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	☒	☒	☒	☒
Chụp hình minh chứng	☒	☒	☒	☒
Giải thích kết quả	☒	☒	☒	☒

3.6. BÀI TẬP ÔN TẬP

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

Tự chấm điểm: 9.5

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

2.5. BÀI TẬP THỰC HÀNH

1. Thực hiện Ví dụ 3-1, Ví dụ 3-2, Ví dụ 3-3, Ví dụ 3-4 giải thích code và kết quả nhận được?

Ví dụ : 3-1

```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);
}
```

```
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.

```c
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)

```c
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:
    - Tiến trình Cha (PID = 5076).
    - Tiến trình Con (PID = 5077).
- Luồng thực thi:
    - 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

```c
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

```c
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) { ... wait(NULL); }` : Đây là cha, in PID, PPID, số tham số, rồi chờ con kết thúc

```c
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:
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
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.

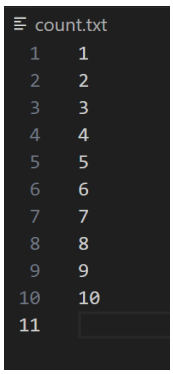
Tiến trình Cha (PARENTS | PID = 12646 | PPID = 692) : Tiến trình cha (Chương trình C) vẫn chạy bình thường, in ra PID của nó và đợi con.

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
```

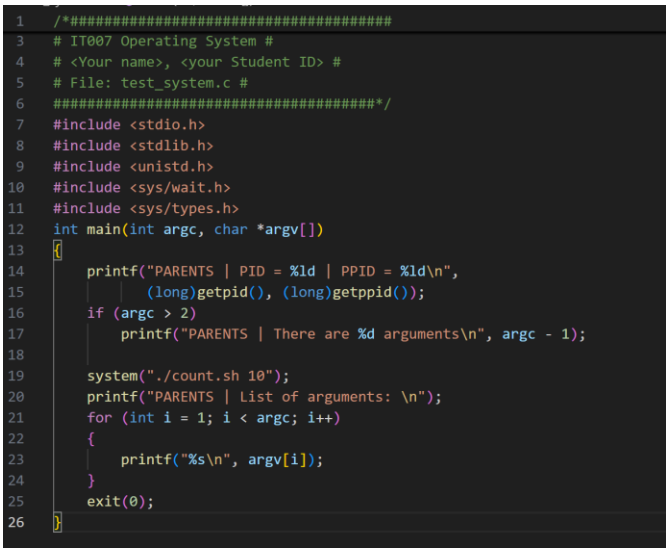
Đây là bằng chứng rõ nhất cho thấy execl đã thành công:

- 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:**
  - Tiến trình chạy script có PID = 12647.
  - 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:



```c
/*###################################
# 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);
}
```

- *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:
  - `printf("PARENTS | List of arguments: \n");`: In tiêu đề.
  - 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:
lenhuth+   28610   28609  0 17:45 pts/5    00:00:00 sh -c -- ./count.sh 10
lenhuth+   28611   28610  0 17:45 pts/5    00:00:00 /bin/bash ./count.sh 10
lenhuth+   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:

```c
int main()
{ /* the size (in bytes) of shared memory object */
    const int SIZE = 4096;
    /* name of the shared memory object */
    const char *name = "OS";
    /* shared memory file descriptor */
    int fd;
    /* pointer to shared memory obect */
    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 (strncmp(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;
}
```

```c
int main()
{
    /* the size (in bytes) of shared memory object */
    const int SIZE = 4096;
    /* name of the shared memory object */
    const char *name = "OS";
    /* shared memory file descriptor */
    int fd;
    /* pointer to shared memory obect */
    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
Waiting Process B update shared memory
Waiting Process B update shared memory
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.

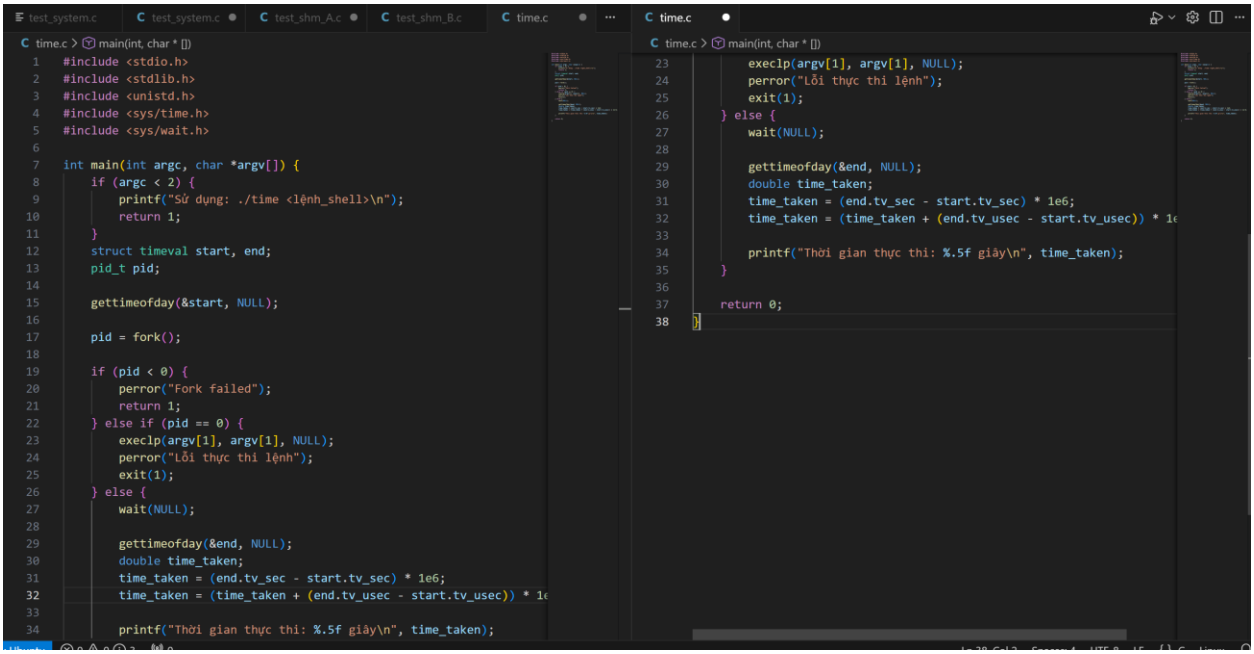
2. Viết chương trình `time.c` thực hiện đo thời gian thực thi của một lệnh shell. Chương trình sẽ được chạy với cú pháp "`./time <command>`" với `<command>` là lệnh shell muốn đo thời gian thực thi.

Ví dụ:

```
$ ./time ls
time.c
time
Thời gian thực thi: 0.25422
```

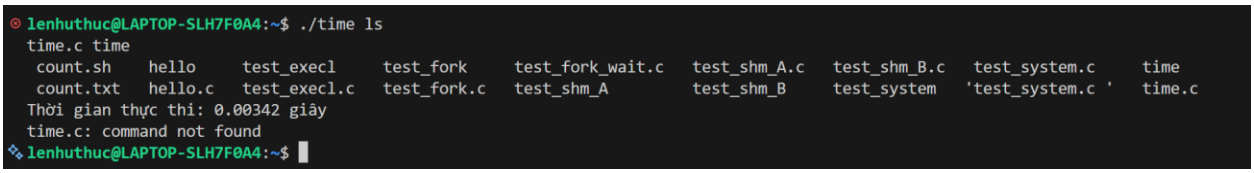
Gợi ý: Tiến trình cha gọi hàm fork() tạo ra tiến trình con rồi wait(). Tiến trình con gọi hàm gettimeofday() để lấy mốc thời gian trước khi thực thi lệnh shell, sau đó sử dụng hàm execl() để thực thi lệnh. Sau khi tiến trình con kết thúc, tiến

trình cha tiếp tục gọi hàm gettimeifday() một lần nữa để lấy mốc thời gian sau khi thực thi lệnh shell và tính toán.



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

int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Sử dụng: ./time <lệnh_shell>\n");
        return 1;
    }
    struct timeval start, end;
    pid_t pid;

    gettimeofday(&start, NULL);

    pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        return 1;
    } else if (pid == 0) {
        execlp(argv[1], argv[1], NULL);
        perror("Lỗi thực thi lệnh");
        exit(1);
    } else {
        wait(NULL);

        gettimeofday(&end, NULL);
        double time_taken;
        time_taken = (end.tv_sec - start.tv_sec) * 1e6;
        time_taken = (time_taken + (end.tv_usec - start.tv_usec)) * 1e

        printf("Thời gian thực thi: %.5f giây\n", time_taken);
    }

    return 0;
}
```

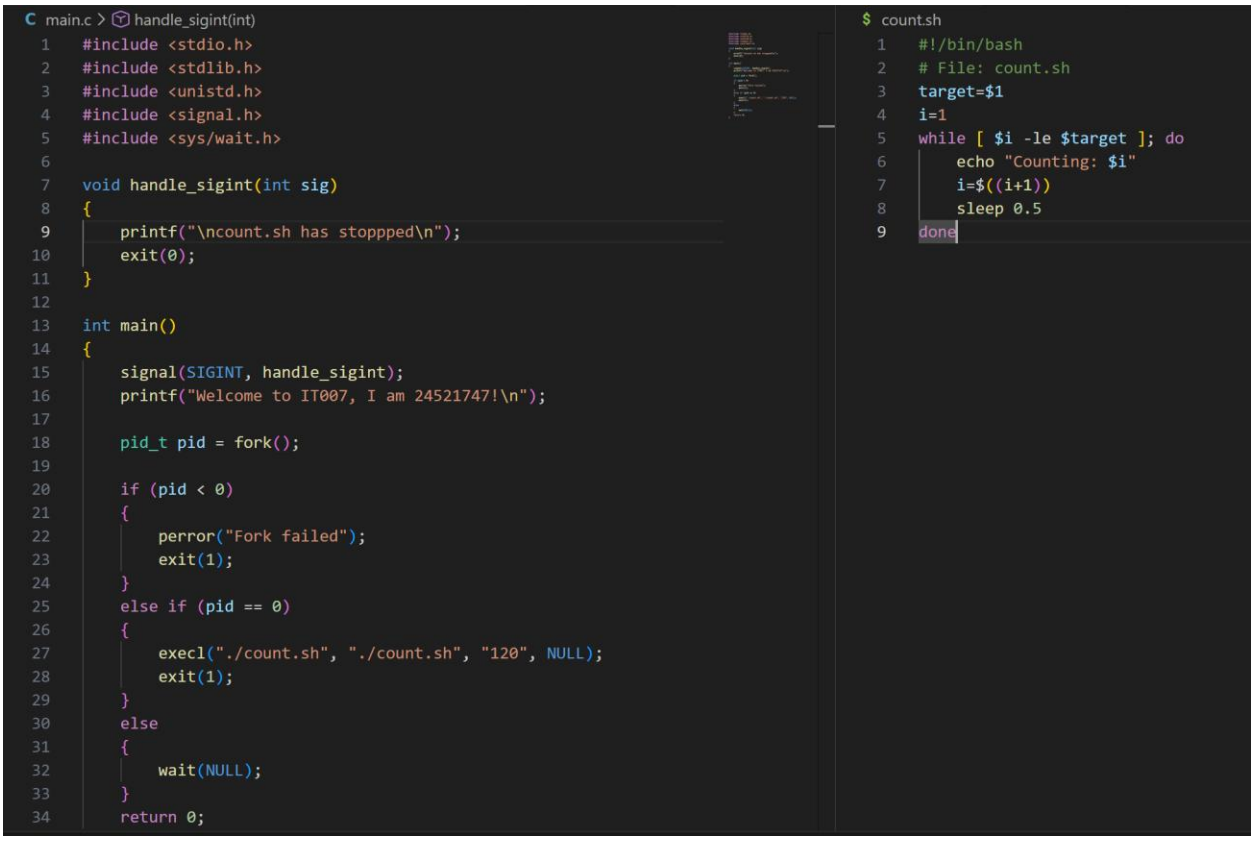
- Hàm *gettimeofday(&start, NULL)* : Bấm đồng hồ bắt đầu tính giờ
- *pid = fork()* dùng để tạo ra một bản sao của tiến trình hiện tại, hệ điều hành nhân đôi chương trình.
- *execlp(argv[1], argv[1], NULL)* :
    - argv[1]: Là chuỗi "ls" (lấy từ tham số dòng lệnh).
    - Cơ chế: Hàm này ra lệnh cho hệ 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) :*
    - Đây là lệnh Blocking, tiến trình Cha sẽ bị sleep, không làm gì cả.
    - 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.



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"**



main.c > handle_sigint(int)

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

void handle_sigint(int sig)
{
    printf("\ncount.sh has stoppped\n");
    exit(0);
}

int main()
{
    signal(SIGINT, handle_sigint);
    printf("Welcome to IT007, I am 24521747!\n");

    pid_t pid = fork();

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

$ count.sh

```bash
#!/bin/bash
# File: count.sh
target=$1
i=1
while [ $i -le $target ]; do
    echo "Counting: $i"
    i=$((i+1))
    sleep 0.5
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.**
- **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**

producer_consumer.c > main()

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

struct SharedData {
    unsigned char buffer[10];
    int in;
    int out;
    int sum;

    sem_t sem_empty;
    sem_t sem_full;
    sem_t sem_mutex;
};

int main() {
    struct SharedData *shared = mmap(NULL, sizeof(struct SharedData),
                                     PROT_READ | PROT_WRITE,
                                     MAP_SHARED | MAP_ANONYMOUS, -1, 0);

    if (shared == MAP_FAILED) {
        perror("mmap failed");
        return 1;
    }

    shared->in = 0;
    shared->out = 0;
    shared->sum = 0;
```

```c
    sem_init(&shared->sem_empty, 1, 10);
    sem_init(&shared->sem_full, 1, 0);
    sem_init(&shared->sem_mutex, 1, 1);

    pid_t pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        return 1;
    }

    if (pid > 0) {
        srand(time(NULL));
        while (1) {
            if (shared->sum > 100) break;

            int val = rand() % 11 + 10;

            sem_wait(&shared->sem_empty);
            sem_wait(&shared->sem_mutex);
            if (shared->sum > 100) {
                sem_post(&shared->sem_mutex);
                sem_post(&shared->sem_empty);
                break;
            }

            shared->buffer[shared->in] = val;
            printf("[Producer] Ghi %d vào buffer[%d]\n", val, shared->in);
            shared->in = (shared->in + 1) % 10;
```

```
            sem_post(&shared->sem_mutex);
            sem_post(&shared->sem_full);

            sleep(1);
        }

        wait(NULL);
        printf("Producer kết thúc.\n");

        sem_destroy(&shared->sem_empty);
        sem_destroy(&shared->sem_full);
        sem_destroy(&shared->sem_mutex);
        munmap(shared, sizeof(struct SharedData));
    }
    else {
        while (1) {
            sem_wait(&shared->sem_full);
            sem_wait(&shared->sem_mutex);

            int val = shared->buffer[shared->out];
            shared->sum += val;

            printf("\t\t[Consumer] Đọc %d từ buffer[%d] -> Tổng = %d\n",
                   val, shared->out, shared->sum);

            shared->out = (shared->out + 1) % 10;
            if (shared->sum > 100) {
                printf("\t\t[Consumer] Tổng > 100. Dừng!\n");
                sem_post(&shared->sem_mutex);
                sem_post(&shared->sem_empty);
                exit(0);
            }
            }

            sem_post(&shared->sem_mutex);
            sem_post(&shared->sem_empty);
        }
    }
    return 0;
}
```

```
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_custommer" 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) lần lượt sinh 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

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

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:**

$$n = \begin{cases} n/2 \ \ \text{nếu } n \text{ là số chẵn} \\ 3 * n + 1 \ \text{nếu } n \text{ là số lẻ} \end{cases}$$

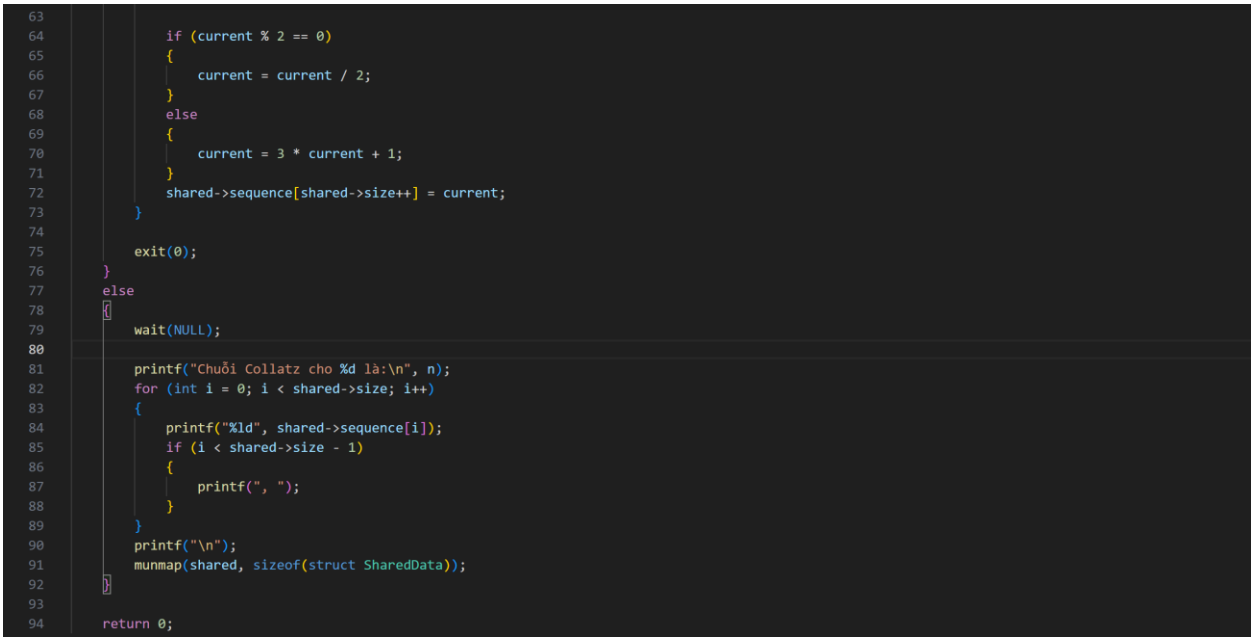
Phỏng đoán phát biểu rằng khi thuật toán này được áp dụng liên tục, tất cả số nguyên dương đều sẽ tiến đến 1. Ví dụ, với n = 35, ta sẽ có chuỗi kết quả như sau:

```
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.

collatz.c > main(int, char * [])

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

#define MAX_ITEMS 200
struct SharedData
{
    long sequence[MAX_ITEMS];
    int size;
};

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

    if (argc < 2)
    {
        printf("Sử dụng: ./collatz <số nguyên dương>\n");
        return 1;
    }

    int n = atoi(argv[1]);
    if (n <= 0)
    {
        printf("Lỗi: Vui lòng nhập một số nguyên dương lớn hơn 0.\n");
        return 1;
    }

    struct SharedData *shared = mmap(NULL, sizeof(struct SharedData),
                                     PROT_READ | PROT_WRITE,
                                     MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    if (shared == MAP_FAILED)
    {
        perror("mmap failed");
        return 1;
    }

    shared->size = 0;

    pid_t pid = fork();

    if (pid < 0)
    {
        perror("Fork failed");
        return 1;
    }
    else if (pid == 0)
    {

        long current = n;
        shared->sequence[shared->size++] = current;

        while (current != 1)
        {

            if (shared->size >= MAX_ITEMS)
            {
                break;
            }
```



```
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):**
    - 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.
    - Định nghĩa một cấu trúc (struct) gồm mảng chứa dãy số và biến đếm số lượng phần tử.
- **Tạo tiến trình (Fork):**
    - Gọi fork() để tách chương trình.
- **Xử lý tại Tiến trình con (Child):**
    - Thực hiện thuật toán Collatz:
        - Nếu n chẵn: n = n / 2
        - Nếu n lẻ: n = 3n + 1
    - Ghi từng giá trị tính toán được vào vùng nhớ chia sẻ.
    - 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 -> 35*3 + 1 = 106. (Ghi 106 vào buffer).
  - 106 là chẵn -> 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.