Họ và tên: Đỗ Phương Duy
Mã số sinh viên: 23520362
Lớp: KHTN2023

HỆ ĐIỀU HÀNH
BÁO CÁO LAB 3

CHECKLIST (Đánh dấu x khi hoàn thành)
Lưu ý mỗi câu phải làm đủ 3 yêu cầu

I. Bài tập thực hành

	BT 1	BT 2	BT 3	BT 4
Trình bày cách làm	☒	☒	☒	☒
Chụp hình minh chứng	☒	☒	☒	☒
Giải thích kết quả	☒	☒	☒	☒

II. Bài tập ôn tập

	a
Trình bày cách làm	☒
Chụp hình minh chứng	☒
Giải thích kết quả	☒

Tự chấm điểm: 10

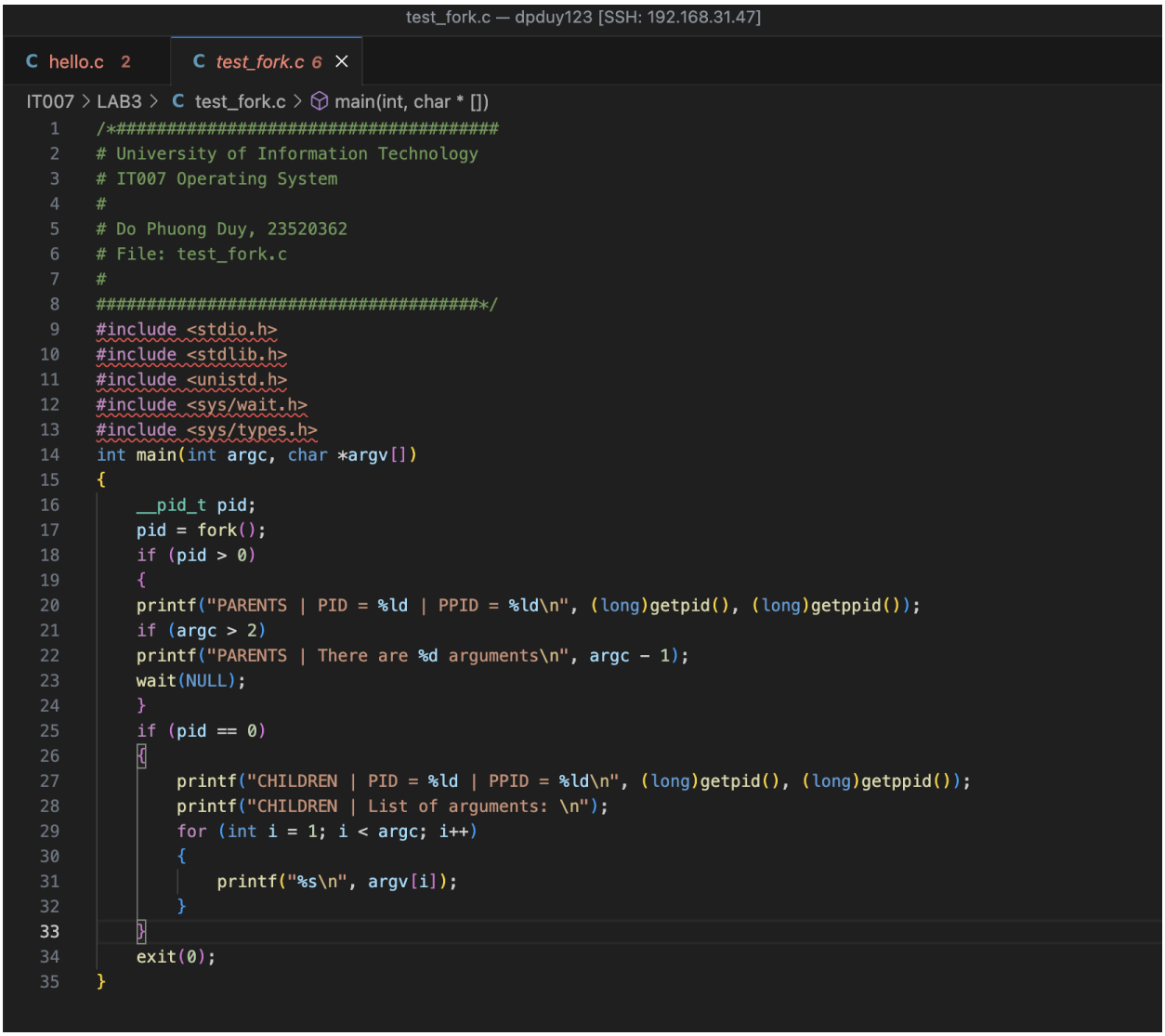
**Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp:*
<MSSV>_LABx.pdf

I. Bài tập thực hành:

1.

A. Ví dụ 3-1:

Code:




```c
/*#####################################
# University of Information Technology
# IT007 Operating System
#
# Do Phuong Duy, 23520362
# 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 code:

Hàm main nhận số lượng tham số argc và mảng các tham số (agrv[])

__pid_t pid và pid = fork() là lệnh tạo tiến trình con.

Fork() tạo ra tiến trình con, gọi X là giá trị hàm này trả về. Nếu:

- X > 0 là tiến trình cha (PID của tiến trình con)
- X = 0 là tiến trình con
- X < 0 là đã có lỗi xảy ra

Nếu `pid` lớn hơn 0, nghĩa là đoạn mã này đang chạy trong tiến trình cha.

Nó in ra PID (Process ID) và PPID (Parent Process ID) của tiến trình cha.

Nếu có nhiều hơn 2 tham số (tính cả tên chương trình), nó sẽ in ra số lượng tham số.

Cuối cùng, tiến trình cha gọi `wait(NULL)` để chờ tiến trình con hoàn thành trước khi tiếp tục.

Nếu pid bằng 0, đoạn mã này đang chạy trong tiến trình con.
Chương trình in ra PID và PPID của tiến trình con.
Sau đó, in ra danh sách các tham số được truyền vào từ argv, bắt đầu từ argv[1] (bỏ qua tên chương trình ở argv[0]).

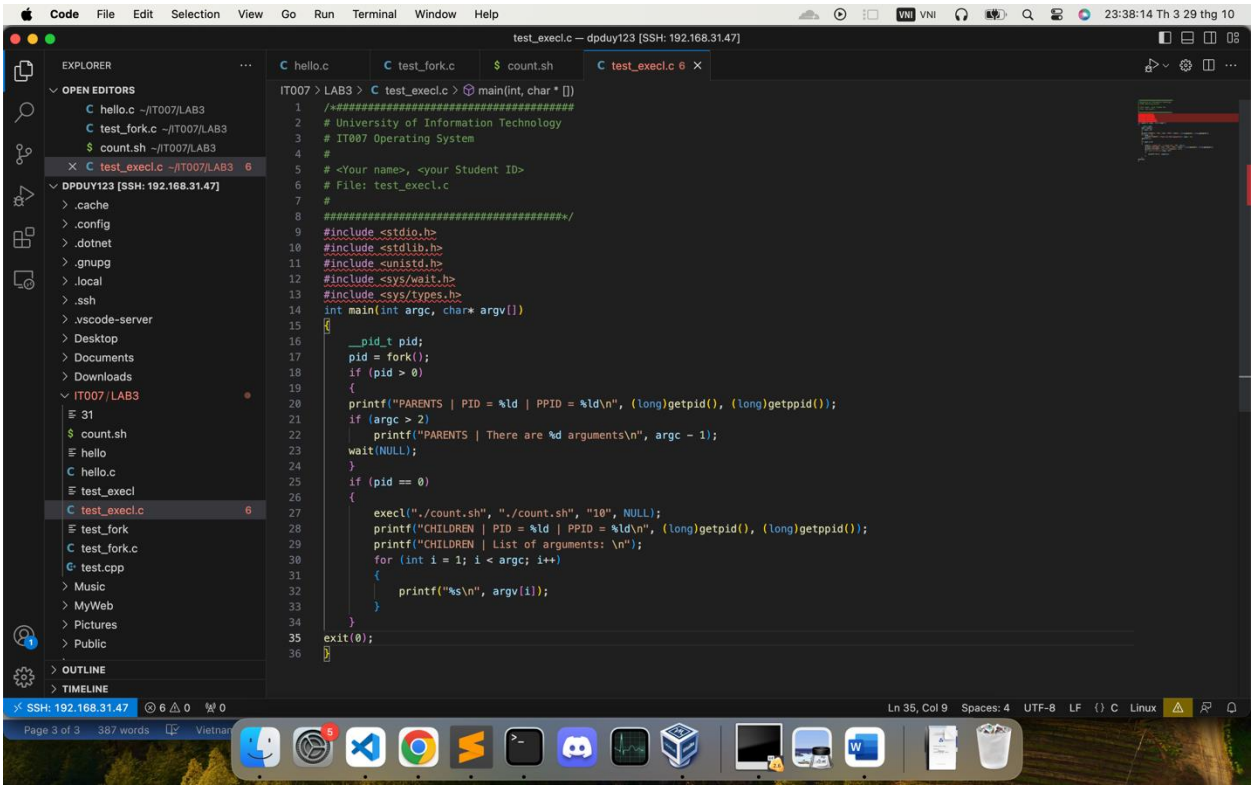
Biên dịch:

```
dpduy123@dpduy123-VirtualBox:~/IT007/LAB3$ gcc test_fork.c -o test_fork
dpduy123@dpduy123-VirtualBox:~/IT007/LAB3$ ./test_fork 128 64 32
PARENTS | PID = 6777 | PPID = 4163
PARENTS | There are 3 arguments
CHILDREN | PID = 6778 | PPID = 6777
CHILDREN | List of arguments:
128
64
32
```

Giải thích: biên dịch code trên bằng lệnh ./test_fork 128 64 32 với 3 tham số là 128 64 32.
PID và PPID của tiến trình cha lần lượt là 6777 và 4163
PID và PPID của tiến trình con lần lượt là 6778 và 6777

B. Ví dụ 3-2:

execl("./count.sh", "./count.sh", "10", NULL);: Thực thi tập lệnh shell count.sh với tham số "10". Nếu execl thành công, mã phía dưới sẽ không được thực thi.
Code:



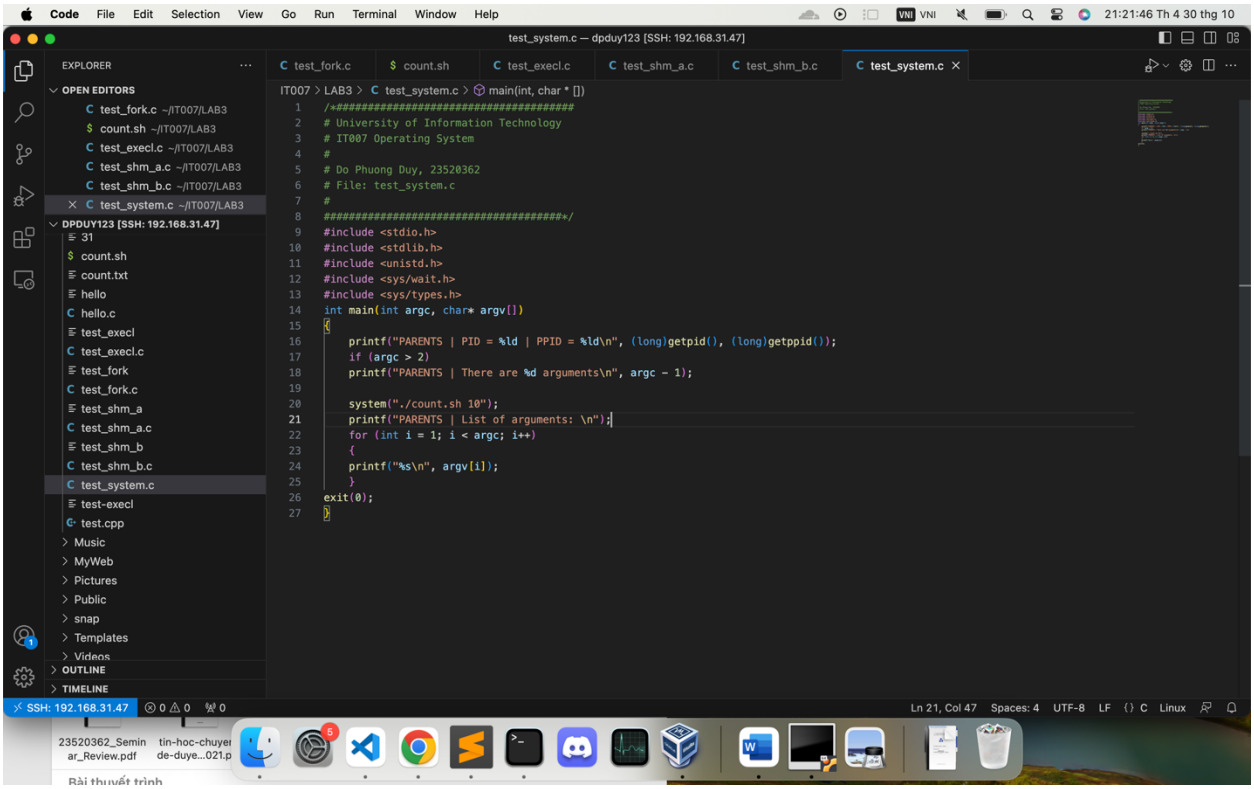
Biên dịch:

```
dpduy123@dpduy123-VirtualBox:~/IT007/LAB3$ gcc test_execl.c -o test_execl
dpduy123@dpduy123-VirtualBox:~/IT007/LAB3$ ./test_execl 3 6 2
PARENTS | PID = 8693 | PPID = 7348
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
dpduy123    8694    8693  0 00:05 pts/5    00:00:00 /bin/bash ./count.sh 10
dpduy123    8696    8694  0 00:05 pts/5    00:00:00 grep count.sh
```

Chương trình này minh họa việc tạo và xử lý tiến trình con trong C. Tiến trình cha in ra thông tin về nó và chờ tiến trình con. Tiến trình con thực thi một tập lệnh shell với tham số, và nếu việc thực thi thành công, nó sẽ không thực hiện các lệnh phía sau `execl()`. Chương trình sử dụng `fork()` để phân chia tiến trình và `execl()` để thay thế tiến trình con bằng một chương trình khác.

C. Ví dụ 3-3:

Code:



Biên dịch:

```
dpduy123@dpduy123-VirtualBox:~/IT007/LAB3$ gcc test_system.c -o test_system
dpduy123@dpduy123-VirtualBox:~/IT007/LAB3$ ./test_system
PARENTS | PID = 3829 | PPID = 3650
Implementing: ./count.sh
PPID of count.sh:
dpduy123    3830    3829  0 21:23 pts/6    00:00:00 sh -c -- ./count.sh 10
dpduy123    3831    3830  0 21:23 pts/6    00:00:00 /bin/bash ./count.sh 10
dpduy123    3833    3831  0 21:23 pts/6    00:00:00 grep count.sh
PARENTS | List of arguments:
```

Giải thích:

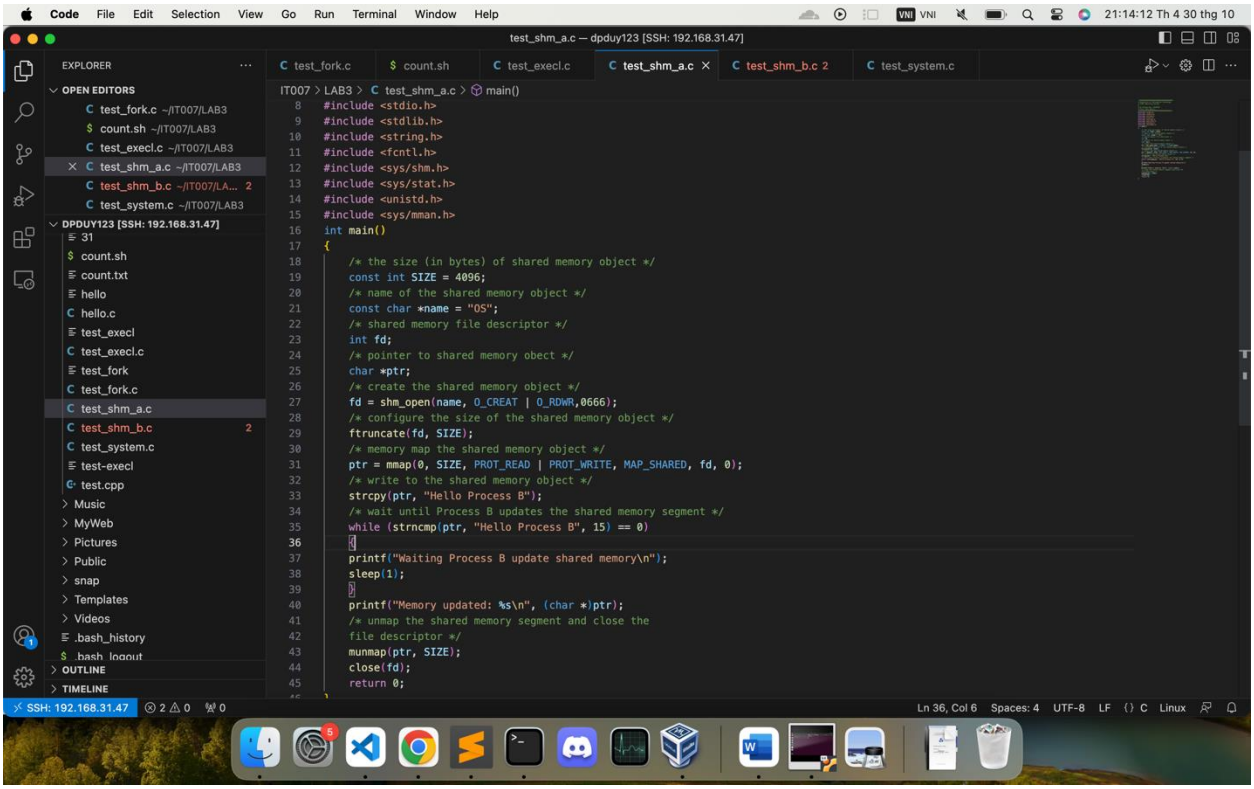
Chương trình này minh họa cách sử dụng hàm `system()` để thực thi một lệnh shell từ trong một chương trình C. Nó in ra thông tin về PID và PPID, kiểm tra số lượng tham số, và sau đó thực thi tập lệnh `count.sh` với tham số "10". Cuối cùng, nó in ra danh sách các tham số đã truyền vào.

D. Ví dụ 3-4:

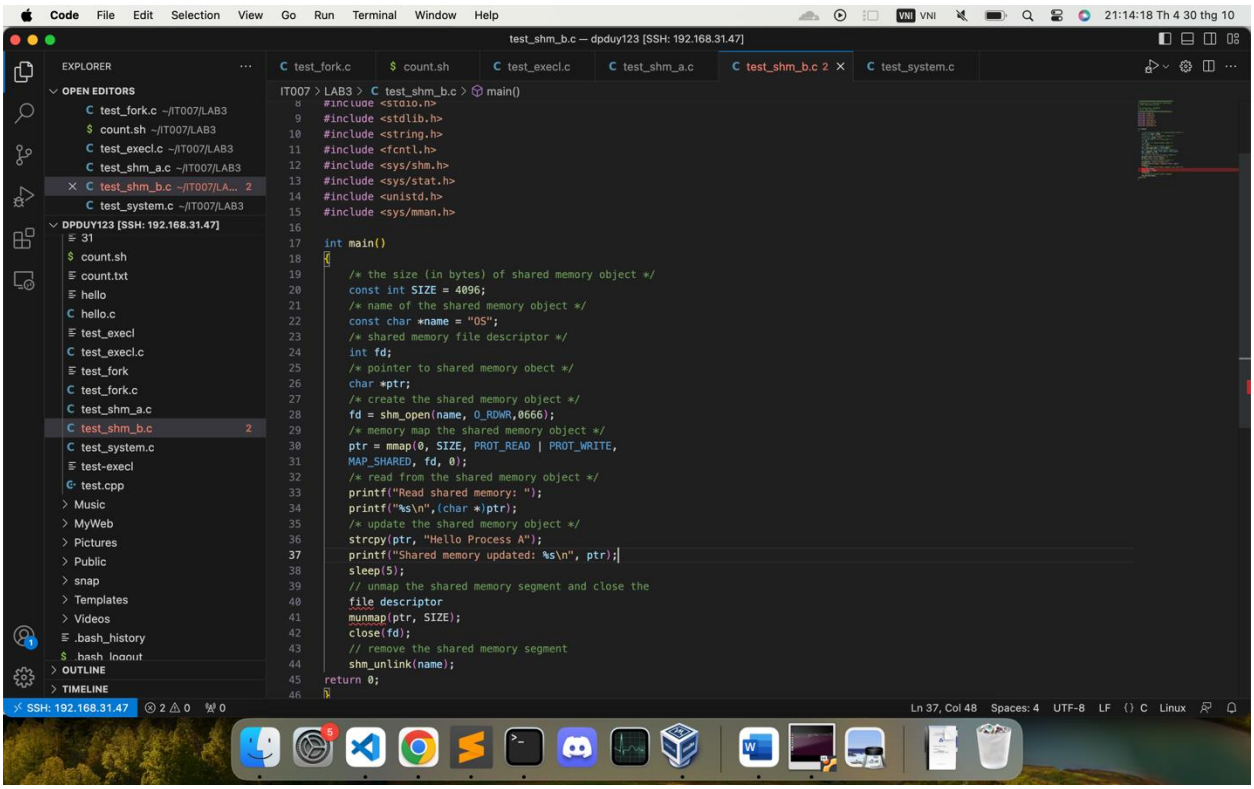
Process A khởi tạo bộ nhớ chia sẻ, ghi vào bộ nhớ "Hello Process B", sau đó chờ bộ nhớ được cập nhật bởi Process B.

Proces B truy cập bộ nhớ chia sẻ, đọc dữ liệu do Process A ghi vào, sau đó cập nhật bộ nhớ với chuỗi "Hello Process A"

Code tiến trình A:



Code tiến trình B:



Chạy process A:

```
dpduy123@dpduy123-VirtualBox:~/IT007/LAB3$ ./test_shm_a
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
```

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.

Code:

IT007 > LAB3 > C time.c > main(int, char * [])

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

int main(int argc, char *argv[])
{
    // Kiểm tra số lượng tham số
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <command>\n", argv[0]);
        return 1;
    }
    __pid_t pid;
    pid = fork();
    if (pid < 0) {
        // Nếu fork thất bại
        perror("Fork Failed");
        return 1;
    }
    if (pid > 0) {
        // Tien Trinh Cha
        int status;
        wait(&status);
        struct timeval start, end;
        gettimeofday(&end, NULL); // Lấy thời gian sau khi thực thi lệnh

        // Tính toán thời gian thực thi
        double elapsedTime = (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", elapsedTime);
    }
```

```c
    else {
        // Tien trinh con
        struct timeval start, end;

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

        // Thực thi lệnh
        execvp(argv[1], &argv[1]);

        // Nếu execvp thất bại
        perror("Exec failed");
        exit(1);
    }
    return 0;
}
```

Giải thích code:

1. **Thư viện bao gồm**:
   - `stdio.h`: Để sử dụng các hàm nhập/xuất.
   - `stdlib.h`: Để sử dụng các hàm chung như `exit()`.
   - `unistd.h`: Để sử dụng `fork()` và `exec()`.
   - `sys/types.h` và `sys/wait.h`: Để làm việc với tiến trình.
   - `sys/time.h`: Để sử dụng `gettimeofday()`.
2. **Kiểm tra tham số đầu vào**:
   - Nếu số lượng tham số nhỏ hơn 2, in ra thông báo hướng dẫn và kết thúc chương trình.
3. **Tạo tiến trình con**:
   - Sử dụng `fork()` để tạo một tiến trình con.
4. **Trong tiến trình con**:
   - Sử dụng `gettimeofday()` để lấy thời gian bắt đầu.
   - Gọi `execvp()` để thực thi lệnh shell. `execvp()` cho phép bạn truyền danh sách các tham số.
   - Nếu `execvp()` thất bại, in ra thông báo lỗi và kết thúc tiến trình con.
5. **Trong tiến trình cha**:
   - Sử dụng `wait()` để chờ tiến trình con hoàn thành.
   - Sau khi tiến trình con kết thúc, lấy thời gian kết thúc bằng `gettimeofday()`.
   - Tính toán thời gian thực thi bằng cách lấy hiệu giữa thời gian kết thúc và thời gian bắt đầu.
   - In ra thời gian thực thi.

Biên dịch và thực thi:

```
dpduy123@dpduy123-VirtualBox:~/IT007/LAB3$ gcc time.c -o time
dpduy123@dpduy123-VirtualBox:~/IT007/LAB3$ ./time ls
31         count.sh   hello    test.cpp    test_execl    test_fork    test_shm_a    test_shm_b    test_system    time
collatz.c  count.txt  hello.c  test-execl  test_execl.c  test_fork.c  test_shm_a.c  test_shm_b.c  test_system.c  time.c
Thời gian thực thi: 1730299252.31717 giây
```

3.

Code:

IT007 > LAB3 > C btth3.c > main(int, char * [])

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

void handle_sigint() {
    printf("count.sh has stopped");
}
int main(int argc, char* argv[])
{
    printf("Welcome to IT007, I am 23520362\n");
    signal(SIGINT, handle_sigint);

    __pid_t pid;
    pid = fork();
    if (pid > 0)
    {
        wait(NULL);
    }
    if (pid == 0)
    {
        execl("./count.sh", "./count.sh", "120", 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]);
        }
    }
    return 0;
}
```

Biên dịch:

```
dpduy123@dpduy123-VirtualBox:~$ cd IT007/LAB3/
dpduy123@dpduy123-VirtualBox:~/IT007/LAB3$ ./btth3
Welcome to IT007, I am 23520362
Implementing: ./count.sh
PPID of count.sh:
dpduy123    5323    5322  0 21:56 pts/8    00:00:00 /bin/bash ./count.sh 120
dpduy123    5325    5323  0 21:56 pts/8    00:00:00 grep count.sh
1
2
3
4
5
^Ccount.sh has stoppeddpduy123@dpduy123-VirtualBox:~/IT007/LAB3$
```

Ở đây vì thời gian có hạn nên em Ctrl C khi count đếm đến 5

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

Code:

IT007 > LAB3 > C btth4.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <string.h>
#include <time.h>

#define BUFFER_SIZE 10 // Kích thước buffer
#define MAX_RANDOM 20
#define MIN_RANDOM 10

// Cấu trúc cho buffer
typedef struct {
    int buffer[BUFFER_SIZE];
    int count; // Số lượng phần tử hiện có trong buffer
} SharedBuffer;

int main() {
    int shmid;
    SharedBuffer *shared_buffer;

    // Tạo bộ nhớ chia sẻ
    shmid = shmget(IPC_PRIVATE, sizeof(SharedBuffer), IPC_CREAT | 0666);
    if (shmid < 0) {
        perror("shmget");
        exit(1);
    }

    shared_buffer = (SharedBuffer *)shmat(shmid, NULL, 0);
    if (shared_buffer == (SharedBuffer *) -1) {
        perror("shmat");
        exit(1);
    }

    shared_buffer->count = 0; // Khởi tạo số lượng phần tử
```

```
39
40      pid_t pid = fork(); // Tạo tiến trình con
41
42      if (pid < 0) {
43          perror("Fork failed");
44          exit(1);
45      } else if (pid == 0) {
46          // Tiến trình con - Consumer
47          int total = 0;
48
49          while (1) {
50              // Chờ cho có phần tử trong buffer
51              while (shared_buffer->count == 0) {
52                  usleep(100); // Ngủ 100 micro giây
53              }
54
55              // Đọc dữ liệu từ buffer
56              int value = shared_buffer->buffer[--shared_buffer->count];
57              printf("Consumer: Read value %d\n", value);
58              total += value;
59
60              // Kiểm tra tổng
61              if (total > 100) {
62                  printf("Consumer: Total is %d, stopping...\n", total);
63                  break;
64              }
65          }
66
67          // Giải phóng bộ nhớ chia sẻ và kết thúc tiến trình con
68          shmdt(shared_buffer);
69          exit(0);
70      } else {
71          // Tiến trình cha - Producer
72          srand(time(NULL));
73
74          while (1) {
75              // Tạo số ngẫu nhiên trong khoảng [10, 20]
76              int value = (rand() % (MAX_RANDOM - MIN_RANDOM + 1)) + MIN_RANDOM;
```

```
77
78              // Chờ cho có chỗ trong buffer
79              while (shared_buffer->count == BUFFER_SIZE) {
80                  usleep(100); // Ngủ 100 micro giây
81              }
82
83              // Ghi dữ liệu vào buffer
84              shared_buffer->buffer[shared_buffer->count++] = value;
85              printf("Producer: Produced value %d\n", value);
86
87              // Kiểm tra nếu Consumer đã dừng
88              if (shared_buffer->count == BUFFER_SIZE && total > 100) {
89                  break;
90              }
91          }
92
93          // Chờ tiến trình con kết thúc
94          wait(NULL);
95
96          // Giải phóng bộ nhớ chia sẻ
97          shmdt(shared_buffer);
98          shmctl(shmid, IPC_RMID, NULL);
99      }
100
101     return 0;
102 }
103
```

II. Bài tập ôn tập:

IT007 > LAB3 > C collatz.c > is_positive_integer(const char *)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
#include <ctype.h>

#define BUFFER_SIZE 1024

// Hàm kiểm tra xem đầu vào có phải là số nguyên dương hay không
int is_positive_integer(const char *str) {
    if (*str == '\0') return 0; // Rỗng
    while (*str) {
        if (!isdigit(*str)) return 0; // Không phải số
        str++;
    }
    return 1; // Là số nguyên dương
}

int main(int argc, char *argv[]) {
    if (argc != 2 || !is_positive_integer(argv[1])) {
        fprintf(stderr, "Usage: %s <positive_integer>\n", argv[0]);
        return 1;
    }

    int n = atoi(argv[1]); // Chuyển đổi chuỗi sang số nguyên
    int shmid;
```

```
30      char *buffer;
31
32      // Tạo bộ nhớ chia sẻ
33      shmid = shmget(IPC_PRIVATE, BUFFER_SIZE, IPC_CREAT | 0666);
34      if (shmid < 0) {
35          perror("shmget");
36          return 1;
37      }
38
39      buffer = shmat(shmid, NULL, 0); // Gán bộ nhớ chia sẻ
40      if (buffer == (char *) -1) {
41          perror("shmat");
42          return 1;
43      }
44
45      pid_t pid = fork(); // Tạo tiến trình con
46
47      if (pid < 0) {
48          perror("Fork failed");
49          return 1;
50      } else if (pid == 0) {
51          // Tiến trình con
52          char temp[32];
53          int index = 0;
54
55          while (n != 1) {
56              index += sprintf(&buffer[index], "%d, ", n); // Ghi số vào buffer
57              if (n % 2 == 0) {
58                  n /= 2;
```

```
            } else {
                n = 3 * n + 1;
            }
        }
        sprintf(&buffer[index], "1"); // Thêm 1 vào buffer

        // Kết thúc tiến trình con
        shmdt(buffer); // Ngắt kết nối bộ nhớ chia sẻ
        exit(0);
    } else {
        // Tiến trình cha
        wait(NULL); // Chờ tiến trình con hoàn thành

        // In kết quả ra màn hình
        printf("Collatz sequence: %s\n", buffer);

        // Giải phóng bộ nhớ chia sẻ
        shmdt(buffer);
        shmctl(shmid, IPC_RMID, NULL);
    }

    return 0;
}

```

Giải thích code:

- **Hàm kiểm tra số nguyên dương**:
  - Hàm `is_positive_integer()` kiểm tra xem chuỗi đầu vào có phải là số nguyên dương không. Nếu không, chương trình sẽ in thông báo lỗi và thoát.
- **Bộ nhớ chia sẻ**:
  - Tạo bộ nhớ chia sẻ bằng `shmget()` và gán nó vào buffer sử dụng `shmat()`.
- **Tiến trình con**:
  - Tiến trình con tính toán chuỗi Collatz từ số `n`. Mỗi giá trị được ghi vào buffer sử dụng `sprintf()`.
  - Khi `n` trở về 1, tiến trình con kết thúc và ngắt kết nối bộ nhớ chia sẻ.
- **Tiến trình cha**:
  - Tiến trình cha chờ cho tiến trình con hoàn thành bằng `wait()`, sau đó in ra chuỗi Collatz từ buffer.
  - Giải phóng bộ nhớ chia sẻ bằng `shmdt()` và `shmctl()`.

Biên dịch và kiểm thử các testcase: