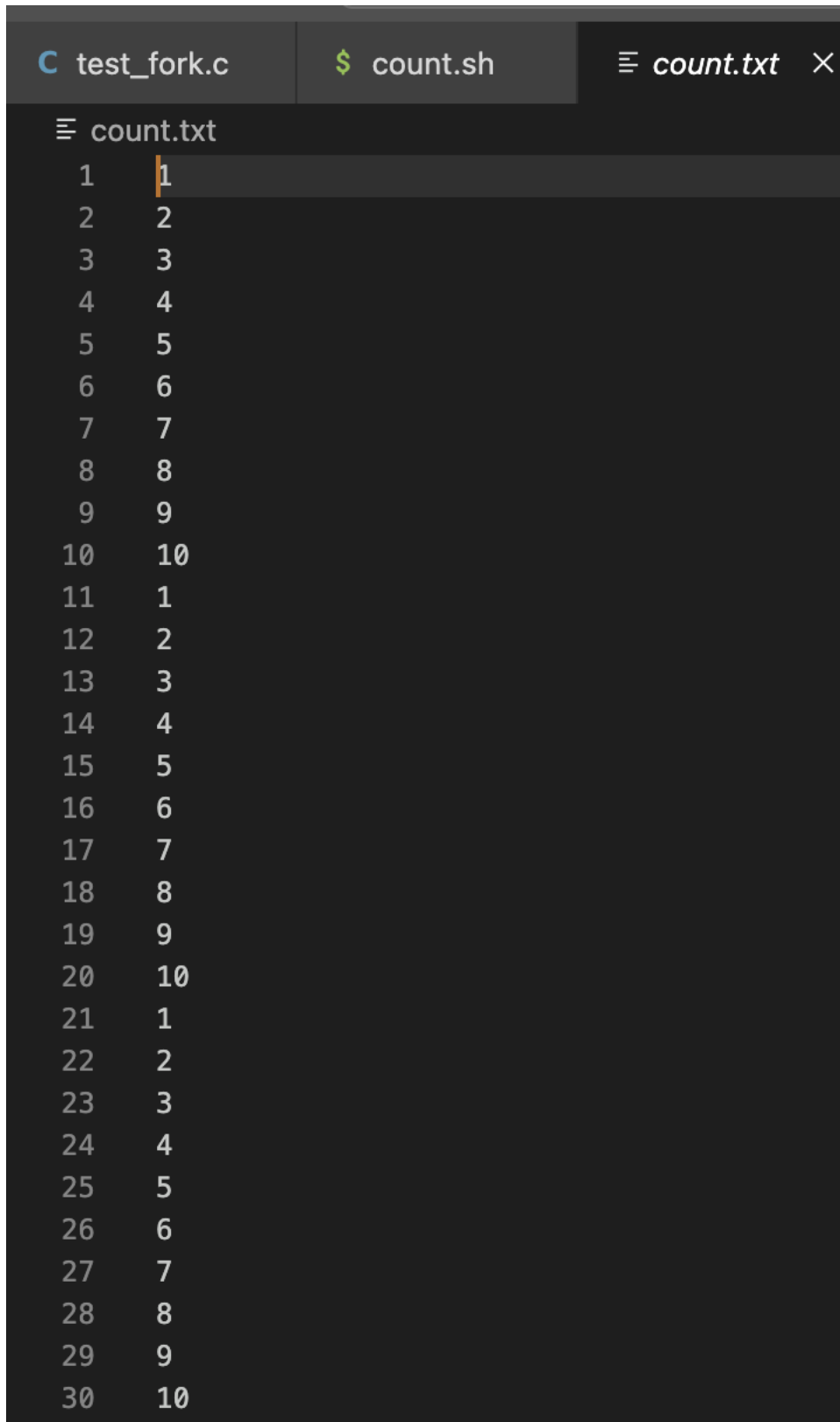
    if (pid > 0) {
        printf("PARENTS | PID = %ld | PPID = %ld\n", (long)getpid(), (long)getppid());
        if (argc > 2) printf("PARENTS | There are %d arguments\n", argc - 1);
        wait(NULL);
    }

    if (pid == 0) {
        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 < argc; i++) {
            printf("%s\n", argv[i]);
        }
    }

    exit(0);
}
```

```
● thy-22521461@thy-22521461:~/Desktop/Lab3$ ./test_exec1 1 2 3
PARENTS | PID = 6263 | PPID = 1893
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
thy-225+ 6264 6263 0 10:38 pts/3 00:00:00 /bin/bash ./count.sh 10
thy-225+ 6266 6264 0 10:38 pts/3 00:00:00 grep count.sh
```



```
C test_fork.c $ count.sh ≡ count.txt ×  
≡ count.txt  
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 1  
22 2  
23 3  
24 4  
25 5  
26 6  
27 7  
28 8  
29 9  
30 10
```

+ Giải thích:

Đoạn code trên tạo một tiến trình con và thực thi shell script trong tiến trình con

Lệnh `execl`: thay tiến trình hiện tại bằng tiến trình khác (ở đoạn code trên có nghĩa là thay tiến trình con thành tiến trình `count.sh`). Vì vậy, các dòng `printf` sau lệnh `execl` sẽ không được thực thi vì lúc này tiến trình con đã chuyển sang thành `count.sh`)

- Ví dụ 3-3

+ Hình minh chứng:

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

int main(int argc, char* argv[]) {
    printf("PARENTS | PID = %ld | PPID = %ld\n", (long)getpid(), (long)getppid());
    if (argc > 2) printf("PARENTS | There are %d arguments\n", argc - 1);

    system("./count.sh 10");

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

    exit(0);
}
```

● thy-22521461@thy-22521461:~/Desktop/Lab3\$./test_system 1 2 3

PARENTS | PID = 7719 | PPID = 1893

PARENTS | There are 3 arguments

Implementing: ./count.sh

PPID of count.sh:

thy-225+	7720	7719	0	11:37	pts/3	00:00:00	sh -c ./count.sh 10
thy-225+	7721	7720	0	11:37	pts/3	00:00:00	/bin/bash ./count.sh 10
thy-225+	7723	7721	0	11:37	pts/3	00:00:00	grep count.sh

PARENTS | List of arguments:

1

2

3

+ Giải thích:

Lệnh `system("./count.sh 10")` sẽ tạo tiến trình mới là `sh` với lệnh là `"./count.sh 10"`

Trong file count.sh, dòng #!/bin/bash được đọc và sẽ tạo tiến trình /bin/bash
./count.sh 10

Vì tạo mới hoàn toàn 1 tiến trình, nên các dòng code sau lệnh system vẫn
được thực thi bình thường

- Ví dụ 3-4

+ Hình minh chứng:

```
C test_shm_A.c
16 #include <unistd.h>
17 #include <sys/mman.h>
18
19 int main() {
20     const int SIZE = 4096;
21
22     const char *name = "0S";
23
24     int fd;
25
26     char *ptr;
27
28     fd = shm_open(name, O_CREAT | O_RDWR, 0666);
29
30     ftruncate(fd, SIZE);
31
32     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
33
34     strcpy(ptr, "Hello Process B");
35
36     while (strncmp(ptr, "Hello Process B", 15) == 0) {
37         printf("Waiting Process B update shared memory\n");
38         sleep(1);
39     }
40
41     printf("Memory updated: %s\n", (char *)ptr);
42
43     munmap(ptr, SIZE);
44     close(fd);
45     return 0;
46 }
```

~/Desktop/Lab

```
C test_shm_B.c
16 // #include <sys/mman.h>
17 #include <sys/mman.h>
18
19 int main() {
20     const int SIZE = 4096;
21
22     const char *name = "0S";
23
24     int fd;
25
26     char *ptr;
27
28     fd = shm_open(name, O_RDWR, 0666);
29     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
30
31     printf("Read shared memory: ");
32     printf("%s\n", (char *)ptr);
33
34     strcpy(ptr, "Hello Process A");
35     printf("Shared memory updated: %s\n", ptr);
36     sleep(5);
37
38     munmap(ptr, SIZE);
39     close(fd);
40
41     shm_unlink(name);
42     return 0;
43 }
```

```
thy-22521461@thy-22521461:~/Desktop/Lab3$ ./test_shm_A
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Memory updated: Hello Process A
thy-22521461@thy-22521461:~/Desktop/Lab3$

thy-22521461@thy-22521461:~/Desktop/Lab3$ ./test_shm_B
Read shared memory: Hello Process B
Shared memory updated: Hello Process A
thy-22521461@thy-22521461:~/Desktop/Lab3$
```

+ Giải thích:

`fd = shm_open(name, O_CREAT | O_RDWR, 0666)` : Khởi tạo vùng nhớ được chia sẻ

`ftruncate(fd, SIZE)`: Cài đặt độ lớn của vùng nhớ chia sẻ.

`mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0)`: Khởi tạo file ánh xạ bộ nhớ có đối tượng vùng nhớ chia sẻ.

`munmap(ptr, SIZE)`;

`close(fd)`;

shm_unlink(name): Thu hồi bộ nhớ
test_shm_A.c: strcpy(ptr, "Hello Process B"): Ghi vào vùng nhớ chia sẻ chuỗi "Hello Process B", while (strncmp(ptr, "Hello Process B"), 15 == 0) kiểm tra vùng nhớ chia sẻ đã được thay đổi hay chưa, sleep(1): đợi 1 giây
test_shm_B.c: strcpy(ptr, "Hello Process A"): Ghi vào vùng nhớ chia sẻ chuỗi "Hello Process A", sleep(5): đợi 5 giây
Vì vậy ở bài này khi thực thi test_shm_A, dòng while sẽ được lặp lại liên tục đến khi test_shm_B được thực thi, đến khi vùng nhớ chia sẻ được đổi thành "Hello Process A" thì dòng while sẽ ngừng.

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...

Bước 1: Xác thực Tham số Đầu Vào

Kiểm tra xem có đủ tham số dòng lệnh để chạy chương trình không. Nếu không, chương trình sẽ in hướng dẫn cách sử dụng và kết thúc.

Bước 2: Lấy Thời Gian Bắt Đầu

Sử dụng hàm `gettimeofday()` để lấy thời gian khi chương trình bắt đầu thực hiện lệnh shell.

Bước 3: Tạo Tiến Trình Con

Sử dụng `fork()` để tạo một tiến trình con mới.

Bước 4: Thực Thi Lệnh Shell

Trong tiến trình con, chương trình sử dụng `execl()` để thực thi lệnh shell được truyền qua tham số dòng lệnh.

Bước 5: Đợi Tiến Trình Con Kết Thúc

Trong tiến trình cha, chương trình sử dụng `wait()` để đợi tiến trình con kết thúc thực thi lệnh.

Bước 6: Lấy Thời Gian Kết Thúc

Sau khi tiến trình con kết thúc, sử dụng `gettimeofday()` để lấy thời gian tại thời điểm tiến trình con hoàn thành công việc.

Bước 7: Tính Thời Gian Thực Thi

Tính toán thời gian thực thi bằng cách lấy sự chênh lệch giữa thời gian bắt đầu và thời gian kết thúc.

Bước 8: Xuất Kết Quả

Kết quả về thời gian thực thi của lệnh shell được in ra màn hình.

Chương trình này sử dụng các hàm hệ thống để tạo và quản lý các tiến trình con, sau đó đo và báo cáo thời gian thực thi của lệnh shell được chạy.

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

```
int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Usage: ./time <command>\n");
        return 1;
    }

    struct timeval start, end;
    gettimeofday(&start, NULL);

    pid_t pid = fork();

    if (pid == 0) {
        // Child process
        execl("/bin/sh", "sh", "-c", argv[1], (char *)0);
    } else if (pid > 0) {
        // Parent process
        wait(NULL);
        gettimeofday(&end, NULL);

        double execution_time = (double)(end.tv_sec - start.tv_sec) + (double)(end.tv_usec - start.tv_usec) / 1000000;

        printf("Execution time: %.5f seconds\n", execution_time);
    } else {
        printf("Fork failed\n");
        return 1;
    }

    return 0;
}
```

Kết quả của chương trình sau khi gõ lệnh ./time ls:

```
phamduylong-20521573@LAPTOP-JF94MS0F:~/ssh$ ./time ls
authorized_keys count.sh id_rsa known_hosts test_execl test_fork time
config count.txt id_rsa.pub known_hosts.old test_execl.c test_fork.c time.c
Thời gian thực thi: 0.02305
phamduylong-20521573@LAPTOP-JF94MS0F:~/ssh$
```

Giải thích kết quả:

- Khi chạy ./time ls, chương trình của bạn thực hiện theo thứ tự sau:
- Bắt đầu đo thời gian (gettimeofday(&start, NULL)).
- Tạo một tiến trình con để thực hiện lệnh ls.
- Tiến trình con chạy lệnh ls để hiển thị danh sách các file trong thư mục.
- Tiến trình cha đợi tiến trình con hoàn thành (bằng cách sử dụng wait()).
- Sau khi tiến trình con kết thúc, tiến trình cha đo thời gian (gettimeofday(&end, NULL)) và tính toán thời gian thực thi.

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

- Cuối cùng, chương trình in ra danh sách các file và sau đó in ra thời gian thực thi.

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...

Ý tưởng

Sử dụng các hàm hệ thống.

```
top > Lab3 > C Bai3.c > main()
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
```

Bước 1: In ra thông điệp chào mừng và lấy thông tin Student ID từ người dùng

- a. Sử dụng printf để hiển thị thông điệp chào mừng.
- b. Sử dụng scanf để lấy thông tin Student ID từ người dùng lưu vào mảng char đã khai báo từ trước.

```
int main() {
    // Get Student ID from the user
    char studentID[50];
    printf("Enter your Student ID: ");
    scanf("%s", studentID);

    printf("Welcome to IT007, I am %s!\n", studentID);
}
```

Bước 2: Thực thi file script count.sh

- Sử dụng `fork()` để tạo một tiến trình con.
- Trong tiến trình con:
- Sử dụng `execl()` để thực thi file script `count.sh` với tham số là 120.

```
pid_t pid = fork();
if (pid == 0) {
    execl("./count.sh", "count.sh", "120", NULL);
```

Bước 3: Xử lý tín hiệu Ctrl+C

Trong tiến trình cha:

- Đăng ký xử lý tín hiệu Ctrl+C bằng hàm `signal(SIGINT, handler_function)`.
- Khi nhận tín hiệu Ctrl+C, chạy hàm xử lý tín hiệu để in ra thông điệp "count.sh has stopped" và kết thúc chương trình.

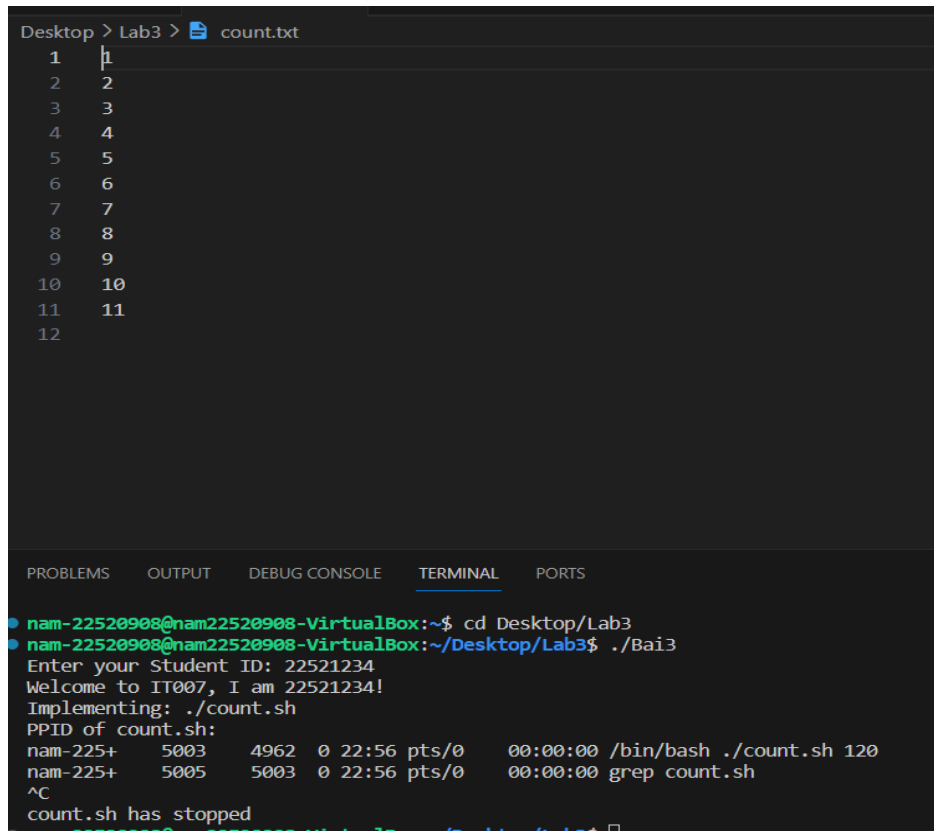
```
else {
    // Register signal handling for Ctrl+C
    signal(SIGINT, handle_ctrlc);
    // Wait for the child process to finish
    wait(NULL);
```

```
return 0;
```

```
void handle_ctrlc(int sig) {
    printf("\n count.sh has stopped \n");
    exit(0);
}
```

Kết quả:

- Nhập thử một mssv: 225212344.
- Theo dõi file `count.txt`, dừng tiến trình ở 11 bằng Ctrl+C. Terminal trả về thông báo "count.sh has stopped" và kết thúc chương trình



The screenshot shows a terminal window with a file editor at the top and a terminal output at the bottom. The file editor is editing a file named `count.txt` in the `Desktop > Lab3` directory. The content of the file is a list of numbers from 1 to 12, with the first line being `1 |`. The terminal output shows the following commands and their results:

```
nam-22520908@nam22520908-VirtualBox:~$ cd Desktop/Lab3
nam-22520908@nam22520908-VirtualBox:~/Desktop/Lab3$ ./Bai3
Enter your Student ID: 22521234
Welcome to IT007, I am 22521234!
Implementing: ./count.sh
PPID of count.sh:
nam-225+ 5003 4962 0 22:56 pts/0 00:00:00 /bin/bash ./count.sh 120
nam-225+ 5005 5003 0 22:56 pts/0 00:00:00 grep count.sh
^C
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

1. Giả định:

- Vì độ dài của buffer là 10 bytes, nên có thể chia buffer thành 10 phần với mỗi phần 1 byte.

- Thực tế, khi tạo ra một buffer, phải đồng thời cấp phát ô nhớ để lưu các thông tin của buffer như điểm start (điểm bắt đầu), end (điểm kết thúc)... Nên lượng tài nguyên cần để tạo ra một buffer thực tế sẽ lớn hơn độ dài của buffer.

2. Ý tưởng:

- Định nghĩa cấu trúc Buffer chứa một buffer và các thông tin liên quan của nó. Trong đó phải có một thông tin cho biết buffer có còn được sử dụng hay không (isFinish).
- Tạo một shared memory và khởi tạo một buffer rỗng.
- Tạo function Producer và Consumer:
 - o Producer: Khi buffer vẫn còn được sử dụng, thực hiện công việc tạo ra số ngẫu nhiên và đẩy số ngẫu nhiên đó vào trong buffer.
 - o Consumer: Khi có sự thay đổi trong buffer, thực hiện công việc in ra các số có trong buffer và tính tổng. Nếu tổng các số trong buffer lớn hơn 100, sẽ chỉnh thông tin buffer lại thành “không còn được sử dụng”.
- Dùng lệnh fork() để tạo tiến trình con.
- Tiến trình cha sẽ gọi function Producer và tiến trình con sẽ gọi function Consumer.
- Cuối cùng là đóng kết nối và giải phóng bộ nhớ.

3. Code:

```
1  #include <stdio.h>
2  #include <stdbool.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <fcntl.h>
6  #include <sys/shm.h>
7  #include <sys/stat.h>
8  #include <unistd.h>
9  #include <sys/mman.h>
10 #include <sys/wait.h>
11 #include <time.h>
12 #include <math.h>
```

Khai báo thư viện

```
14  #define BUFFER_SIZE 10
15
16  // Define a Buffer structure
17  struct Buffer
18  {
19      char buffer[BUFFER_SIZE];
20      int length;
21      bool isFinish;
22  };
```

Khai báo cấu trúc của buffer

- Bao gồm:
 - **buffer**: chứa buffer.
 - **length**: chứa độ dài của buffer.
 - **isFinish**: cho biết đã không còn được sử dụng hay không. (**false** → buffer đang được sử dụng; **true** → buffer không còn được sử dụng).

```
24  // Min both numbers
25  int min(int a, int b) {
26      return (a < b ? a : b);
27  }
```

Khai báo các hàm helper

- Ở đây sẽ sử dụng hàm **min** để xử lý bounded cho buffer. Được sử dụng trong function xử lý của Consumer.


```
30 // The producer function simulates the role of a Producer
31 void producer(struct Buffer *ptr)
32 {
33
34     while (!ptr->isFinish)
35     {
36         // Generate a random number in the range [10, 20]
37         char num = rand() % 11 + 10;
38
39         // Add the number to the buffer
40         ptr->buffer[(ptr->length++) % BUFFER_SIZE] = num;
41
42         sleep(1);
43     }
44
45     // End the Consumer when the Producer has finished
46     wait(NULL);
47     printf("Producer is closed\n");
48 }
```

Hàm xử lí của Producer

- Input nhận vào một con trỏ ánh xạ đến vùng shared memory chứa buffer.
- Điều kiện của dòng while có nghĩa là nếu buffer vẫn còn được sử dụng thì vẫn thực hiện các lệnh bên trong.
- Bên trong dòng while bao gồm:
 - o Tạo ra số ngẫu nhiên.
 - o Thêm số vào buffer. Có hỗ trợ cơ chế bounded.
 - o Sau khi thêm số vào buffer thì chờ cho bên **Consumer** xử lí rồi mới thêm số khác vào. Thời gian chờ đợi ở code trên là **1 giây**.
- Sau khi dòng while đã dừng thì chờ cho bên Consumer off trước rồi Producer mới thông báo off sau.

```
47 // The Consumer
48 void consumer(struct Buffer* ptr) {
49     int length = 0;
50
51     while (1) {
52         if (ptr->length <= length) {
53             continue;
54         }
55
56         length = ptr->length;
57
58         int sum = 0;
59         printf("buffer: [ ");
60         // Loop for print buffer and sum
61         for (int i = 0; i < min(length, 10); i++)
62         {
63             printf("%d ", ptr->buffer[i]);
64             sum += ptr->buffer[i];
65         }
66         printf("]\n");
67         printf("sum: %d\n\n", sum);
68
69         // Check sum & close Consumer
70         if (sum >= 100)
71         {
72             ptr->isFinish = true;
73             printf("Consumer is closed\n");
74             return;
75         }
76     }
77 }
```

Hàm xử lý của Consumer

- Input đầu vào là một con trỏ ánh xạ đến shared memory chứa buffer.
- Để xét được buffer có bị thay đổi dữ liệu không thì cần có một dependency để so sánh sự thay đổi. Ở đây chọn thuộc tính **length** làm dependency (do buffer chỉ có thể được thêm, chứ không được chỉnh sửa nên sự thay đổi dễ thấy nhất là độ dài của buffer). Vì thế, cần có một biến để lưu lại giá trị length ở vòng lặp trước đó để so sánh.

- Đây là một listener nên cần phải lặp vô tận để bắt các sự thay đổi. Vì vậy, nên vòng lặp sẽ là **while(1)**.
- Bên trong dòng while:
 - o Nếu dòng length mới không khác length cũ, thì sẽ không thực hiện việc xử lý.
 - o Gán length = length mới.
 - o Dòng for có tác dụng in ra các số có trong buffer và tính tổng.
 - o Sau khi đã tính tổng, in tổng ra.
 - o Xét nếu tổng đã lớn hơn 100 thì cho **isFinish = true** (nghĩa là buffer này không còn được sử dụng nữa). Sau đó thông báo Consumer off và **return**.

```
85 int main()
86 {
87     // Define a name of shared memory
88     const char *name = "Provider - Consumer";
89
90     // Create and configure shared memory
91     int fd = shm_open(name, O_CREAT | O_RDWR, 0666);
92     ftruncate(fd, sizeof(struct Buffer));
93
94     // Connect shared memory to pointer & initialize the state
95     struct Buffer *ptr = mmap(0, sizeof(struct Buffer), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
96     ptr->isFinish = false;
97     ptr->length = 0;
98
99     // Initialize a random number generator
100    srand(time(NULL));
```

Hàm main: set up buffer và random

- Tạo ra một shared memory bằng shm_open với quyền **O_CREAT**, cấp quyền đọc ghi bằng **O_RDWR** và truyền mã quyền là **0666**.
- Cấp phát không gian cho shared memory bằng **ftruncate** với kích thước được cấp phát là độ lớn của cấu trúc Buffer.
- Ánh xạ vùng nhớ buffer vào con trỏ ptr bằng **mmap**, với vị trí ánh xạ đầu là **0** và độ lớn ánh xạ bằng với độ lớn của cấu trúc Buffer. Cấp quyền đọc ghi bằng **PROT_READ** và **PROT_WRITE**, cấp quyền chia sẻ thay đổi bằng **MAP_SHARED**.

- Khởi tạo trạng thái ban đầu cho buffer với **isFinish = false** (buffer đang được sử dụng) và **length = 0** (buffer chưa có bất kỳ phần tử nào).
- Khởi tạo seed cho random bằng **srand**, seed sẽ là thời gian hiện tại.

```
102 // Fork
103 pid_t pid = fork();
104 if (pid == 0)
105 {
106     // The child process as the Consumer
107     consumer(ptr);
108 }
109 else if (pid > 0)
110 {
111     // The parent process as the Producer
112     producer(ptr);
113 }
114 else
115 {
116     perror("fork");
117     return 1;
118 }
```

Hàm main: Tạo tiến trình con và chia Producer – Consumer

- Lệnh **fork** để tạo tiến trình con.
- Với **pid == 0**, đây là ngữ cảnh của tiến trình con. Thực hiện function xử lý của **Consumer**.
- Với **pid > 0**, đây là ngữ cảnh của tiến trình cha. Thực hiện function xử lý của **Producer**.
- Trường hợp **pid < 0**, fork thất bại, thông báo lỗi và **return 1** (lỗi).

```
120 // Close the connection to shared memory
121 munmap(ptr, BUFFER_SIZE);
122 close(fd);
123
124 return 0;
125 }
```

Hàm main: Đóng kết nối với shared memory và thu hồi bộ nhớ

- Đóng kết nối với shared memory bằng **munmap**.
- Thu hồi bộ nhớ bằng **close**.

4. Kết quả:

```
● truongthienloc-21520330@LAPTOP-58S08A0P:~/HDH/Lab03/Bai04$ gcc Bai04.c -o Bai04 -lrt
● truongthienloc-21520330@LAPTOP-58S08A0P:~/HDH/Lab03/Bai04$ ls
Bai04  Bai04.c  Bai04_test.c  Test
```

Build file thực thi

- Build file **Bai04.c** (file chứa code) thành file **Bai04** (file thực thi).
- Khi build file, phải gán thêm cờ **-lrt**, nếu không gán thì gcc sẽ không cho phép build.
- Như trên hình, ta thấy file **Bai04** đã được build và khi dùng lệnh **ls**, nó xuất hiện có chứa **màu xanh** (nghĩa là có quyền thực thi).

```
● truongthienloc-21520330@LAPTOP-58S08A0P:~/HDH/Lab03/Bai04$ ./Bai04
buffer: [ 18 ]
sum: 18

buffer: [ 18 11 ]
sum: 29

buffer: [ 18 11 17 ]
sum: 46

buffer: [ 18 11 17 12 ]
sum: 58

buffer: [ 18 11 17 12 12 ]
sum: 70

buffer: [ 18 11 17 12 12 13 ]
sum: 83

buffer: [ 18 11 17 12 12 13 13 ]
sum: 96

buffer: [ 18 11 17 12 12 13 13 17 ]
sum: 113

Consumer is closed
Producer is closed
```

Chạy file thực thi

- Khi thực thi file Bai04, cứ sau **1 giây**, màn hình sẽ in ra giá trị của buffer và tổng của các số trong buffer.
- Lần đầu tiên, buffer có một số duy nhất là **18** và có tổng là **18**. Sau **1 giây**, giá trị bên trong buffer có thêm số **11** và tổng là **29**... Đến khi tổng các số bên trong buffer **113** (>100) thì **Consumer thông báo đóng**, kế tiếp là **Producer thông báo đóng**.

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...

1. Giả định

- Buffer không giới hạn độ dài.
- Mỗi ngăn chứa của buffer không giới hạn kiểu dữ liệu.

2. Ý tưởng

- Định nghĩa cấu trúc Buffer chứa một buffer và các thông tin liên quan khác. Trong đó có thông tin độ dài của buffer (length).
- Lấy input của chương trình, kiểm tra input đó, nếu không phải là số nguyên dương thì xuất thông báo và kết thúc chương trình.
- Tạo ra một shared memory và khởi tạo một buffer rỗng.
- Tạo 2 function parent_process và child_process:
 - o parent_process: mỗi khi buffer có một con số mới được thêm vào, xuất ra màn hình con số đó, đồng thời kiểm tra con số đó nếu bằng 1 thì kết thúc function.

- `child_process`: Liên tục tạo ra con số kế tiếp theo thuật toán mà đề đã cho, thêm con số vào `buffer`, đồng thời kiểm tra con số đó nếu bằng 1 thì kết thúc function.
- Dùng lệnh `fork()` để tạo tiến trình con.
- Tiến trình cha sẽ gọi function `parent_process` và tiến trình con sẽ gọi function `child_process`.
- Cuối cùng là đóng kết nối và giải phóng bộ nhớ.

3. Code

```
1  #include <stdio.h>
2  #include <stdbool.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <fcntl.h>
6  #include <sys/shm.h>
7  #include <sys/stat.h>
8  #include <unistd.h>
9  #include <sys/mman.h>
10 #include <sys/wait.h>
11 #include <time.h>
```

Khai báo thư viện

```
12
13  #define BUFFER_SIZE 1001
14
15  struct Buffer
16  {
17      int buffer[BUFFER_SIZE];
18      int length;
19  };
20
```

Khai báo cấu trúc Buffer

- Cấu trúc của **Buffer** bao gồm 1 **buffer** có độ dài **1001 phần tử** và thuộc tính **length** cho biết độ dài đã ghi vào buffer.


```
21 void parent_process(struct Buffer* ptr)
22 {
23     // Dependencies
24     int length = 0;
25
26     while (1)
27     {
28         while (length == ptr->length);
29
30         int num = ptr->buffer[length++];
31         if (num == 1) {
32             printf("%d\n", num);
33             break;
34         }
35         printf("%d, ", num);
36     }
37
38     wait(NULL);
39 }
```

Function parent_process

- Được chạy bởi tiến trình cha.
- Tham số đầu vào bao gồm 1 con trỏ **Buffer**.
- Khởi tạo một dependency là **length**, mỗi khi thuộc tính **length** của **buffer** bị thay đổi thì sẽ thực thi đoạn code xử lý.
- Bên trong dòng **while(1)**, dòng **while đầu tiên** được dùng để đợi cho đến khi thuộc tính **length** của **Buffer** bị thay đổi.
- Lấy ra phần tử tiếp theo của **buffer** mà tiến trình cha chưa xuất ra màn hình. Kiểm tra số đó nếu bằng 1 thì xuất ra màn hình như là một phần tử cuối cùng và kết thúc dòng **while(1)**, còn nếu khác 1 thì vẫn xuất ra màn hình như một phần tử bình thường.
- Kết thúc dòng while, đợi cho đến khi tiến trình con kết thúc thì mới thoát function **parent_process**.

```
41 void child_process(struct Buffer* ptr, int start)
42 {
43     int num = start;
44     ptr->buffer[ptr->length++] = num;
45
46     while (num > 1)
47     {
48         if (num % 2 == 0)
49         {
50             num = num / 2;
51         }
52         else {
53             num = 3 * num + 1;
54         }
55
56         ptr->buffer[ptr->length++] = num;
57     }
58 }
```

Function child_process

- Được sử dụng với tiến trình con.
- Tham số đầu vào bao gồm 1 con trỏ **Buffer** và một **giá trị xuất phát** (do người dùng nhập vào).
- Lưu giá trị xuất phát đó như phân tử đầu tiên và lưu nó vào biến **num** để tính toán.
- **While(num > 1)**: kiểm tra nếu **num > 1** thì sẽ thực thi thuật toán bên trong.
- Dòng **if else** bên trong dòng while: Dùng biến **num** để thực thi thuật toán của **Collatz** và lưu kết quả vào biến **num**.
- Sau đó, lưu biến **num** vào phần tử tiếp theo của **Buffer**.

```
60 int main(int argc, char* agrv[])
61 {
62     // Get input
63     int start = atoi(agrv[1]);
64
65     // Check input
66     if (start <= 0) {
67         printf("%s không phải là số nguyên dương\n", agrv[1]);
68         return 1;
69     }
70 }
```

Hàm main: Lấy và check input

- Input của người dùng sẽ được lưu vào **agrv[1]** dưới dạng **char***.
- Để chuyển từ **char*** thành **int**, sử dụng hàm **atoi** được cung cấp sẵn trong bộ thư viện chuẩn.
- Kiểm tra nếu **start <= 0** thì xuất ra thông báo và kết thúc chương trình.

```
71 // Define a name of shared memory
72 const char* name = "collatz";
73
74 // Create and configure shared memory
75 int fd = shm_open(name, O_CREAT | O_RDWR, 0666);
76 ftruncate(fd, sizeof(struct Buffer));
77
78 // Connect shared memory to pointer & initialize the state
79 struct Buffer *ptr = mmap(0, sizeof(struct Buffer), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
80 ptr->length = 0;
```

Hàm main: Khởi tạo, cấp phát shared memory và Buffer.

- Tạo ra một shared memory bằng **shm_open** với quyền **O_CREAT**, cấp quyền đọc ghi bằng **O_RDWR** và truyền mã quyền là **0666**.
- Cấp phát không gian cho shared memory bằng **ftruncate** với kích thước được cấp phát là độ lớn của cấu trúc **Buffer**.
- Ánh xạ vùng nhớ buffer vào con trỏ **ptr** bằng **mmap**, với vị trí ánh xạ đầu là **0** và độ lớn ánh xạ bằng với độ lớn của cấu trúc **Buffer**. Cấp quyền đọc ghi bằng **PROT_READ** và **PROT_WRITE**, cấp quyền chia sẻ thay đổi bằng **MAP_SHARED**.

- Khởi tạo trạng thái ban đầu cho **Buffer** với **length = 0** (buffer chưa có phần tử nào được thêm vào).

```
82     pid_t pid = fork();
83     if (pid > 0) {
84         parent_process(ptr);
85     }
86     else if (pid == 0) {
87         child_process(ptr, start);
88     }
89     else {
90         perror("fork");
91         return 1;
92     }
```

Hàm main: Tạo tiến trình con và chia công việc

- Lệnh **fork** để tạo tiến trình con.
- Với **pid == 0**, đây là ngữ cảnh của tiến trình con. Thực hiện function xử lí của **Consumer**.
- Với **pid > 0**, đây là ngữ cảnh của tiến trình cha. Thực hiện function xử lí của **Producer**.
- Trường hợp **pid < 0**, fork thất bại, thông báo lỗi và **return 1** (lỗi).

```
94     // Close the connection to shared memory
95     munmap(ptr, BUFFER_SIZE);
96     close(fd);
97
98     return 0;
99 }
```

Hàm main: Đóng kết nối với shared memory và thu hồi bộ nhớ

- Đóng kết nối với shared memory bằng **munmap**.
- Thu hồi bộ nhớ bằng **close**.

4. Kết quả

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
• truongthienloc-21520330@LAPTOP-58S08A0P:~/HDH/Lab03/BaiOT01$ gcc BaiOT01.c -o collatz -lrt
• truongthienloc-21520330@LAPTOP-58S08A0P:~/HDH/Lab03/BaiOT01$ ls
BaiOT01.c  collatz
```

Biên dịch file thực thi

- Biên dịch file **BaiOT01.c** thành file thực thi **collatz**.
- Kiểm tra bằng lệnh **ls**, thấy file **collatz** có màu xanh (nghĩa là file này có quyền thực thi).

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
• truongthienloc-21520330@LAPTOP-58S08A0P:~/HDH/Lab03/BaiOT01$ ./collatz 8
8, 4, 2, 1
• truongthienloc-21520330@LAPTOP-58S08A0P:~/HDH/Lab03/BaiOT01$ ./collatz 35
35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1
```

Chạy file thực thi

- Thực thi với input bằng **8** và input bằng **35**. Thấy kết quả giống như trong đề.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
⊗ truongthienloc-21520330@LAPTOP-58S08A0P:~/HDH/Lab03/BaiOT01$ ./collatz -5
-5 không phải là số nguyên dương
⊗ truongthienloc-21520330@LAPTOP-58S08A0P:~/HDH/Lab03/BaiOT01$ ./collatz 0
0 không phải là số nguyên dương
⊗ truongthienloc-21520330@LAPTOP-58S08A0P:~/HDH/Lab03/BaiOT01$ ./collatz hello
hello không phải là số nguyên dương
```

Trường hợp lỗi

- Các trường hợp không phải số nguyên dương được test ở trên bao gồm **-5** (số âm), **0** (số 0) và **hello** (một đoạn string bất kỳ).
- Khi nhập vào một số không phải số nguyên dương, chương trình sẽ xuất ra đoạn thông báo trên và dừng thực thi chương trình.