LAB 03 – GROUP 5

CLASS : IT007.019.1 && IT007.019.2

Date: 23/10/2023

Members:

1. Hà Huy Hoàng | 22520460
2. Nguyễn Duy Hoàng | 22520467
3. Nguyễn Hoàng Hiệp | 22520452
4. Nguyễn Hoàng Phúc | 22521129

SUMMARY

A. 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:

File test.c:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>  //Sử dụng hàm fork()
#include <sys/wait.h>
#include <sys/types.h>
int main(int argc, char *argv[])   // argc là số lượng đối số được truyền vào
{
    // Tạo một tiến trình con
    __pid_t pid;
    pid = fork();
    // Tiến trình cha
    if (pid > 0)
    {
```

```
        printf("PARENTS | PID = %ld | PPID = %ld\n", (long)getpid(),
(long)getppid()); //in ra pid và ppid của tiến trình hiện tại
        if (argc > 2)
            printf("PARENTS | There are %d arguments\n", argc - 1);
        wait(NULL); // Đợi tiến trình con kết thúc
    }
    // Tiến trình con
    if (pid == 0)   //thông tin về quy trình con và các đối số được in ra màn
hình
    {
        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]); // In ra các đối số từ dòng lệnh
        }
    }
    exit(0);   //Kết thúc tiến trình.
}
```

1. Đây là thông tin về tiến trình cha (parent process). PID = 4314 là ID của tiến trình cha và PPID = 3929 là ID của tiến trình ông nội (grandparent process).
2. Đây cho biết rằng bạn đã chạy chương trình với 3 đối số đầu vào, là ThamSo1, ThamSo2, và ThamSo3.
3. Đây là thông tin về tiến trình con (child process). PID = 4315 là ID của tiến trình con và PPID = 4314 là ID của tiến trình cha.
4. Đây là danh sách các đối số đầu vào mà bạn đã truyền vào chương trình. Trong trường hợp này, các đối số là ThamSo1, ThamSo2, và ThamSo3.

```
nguyenhoangphuc_22521129@LAPTOP-021S54DV:~$ gcc -o test test.c
nguyenhoangphuc_22521129@LAPTOP-021S54DV:~$ ./test ThamSo1 ThamSo2 ThamSo3
1. PARENTS | PID = 4314 | PPID = 3929
2. PARENTS | There are 3 arguments
3. CHILDREN | PID = 4315 | PPID = 4314
4. CHILDREN | List of arguments:
   ThamSo1
   ThamSo2
   ThamSo3
```

- Ví dụ 3-2:

File count.sh

```
#!/bin/bash
# Đây là shebang, nó chỉ định rằng script này sẽ được thực thi bằng /bin/bash

echo "Implementing: $0"
# In ra tên của script hiện tại

echo "PPID of count.sh: "
# In ra chuỗi "PPID of count.sh: "

ps -ef | grep count.sh
# Lệnh 'ps -ef' sẽ liệt kê tất cả các tiến trình đang chạy.
# Kết quả sau đó được chuyển tới lệnh 'grep count.sh',
# lọc ra những dòng có chứa chuỗi 'count.sh'

i=1
# Khởi tạo biến i với giá trị là 1

while [ $i -le $1 ]
# Vòng lặp while sẽ chạy miễn là giá trị của i nhỏ hơn hoặc bằng tham số đầu tiên được
truyền vào script ($1)

do
    echo $i >> count.txt
    # In giá trị của i vào file count.txt. Dấu '>>' có nghĩa là nối vào cuối file nếu
file đã tồn tại.

    i=$((i + 1))
    # Tăng giá trị của i lên 1

    sleep 1
    # Dừng script trong 1 giây

done
# Kết thúc vòng lặp while

exit 0
# Kết thúc script với mã thoát là 0, biểu thị rằng script đã hoàn thành mà không có
lỗi xảy ra.
```

File test_execl.c:

```c
/*#######################################
# University of Information Technology
# IT007 Operating System
#
# <Ha Huy Hoang>, <22520460>
# 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();
    // Tạo một tiến trình con bằng cách sử dụng hàm fork()

    if (pid > 0)
    {
        // Đây là phần mã của tiến trình cha

        printf("PARENTS | PID = %ld | PPID = %ld\n", (long)getpid(),
(long)getppid());
        // In ra PID và PPID của tiến trình cha

        if (argc > 2)
            printf("PARENTS | There are %d arguments\n", argc - 1);
            // Nếu có nhiều hơn 2 đối số, in ra số lượng đối số

        wait(NULL);
        // Tiến trình cha chờ tiến trình con kết thúc
    }

    if (pid == 0)
    {
        // Đây là phần mã của tiến trình con

        execl("./count.sh", "./count.sh", "10", NULL);
        // Tiến trình con thực thi script bash "count.sh" với đối số là "10"
```

```
        printf("CHILDREN | PID = %ld | PPID = %ld\n", (long)getpid(),
(long)getppid());
        // In ra PID và PPID của tiến trình con

        printf("CHILDREN | List of arguments: \n");
        for (int i = 1; i < argc; i++)
        {
            printf("%s\n", argv[i]);
            // In ra danh sách các đối số được truyền vào chương trình
        }
    }

    exit(0);
    // Kết thúc chương trình với mã thoát là 0, biểu thị rằng chương trình đã
hoàn thành mà không có lỗi xảy ra.
}
```

```
hoang-22520460@DESKTOP-PA2DBO6:~/Lab3$ chmod +x count.sh
hoang-22520460@DESKTOP-PA2DBO6:~/Lab3$ ./test_execl ThamSo1 ThamSo2 ThamSo3
1. PARENTS | PID = 2179 | PPID = 1923
2. PARENTS | There are 3 arguments
3. Implementing: ./count.sh
4. PPID of count.sh:
5. hoang-2+    2180    2179  0 13:57 pts/5    00:00:00 /bin/bash ./count.sh 10
6. hoang-2+    2182    2180  0 13:57 pts/5    00:00:00 grep count.sh
hoang-22520460@DESKTOP-PA2DBO6:~/Lab3$
```

1. Đây là output từ tiến trình cha trong chương trình test_execl.c. Nó in ra PID (Process ID) của tiến trình cha là 2179 và PPID (Parent Process ID) của nó là 1923.
2. Đây là output từ tiến trình cha, nó in ra số lượng đối số được truyền vào chương trình test_execl.c (không tính tên chương trình). Trong trường hợp này, bạn đã truyền vào 3 đối số: “ThamSo1”, “ThamSo2”, và “ThamSo3”.
3. Đây là output từ script bash count.sh, nó in ra tên của script hiện tại.
4. Tiếp tục là output từ script bash count.sh, nó in ra chuỗi "PPID of count.sh: ".
5. Đây là output từ lệnh ps -ef | grep count.sh trong script bash count.sh. Nó in ra thông tin về tiến trình đang chạy script count.sh. Trong đó, PID của tiến trình này là 2180 và PPID của nó (là tiến trình cha đã tạo ra nó) là 2179.
6. Đây cũng là output từ lệnh ps -ef | grep count.sh, nhưng nó liên quan đến tiến trình grep mà lệnh này đã tạo ra để lọc kết quả. PID của tiến trình grep này là 2182 và PPID của nó (là tiến trình đã tạo ra nó) là 2180.

- Ví dụ 3-3:

File test_system.c:

```c
/*#######################################
# University of Information Technology
# IT007 Operating System
#
# Nguyen Duy Hoang, 22520467
# 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[]) {
    // In ra ID của tiến trình hiện tại và ID của tiến trình cha (nếu có)
    printf("PARENTS | PID = %ld | PPID = %ld\n", (long)getpid(),
(long)getppid());

    // Nếu có nhiều hơn 2 tham số đầu vào, in ra số lượng tham số
    if (argc > 2) {
        printf("PARENTS | There are %d arguments\n", argc - 1);
    }

    // Thực thi script shell count.sh với tham số là 10 trong một tiến trình con
    system("./count.sh 10");

    // In ra danh sách các tham số đầu vào
    printf("PARENTS | List of arguments: \n");
    for (int i = 1; i < argc; i++) {
        printf("%s\n", argv[i]);
    }

    // Kết thúc chương trình với mã lỗi 0 (tức là không có lỗi xảy ra)
    exit(0);
}
```

```
nguyenduyhoang-22520467@HOANGND04:~$ gcc test_system.c -o test_system
nguyenduyhoang-22520467@HOANGND04:~$ ./test_system ThamSo1 ThamSo2 ThamSo3
1.PARENTS | PID = 1940 | PPID = 1807
2.PARENTS | There are 3 arguments
3.Implementing: ./count.sh
4.PPID of count.sh:
  nguyend+  1941  1940  0 13:58 pts/1    00:00:00 sh -c ./count.sh 10
  nguyend+  1942  1941  0 13:58 pts/1    00:00:00 /bin/bash ./count.sh 10
  nguyend+  1944  1942  0 13:58 pts/1    00:00:00 grep count.sh
5. PARENTS | List of arguments:
  ThamSo1
  ThamSo2
  ThamSo3
```

1. Đây là thông tin về tiến trình cha (parent process). PID = 1940 là ID của tiến trình cha và PPID = 1807 là ID của tiến trình ông nội (grandparent process).
2. Đây cho biết rằng bạn đã chạy chương trình với 3 đối số đầu vào, là ThamSo1, ThamSo2, và ThamSo3.
3. Đây cho biết rằng chương trình đang thực thi script shell count.sh.
4. Đây là thông tin về các tiến trình liên quan đến việc thực thi script shell count.sh. Cụ thể, sh -c ./count.sh 10 là tiến trình shell được tạo ra bởi hàm system() để thực thi script, và /bin/bash ./count.sh 10 là tiến trình của script shell count.sh đang chạy.
5. Đây là danh sách các đối số đầu vào mà bạn đã truyền vào chương trình.

- Ví dụ 3-4:

a) Process A:

```c
#include <stdio.h>  // Thư viện chuẩn C cho I/O
#include <stdlib.h> // Thư viện chuẩn C cho các hàm như malloc, free, exit...
#include <string.h> // Thư viện chuẩn C cho các hàm xử lý chuỗi
#include <fcntl.h>  // Thư viện POSIX cho file control
#include <sys/shm.h> // Thư viện POSIX cho shared memory
#include <sys/stat.h> // Thư viện POSIX cho các hàm xử lý file status
#include <unistd.h>  // Thư viện POSIX cho các hàm như close, write, read...

#include <sys/mman.h>  // Thư viện POSIX cho memory management

int main()
{
    /* the size (in bytes) of shared memory object */
    const int SIZE = 4096;  // Kích thước (theo byte) của đối tượng bộ nhớ chia sẻ
    /* name of the shared memory object */
    const char *name = "OS";  // Tên của đối tượng bộ nhớ chia sẻ
    /* shared memory file descriptor */
```

```c
    int fd;  // File descriptor cho bộ nhớ chia sẻ
    /* pointer to shared memory obect */
    char *ptr;  // Con trỏ đến đối tượng bộ nhớ chia sẻ
    /* create the shared memory object */
    fd = shm_open(name, O_CREAT | O_RDWR,0666);  // Tạo đối tượng bộ nhớ chia
sẻ
    /* configure the size of the shared memory object */
    ftruncate(fd, SIZE);  // Cấu hình kích thước của đối tượng bộ nhớ chia sẻ
    /* memory map the shared memory object */
    ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);  // Ánh xạ
bộ nhớ của đối tượng bộ nhớ chia sẻ
    /* write to the shared memory object */
    strcpy(ptr, "Hello Process B");  // Ghi vào đối tượng bộ nhớ chia sẻ
       /* wait until Process B updates the shared memory segment */
    while (strncmp(ptr, "Hello Process B", 15) == 0)  // Chờ cho đến khi
Process B cập nhật phân đoạn bộ nhớ chia sẻ
    {
        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);  // Hủy ánh xạ phân đoạn bộ nhớ chia sẻ và đóng file
descriptor
    close(fd);
       return 0;
}
```

Chương trình A sẽ tạo ra một đối tượng bộ nhớ chia sẻ và ghi chuỗi "Hello Process B" vào đó, sau đó liên tục in ra thông báo "Waiting Process B update shared memory" cho đến khi chương trình B cập nhật bộ nhớ chia sẻ.

```
nguyenhoanghiep-22520452@HiepNH:~$ gcc processA.c -o testA
nguyenhoanghiep-22520452@HiepNH:~$ ./testA
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
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
nguyenhoanghiep-22520452@HiepNH:~$
```

b) Process B:

```
#include <stdio.h>  // Thư viện chuẩn C cho I/O
#include <stdlib.h> // Thư viện chuẩn C cho các hàm như malloc, free, exit...
#include <string.h> // Thư viện chuẩn C cho các hàm xử lý chuỗi
#include <fcntl.h>  // Thư viện POSIX cho file control
#include <sys/shm.h> // Thư viện POSIX cho shared memory
#include <sys/stat.h> // Thư viện POSIX cho các hàm xử lý file status
#include <unistd.h>  // Thư viện POSIX cho các hàm như close, write, read...
```

```c
#include <sys/mman.h>  // Thư viện POSIX cho memory management

int main()
{
    /* the size (in bytes) of shared memory object */
    const int SIZE = 4096;  // Kích thước (theo byte) của đối tượng bộ nhớ chia sẻ
    /* name of the shared memory object */
    const char *name = "OS";  // Tên của đối tượng bộ nhớ chia sẻ
    /* shared memory file descriptor */
    int fd;  // File descriptor cho bộ nhớ chia sẻ
    /* pointer to shared memory obect */
    char *ptr;  // Con trỏ đến đối tượng bộ nhớ chia sẻ
    /* create the shared memory object */
    fd = shm_open(name, O_RDWR,0666);  // Mở đối tượng bộ nhớ chia sẻ đã tồn tại
    /* memory map the shared memory object */
    ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);  // Ánh xạ bộ nhớ của đối tượng bộ nhớ chia sẻ
    /* read from the shared memory object */
    printf("Read shared memory: ");
    printf("%s\n",(char *)ptr);  // Đọc từ đối tượng bộ nhớ chia sẻ
    /* update the shared memory object */
    strcpy(ptr, "Hello Process A");  // Cập nhật đối tượng bộ nhớ chia sẻ
    printf("Shared memory updated: %s\n", ptr);
    sleep(5);
    // unmap the shared memory segment and close the file descriptor
    munmap(ptr, SIZE);  // Hủy ánh xạ phân đoạn bộ nhớ chia sẻ và đóng file descriptor
    close(fd);
   // remove the shared memory segment
    shm_unlink(name);  // Gỡ bỏ đối tượng bộ nhớ chia sẻ
    return 0;
}
```

Chương trình B chia chạy sẽ đọc chuỗi từ bộ nhớ chia sẻ ("Hello Process B"), sau đó in ra chuỗi đã đoc và cập nhật bộ nhớ chia sẻ với chuỗi “Hello Process A” và in ra chuỗi mới.

```
nguyenhoanghiep-22520452@HiepNH:~$ gcc processB.c -o testB
nguyenhoanghiep-22520452@HiepNH:~$ ./testB
Read shared memory: Hello Process B
Shared memory updated: Hello Process A
nguyenhoanghiep-22520452@HiepNH:~$
```

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.

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

int main(int argc, char *argv[]) {
    // Kiểm tra xem có đúng một đối số được truyền vào không
    if (argc != 2) {
        printf("Cách sử dụng: ./time <lệnh>\n");
        return 1;
    }

    struct timeval start, end;

    // Lấy thời gian hiện tại và lưu vào biến start
    gettimeofday(&start, NULL);

    // Tạo một tiến trình con
    pid_t pid = fork();

    // Nếu đây là tiến trình con
    if (pid == 0) {
        // Thực thi lệnh được truyền vào từ dòng lệnh
        if (execl("/bin/sh", "/bin/sh", "-c", argv[1], NULL) == -1) {
            printf("Lỗi: không thể thực thi lệnh\n");
            return 1;
        }
    }
    // Nếu đây là tiến trình cha
    else if (pid > 0) {
        // Chờ tiến trình con kết thúc
        wait(NULL);

        // Lấy thời gian hiện tại và lưu vào biến end
        gettimeofday(&end, NULL);

        // Tính toán thời gian thực thi bằng cách lấy thời gian kết thúc trừ đi thời
gian bắt đầu
```

```
        double ans = (end.tv_sec - start.tv_sec) + ((end.tv_usec -
start.tv_usec)/1000000.0);

        // In ra thời gian thực thi
        printf("Thời gian thực thi: %.5f giây\n", ans);
    }
    // Nếu có lỗi xảy ra khi tạo tiến trình con
    else {
        printf("Lỗi: không thể tạo tiến trình con\n");
        return 1;
    }

    return 0;
}
```

```
nguyenduyhoang-22520467@HOANGND04:~$ gcc time.c -o time
nguyenduyhoang-22520467@HOANGND04:~$ ./time ls
'C:\Users\Legion 5 Pro\.ssh\id_rsa'      'C:\Users\Legion 5 pro\.ssh'      Myweb     count.txt   elif_control2.sh   for_loop2.sh   hello.c   monhoc.txt          password.sh   test_execl.c   test_system.c   time.c
'C:\Users\Legion 5 Pro\.ssh\id_rsa.pub'  'C:\Users\Legion 5 pro\.ssh.pub'  count.sh  dkhp.sh     for_loop.sh        hello          lab_2.sh  'nguyen duy hoang'   test_execl    test_system    time
Thời gian thực thi: 0.13540 giây
```

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>!”

```
#include <stdio.h>
int main() {
    char studentID[] = "22521129";
    printf("Welcome to IT007, I am %s!\n", studentID);
    return 0;
}
```

```
nguyenhoangphuc_22521129@LAPTOP-021S54DV:~$ gcc -o bt4 bt4.c
nguyenhoangphuc_22521129@LAPTOP-021S54DV:~$ ./bt4
Welcome to IT007, I am 22521129!
```

➢ Thực thi file script count.sh với số lần đếm là 120

```
#!/bin/bash
echo "Implementing: $0"
echo "PPID of count.sh: "
ps -ef | grep count.sh
i=1
while [ $i -le $1 ]
do
echo $i >> count.txt
i=$((i + 1))
sleep 1
done
exit 0
```

```
nguyenhoangphuc_22521129@LAPTOP-021S54DV:~$ chmod +x count.sh
nguyenhoangphuc_22521129@LAPTOP-021S54DV:~$ ./count.sh 120
Implementing: ./count.sh
PPID of count.sh:
nguyenh+    6268    6123  0 14:59 pts/2    00:00:00 /bin/bash ./count.sh 120
nguyenh+    6270    6268  0 14:59 pts/2    00:00:00 grep count.sh
```

➢ Trước khi count.sh đếm đến 120, bấm CTRL+C để dừng tiến trình này

```
nguyenhoangphuc_22521129@LAPTOP-021S54DV:~$ chmod +x count.sh
nguyenhoangphuc_22521129@LAPTOP-021S54DV:~$ ./count.sh 120
Implementing: ./count.sh
PPID of count.sh:
nguyenh+    6593    6123  0 15:04 pts/2    00:00:00 /bin/bash ./count.sh 120
nguyenh+    6595    6593  0 15:04 pts/2    00:00:00 grep count.sh
^C
nguyenhoangphuc_22521129@LAPTOP-021S54DV:~$
```

➢ File count.sh in ra "count.sh has stopped" khi có tín hiệu dừng

```bash
#!/bin/bash

echo "Implementing: $0"
echo "PPID of count.sh: "
ps -ef | grep count.sh

cleanup() {
    echo "count.sh has stopped"
    exit 1
}

trap cleanup INT

i=1
while [ $i -le $1 ]
do
    echo $i >> count.txt
    i=$((i + 1))
    sleep 1
done
```

```
nguyenhoangphuc_22521129@LAPTOP-021S54DV:~$ chmod +x count.sh
nguyenhoangphuc_22521129@LAPTOP-021S54DV:~$ ./count.sh 120
Implementing: ./count.sh
PPID of count.sh:
nguyenh+    6268    6123  0 14:59 pts/2    00:00:00 /bin/bash ./count.sh 120
nguyenh+    6270    6268  0 14:59 pts/2    00:00:00 grep count.sh
^Ccount.sh has stopped
```

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

- Sử dụng kỹ thuật shared-memory để tạo một boundedbuffer 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

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/wait.h>
int main() {
    const int BUFFER_SIZE = 10; // Định nghĩa kích thước buffer
    const char *name = "OS"; // Tên của shared memory object

    // Tạo shared memory object
    int shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

    // Cấu hình kích thước của shared memory object
    ftruncate(shm_fd, sizeof(int) * (BUFFER_SIZE + 1));

    // Mapping shared memory object
    int *buffer = mmap(0, sizeof(int) * (BUFFER_SIZE + 1), PROT_READ |
PROT_WRITE, MAP_SHARED, shm_fd, 0);

    // Kiểm tra xem mmap có thành công không
    if (buffer == MAP_FAILED) {
        perror("mmap");
        return 1;
```

```
    }

    int *sum = buffer + BUFFER_SIZE; // Lưu tổng vào cuối buffer
    *sum = 0; // Khởi tạo tổng bằng 0

    pid_t pid = fork(); // Tạo một tiến trình con

    if (pid == 0) { // Tiến trình con (Consumer)
        while (*sum <= 100) { // Chạy cho đến khi tổng lớn hơn 100
            for (int i = 0; i < BUFFER_SIZE; i++) { // Duyệt qua từng phần tử
trong buffer
                if (buffer[i] != 0) { // Nếu phần tử không rỗng
                    printf("Consumer reads %d\n", buffer[i]); // In ra giá trị
phần tử
                    *sum += buffer[i]; // Cộng giá trị phần tử vào tổng
                    buffer[i] = 0; // Đặt lại giá trị phần tử thành 0
                }
            }
        }
        printf("Sum > 100, Consumer stops\n"); // In ra thông báo khi tổng lớn
hơn 100
    } else if (pid > 0) { // Tiến trình cha (Producer)
        while (*sum <= 100) { // Chạy cho đến khi tổng lớn hơn 100
            for (int i = 0; i < BUFFER_SIZE; i++) { // Duyệt qua từng phần tử
trong buffer
                if (buffer[i] == 0) { // Nếu phần tử rỗng
                    buffer[i] = rand() % 11 + 10; // Tạo một số ngẫu nhiên trong
khoảng [10,20] và gán vào phần tử
                    printf("Producer writes %d\n", buffer[i]); // In ra giá trị
đã ghi vào phần tử
                }
            }
        }
        printf("Sum > 100, Producer stops\n"); // In ra thông báo khi tổng lớn
hơn 100
        wait(NULL); // Đợi tiến trình con kết thúc trước khi tiến trình cha kết
thúc

        // Xóa shared memory object
        shm_unlink(name);
    } else { // fork failed
        perror("fork");
        return 1;
    }

    return 0;
```

```
}
```

```
nguyenhoanghiep-22520452@HiepNH:~$ gcc producer_consumer.c -o bai4
producer_consumer.c: In function 'main':
producer_consumer.c:55:9: warning: implicit declaration of function 'wait' [-Wimplicit-function-declaration]
   55 |         wait(NULL); // Đợi tiến trình con kết thúc trước khi tiến trình cha kết thúc
      |         ^~~~
nguyenhoanghiep-22520452@HiepNH:~$ gcc producer_consumer.c -o bai4
nguyenhoanghiep-22520452@HiepNH:~$ ./bai4
Producer writes 16
Producer writes 20
Producer writes 16
Producer writes 12
Producer writes 11
Producer writes 14
Producer writes 10
Producer writes 16
Producer writes 13
Producer writes 11
Consumer reads 16
Consumer reads 20
Producer writes 18
Consumer reads 16
Producer writes 17
Consumer reads 12
Producer writes 15
Consumer reads 11
Producer writes 13
Consumer reads 14
Producer writes 17
Consumer reads 10
Producer writes 14
Consumer reads 16
Producer writes 19
Consumer reads 13
Producer writes 20
Consumer reads 11
Producer writes 12
Sum > 100, Consumer stops
Producer writes 10
Sum > 100, Producer stops
nguyenhoanghiep-22520452@HiepNH:~$
```

B. 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

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

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        printf("Vui long nhap mot so duong");
        return 1;
    }

    int n = atoi(argv[1]);
    if (n <= 0)
    {
        printf("Vui long nhap mot so duong\n");
        return 1;
    }

    pid_t pid;
    int *buffer;

    // Tạo bộ nhớ chia sẻ
    buffer = mmap(NULL, sizeof(int) * (n + 1), PROT_READ | PROT_WRITE,
MAP_SHARED | MAP_ANONYMOUS, -1, 0);

    pid = fork();
    if (pid == 0)
    { // Tiến trình con
        int i = 0;
        while (n != 1)
        {
```

```
            buffer[i++] = n;
            if (n % 2 == 0)
                n /= 2;
            else
                n = 3 * n + 1;
        }
        buffer[i] = n; // Thêm số cuối cùng (1) vào chuỗi
        buffer[i + 1] = -1; // Đánh dấu kết thúc chuỗi
    }
    else
    { // Tiến trình cha
        wait(NULL); // Đợi tiến trình con kết thúc

        // In chuỗi
        for (int i = 0; buffer[i] != -1; i++)
            printf("%d ", buffer[i]);
        printf("\n");

        // Dọn dẹp bộ nhớ chia sẻ
        munmap(buffer, sizeof(int) * (n + 1));
    }

    return 0;
}
```

```
hoang-22520460@DESKTOP-PA2DBO6:~/Lab3$ gcc 4_1.c -o collatz
hoang-22520460@DESKTOP-PA2DBO6:~/Lab3$ ./collatz 8
8 4 2 1
hoang-22520460@DESKTOP-PA2DBO6:~/Lab3$
```