

Họ và tên	MSSV	Lớp
Lại Quan Thiên	22521385	IT007.O21.1
Đặng Đức Tài	22521270	
Mai Nguyễn Nam Phương	22521164	
Phùng Trần Thế Nam	21522366	

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 checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Chụp hình minh chứng	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Giải thích kết quả	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

3.6. BÀI TẬP ÔN TẬP

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

Tư chấm điểm: 10/10

**Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp: <MSSV>_LAB3.pdf*

3.7. 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:

```
/*#####  
26  
# 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 source code:

- Hàm fork() tạo một bản sao của tiến trình gọi nó. Trong trường hợp này, nó tạo ra một tiến trình con.
- Xử lý trong tiến trình cha: Nếu pid > 0, đây là tiến trình cha. Chương trình in ra PID và PPID của tiến trình cha. Nếu số lượng tham số truyền vào lớn hơn 2, chương trình in ra số lượng tham số đó. Sau đó, tiến trình cha chờ tiến trình con kết thúc bằng hàm wait(NULL).
- Xử lý trong tiến trình con: Nếu pid == 0, đây là tiến trình con. Chương trình in ra PID và PPID của tiến trình con, sau đó in ra danh sách các tham số truyền vào.

+ Kết quả:

```
● thenam@21522366:~/Documents/Lab_3$ gcc test_fork.c -o test_fork
● thenam@21522366:~/Documents/Lab_3$ ./test_fork ThamSo1 ThamSo2 ThamSo3
PARENTS | PID = 7131 | PPID = 6512
PARENTS | There are 3 arguments
CHILDREN | PID = 7132 | PPID = 7131
CHILDREN | List of arguments:
ThamSo1
ThamSo2
ThamSo3
```

- Ví dụ: 3-2:

```
/*#####
#University of Information Technology
#IT007 Operating System
#
#<Your name>, <your Student ID>
#File : test_execl.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)
    {
        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);
}
```

```
$ count.sh
1  #!/bin/bash
2
3  echo "Implementing: $0"
4  echo "PPID of count.sh: "
5  ps -ef | grep count.sh
6
7  i=1
8  while [ $i -le $1 ]
9  do
10     echo $i >> count.txt
11     i=$((i+1))
12     sleep 1
13 done
14 exit 0
```

+ Giải thích source code:

- Tạo tiến trình con với fork(): Sử dụng hàm fork() để tạo một tiến trình con. Giá trị trả về của fork() là 0 trong tiến trình con và một số khác trong tiến trình cha.
- Xử lý trong tiến trình cha: Nếu pid lớn hơn 0, tức là đang ở trong tiến trình cha, in ra thông tin về PID và PPID của tiến trình cha. Nếu có hơn 2 tham số dòng lệnh, in ra số lượng tham số.
- Xử lý trong tiến trình con: Nếu pid bằng 0, tức là đang ở trong tiến trình con, sử dụng hàm execl() để thực thi tập lệnh count.sh với tham số là "10". In ra thông tin về PID và PPID của tiến trình con. In ra các tham số dòng lệnh truyền vào.

+ Kết quả:

```
● thenam@21522366:~/Documents/Lab_3$ gcc test_execl.c -o test_execl
● thenam@21522366:~/Documents/Lab_3$ ./test_execl ThamSo1 ThamSo2 ThamSo3
PARENTS | PID = 2436 | PPID = 2083
PARENTS | There are 3 arguments
CHILDREN | PID = 2437 | PPID = 2436
CHILDREN | List of arguments:
ThamSo1
ThamSo2
ThamSo3
```

- Ví dụ: 3-3:

```
/*#####  
# University of Information Technology  
# IT007 Operating System  
#  
# <Your name>, <your Student ID>  
# File: test_system.c  
#  
#####*/  
#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);  
}
```

```
$ count.sh
1  #!/bin/bash
2
3  echo "Implementing: $0"
4  echo "PPID of count.sh: "
5  ps -ef | grep count.sh
6
7  i=1
8  while [ $i -le $1 ]
9  do
10     echo $i >> count.txt
11     i=$((i+1))
12     sleep 1
13 done
14 exit 0
```

+ **Giải thích source code:**

- Xử lý trong tiến trình cha: In ra thông tin về PID và PPID của tiến trình cha. Nếu có hơn 2 tham số dòng lệnh, in ra số lượng tham số.
- Thực thi lệnh với hàm system(): Sử dụng hàm system() để thực thi lệnh ./count.sh 10. Điều này sẽ gọi script count.sh với tham số là 10.

+ **Kết quả:**

```
● thenam@21522366:~/Documents/Lab_3$ gcc test_system.c -o test_system
● thenam@21522366:~/Documents/Lab_3$ ./test_system ThamSo1 ThamSo2 ThamSo3
PARENTS | PID = 5655 | PPID = 5549
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
thenam      5656    5655  0 14:26 pts/2    00:00:00 sh -c ./count.sh 10
thenam      5657    5656  0 14:26 pts/2    00:00:00 /bin/bash ./count.sh 10
thenam      5659    5657  0 14:26 pts/2    00:00:00 grep count.sh
PARENTS | List of arguments:
ThamSo1
ThamSo2
ThamSo3
```

- Ví dụ: 3-4:

```
C test_shm_A.c > ...
1  /*#####
2  # University of Information Technology
3  # IT007 Operating System
4  #
5  # <Your name>, <your Student ID>
6  # File: test_shm_A.c
7  #####*/
8
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12 #include <fcntl.h>
13 #include <sys/shm.h>
14 #include <sys/stat.h>
15 #include <unistd.h>
16 #include <sys/mman.h>
17 int main()
18 {
19     /* the size (in bytes) of shared memory object */
20     const int SIZE = 4096;
21     /* name of the shared memory object */
22     const char *name = "OS";
23     /* shared memory file descriptor */
24     int fd;
25     /* pointer to shared memory object */
26     char *ptr;
27     /* create the shared memory object */
28     fd = shm_open(name, O_CREAT | O_RDWR, 0666);
29     /* configure the size of the shared memory object */
30     ftruncate(fd, SIZE);
31     /* memory map the shared memory object */
32     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE,
33               MAP_SHARED, fd, 0);
34     /* write to the shared memory object */
35     strcpy(ptr, "Hello Process B");
36     /* wait until Process B updates the shared memory
37     segment */
38     while (strncmp(ptr, "Hello Process B", 15) == 0)
39     {
40         printf("Waiting Process B update shared memory\n");
41         sleep(1);
42     }
43     printf("Memory updated: %s\n", (char *)ptr);
44     /* unmap the shared memory segment and close the
45     file descriptor */
46     munmap(ptr, SIZE);
47     close(fd);
48     return 0;
49 }
```

```
C test_shm_B.c > ...
1  /*#####
2  # University of Information Technology
3  # IT007 Operating System
4  #
5  # <Your name>, <your Student ID>
6  # File: test_shm_B.c
7  #####*/
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include <fcntl.h>
12 #include <sys/shm.h>
13 #include <sys/stat.h>
14 #include <unistd.h>
15 #include <sys/mman.h>
16 int main()
17 {
18     /* the size (in bytes) of shared memory object */
19     const int SIZE = 4096;
20     /* name of the shared memory object */
21     const char *name = "OS";
22     /* shared memory file descriptor */
23     int fd;
24     /* pointer to shared memory object */
25     char *ptr;
26     /* create the shared memory object */
27     fd = shm_open(name, O_RDWR, 0666);
28     /* memory map the shared memory object */
29     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE,
30              MAP_SHARED, fd, 0);
31     /* read from the shared memory object */
32     printf("Read shared memory: ");
33     printf("%s\n", (char *)ptr);
34     /* update the shared memory object */
35     strcpy(ptr, "Hello Process A");
36     printf("Shared memory updated: %s\n", ptr);
37     sleep(5);
38     // unmap the shared memory segment and close the file descriptor
39     munmap(ptr, SIZE);
40     close(fd);
41     // remove the shared memory segment
42     shm_unlink(name);
43     return 0;
44 }
```

+ Giải thích source code:

Process A (test_shm_A.c):

- Mô tả: Quy trình A tạo một đối tượng shared memory, ghi dữ liệu vào nó và sau đó chờ quy trình B cập nhật dữ liệu trong đối tượng shared memory. Khi quy trình B đã cập nhật xong, quy trình A đọc và in ra dữ liệu đã được cập nhật.
- Bước thực thi:
 - Tạo một đối tượng shared memory bằng hàm shm_open().
 - Cấu hình kích thước của đối tượng shared memory bằng ftruncate().
 - Ánh xạ đối tượng shared memory vào không gian địa chỉ của quy trình bằng mmap().

- Ghi dữ liệu vào đối tượng shared memory bằng cách sao chép chuỗi "Hello Process B" vào con trỏ ptr.
- Chờ đợi cho đến khi quy trình B cập nhật dữ liệu trong shared memory thông qua vòng lặp while.
- Khi dữ liệu đã được cập nhật, đọc và in ra dữ liệu đã được cập nhật.
- Hủy ánh xạ và đóng file descriptor bằng munmap() và close().

Process B (test_shm_B.c):

- Mô tả: Quy trình B mở đối tượng shared memory được tạo bởi quy trình A, đọc dữ liệu từ đối tượng shared memory, sau đó cập nhật dữ liệu trong nó. Sau đó, quy trình B chờ một thời gian trước khi hủy bỏ đối tượng shared memory.
- Bước thực thi:
 - Mở đối tượng shared memory bằng hàm shm_open() với quyền đọc và ghi.
 - Ánh xạ đối tượng shared memory vào không gian địa chỉ của quy trình bằng mmap().
 - Đọc dữ liệu từ đối tượng shared memory và in ra màn hình.
 - Cập nhật dữ liệu trong đối tượng shared memory bằng cách sao chép chuỗi "Hello Process A" vào con trỏ ptr.
 - Chờ một khoảng thời gian bằng hàm sleep().
 - Hủy ánh xạ và đóng file descriptor bằng munmap() và close().
 - Xóa đối tượng shared memory bằng hàm shm_unlink().
 - Một khi quy trình B đã cập nhật xong dữ liệu trong shared memory, quy trình A sẽ tiếp tục thực hiện và in ra dữ liệu đã được cập nhật. Điều này minh họa cơ chế truyền thông qua shared memory giữa các quy trình trong hệ điều hành.

+ **Kết quả:**

```
● thenam@21522366:~/Documents/Lab_3$ gcc test_shm_A.c -o test_shm_A
● thenam@21522366:~/Documents/Lab_3$ ./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
Memory updated: Hello Process A

● thenam@21522366:~/Documents/Lab_3$ gcc test_shm_B.c -o test_shm_B
● thenam@21522366:~/Documents/Lab_3$ ./test_shm_B
Read shared memory: Hello Process B
Shared memory updated: Hello Process A
```

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:

- **Source code:** (giải thích trong comments)

```
int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Usage: %s <command>\n", argv[0]);
        return 1;
    }

    struct timeval start, end;

    // Lấy mốc thời gian trước khi thực thi lệnh shell
    gettimeofday(&start, NULL);

    pid_t pid = fork();

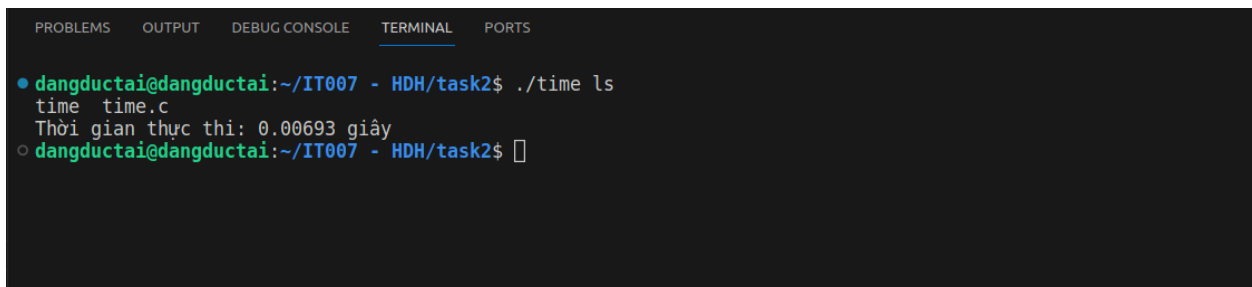
    if (pid < 0) {
        perror("fork");
        return 1;
    } else if (pid == 0) {
        // Tiến trình con: thực thi lệnh shell được chỉ định
        execl("/bin/sh", "sh", "-c", argv[1], NULL);
        perror("execl");
        exit(1);
    } else {
        // Tiến trình cha: chờ tiến trình con kết thúc
        int status;
        waitpid(pid, &status, 0);

        // Lấy mốc thời gian sau khi thực thi lệnh shell
        gettimeofday(&end, NULL);

        if (WIFEXITED(status)) { //Kiểm tra xem tiến trình con có kết thúc như bình thường không (dùng exit() hoặc return)
            //Tính toán thời gian thực thi
            double execution_time = (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec) / 1000000.0;
            printf("Thời gian thực thi: %.5f giây\n", execution_time);
        }
    }

    return 0;
}
```

- Kết quả:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
dangductai@dangductai:~/IT007 - HDH/task2$ ./time ls
time time.c
Thời gian thực thi: 0.00693 giây
dangductai@dangductai:~/IT007 - HDH/task2$
```

- Giải thích:

- + Trước khi thực thi lệnh shell, chương trình sẽ lấy mốc thời gian bắt đầu bằng cách sử dụng hàm gettimeofday().
- + Tiến trình cha sẽ tạo một tiến trình con bằng cách sử dụng hàm fork().
- + Trong tiến trình con, lệnh shell được thực thi bằng hàm execl() và tham số dòng lệnh được truyền vào.
- + Tiến trình cha sẽ đợi tiến trình con kết thúc bằng hàm waitpid().

- + Sau khi tiến trình con kết thúc, tiến trình cha lấy mốc thời gian kết thúc bằng cách sử dụng lại hàm `gettimeofday()`.
- + Sau đó, chương trình tính toán thời gian thực thi bằng cách lấy hiệu của thời gian kết thúc và thời gian bắt đầu.
- + Kết quả thời gian thực thi được in ra màn hình với định dạng số thập phân có 5 chữ số sau dấu 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”s

Trả lời:

- Source code: (giải thích trong comments)

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

// Handler cho tín hiệu SIGINT (CTRL+C)
void sigint_handler(int signum) {
    printf("count.sh has stoppped\n");
    exit(0);
}

int main() {
    // In ra dòng chữ chào mừng với Student ID
    printf("Welcome to IT007, I am 22521270!\n");

    // Thiết lập handler cho tín hiệu SIGINT (CTRL+C) sử dụng sigaction()
    struct sigaction act;
    act.sa_handler = sigint_handler;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    sigaction(SIGINT, &act, NULL);

    // Thực thi file script count.sh với số lần đếm là 120
    pid_t pid = fork();

    if (pid < 0) {
        perror("fork");
        return 1;
    } else if (pid == 0) {
        execl("./count.sh", "./count.sh", "120", NULL);
        sleep(1);
        perror("execl");
        exit(1);
    } else {
        wait(NULL);
    }


    return 0;
}
```

```
$ cat count.sh
#!/bin/bash

# Hàm để xử lý tín hiệu SIGINT (CTRL+C)
trap 'echo ""; echo "count.sh has stopped"; exit' INT

# Đếm từ 1 đến 120
for ((i = 1; i <= 120; i++)); do
    echo $i
    sleep 0.1 # Dừng một giây trước khi in số tiếp theo
done
```

- Kết quả:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
• dangductai@angductai:~/IT007 - HDH/task2$ ./time ls
time time.c
Thời gian thực thi: 0.00693 giây
○ dangductai@angductai:~/IT007 - HDH/task2$
```

- Giải thích:

- + In ra dòng chữ chào mừng với Student ID.
- + Thiết lập một handler cho tín hiệu SIGINT (CTRL+C) bằng cách sử dụng `sigaction()`.
- + Thực thi file script `count.sh` với số lần đếm là 120 bằng cách sử dụng hàm `exec1()`.
- + Nếu không có lỗi xảy ra trong quá trình thực thi file script, tiến trình con sẽ kết thúc và tiến trình cha sẽ đợi cho đến khi tiến trình con kết thúc.
- + Nếu có lỗi xảy ra trong quá trình thực thi file script, tiến trình con sẽ in ra thông báo lỗi và thoát với mã lỗi 1.
- + Khi tiến trình cha nhận được tín hiệu SIGINT (CTRL+C), nó sẽ kích hoạt handler `sigint_handler()`, in ra thông báo "count.sh has stopped" và thoát chương trình.
- + Nếu tập lệnh `count.sh` thực hiện đúng, chương trình sẽ kết thúc và không cần phải in ra thông báo lỗi từ `perror("exec1")`.

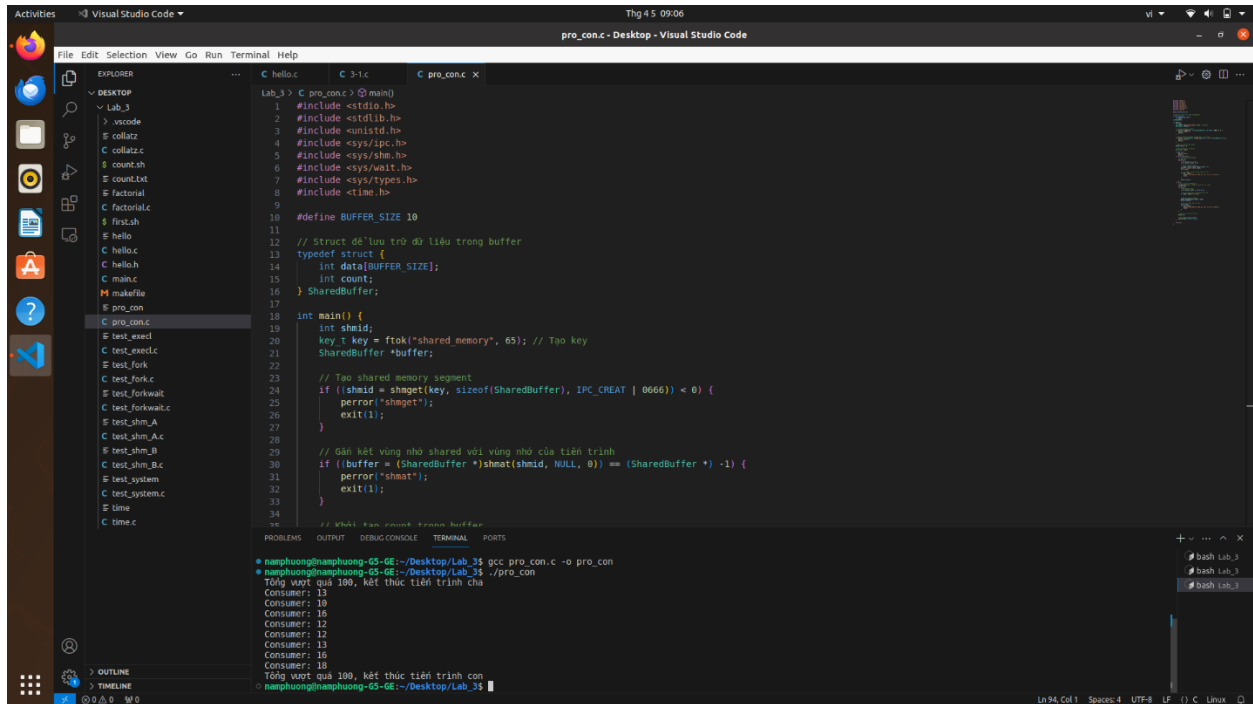
4. Viết chương trình mô phỏng bài toán Producer - Consumer như sau:

- 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:

- Trình bày cách làm: đã comment trong code

- Chụp hình minh chứng:



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/ipc.h>
5 #include <sys/shm.h>
6 #include <sys/wait.h>
7 #include <sys/types.h>
8 #include <time.h>
9
10 #define BUFFER_SIZE 10
11
12 // Struct để lưu trữ dữ liệu trong buffer
13 typedef struct {
14     int data[BUFFER_SIZE];
15     int count;
16 } SharedBuffer;
17
18 int main() {
19     int shmid;
20     key_t key = ftok("shared_memory", 0); // Tạo key
21     SharedBuffer *buffer;
22
23     // Tạo shared memory segment
24     if ((shmid = shmget(key, sizeof(SharedBuffer), IPC_CREAT | 0666)) < 0) {
25         perror("shmget");
26         exit(1);
27     }
28
29     // Gán kết vùng nhớ shared với vùng nhớ của tiến trình
30     if ((buffer = (SharedBuffer *)shmat(shmid, NULL, 0)) == (SharedBuffer *) -1) {
31         perror("shmat");
32         exit(1);
33     }
34
35     // Khởi tạo biến count trong buffer
36     buffer->count = 0;
37
38     // Tạo tiến trình con
39     pid_t pid = fork();
40
41     if (pid < 0) {
42         perror("fork");
43         exit(1);
44     }
45
46     if (pid == 0) {
47         // Tiến trình con (Consumer)
48         int sum = 0;
49         while (1) {
50             int i = 0;
51             while (i < buffer->count) {
52                 int val = buffer->data[i];
53                 printf("Consumer: %d\n", val);
54                 sum += val;
55                 i++;
56             }
57             if (sum > 100) {
58                 printf("Tổng vượt quá 100, kết thúc tiến trình con\n");
59                 exit(0);
60             }
61             // Dừng lại nếu buffer trống
62             while (buffer->count == 0) {
63                 sleep(1);
64             }
65         }
66     }
67
68     // Tiến trình cha (Producer)
69     while (1) {
70         int val = rand() % 11 + 10; // Tạo số ngẫu nhiên trong khoảng [10, 20]
71         while (i < buffer->count) {
72             sleep(1);
73         }
74         buffer->data[i] = val;
75         buffer->count++;
76         printf("Producer: %d\n", val);
77     }
78
79     // Dừng lại nếu buffer đầy
80     while (buffer->count == BUFFER_SIZE) {
81         sleep(1);
82     }
83
84     // Kết thúc tiến trình cha
85     exit(0);
86 }
```

nanphuong@nanphuong-GS-GE:~/Desktop/Lab_3\$ gcc pro_con.c -o pro_con

nanphuong@nanphuong-GS-GE:~/Desktop/Lab_3\$./pro_con

Tổng vượt quá 100, kết thúc tiến trình cha

Consumer: 12

Consumer: 19

Consumer: 16

Consumer: 12

Consumer: 12

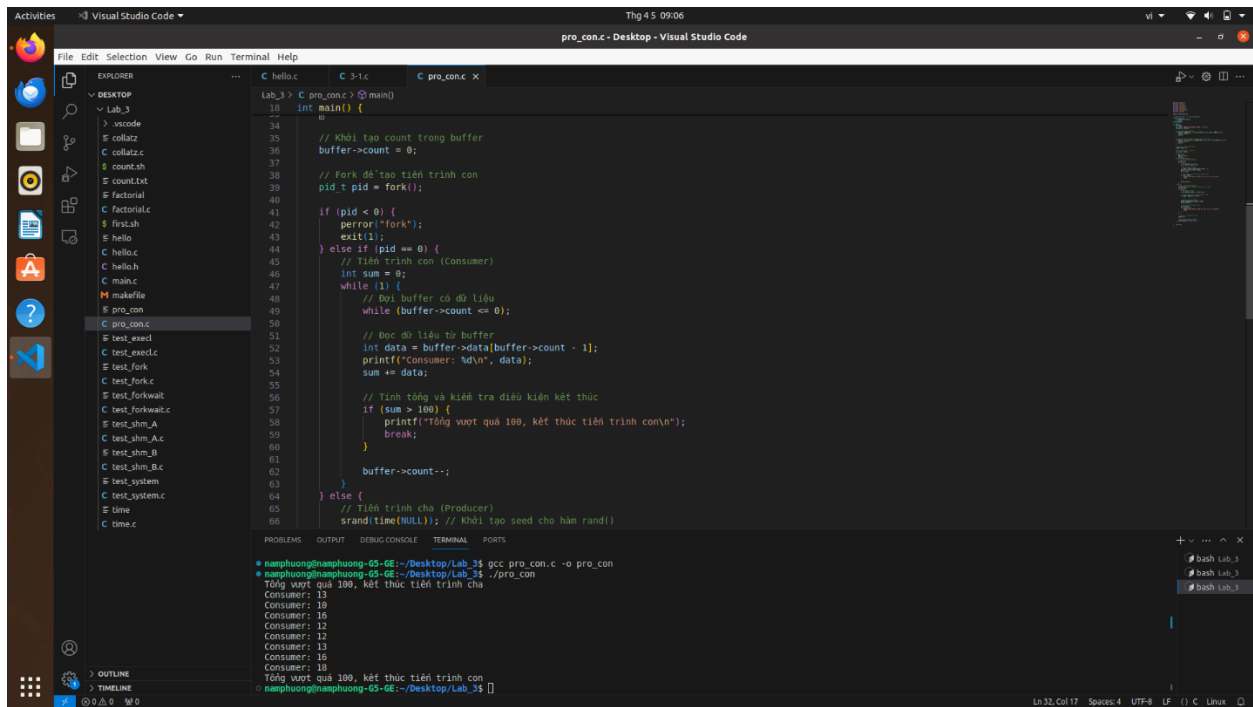
Consumer: 13

Consumer: 16

Tổng vượt quá 100, kết thúc tiến trình con

nanphuong@nanphuong-GS-GE:~/Desktop/Lab_3\$

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Lê Hoài Nghĩa.



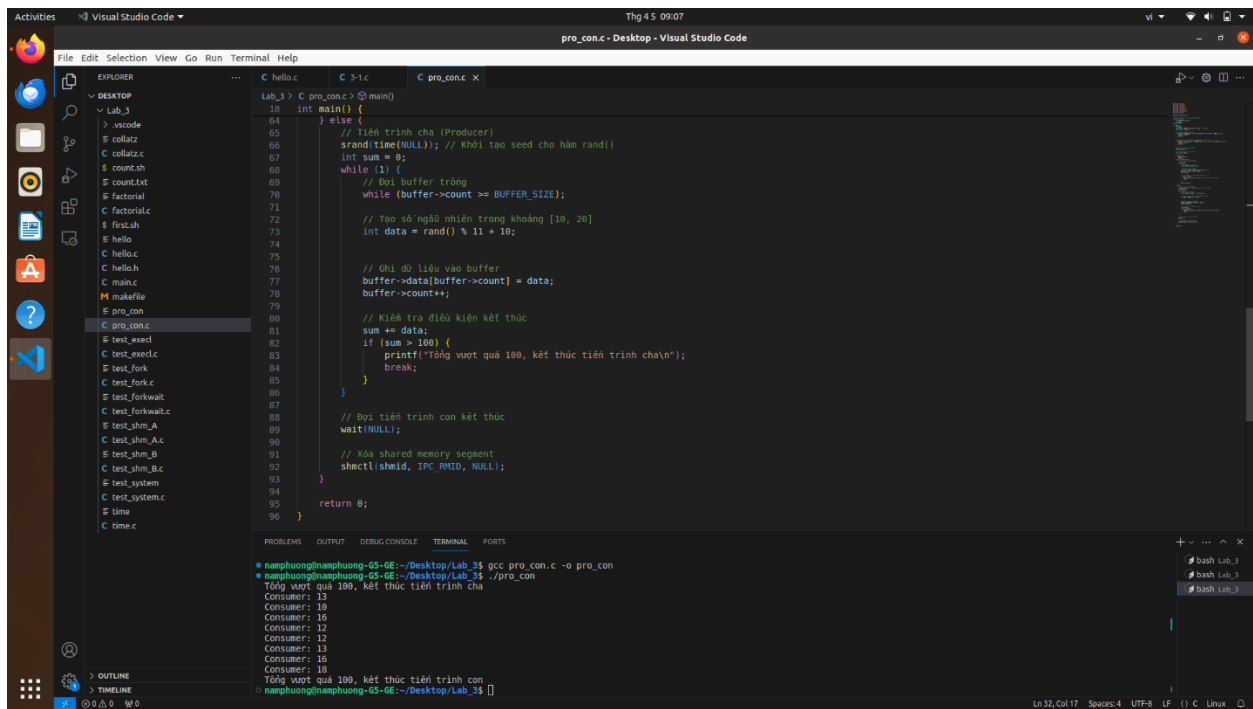
```
File Edit Selection View Go Run Terminal Help
pro_con.c - Desktop - Visual Studio Code

EXPLORER
DESKTOP
Lab_3
> .vscode
F collatz
C collatz.c
C countsh
F count.txt
F factorial
C factorial.c
F firstsh
F hello
C hello.c
C hello.h
C main.c
M makefile
E pro_con
C pro_con.c
F test_exec
C test_exec.c
F test_fork
C test_fork.c
F test_forkwait
C test_forkwait.c
F test_shm_A
C test_shm_A.c
E test_shm_B
C test_shm_B.c
F test_system
C test_system.c
F time
C time.c

18 int main() {
19     // Khởi tạo count trong buffer
20     buffer->count = 0;
21     // Fork để tạo tiến trình con
22     pid_t pid = fork();
23     if (pid < 0) {
24         perror("fork");
25         exit(1);
26     } else if (pid == 0) {
27         // Tiến trình con (Consumer)
28         int sum = 0;
29         while (1) {
30             // Đợi buffer có đủ dữ liệu
31             while (buffer->count <= 0) {
32                 // Đọc dữ liệu từ buffer
33                 int data = buffer->data[buffer->count - 1];
34                 printf("Consumer: %d\n", data);
35                 sum += data;
36             }
37             // Tính tổng và kiểm tra điều kiện kết thúc
38             if (sum > 100) {
39                 printf("Tổng vượt quá 100, kết thúc tiến trình con\n");
40                 break;
41             }
42             buffer->count--;
43         }
44     } else {
45         // Tiến trình cha (Producer)
46         srand(time(NULL)); // Khởi tạo seed cho hàm rand()
47         int sum = 0;
48         while (1) {
49             // Tạo số ngẫu nhiên trong khoảng [10, 20]
50             int data = rand() % 11 + 10;
51             // Ghi dữ liệu vào buffer
52             buffer->data[buffer->count] = data;
53             buffer->count++;
54             // Kiểm tra điều kiện kết thúc
55             sum += data;
56             if (sum > 100) {
57                 printf("Tổng vượt quá 100, kết thúc tiến trình cha\n");
58                 break;
59             }
60         }
61         // Đợi tiến trình con kết thúc
62         wait(NULL);
63         // Khóa shared memory segment
64         shmctl(shmid, IPC_RMID, NULL);
65     }
66     return 0;
67 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- nanphuong@nanphuong-GS-GE:~/Desktop/Lab_3\$ gcc pro_con.c -o pro_con
- nanphuong@nanphuong-GS-GE:~/Desktop/Lab_3\$./pro_con
- Consumer: 13
- Consumer: 19
- Consumer: 16
- Consumer: 12
- Consumer: 12
- Consumer: 12
- Consumer: 16
- Consumer: 18
- Tổng vượt quá 100, kết thúc tiến trình con
- nanphuong@nanphuong-GS-GE:~/Desktop/Lab_3\$



```
File Edit Selection View Go Run Terminal Help
pro_con.c - Desktop - Visual Studio Code

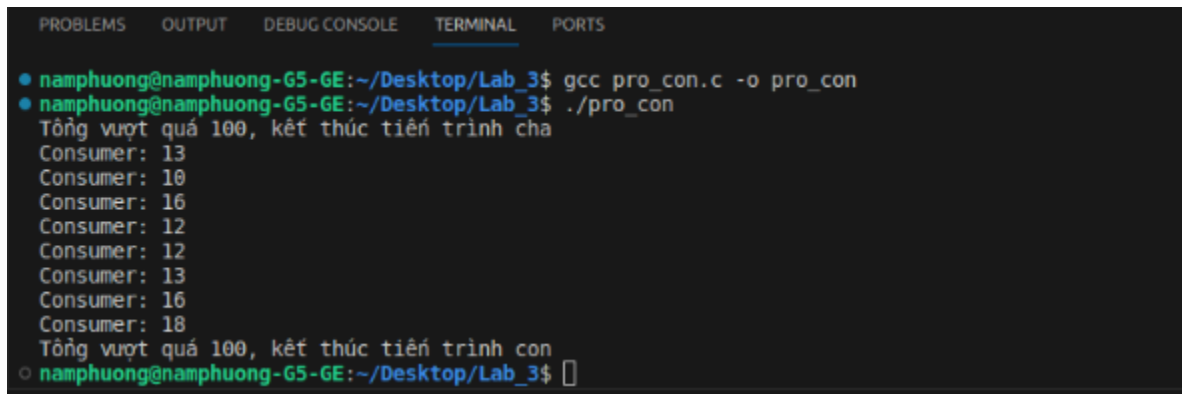
EXPLORER
DESKTOP
Lab_3
> .vscode
F collatz
C collatz.c
C countsh
F count.txt
F factorial
C factorial.c
F firstsh
F hello
C hello.c
C hello.h
C main.c
M makefile
E pro_con
C pro_con.c
F test_exec
C test_exec.c
F test_fork
C test_fork.c
F test_forkwait
C test_forkwait.c
F test_shm_A
C test_shm_A.c
E test_shm_B
C test_shm_B.c
F test_system
C test_system.c
F time
C time.c

18 int main() {
19     // Khởi tạo count trong buffer
20     buffer->count = 0;
21     // Fork để tạo tiến trình con
22     pid_t pid = fork();
23     if (pid < 0) {
24         perror("fork");
25         exit(1);
26     } else if (pid == 0) {
27         // Tiến trình con (Consumer)
28         int sum = 0;
29         while (1) {
30             // Đợi buffer trống
31             while (buffer->count >= BUFFER_SIZE) {
32                 // Tạo số ngẫu nhiên trong khoảng [10, 20]
33                 int data = rand() % 11 + 10;
34             }
35             // Ghi dữ liệu vào buffer
36             buffer->data[buffer->count] = data;
37             buffer->count++;
38             // Kiểm tra điều kiện kết thúc
39             sum += data;
40             if (sum > 100) {
41                 printf("Tổng vượt quá 100, kết thúc tiến trình cha\n");
42                 break;
43             }
44         }
45         // Đợi tiến trình con kết thúc
46         wait(NULL);
47         // Khóa shared memory segment
48         shmctl(shmid, IPC_RMID, NULL);
49     }
50     return 0;
51 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- nanphuong@nanphuong-GS-GE:~/Desktop/Lab_3\$ gcc pro_con.c -o pro_con
- nanphuong@nanphuong-GS-GE:~/Desktop/Lab_3\$./pro_con
- Consumer: 13
- Consumer: 19
- Consumer: 16
- Consumer: 12
- Consumer: 12
- Consumer: 12
- Consumer: 16
- Consumer: 18
- Tổng vượt quá 100, kết thúc tiến trình con
- nanphuong@nanphuong-GS-GE:~/Desktop/Lab_3\$

- Kết quả:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
● namphuong@namphuong-G5-GE:~/Desktop/Lab_3$ gcc pro_con.c -o pro_con
● namphuong@namphuong-G5-GE:~/Desktop/Lab_3$ ./pro_con
Tổng vượt quá 100, kết thúc tiến trình cha
Consumer: 13
Consumer: 10
Consumer: 16
Consumer: 12
Consumer: 12
Consumer: 13
Consumer: 16
Consumer: 18
Tổng vượt quá 100, kết thúc tiến trình con
○ namphuong@namphuong-G5-GE:~/Desktop/Lab_3$
```

- Giải thích kết quả:

+ Chương trình trên tạo hai tiến trình, một tiến trình cha là Producer và một tiến trình con là Consumer, sử dụng shared memory để trao đổi dữ liệu giữa chúng.

+ Tiến trình cha (Producer): Tiến trình cha sinh ra các số ngẫu nhiên trong khoảng từ 10 đến 20 và ghi chúng vào buffer được chia sẻ. Sau đó, tiến trình cha kiểm tra tổng của các số đã ghi vào buffer, nếu tổng vượt quá 100, tiến trình cha kết thúc.

+ Tiến trình con (Consumer): Tiến trình con đợi cho đến khi buffer có ít nhất một số được ghi vào. Sau đó, nó lấy số cuối cùng từ buffer, in ra màn hình và cộng vào tổng. Nếu tổng vượt quá 100, tiến trình con kết thúc.

- Các bước thực hiện:

+ Tiến trình cha (Producer) sẽ sinh ra các số ngẫu nhiên và in chúng ra màn hình.

+ Tiến trình con (Consumer) sẽ tiếp tục in ra màn hình các số mà tiến trình cha ghi vào buffer.

+ Khi tổng của các số vượt quá 100, cả hai tiến trình đều sẽ kết thúc.

+ Lưu ý: Đối với các chương trình sử dụng shared memory như vậy, kết quả cuối cùng có thể thay đổi mỗi lần chạy do sự ngẫu nhiên trong sinh số ngẫu nhiên và thời điểm các tiến trình kết thúc.

2.6. BÀI TẬP ÔN TẬP

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:

- Chương trình được gọi với một đối số là số nguyên dương. Nếu không, màn hình sẽ hiển thị một thông báo lỗi và thoát.
- Tiến trình cha tạo một shared memory segment và gắn kết nó vào vùng nhớ của tiến trình. Sau đó, nó gọi fork() để tạo một tiến trình con.
- Tiến trình con tính toán chuỗi Collatz cho số nguyên được cung cấp và lưu trữ nó trong shared memory segment.
- Tiến trình cha đợi cho tiến trình con kết thúc, sau đó in ra chuỗi kết quả từ shared memory segment.
- Cuối cùng, chúng ta xóa shared memory segment.

- Minh họa:

```
wanthinnn@laiquanthien-22521385:~/Documents/IT007.0211/Lab_3$ gcc collatz.c -o collatz
wanthinnn@laiquanthien-22521385:~/Documents/IT007.0211/Lab_3$ ./collatz 35
Chuỗi Collatz: 35 106 53 160 80 40 20 10 5 16 8 4 2 1
wanthinnn@laiquanthien-22521385:~/Documents/IT007.0211/Lab_3$ ./collatz 3
Chuỗi Collatz: 3 10 5 16 8 4 2 1
wanthinnn@laiquanthien-22521385:~/Documents/IT007.0211/Lab_3$ ./collatz -9
Vui lòng nhập số nguyên dương
wanthinnn@laiquanthien-22521385:~/Documents/IT007.0211/Lab_3$ ./collatz 9
Chuỗi Collatz: 9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
wanthinnn@laiquanthien-22521385:~/Documents/IT007.0211/Lab_3$
```

- Source code:

```
C: collatz.c > main(int, char *[])
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/ipc.h>
5  #include <sys/shm.h>
6  #include <sys/wait.h>
7  #include <sys/types.h>
8
9  #define BUFFER_SIZE 100
10
11 // Struct để lưu trữ dữ liệu trong buffer
12 typedef struct {
13     int sequence[BUFFER_SIZE];
14     int length;
15 } CollatzData;
16
17 int main(int argc, char *argv[]) {
18     if (argc != 2) {
19         printf("Sử dụng: ./collatz <số nguyên dương>\n");
20         return 1;
21     }
22
23     int n = atoi(argv[1]);
24     if (n <= 0) {
25         printf("Vui lòng nhập số nguyên dương\n");
26         return 1;
27     }
28
29     int shmid;
30     key_t key = ftok("collatz", 65); // Tạo key
31     CollatzData *collatzData;
32
33     // Tạo shared memory segment
34     if ((shmid = shmget(key, sizeof(CollatzData), IPC_CREAT | 0666)) < 0) {
35         perror("shmget");
36         exit(1);
37     }
38
39     // Gắn kết vùng nhớ shared với vùng nhớ của tiến trình
40     if ((collatzData = (CollatzData *)shmat(shmid, NULL, 0)) == (CollatzData *) -1) {
41         perror("shmat");
42         exit(1);
43     }
44
45     // Tiến trình con (Producer) tính toán chuỗi Collatz
46     pid_t pid = fork();
47
48     if (pid < 0) {
49         perror("fork");
50         exit(1);
51     } else if (pid == 0) {
52         int length = 0;
53         while (n != 1 && length < BUFFER_SIZE) {
54             collatzData->sequence[length++] = n;
55             if (n % 2 == 0) {
56                 n /= 2;
57             } else {
58                 n = 3 * n + 1;
59             }
60         }
61         collatzData->sequence[length++] = 1;
62         collatzData->length = length;
63         exit(0);
64     } else {
65         // Tiến trình cha (Consumer) in chuỗi kết quả
66         wait(NULL);
67         printf("Chuỗi Collatz: ");
68         for (int i = 0; i < collatzData->length; i++) {
69             printf("%d ", collatzData->sequence[i]);
70         }
71         printf("\n");
72
73         // Xóa shared memory segment
74         shmctl(shmid, IPC_RMID, NULL);
75     }
76
77     return 0;
78 }
```