

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Phạm Quốc Hùng.

Họ và tên: Lê Như Thực

Mã số sinh viên: 24521747

Lớp: IT007.Q112.1

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 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"/>

Tự chấm điểm: 9.5

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

<MSV>_LAB3.pdf

2.5. BÀI TẬP THỰC HÀNH

- 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

```
#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);
}
```

```
int main(int argc, char *argv[])
```

argc = số lượng tham số dòng lệnh + 1 (tên chương trình).

argv[] = mảng các chuỗi chứa tham số dòng lệnh.

```
__pid_t pid;  
pid = fork();
```

fork() tạo một tiến trình con (child process) từ tiến trình cha hiện tại.

Giá trị trả về của fork():

- pid > 0 → đang ở cha, pid = PID của tiến trình con
- pid == 0 → đang ở con, pid = 0
- pid < 0 → lỗi tạo process

```
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) : Chỉ chạy trong process cha.

(long)getpid(): PID của process hiện tại (cha)

(long)getppid() : PID của process cha của cha (grandparent), thường là terminal/parent shell.

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

Nếu có hơn 1 tham số (argc > 2) → in ra số lượng tham số (argc - 1)

wait (NULL) : cha chờ tiến trình con kết thúc trước khi kết thúc.

```
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]);
    }
}
```

if (pid == 0) : Chỉ chạy trong **tiến trình con**.

getpid() : PID con

getppid() : PID của cha (process cha, tức tiến trình gọi fork)

```
for (int i = 1; i < argc; i++)
{
    printf("%s\n", argv[i]);
```

In danh sách tham số: từ argv[1] đến argv[argc-1].

exit(0) : Kết thúc tiến trình, trả về 0.

```
lenhuthuc@LAPTOP-SLH7F0A4:~$ gcc ./test_fork.c -o ./test_fork
lenhuthuc@LAPTOP-SLH7F0A4:~$ ./test_fork ThamSo1 ThamSo2 ThamSo3
PARENTS | PID = 5076 | PPID = 692
PARENTS | There are 3 arguments
CHILDREN | PID = 5077 | PPID = 5076
CHILDREN | List of arguments:
ThamSo1
ThamSo2
ThamSo3
```

- Lệnh gcc được sử dụng để biên dịch mã nguồn thành file thực thi.
- Hàm fork() phân tách chương trình thành hai tiến trình chạy song song:
 - o Tiến trình Cha (PID = 5076).
 - o Tiến trình Con (PID = 5077).
- Luồng thực thi:
 - o Cha: In PID của mình, đếm số lượng tham số và gọi hàm wait() để chờ con kết thúc.

- Con: In PID của mình, in PPID của cha (xác nhận mối quan hệ cha-con) và liệt kê danh sách các tham số dòng lệnh.
- Kết quả: Màn hình hiển thị đầy đủ thông tin định danh của cả hai tiến trình, kết thúc bằng danh sách các đối số mà tiến trình con đã xử lý.

Ví dụ : 3-2

```
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);
}
```

pid = fork() : tạo ra một tiến trình con

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

if (pid > 0) { ... wait(NULL); } : Đây là cha, in PID, PPID, số tham số, rồi chờ con kết thúc

```
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]);
    }
}
```

- Đây là con.
 - execl() sẽ thay thế toàn bộ tiến trình con bằng chương trình ./count.sh.
 - Tất cả code sau execl() sẽ không bao giờ chạy nếu execl() thành công.
 - Nếu execl() thất bại (file không tồn tại hoặc không có quyền thực thi), lúc đó code sau mới chạy.

```
lenhuthuc@LAPTOP-SLH7F0A4:~$ chmod +x count.sh
lenhuthuc@LAPTOP-SLH7F0A4:~$ gcc ./test_execl.c -o ./test_execl
lenhuthuc@LAPTOP-SLH7F0A4:~$ ./test_execl ThamSo1 ThamSo2 ThamSo3
PARENTS | PID = 12646 | PPID = 692
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
lenhuthuc+ 12647 12646 0 16:16 pts/5 00:00:00 /bin/bash ./count.sh 10
lenhuthuc+ 12649 12647 0 16:16 pts/5 00:00:00 grep count.sh
lenhuthuc@LAPTOP-SLH7F0A4:~$
```

Lệnh chmod +x count.sh : Tiến hành cấp quyền thực thi cho file script nhờ vậy hàm execl mới có thể chạy file này.

Tiến trình Cha (PARENTS | PID = 12646 | PPID = 692) : Tiến trình cha (Chương trình C) vẫn chạy bình thường, in ra PID của nó và đợi con.

Phần Script chạy (Tiến trình Con sau khi biến hình)

```
Implementing: ./count.sh
PPID of count.sh:
lenhuth+ 12647 12646 0 16:16 pts/5    00:00:00 /bin/bash ./count.sh 10
lenhuth+ 12649 12647 0 16:16 pts/5    00:00:00 grep count.sh
```

Đây là bằng chứng rõ nhất cho thấy execl đã thành công:

- Dòng "CHILDREN..." biến mất nên giờ đây không còn thấy các dòng CHILDREN | PID... hay List of arguments của code C nữa. Lý do là execl đã xóa bỏ code C trong bộ nhớ của tiến trình con và thay thế bằng code của count.sh.
- **Thông tin định danh:**
 - o Tiến trình chạy script có PID = 12647.
 - o Tiến trình cha (PPID) là 12646 (trùng khớp với PID của chương trình C cha).
- **Tham số truyền vào:** Cột lệnh hiển thị: /bin/bash ./count.sh 10. Điều này xác nhận hàm execl đã truyền thành công tham số "10" vào script.
- **Hiện tượng dòng grep xuất hiện:** Việc dòng lệnh grep count.sh xuất hiện (với PID = 12649) là do cơ chế lọc song song: khi lệnh ps quét hệ thống, lệnh grep cũng đang thực thi nên nó tự bắt được chính mình.

Điều này khớp hoàn toàn với tham số bạn truyền trong code C: execl(..., "10", NULL).

Ví dụ 3-3:

```
1  #####  
2  # IT007 Operating System #  
3  # <Your name>, <your Student ID> #  
4  # File: test_system.c #  
5  #####  
6  #include <stdio.h>  
7  #include <stdlib.h>  
8  #include <unistd.h>  
9  #include <sys/wait.h>  
10 #include <sys/types.h>  
11  
12 int main(int argc, char *argv[])  
13 {  
14     printf("PARENTS | PID = %ld | PPID = %ld\n",  
15            (long)getpid(), (long)getppid());  
16     if (argc > 2)  
17         printf("PARENTS | There are %d arguments\n", argc - 1);  
18  
19     system("./count.sh 10");  
20     printf("PARENTS | List of arguments: \n");  
21     for (int i = 1; i < argc; i++)  
22     {  
23         printf("%s\n", argv[i]);  
24     }  
25     exit(0);  
26 }
```

- *printf("PARENTS | PID = ... ")*: Chương trình in ra PID (Mã tiến trình) của chính nó và PPID (Mã của cha nó).
- *if (argc > 2)...*: Kiểm tra và in số lượng tham số dòng lệnh truyền vào khi chạy file C (ví dụ: ThamSo1, ThamSo2...).
- *system("./count.sh 10")*: Hàm system() sẽ tạm dừng chương trình C hiện tại. Nó gọi hệ điều hành để tạo một tiến trình con (thường là /bin/sh) để chạy lệnh ./count.sh 10. Script count.sh sẽ chạy, đếm số, in kết quả. Chương trình C sẽ Wait cho đến khi lệnh system chạy xong hoàn toàn.
- Sau khi count.sh chạy xong, quyền điều khiển được trả lại cho chương trình C. Chương trình tiếp tục chạy các dòng code bên dưới:
 - o *printf ("PARENTS | List of arguments: \n")*;: In tiêu đề.
 - o Vòng lặp for: In ra danh sách các tham số (ThamSo1, ThamSo2...) đã truyền vào ban đầu.

```
lenhuthuc@LAPTOP-SLH7F0A4:~$ gcc ./test_system.c -o ./test_system  
lenhuthuc@LAPTOP-SLH7F0A4:~$ ./test_system ThamSo1 ThamSo2 ThamSo3  
PARENTS | PID = 28609 | PPID = 692  
PARENTS | There are 3 arguments  
Implementing: ./count.sh  
PPID of count.sh:  
lenhuthuc 28610 28609 0 17:45 pts/5 00:00:00 sh -c -- ./count.sh 10  
lenhuthuc 28611 28610 0 17:45 pts/5 00:00:00 /bin/bash ./count.sh 10  
lenhuthuc 28613 28611 0 17:45 pts/5 00:00:00 grep count.sh  
PARENTS | List of arguments:  
ThamSo1  
ThamSo2  
ThamSo3
```

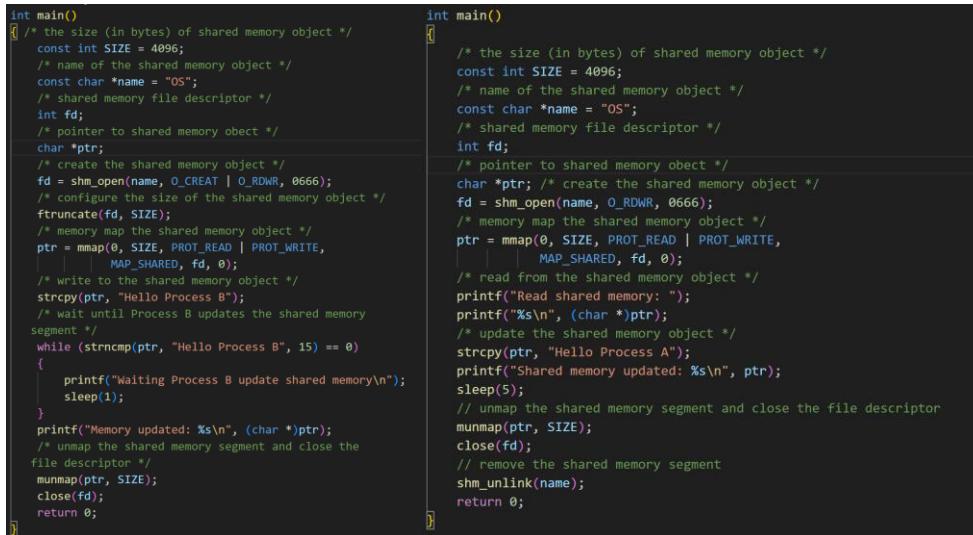
PARENTS | PID = 28609: Đây là tiến trình chính của file test_system.

Lệnh system() bắt đầu hoạt động. Khi gọi system("./count.sh 10"), nó không tự chạy ngay mà nhờ một "người trung gian" là shell (sh):

- *sh -c -- ./count.sh 10 (PID 28610, PPID 28609)* : Đây là tiến trình con do system tạo ra. Cha của nó chính là chương trình C (28609).
- */bin/bash ./count.sh 10 (PID 28611, PPID 28610)* : Shell (sh) gọi trình thông dịch bash để chạy script. Cha của nó là sh (28610).
- *grep count.sh (PID 28613, PPID 28611)* : Lệnh tìm kiếm nằm trong script. Cha của nó là script bash (28611).

Cuối cùng là quay lại file C in ra các tham số.

Ví dụ 3-4:



```
int main()
{
    /* the size (in bytes) of shared memory object */
    const int SIZE = 4096;
    /* name of the shared memory object */
    const char *name = "05";
    /* shared memory file descriptor */
    int fd;
    /* pointer to shared memory object */
    char *ptr;
    /* create the shared memory object */
    fd = shm_open(name, O_CREAT | O_RDWR, 0666);
    /* configure the size of the shared memory object */
    ftruncate(fd, SIZE);
    /* memory map the shared memory object */
    ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE,
               MAP_SHARED, fd, 0);
    /* write to the shared memory object */
    strcpy(ptr, "Hello Process B");
    /* wait until Process B updates the shared memory
       segment */
    while (strcmp(ptr, "Hello Process B", 15) == 0)
    {
        printf("Waiting Process B update shared memory\n");
        sleep(1);
    }
    printf("Memory updated: %s\n", (char *)ptr);
    /* unmap the shared memory segment and close the
       file descriptor */
    munmap(ptr, SIZE);
    close(fd);
    return 0;
}

int main()
{
    /* the size (in bytes) of shared memory object */
    const int SIZE = 4096;
    /* name of the shared memory object */
    const char *name = "05";
    /* shared memory file descriptor */
    int fd;
    /* pointer to shared memory object */
    char *ptr; /* create the shared memory object */
    fd = shm_open(name, O_RDWR, 0666);
    /* memory map the shared memory object */
    ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE,
               MAP_SHARED, fd, 0);
    /* read from the shared memory object */
    printf("Read shared memory: ");
    printf("%s\n", (char *)ptr);
    /* update the shared memory object */
    strcpy(ptr, "Hello Process A");
    printf("Shared memory updated: %s\n", ptr);
    sleep(5);
    /* unmap the shared memory segment and close the file descriptor */
    munmap(ptr, SIZE);
    close(fd);
    /* remove the shared memory segment */
    shm_unlink(name);
    return 0;
}
```

- *test_shm_A.c* :
 - Tạo vùng nhớ chung, thiết lập kích thước và ghi dòng chữ "Hello Process B".
 - Vòng lặp: Nó liên tục kiểm tra vùng nhớ (polling). Nếu nội dung vẫn là "Hello Process B" thì in ra "Waiting..." và ngủ tiếp. Nó chỉ thoát khi nội dung bị ai đó thay đổi.
- File B (*test_shm_B.c*) - Người phản hồi:
 - Mở vùng nhớ chung đã có sẵn (do A tạo).
 - Đọc nội dung cũ ("Hello Process B") in ra màn hình.

- Ghi đè nội dung mới "Hello Process A" vào chính vị trí đó (đây là hành động giúp A thoát khỏi vòng lặp chờ).
- Cuối cùng, B dọn dẹp và xóa vùng nhớ (shm_unlink).

```
● lenhuthuc@LAPTOP-SLH7F0A4:~$ gcc ./test_shm_A.c -o ./test_shm_A
● lenhuthuc@LAPTOP-SLH7F0A4:~$ ./test_shm_A
Waiting Process B update shared memory
Memory updated: Hello Process A
```

```
● lenhuthuc@LAPTOP-SLH7F0A4:~$ gcc ./test_shm_B.c -o ./test_shm_B
● lenhuthuc@LAPTOP-SLH7F0A4:~$ ./test_shm_B
Read shared memory: Hello Process B
Shared memory updated: Hello Process A
```

- **Terminal A (Chạy trước):** Tạo vùng nhớ, ghi dòng chữ "Hello Process B" rồi vào vòng lặp chờ (in Waiting... liên tục).
- **Terminal B (Chạy sau):** Đọc được tin nhắn của A, sau đó ghi đè nội dung mới là "Hello Process A" vào vùng nhớ đó.
- **Kết quả tại A:** Ngay khi B ghi đè xong, A phát hiện dữ liệu đã thay đổi -> Thoát vòng lặp chờ -> In ra "Memory updated: Hello Process A" và kết thúc.

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.

```

test_system.c | C test_system.c | C test_shm_A.c | C test_shm_B.c | C time.c | ...
C time.c > main(int, char **)
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/time.h>
5 #include <sys/wait.h>
6
7 int main(int argc, char *argv[]) {
8     if (argc < 2) {
9         printf("Sử dụng: ./time <lệnh_shell>\n");
10        return 1;
11    }
12    struct timeval start, end;
13    pid_t pid;
14
15    gettimeofday(&start, NULL);
16
17    pid = fork();
18
19    if (pid < 0) {
20        perror("Fork failed");
21        return 1;
22    } else if (pid == 0) {
23        execvp(argv[1], argv[1], NULL);
24        perror("Lỗi thực thi lệnh");
25        exit(1);
26    } else {
27        wait(NULL);
28
29        gettimeofday(&end, NULL);
30        double time_taken;
31        time_taken = (end.tv_sec - start.tv_sec) * 1e6;
32        time_taken = (time_taken + (end.tv_usec - start.tv_usec)) * 1e-6;
33
34        printf("Thời gian thực thi: %.5f giây\n", time_taken);
35    }
36
37    return 0;
38}

```

- Hàm `gettimeofday(&start, NULL)` : Bấm đồng hồ bắt đầu tính giờ
- `pid = fork()` dùng để tạo ra một bản sao của tiến trình hiện tại, hệ điều hành nhân đôi chương trình.
- `execvp(argv[1], argv[1], NULL)` :
 - o `argv[1]`: Là chuỗi "ls" (lấy từ tham số dòng lệnh).
 - o Cơ chế: Hàm này ra lệnh cho hệ điều hành: "*Hãy xóa sạch code của chương trình hiện tại (time.c đang chạy) trong RAM, và tải code của chương trình ls vào thay thế, rồi chạy nó ngay lập tức*".
- `wait(NULL)` :
 - o Đây là lệnh Blocking, tiến trình Cha sẽ bị sleep, không làm gì cả.
 - o Nó chờ tín hiệu SIGCHLD từ hệ điều hành báo rằng tiến trình con của bản thân báo rằng tiến trình con đã thực thi xong và tắt. Lúc đó Cha mới chạy tiếp.
- `gettimeofday(&end, NULL)` : Lấy thời gian kết thúc ngay khi con vừa tắt.
- `time_taken = (Giây cuối - Giây đầu) * 1 triệu + (Microgiây cuối - Microgiây đầu)`.
- Kết quả chia cho `1e6` (1.000.000) để quy đổi tổng số micro giây về đơn vị Giây.

```
① lenhuthuc@LAPTOP-SLH7F0A4:~$ ./time ls
time.c time
count.sh hello test_exec1 test_fork test_fork_wait.c test_shm_A.c test_shm_B.c test_system.c time
count.txt hello.c test_exec1.c test_fork.c test_shm_A test_shm_B test_system 'test_system.c' time.c
Thời gian thực thi: 0.00342 giây
time.c: command not found
② lenhuthuc@LAPTOP-SLH7F0A4:~$
```

Khi thực thi lệnh `./time ls`, kết quả hiển thị trên màn hình được chia thành ba phần nối tiếp nhau: đầu tiên là danh sách các tập tin hiện có trong thư mục do tiến trình con (đã hóa thân thành lệnh `ls`) in ra màn hình, ngay sau khi việc liệt kê hoàn tất, dòng thông báo "Thời gian thực thi: 0.00342 giây" xuất hiện, đây là kết quả do tiến trình cha tính toán và in ra sau khi đã chờ tiến trình con kết thúc, cuối cùng là dòng lỗi "`time.c: command not found`", xuất hiện do người dùng vô tình nhập tên file mã nguồn vào terminal sau khi chương trình chính đã chạy xong, không liên quan đến logic hoạt động của mã nguồn.

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”

The screenshot shows a terminal window with two code snippets. On the left, the code for `main.c` is displayed, and on the right, the code for `count.sh` is displayed. The `main.c` code handles a SIGINT signal to print "count.sh has stopped" and exit. It then forks, with the child process executing `count.sh` with argument "120". The parent process waits for the child to finish. The `count.sh` script uses a loop to count from 1 to 120, printing each number and pausing for 0.5 seconds between prints.

```
main.c > handle_sigint(int)
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <signal.h>
5 #include <sys/wait.h>
6
7 void handle_sigint(int sig)
8 {
9     printf("\ncount.sh has stoppped\n");
10    exit(0);
11 }
12
13 int main()
14 {
15     signal(SIGINT, handle_sigint);
16     printf("Welcome to IT007, I am 24521747!\n");
17
18     pid_t pid = fork();
19
20     if (pid < 0)
21     {
22         perror("Fork failed");
23         exit(1);
24     }
25     else if (pid == 0)
26     {
27         execl("./count.sh", "./count.sh", "120", NULL);
28         exit(1);
29     }
30     else
31     {
32         wait(NULL);
33     }
34     return 0;
}

$ count.sh
1 #!/bin/bash
2 # File: count.sh
3 target=$1
4 i=1
5 while [ $i -le $target ]; do
6     echo "Counting: $i"
7     i=$((i+1))
8     sleep 0.5
9 done
```

file main.c :

- *signal(SIGINT, handle_sigint)* : Đăng ký xử lý tín hiệu. Khi người dùng bấm Ctrl + C, thay vì chương trình bị kill ngay lập tức, nó sẽ chạy hàm handle_sigint để in dòng chữ "count.sh has stopped" rồi mới thoát an toàn.
- *fork()* tách chương trình thành 2 nhánh chạy song song:
 - Tiến trình Cha ($pid > 0$): Chạy vào nhánh else, dùng lệnh `wait(NULL)` để tạm dừng và chờ đợi tiến trình con thực hiện xong nhiệm vụ.

- Tiến trình Con (pid == 0): Chạy vào nhánh if, dùng lệnh execl để thay thế toàn bộ mã nguồn hiện tại thành file script count.sh. Nó truyền tham số "120" cho script này.

File count.sh - File này là "nhân viên thực thi": nhận lệnh và thực hiện việc đếm :

- **target=\$1:** Nhận tham số đầu tiên được truyền từ lệnh execl bên C (chính là số 120).
- **Vòng lặp while:** Thực hiện đếm từ 1 đến 120 (\$target).
- **sleep 0.5:** Mỗi lần đếm sẽ tạm dừng 0.5 giây. Mục đích là để quá trình đếm diễn ra chậm rãi, giúp bạn có đủ thời gian để bấm **Ctrl + C**.

```
● lenhuthuc@LAPTOP-SLH7F0A4:~$ gcc ./main.c -o ./main
● lenhuthuc@LAPTOP-SLH7F0A4:~$ ./main
Welcome to IT007, I am 24521747!
Counting: 1
Counting: 2
Counting: 3
Counting: 4
Counting: 5
Counting: 6
^C
count.sh has stoppped
❖ lenhuthuc@LAPTOP-SLH7F0A4:~$
```

Chương trình main tạo ra một tiến trình con để thực thi script count.sh đếm từ 1 đến 120. Hàm signal() được sử dụng ở tiến trình cha để bắt tín hiệu SIGINT sinh ra khi người dùng nhấn tổ hợp phím Ctrl + C. Thay vì bị hệ điều hành tắt đột ngột, chương trình sẽ chạy hàm handle_sigint để in thông báo "count.sh has stopped" rồi mới kết thúc an toàn.

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.
- Tiền trình cha đóng vai trò là Producer, tạo một số ngẫu nhiên trong khoảng [10, 20] và ghi dữ liệu vào buffer
- Tiền 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

```
C producer_consumer.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/mman.h>
5  #include <sys/wait.h>
6  #include <semaphore.h>
7  #include <time.h>
8  #include <fcntl.h>
9
10 struct SharedData {
11     unsigned char buffer[10];
12     int in;
13     int out;
14     int sum;
15
16     sem_t sem_empty;
17     sem_t sem_full;
18     sem_t sem_mutex;
19 };
20
21 int main() {
22     struct SharedData *shared = mmap(NULL, sizeof(struct SharedData),
23                                     PROT_READ | PROT_WRITE,
24                                     MAP_SHARED | MAP_ANONYMOUS, -1, 0);
25
26     if (shared == MAP_FAILED) {
27         perror("mmap failed");
28         return 1;
29     }
30
31     shared->in = 0;
32     shared->out = 0;
33     shared->sum = 0;
34
35     sem_init(&shared->sem_empty, 1, 10);
36     sem_init(&shared->sem_full, 1, 0);
37     sem_init(&shared->sem_mutex, 1, 1);
38
39     pid_t pid = fork();
40
41     if (pid < 0) {
42         perror("Fork failed");
43         return 1;
44     }
45
46     if (pid > 0) {
47         srand(time(NULL));
48         while (1) {
49             if (shared->sum > 100) break;
50
51             int val = rand() % 11 + 10;
52
53             sem_wait(&shared->sem_empty);
54             sem_wait(&shared->sem_mutex);
55             if (shared->sum > 100) {
56                 sem_post(&shared->sem_mutex);
57                 sem_post(&shared->sem_empty);
58                 break;
59             }
60
61             shared->buffer[shared->in] = val;
62             printf("[Producer] Ghi %d vào buffer[%d]\n", val, shared->in);
63             shared->in = (shared->in + 1) % 10;
64         }
65     }
66
67     waitpid(pid, NULL, 0);
68
69     sem_destroy(&shared->sem_empty);
70     sem_destroy(&shared->sem_full);
71     sem_destroy(&shared->sem_mutex);
72
73     munmap(shared, sizeof(struct SharedData));
74
75     exit(0);
76 }
```

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Phạm Quốc Hùng.

```
64         sem_post(&shared->sem_mutex);
65         sem_post(&shared->sem_full);
66     }
67     sleep(1);
68 }
69 wait(NULL);
70 printf("Producer kết thúc.\n");
71
72 sem_destroy(&shared->sem_empty);
73 sem_destroy(&shared->sem_full);
74 sem_destroy(&shared->sem_mutex);
75 munmap(shared, sizeof(struct SharedData));
76 }
77 else {
78     while (1) {
79         sem_wait(&shared->sem_full);
80         sem_wait(&shared->sem_mutex);
81
82         int val = shared->buffer[shared->out];
83         shared->sum += val;
84
85         printf("\t\t[Consumer] Đọc %d từ buffer[%d] -> Tổng = %d\n",
86               val, shared->out, shared->sum);
87
88         shared->out = (shared->out + 1) % 10;
89         if (shared->sum > 100) {
90             printf("\t\t[Consumer] Tổng > 100. Dừng!\n");
91             sem_post(&shared->sem_mutex);
92             sem_post(&shared->sem_empty);
93             exit(0);
94         }
95     }
96 }
97 sem_post(&shared->sem_mutex);
98 sem_post(&shared->sem_empty);
99 }
100 }
101 return 0;
102 }
```

```
● lenhuthuc@LAPTOP-SLH7F0A4:~$ gcc ./producer_consumer.c -o ./producer_consumer
● lenhuthuc@LAPTOP-SLH7F0A4:~$ ./producer_consumer
[Producer] Ghi 18 vào buffer[0]
    [Consumer] Đọc 18 từ buffer[0] -> Tổng = 18
[Producer] Ghi 17 vào buffer[1]
    [Consumer] Đọc 17 từ buffer[1] -> Tổng = 35
[Producer] Ghi 16 vào buffer[2]
    [Consumer] Đọc 16 từ buffer[2] -> Tổng = 51
[Producer] Ghi 19 vào buffer[3]
    [Consumer] Đọc 19 từ buffer[3] -> Tổng = 70
[Producer] Ghi 17 vào buffer[4]
    [Consumer] Đọc 17 từ buffer[4] -> Tổng = 87
[Producer] Ghi 10 vào buffer[5]
    [Consumer] Đọc 10 từ buffer[5] -> Tổng = 97
[Producer] Ghi 13 vào buffer[6]
    [Consumer] Đọc 13 từ buffer[6] -> Tổng = 110
    [Consumer] Tổng > 100. Dừng!
Producer kết thúc.
✧ lenhuthuc@LAPTOP-SLH7F0A4:~$
```

Chương trình hiện thực hóa bài toán "producer_consumer" thông qua kỹ thuật bộ nhớ chia sẻ kết hợp với semaphore để đồng bộ hóa quyền truy cập vào bộ đệm vòng, đảm bảo dữ liệu không bị ghi đè hay đọc sai lệch giữa hai tiến trình song song. Trong kết quả hiển thị, tiến trình cha (Producer) lặp lại ghi các số ngẫu nhiên trong khoảng [10, 20] và ghi vào bộ đệm, ngay sau đó tiến trình con (Consumer) lấy ra để cộng dồn vào biến sum, quy trình này diễn ra tuần tự nhịp nhàng (Ghi 18 -> Đọc 18 -> Tổng 18...) cho đến khi con số

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Phạm Quốc Hùng.

13 được thêm vào, nâng tổng lên 110 (vượt ngưỡng 100), khiến Consumer phát hiện điều kiện dừng, in thông báo và đánh thức Producer để cùng kết thúc chương trình.

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.

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Phạm Quốc Hùng.

```
C collatz.c > main(int argc, char *argv[])
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/mman.h>
5  #include <sys/wait.h>
6  #include <sys/types.h>
7
8  #define MAX_ITEMS 200
9  struct SharedData
10 {
11     long sequence[MAX_ITEMS];
12     int size;
13 };
14
15 int main(int argc, char *argv[])
16 {
17
18     if (argc < 2)
19     {
20         printf("Sử dụng: ./collatz <số nguyên dương>\n");
21         return 1;
22     }
23
24     int n = atoi(argv[1]);
25     if (n <= 0)
26     {
27         printf("Lỗi: Vui lòng nhập một số nguyên dương lớn hơn 0.\n");
28         return 1;
29     }
30
31     struct SharedData *shared = mmap(NULL, sizeof(struct SharedData),
32                                     PROT_READ | PROT_WRITE,
33                                     MAP_SHARED | MAP_ANONYMOUS, -1, 0);
34
35     if (shared == MAP_FAILED)
36     {
37         perror("mmap failed");
38         return 1;
39     }
40
41     shared->size = 0;
42
43     pid_t pid = fork();
44
45     if (pid < 0)
46     {
47         perror("Fork failed");
48         return 1;
49     }
50     else if (pid == 0)
51     {
52
53         long current = n;
54         shared->sequence[shared->size++] = current;
55
56         while (current != 1)
57         {
58
59             if (shared->size >= MAX_ITEMS)
60             {
61                 break;
62             }
63
64             if (current % 2 == 0)
65             {
66                 current /= 2;
67             }
68             else
69             {
70                 current = 3 * current + 1;
71             }
72
73             shared->sequence[shared->size++] = current;
74
75         }
76
77         if (shared->size >= MAX_ITEMS)
78         {
79             break;
80         }
81
82     }
83
84     if (pid > 0)
85     {
86         waitpid(pid, NULL, 0);
87     }
88
89     if (shared->size >= MAX_ITEMS)
90     {
91         break;
92     }
93
94     if (shared->size >= MAX_ITEMS)
95     {
96         break;
97     }
98
99     if (shared->size >= MAX_ITEMS)
100    {
101        break;
102    }
103
104    if (shared->size >= MAX_ITEMS)
105    {
106        break;
107    }
108
109    if (shared->size >= MAX_ITEMS)
110    {
111        break;
112    }
113
114    if (shared->size >= MAX_ITEMS)
115    {
116        break;
117    }
118
119    if (shared->size >= MAX_ITEMS)
120    {
121        break;
122    }
123
124    if (shared->size >= MAX_ITEMS)
125    {
126        break;
127    }
128
129    if (shared->size >= MAX_ITEMS)
130    {
131        break;
132    }
133
134    if (shared->size >= MAX_ITEMS)
135    {
136        break;
137    }
138
139    if (shared->size >= MAX_ITEMS)
140    {
141        break;
142    }
143
144    if (shared->size >= MAX_ITEMS)
145    {
146        break;
147    }
148
149    if (shared->size >= MAX_ITEMS)
150    {
151        break;
152    }
153
154    if (shared->size >= MAX_ITEMS)
155    {
156        break;
157    }
158
159    if (shared->size >= MAX_ITEMS)
160    {
161        break;
162    }
163
164    if (shared->size >= MAX_ITEMS)
165    {
166        break;
167    }
168
169    if (shared->size >= MAX_ITEMS)
170    {
171        break;
172    }
173
174    if (shared->size >= MAX_ITEMS)
175    {
176        break;
177    }
178
179    if (shared->size >= MAX_ITEMS)
180    {
181        break;
182    }
183
184    if (shared->size >= MAX_ITEMS)
185    {
186        break;
187    }
188
189    if (shared->size >= MAX_ITEMS)
190    {
191        break;
192    }
193
194    if (shared->size >= MAX_ITEMS)
195    {
196        break;
197    }
198
199    if (shared->size >= MAX_ITEMS)
200    {
201        break;
202    }
203
204    if (shared->size >= MAX_ITEMS)
205    {
206        break;
207    }
208
209    if (shared->size >= MAX_ITEMS)
210    {
211        break;
212    }
213
214    if (shared->size >= MAX_ITEMS)
215    {
216        break;
217    }
218
219    if (shared->size >= MAX_ITEMS)
220    {
221        break;
222    }
223
224    if (shared->size >= MAX_ITEMS)
225    {
226        break;
227    }
228
229    if (shared->size >= MAX_ITEMS)
230    {
231        break;
232    }
233
234    if (shared->size >= MAX_ITEMS)
235    {
236        break;
237    }
238
239    if (shared->size >= MAX_ITEMS)
240    {
241        break;
242    }
243
244    if (shared->size >= MAX_ITEMS)
245    {
246        break;
247    }
248
249    if (shared->size >= MAX_ITEMS)
250    {
251        break;
252    }
253
254    if (shared->size >= MAX_ITEMS)
255    {
256        break;
257    }
258
259    if (shared->size >= MAX_ITEMS)
260    {
261        break;
262    }
263
264    if (shared->size >= MAX_ITEMS)
265    {
266        break;
267    }
268
269    if (shared->size >= MAX_ITEMS)
270    {
271        break;
272    }
273
274    if (shared->size >= MAX_ITEMS)
275    {
276        break;
277    }
278
279    if (shared->size >= MAX_ITEMS)
280    {
281        break;
282    }
283
284    if (shared->size >= MAX_ITEMS)
285    {
286        break;
287    }
288
289    if (shared->size >= MAX_ITEMS)
290    {
291        break;
292    }
293
294    if (shared->size >= MAX_ITEMS)
295    {
296        break;
297    }
298
299    if (shared->size >= MAX_ITEMS)
300    {
301        break;
302    }
303
304    if (shared->size >= MAX_ITEMS)
305    {
306        break;
307    }
308
309    if (shared->size >= MAX_ITEMS)
310    {
311        break;
312    }
313
314    if (shared->size >= MAX_ITEMS)
315    {
316        break;
317    }
318
319    if (shared->size >= MAX_ITEMS)
320    {
321        break;
322    }
323
324    if (shared->size >= MAX_ITEMS)
325    {
326        break;
327    }
328
329    if (shared->size >= MAX_ITEMS)
330    {
331        break;
332    }
333
334    if (shared->size >= MAX_ITEMS)
335    {
336        break;
337    }
338
339    if (shared->size >= MAX_ITEMS)
340    {
341        break;
342    }
343
344    if (shared->size >= MAX_ITEMS)
345    {
346        break;
347    }
348
349    if (shared->size >= MAX_ITEMS)
350    {
351        break;
352    }
353
354    if (shared->size >= MAX_ITEMS)
355    {
356        break;
357    }
358
359    if (shared->size >= MAX_ITEMS)
360    {
361        break;
362    }
363
364    if (shared->size >= MAX_ITEMS)
365    {
366        break;
367    }
368
369    if (shared->size >= MAX_ITEMS)
370    {
371        break;
372    }
373
374    if (shared->size >= MAX_ITEMS)
375    {
376        break;
377    }
378
379    if (shared->size >= MAX_ITEMS)
380    {
381        break;
382    }
383
384    if (shared->size >= MAX_ITEMS)
385    {
386        break;
387    }
388
389    if (shared->size >= MAX_ITEMS)
390    {
391        break;
392    }
393
394    if (shared->size >= MAX_ITEMS)
395    {
396        break;
397    }
398
399    if (shared->size >= MAX_ITEMS)
400    {
401        break;
402    }
403
404    if (shared->size >= MAX_ITEMS)
405    {
406        break;
407    }
408
409    if (shared->size >= MAX_ITEMS)
410    {
411        break;
412    }
413
414    if (shared->size >= MAX_ITEMS)
415    {
416        break;
417    }
418
419    if (shared->size >= MAX_ITEMS)
420    {
421        break;
422    }
423
424    if (shared->size >= MAX_ITEMS)
425    {
426        break;
427    }
428
429    if (shared->size >= MAX_ITEMS)
430    {
431        break;
432    }
433
434    if (shared->size >= MAX_ITEMS)
435    {
436        break;
437    }
438
439    if (shared->size >= MAX_ITEMS)
440    {
441        break;
442    }
443
444    if (shared->size >= MAX_ITEMS)
445    {
446        break;
447    }
448
449    if (shared->size >= MAX_ITEMS)
450    {
451        break;
452    }
453
454    if (shared->size >= MAX_ITEMS)
455    {
456        break;
457    }
458
459    if (shared->size >= MAX_ITEMS)
460    {
461        break;
462    }
463
464    if (shared->size >= MAX_ITEMS)
465    {
466        break;
467    }
468
469    if (shared->size >= MAX_ITEMS)
470    {
471        break;
472    }
473
474    if (shared->size >= MAX_ITEMS)
475    {
476        break;
477    }
478
479    if (shared->size >= MAX_ITEMS)
480    {
481        break;
482    }
483
484    if (shared->size >= MAX_ITEMS)
485    {
486        break;
487    }
488
489    if (shared->size >= MAX_ITEMS)
490    {
491        break;
492    }
493
494    if (shared->size >= MAX_ITEMS)
495    {
496        break;
497    }
498
499    if (shared->size >= MAX_ITEMS)
500    {
501        break;
502    }
503
504    if (shared->size >= MAX_ITEMS)
505    {
506        break;
507    }
508
509    if (shared->size >= MAX_ITEMS)
510    {
511        break;
512    }
513
514    if (shared->size >= MAX_ITEMS)
515    {
516        break;
517    }
518
519    if (shared->size >= MAX_ITEMS)
520    {
521        break;
522    }
523
524    if (shared->size >= MAX_ITEMS)
525    {
526        break;
527    }
528
529    if (shared->size >= MAX_ITEMS)
530    {
531        break;
532    }
533
534    if (shared->size >= MAX_ITEMS)
535    {
536        break;
537    }
538
539    if (shared->size >= MAX_ITEMS)
540    {
541        break;
542    }
543
544    if (shared->size >= MAX_ITEMS)
545    {
546        break;
547    }
548
549    if (shared->size >= MAX_ITEMS)
550    {
551        break;
552    }
553
554    if (shared->size >= MAX_ITEMS)
555    {
556        break;
557    }
558
559    if (shared->size >= MAX_ITEMS)
560    {
561        break;
562    }
563
564    if (shared->size >= MAX_ITEMS)
565    {
566        break;
567    }
568
569    if (shared->size >= MAX_ITEMS)
570    {
571        break;
572    }
573
574    if (shared->size >= MAX_ITEMS)
575    {
576        break;
577    }
578
579    if (shared->size >= MAX_ITEMS)
580    {
581        break;
582    }
583
584    if (shared->size >= MAX_ITEMS)
585    {
586        break;
587    }
588
589    if (shared->size >= MAX_ITEMS)
590    {
591        break;
592    }
593
594    if (shared->size >= MAX_ITEMS)
595    {
596        break;
597    }
598
599    if (shared->size >= MAX_ITEMS)
600    {
601        break;
602    }
603
604    if (shared->size >= MAX_ITEMS)
605    {
606        break;
607    }
608
609    if (shared->size >= MAX_ITEMS)
610    {
611        break;
612    }
613
614    if (shared->size >= MAX_ITEMS)
615    {
616        break;
617    }
618
619    if (shared->size >= MAX_ITEMS)
620    {
621        break;
622    }
623
624    if (shared->size >= MAX_ITEMS)
625    {
626        break;
627    }
628
629    if (shared->size >= MAX_ITEMS)
630    {
631        break;
632    }
633
634    if (shared->size >= MAX_ITEMS)
635    {
636        break;
637    }
638
639    if (shared->size >= MAX_ITEMS)
640    {
641        break;
642    }
643
644    if (shared->size >= MAX_ITEMS)
645    {
646        break;
647    }
648
649    if (shared->size >= MAX_ITEMS)
650    {
651        break;
652    }
653
654    if (shared->size >= MAX_ITEMS)
655    {
656        break;
657    }
658
659    if (shared->size >= MAX_ITEMS)
660    {
661        break;
662    }
663
664    if (shared->size >= MAX_ITEMS)
665    {
666        break;
667    }
668
669    if (shared->size >= MAX_ITEMS)
670    {
671        break;
672    }
673
674    if (shared->size >= MAX_ITEMS)
675    {
676        break;
677    }
678
679    if (shared->size >= MAX_ITEMS)
680    {
681        break;
682    }
683
684    if (shared->size >= MAX_ITEMS)
685    {
686        break;
687    }
688
689    if (shared->size >= MAX_ITEMS)
690    {
691        break;
692    }
693
694    if (shared->size >= MAX_ITEMS)
695    {
696        break;
697    }
698
699    if (shared->size >= MAX_ITEMS)
700    {
701        break;
702    }
703
704    if (shared->size >= MAX_ITEMS)
705    {
706        break;
707    }
708
709    if (shared->size >= MAX_ITEMS)
710    {
711        break;
712    }
713
714    if (shared->size >= MAX_ITEMS)
715    {
716        break;
717    }
718
719    if (shared->size >= MAX_ITEMS)
720    {
721        break;
722    }
723
724    if (shared->size >= MAX_ITEMS)
725    {
726        break;
727    }
728
729    if (shared->size >= MAX_ITEMS)
730    {
731        break;
732    }
733
734    if (shared->size >= MAX_ITEMS)
735    {
736        break;
737    }
738
739    if (shared->size >= MAX_ITEMS)
740    {
741        break;
742    }
743
744    if (shared->size >= MAX_ITEMS)
745    {
746        break;
747    }
748
749    if (shared->size >= MAX_ITEMS)
750    {
751        break;
752    }
753
754    if (shared->size >= MAX_ITEMS)
755    {
756        break;
757    }
758
759    if (shared->size >= MAX_ITEMS)
760    {
761        break;
762    }
763
764    if (shared->size >= MAX_ITEMS)
765    {
766        break;
767    }
768
769    if (shared->size >= MAX_ITEMS)
770    {
771        break;
772    }
773
774    if (shared->size >= MAX_ITEMS)
775    {
776        break;
777    }
778
779    if (shared->size >= MAX_ITEMS)
780    {
781        break;
782    }
783
784    if (shared->size >= MAX_ITEMS)
785    {
786        break;
787    }
788
789    if (shared->size >= MAX_ITEMS)
790    {
791        break;
792    }
793
794    if (shared->size >= MAX_ITEMS)
795    {
796        break;
797    }
798
799    if (shared->size >= MAX_ITEMS)
800    {
801        break;
802    }
803
804    if (shared->size >= MAX_ITEMS)
805    {
806        break;
807    }
808
809    if (shared->size >= MAX_ITEMS)
810    {
811        break;
812    }
813
814    if (shared->size >= MAX_ITEMS)
815    {
816        break;
817    }
818
819    if (shared->size >= MAX_ITEMS)
820    {
821        break;
822    }
823
824    if (shared->size >= MAX_ITEMS)
825    {
826        break;
827    }
828
829    if (shared->size >= MAX_ITEMS)
830    {
831        break;
832    }
833
834    if (shared->size >= MAX_ITEMS)
835    {
836        break;
837    }
838
839    if (shared->size >= MAX_ITEMS)
840    {
841        break;
842    }
843
844    if (shared->size >= MAX_ITEMS)
845    {
846        break;
847    }
848
849    if (shared->size >= MAX_ITEMS)
850    {
851        break;
852    }
853
854    if (shared->size >= MAX_ITEMS)
855    {
856        break;
857    }
858
859    if (shared->size >= MAX_ITEMS)
860    {
861        break;
862    }
863
864    if (shared->size >= MAX_ITEMS)
865    {
866        break;
867    }
868
869    if (shared->size >= MAX_ITEMS)
870    {
871        break;
872    }
873
874    if (shared->size >= MAX_ITEMS)
875    {
876        break;
877    }
878
879    if (shared->size >= MAX_ITEMS)
880    {
881        break;
882    }
883
884    if (shared->size >= MAX_ITEMS)
885    {
886        break;
887    }
888
889    if (shared->size >= MAX_ITEMS)
890    {
891        break;
892    }
893
894    if (shared->size >= MAX_ITEMS)
895    {
896        break;
897    }
898
899    if (shared->size >= MAX_ITEMS)
900    {
901        break;
902    }
903
904    if (shared->size >= MAX_ITEMS)
905    {
906        break;
907    }
908
909    if (shared->size >= MAX_ITEMS)
910    {
911        break;
912    }
913
914    if (shared->size >= MAX_ITEMS)
915    {
916        break;
917    }
918
919    if (shared->size >= MAX_ITEMS)
920    {
921        break;
922    }
923
924    if (shared->size >= MAX_ITEMS)
925    {
926        break;
927    }
928
929    if (shared->size >= MAX_ITEMS)
930    {
931        break;
932    }
933
934    if (shared->size >= MAX_ITEMS)
935    {
936        break;
937    }
938
939    if (shared->size >= MAX_ITEMS)
940    {
941        break;
942    }
943
944    if (shared->size >= MAX_ITEMS)
945    {
946        break;
947    }
948
949    if (shared->size >= MAX_ITEMS)
950    {
951        break;
952    }
953
954    if (shared->size >= MAX_ITEMS)
955    {
956        break;
957    }
958
959    if (shared->size >= MAX_ITEMS)
960    {
961        break;
962    }
963
964    if (shared->size >= MAX_ITEMS)
965    {
966        break;
967    }
968
969    if (shared->size >= MAX_ITEMS)
970    {
971        break;
972    }
973
974    if (shared->size >= MAX_ITEMS)
975    {
976        break;
977    }
978
979    if (shared->size >= MAX_ITEMS)
980    {
981        break;
982    }
983
984    if (shared->size >= MAX_ITEMS)
985    {
986        break;
987    }
988
989    if (shared->size >= MAX_ITEMS)
990    {
991        break;
992    }
993
994    if (shared->size >= MAX_ITEMS)
995    {
996        break;
997    }
998
999    if (shared->size >= MAX_ITEMS)
1000   {
1001      break;
1002   }
1003 }
```

```
63
64     if (current % 2 == 0)
65     {
66         current = current / 2;
67     }
68     else
69     {
70         current = 3 * current + 1;
71     }
72     shared->sequence[shared->size++] = current;
73 }
74
75 exit(0);
76 }
77 else
78 {
79     wait(NULL);
80
81     printf("Chuỗi Collatz cho %d là:\n", n);
82     for (int i = 0; i < shared->size; i++)
83     {
84         printf("%ld", shared->sequence[i]);
85         if (i < shared->size - 1)
86         {
87             printf(", ");
88         }
89     }
90     printf("\n");
91     munmap(shared, sizeof(struct SharedData));
92
93 }
94 return 0;
```

- **Xử lý tham số đầu vào:** Kiểm tra người dùng có nhập tham số hay không và tham số đó có phải số nguyên dương (> 0) không.

- **Thiết lập bộ nhớ chia sẻ (Shared Memory):**

- o Sử dụng hàm mmap() với cờ MAP_SHARED | MAP_ANONYMOUS.
Điều này tạo ra một vùng nhớ trong RAM mà cả tiến trình cha và con đều có thể đọc/ghi.
- o Định nghĩa một cấu trúc (struct) gồm mảng chứa dãy số và biến đếm số lượng phần tử.

- **Tạo tiến trình (Fork):**

- o Gọi fork() để tách chương trình.

- **Xử lý tại Tiến trình con (Child):**

- o Thực hiện thuật toán Collatz:
 - Nếu n chẵn: $n = n / 2$
 - Nếu n lẻ: $n = 3n + 1$
- o Ghi từng giá trị tính toán được vào vùng nhớ chia sẻ.
- o Dừng khi $n = 1$.

2. Xử lý tại Tiến trình cha (Parent):

- Gọi wait(NULL) để đảm bảo con đã tính toán và ghi xong dữ liệu.
- Đọc dữ liệu từ vùng nhớ chia sẻ và in ra màn hình.
- Dọn dẹp bộ nhớ (munmap).

```
● lenhuthuc@LAPTOP-SLH7F0A4:~$ ./collatz 8
Chuỗi Collatz cho 8 là:
8, 4, 2, 1
● lenhuthuc@LAPTOP-SLH7F0A4:~$ ./collatz 35
Chuỗi Collatz cho 35 là:
35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1
✧ lenhuthuc@LAPTOP-SLH7F0A4:~$ █
```

- **Khởi tạo:** Khi lệnh ./collatz 35 được chạy, tiến trình cha xin hệ điều hành một vùng nhớ chung (mmap).
- **Tách nhánh:** Hàm fork() tạo ra tiến trình con. Lúc này cả hai tiến trình cùng nhìn thấy vùng nhớ chung đó.
- **Tính toán (Con):**
 - Tiến trình con bắt đầu với số 35.
 - 35 là lẻ $\rightarrow 35 \times 3 + 1 = 106$. (Ghi 106 vào buffer).
 - 106 là chẵn $\rightarrow 106 / 2 = 53$. (Ghi 53 vào buffer).
 - ... Quá trình lặp lại cho đến khi gặp số 1.
 - Sau khi ghi số 1 vào buffer, tiến trình con thoát (exit).
- **Đồng bộ và Hiển thị (Cha):**
 - Tiến trình cha bị chặn lại ở dòng wait(NULL). Nó không làm gì cả cho đến khi con thoát.
 - Ngay khi con thoát (nghĩa là đã tính xong dãy số), cha "tỉnh dậy".
 - Cha truy cập vào vùng nhớ chung (nơi con vừa ghi dữ liệu) và in toàn bộ dãy số ra màn hình.