

**Họ và tên:**

Phạm Đông Hưng

Lương Anh Huy

Phan Công Minh

Hồng Khải Nguyên

**MSSV:**

22520521

22520550

22520884

22520967

**Lớp:** IT007.O14.1

## 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"/> |

**Tư chấm điểm:** 10

*\*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?

#### 3-1:

**Cách làm:** Sử dụng hàm fork() để tạo tiến trình, Tiến trình gọi hàm fork() được gọi là tiến trình cha, tiến trình mới được tạo ra là tiến trình con

**Minh chứng:**

```
C test_fork.c > main(int, char * [])
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5  #include <sys/types.h>
6  int main(int argc, char *argv[])
7  {
8      __pid_t pid;
9      pid = fork();
10     if (pid > 0)
11     {
12         printf("PARENTS | PID = %ld | PPID = %ld\n",
13             (long)getpid(), (long)getppid());
14         if (argc > 2)
15             printf("PARENTS | There are %d arguments\n",
16                 argc - 1);
17         wait(NULL);
18     }
19     if (pid == 0)
20     {
21         printf("CHILDREN | PID = %ld | PPID = %ld\n",
22             (long)getpid(), (long)getppid());
23         printf("CHILDREN | List of arguments: \n");
24         for (int i = 1; i < argc; i++)
25         {
26             printf("%s\n", argv[i]);
27         }
28     }
29     exit(0);
30 }
```

PROBLEMS 19 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
congminh-22520884@Test1:~$ ./test_fork ThamSo1 ThamSo2 ThamSo3
PARENTS | PID = 9330 | PPID = 9172
PARENTS | There are 3 arguments
CHILDREN | PID = 9331 | PPID = 9330
CHILDREN | List of arguments:
ThamSo1
ThamSo2
ThamSo3
congminh-22520884@Test1:~$
```

### Giải thích:

- `int main(int argc, char *argv[])`: hàm `main()` nhận 2 đối số, `argc` là số lượng đối số truyền vào chương trình và `argv` là mảng các chuỗi chứa các đối số truyền vào (chứa `ThamSo1`, `ThamSo2`, `ThamSo3`)
- `__pid_t` là kiểu dữ liệu để lưu trữ Process ID (PID), biến `pid` được sử dụng để lưu trữ giá trị trả về từ hàm `fork()`
- `pid = fork()`: hàm `fork` sẽ tạo một bản sao của tiến trình hiện tại:
- + `if (pid > 0)` thì đây là tiến trình cha, sau đó in ra PID và PPID của nó bằng hàm `getpid()` và `getppid()`, nếu đối số truyền vào `argc > 2` thì chương trình sẽ in ra số lượng đối số truyền vào, cuối cùng chờ tất cả các tiến trình con kết thúc bằng hàm `wait(NULL)`

```
C test_exec1.c > main(int, char * [])
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5  #include <sys/types.h>
6  int main(int argc, char *argv[])
7  {
8      __pid_t pid;
9      pid = fork();
10     if (pid > 0)
11     {
12         printf("PARENTS | PID = %ld | PPID = %ld\n",
13              (long)getpid(), (long)getppid());
14         if (argc > 2)
15             printf("PARENTS | There are %d arguments\n",
16                  argc - 1);
17         wait(NULL);
18     }
19     if (pid == 0)
20     {
21         execl("./count.sh", "./count.sh", "10", NULL);
22
23         printf("CHILDREN | PID = %ld | PPID = %ld\n",
24              (long)getpid(), (long)getppid());
25         printf("CHILDREN | List of arguments: \n");
26         for (int i = 1; i < argc; i++)
27         {
28             printf("%s\n", argv[i]);
29         }
30     }
31     exit(0);
32 }
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
congminh-22520884@Test1:~$ ./test_exec1 ThamSo1 ThamSo2 ThamSo3
PARENTS | PID = 8791 | PPID = 5627
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
congmin+ 8792 8791 0 10:13 pts/2 00:00:00 /bin/bash ./count.sh 10
congmin+ 8794 8792 0 10:13 pts/2 00:00:00 grep count.sh
congminh-22520884@Test1:~$
```

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Trần Hoàng Lộc.

+ if (pid==0) thì đây là tiến trình con, sau đó in ra PID và PPID của nó bằng hàm getpid() và getppid(), tiếp theo in ra số lượng các đối số truyền vào bằng một vòng lặp for

\*Lưu ý: argc sẽ đếm tất cả đối số trong câu lệnh được nhập, bao gồm cả tên của chương trình (không xét tên của chương trình là một đối số riêng lẻ) nên dòng lệnh nhập “./test\_fork ThamSo1 ThamSo2 ThamSo3” sẽ cho argc = 4 và argv[0] = ./test\_fork (tên chương trình), vì vậy khi muốn in ra số lượng và giá trị đối số đúng ở phần thực hiện tiến trình cha và con thì cần dùng argc – 1 và vòng lặp for với i = 1

### 3-2:

**Cách làm:** Tương tự ví dụ 3-1, có thêm lệnh execl ở phần pid = 0

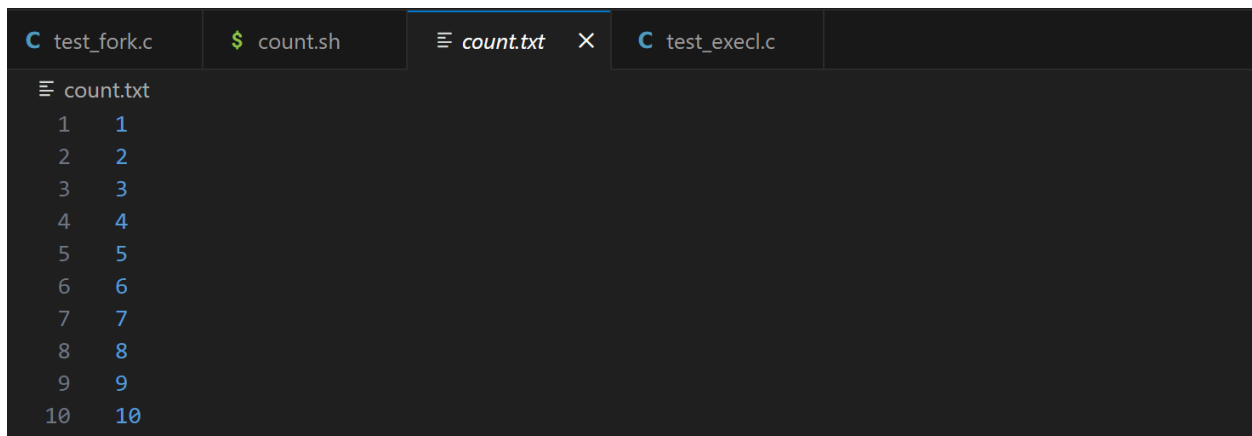
**Minh chứng:**

```
C test_execl.c > main(int, char * [])
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5  #include <sys/types.h>
6  int main(int argc, char *argv[])
7  {
8      __pid_t pid;
9      pid = fork();
10     if (pid > 0)
11     {
12         printf("PARENTS | PID = %ld | PPID = %ld\n",
13             (long)getpid(), (long)getppid());
14         if (argc > 2)
15             printf("PARENTS | There are %d arguments\n",
16                 argc - 1);
17         wait(NULL);
18     }
19     if (pid == 0)
20     {
21         execl("./count.sh", "./count.sh", "10", NULL);
22
23         printf("CHILDREN | PID = %ld | PPID = %ld\n",
24             (long)getpid(), (long)getppid());
25         printf("CHILDREN | List of arguments: \n");
26         for (int i = 1; i < argc; i++)
27         {
28             printf("%s\n", argv[i]);
29         }
30     }
31     exit(0);
32 }
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
congminh-22520884@Test1:~$ ./test_execl ThamSo1 ThamSo2 ThamSo3
PARENTS | PID = 8791 | PPID = 5627
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
congmin+ 8792 8791 0 10:13 pts/2 00:00:00 /bin/bash ./count.sh 10
congmin+ 8794 8792 0 10:13 pts/2 00:00:00 grep count.sh
congminh-22520884@Test1:~$
```

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



```
count.txt
1  1
2  2
3  3
4  4
5  5
6  6
7  7
8  8
9  9
10 10
```

### Giải thích:

- Cấu trúc chương trình giống với ví dụ 3-1, khác ở chỗ có thêm dòng lệnh `execl("./count.sh", "./count.sh", "10", NULL);`. Khi lệnh `execl` được thực thi, nó sẽ thay thế quá trình con hiện tại bằng chương trình `count.sh`, nên các lệnh sau `execl` của quá trình con sẽ không được chạy. Dòng lệnh này sẽ thực hiện chương trình `count.sh` đã được tạo ra sẵn với đối số truyền vào là 10, và `NULL` là để đánh dấu kết thúc danh sách đối số. Trong trường hợp này chương trình `count.sh` sẽ in ra các số từ 1 đến 10

### 3-3:

**Cách làm:** Để thực thi tệp count.sh ta dùng system()

**Minh chứng:**

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
congminh-22520884@Test1:~$ ./test_system ThamSo1 ThamSo2 ThamSo3
PARENTS | PID = 14650 | PPID = 2983
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
congmin+ 14651 14650 0 18:45 pts/1 00:00:00 sh -c ./count.sh 10
congmin+ 14652 14651 0 18:45 pts/1 00:00:00 /bin/bash ./count.sh 10
congmin+ 14654 14652 0 18:45 pts/1 00:00:00 grep count.sh
PARENTS | List of arguments:
ThamSo1
ThamSo2
ThamSo3
congminh-22520884@Test1:~$
```

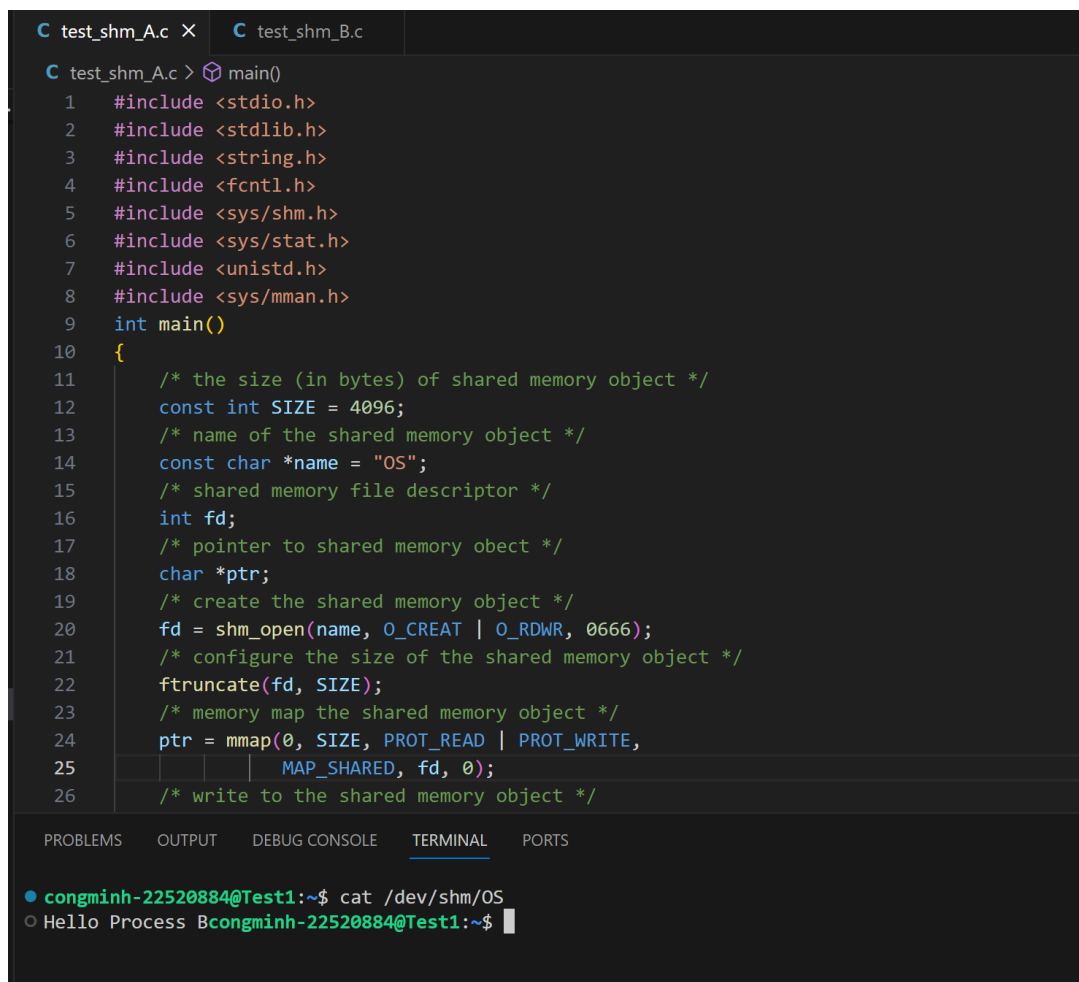
**Giải thích:** hàm system được dùng để thực thi một tệp (một lệnh) bên ngoài chương trình hiện tại, nó tạo một tiến trình riêng biệt (không liên quan đến tiến trình hiện tại) để thực hiện một vài công việc cụ thể nào đó, và sau khi hoàn thành, nó sẽ kết thúc tiến trình con và trả về kết quả của các công việc đó, nên tiến trình cha trước đó vẫn thực hiện các lệnh của nó sau dòng lệnh gọi system

### 3-4: a & b

**Cách làm:** cài đặt bộ nhớ chia sẻ thông qua ví dụ về hai tiến trình A và B được tạo và truyền nội dung vào vùng nhớ chia sẻ chung

**Minh chứng:**

- Khi chỉ mới chạy file test\_shm\_A.c:

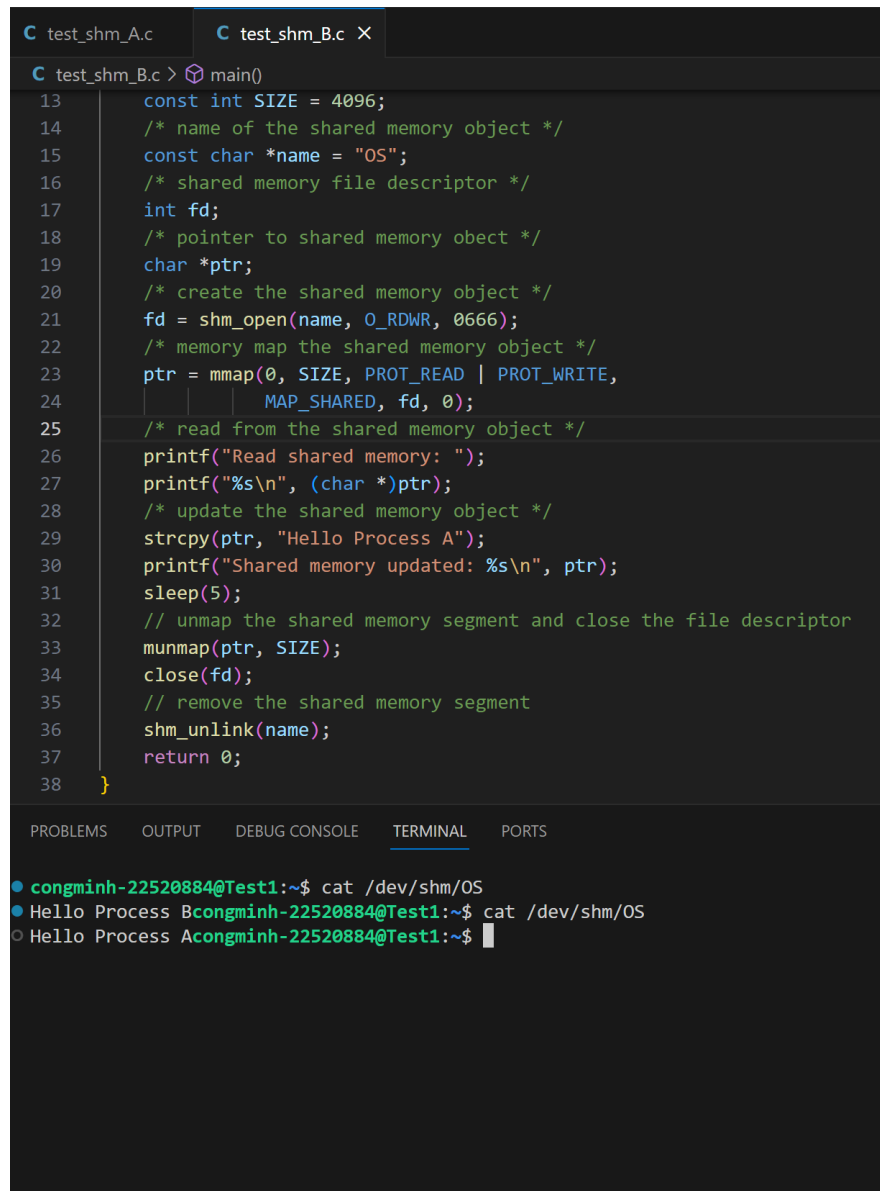


```
C test_shm_A.c X C test_shm_B.c
C test_shm_A.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <fcntl.h>
5  #include <sys/shm.h>
6  #include <sys/stat.h>
7  #include <unistd.h>
8  #include <sys/mman.h>
9  int main()
10 {
11     /* the size (in bytes) of shared memory object */
12     const int SIZE = 4096;
13     /* name of the shared memory object */
14     const char *name = "OS";
15     /* shared memory file descriptor */
16     int fd;
17     /* pointer to shared memory object */
18     char *ptr;
19     /* create the shared memory object */
20     fd = shm_open(name, O_CREAT | O_RDWR, 0666);
21     /* configure the size of the shared memory object */
22     ftruncate(fd, SIZE);
23     /* memory map the shared memory object */
24     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE,
25               MAP_SHARED, fd, 0);
26     /* write to the shared memory object */

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● congminh-22520884@Test1:~$ cat /dev/shm/OS
○ Hello Process Bcongminh-22520884@Test1:~$
```

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Trần Hoàng Lộc.

- Khi chạy cả 2 file ( test\_shm\_A.c và test\_shm\_B.c ):



```
C test_shm_A.c  C test_shm_B.c X
C test_shm_B.c > main()
13  const int SIZE = 4096;
14  /* name of the shared memory object */
15  const char *name = "OS";
16  /* shared memory file descriptor */
17  int fd;
18  /* pointer to shared memory object */
19  char *ptr;
20  /* create the shared memory object */
21  fd = shm_open(name, O_RDWR, 0666);
22  /* memory map the shared memory object */
23  ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE,
24            MAP_SHARED, fd, 0);
25  /* read from the shared memory object */
26  printf("Read shared memory: ");
27  printf("%s\n", (char *)ptr);
28  /* update the shared memory object */
29  strcpy(ptr, "Hello Process A");
30  printf("Shared memory updated: %s\n", ptr);
31  sleep(5);
32  // unmap the shared memory segment and close the file descriptor
33  munmap(ptr, SIZE);
34  close(fd);
35  // remove the shared memory segment
36  shm_unlink(name);
37  return 0;
38  }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● congminh-22520884@Test1:~$ cat /dev/shm/OS
● Hello Process Bcongminh-22520884@Test1:~$ cat /dev/shm/OS
○ Hello Process Acongminh-22520884@Test1:~$
```

### Giải thích:

- Lệnh `strcpy(ptr, "Hello Process B");` sẽ sao chép chuỗi "Hello Process B" vào vùng nhớ chia sẻ thông qua con trỏ `ptr`, tức là đưa nội dung "Hello Process B" vào vùng nhớ chia sẻ
- Vòng lặp `while` được sử dụng để kiểm tra xem tiến trình B đã cập nhật nội dung trong vùng nhớ chia sẻ hay chưa, nếu nội dung trong vùng nhớ vẫn là "Hello Process B" thì sẽ chờ một giây và tiếp tục kiểm tra lại như thế đến khi vùng nhớ được cập nhật nội dung khác, với nội dung mà tiến trình B cập nhật ở đây là "Hello Process A"



- Ta có thể dùng lệnh `cat /dev/shm/OS` để kiểm tra vùng nhớ OS (`/dev/shm/` là một thư mục đặc biệt trong hệ thống tệp dựa trên `tmpfs`, thường được sử dụng để lưu trữ vùng nhớ chia sẻ. Trong ví dụ 3-4, ta đã đặt tên cho vùng nhớ chia sẻ là `OS` nên nó sẽ nằm trong thư mục `/dev/shm` với tên tương tự)

## 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
```

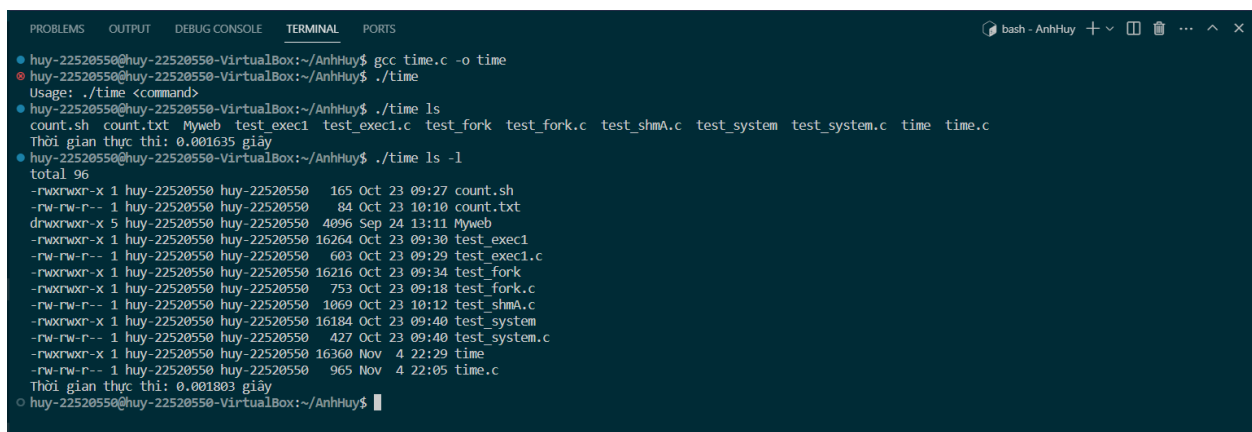
```
time
```

```
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.

Cách làm: Ta sử dụng `gettimeofday` để lấy thời gian trước khi thực thi lệnh shell, sau đó tạo tiến trình con, trong tiến trình con thì sử dụng `execl` để thực thi lệnh shell, trong tiến trình cha thì dùng `waitpid` để đợi tiến trình con kết thúc và lấy thời gian sau khi thực hiện shell. Cuối cùng là tính toán thời gian thực thi bằng cách lấy hiệu 2 thời gian.

Hình ảnh:



```
huy-22520550@huy-22520550-VirtualBox:~/AnhHuy$ gcc time.c -o time
huy-22520550@huy-22520550-VirtualBox:~/AnhHuy$ ./time
Usage: ./time <command>
huy-22520550@huy-22520550-VirtualBox:~/AnhHuy$ ./time ls
count.sh count.txt Myweb test_exec1 test_exec1.c test_fork test_fork.c test_shm.c test_system test_system.c time time.c
Thời gian thực thi: 0.001635 giây
huy-22520550@huy-22520550-VirtualBox:~/AnhHuy$ ./time ls -l
total 96
-rwxrwxr-x 1 huy-22520550 huy-22520550 165 Oct 23 09:27 count.sh
-rw-rw-r-- 1 huy-22520550 huy-22520550 84 Oct 23 10:10 count.txt
drwxrwxr-x 5 huy-22520550 huy-22520550 4096 Sep 24 13:11 Myweb
-rwxrwxr-x 1 huy-22520550 huy-22520550 16264 Oct 23 09:30 test_exec1
-rw-rw-r-- 1 huy-22520550 huy-22520550 603 Oct 23 09:29 test_exec1.c
-rwxrwxr-x 1 huy-22520550 huy-22520550 16216 Oct 23 09:34 test_fork
-rwxrwxr-x 1 huy-22520550 huy-22520550 753 Oct 23 09:18 test_fork.c
-rw-rw-r-- 1 huy-22520550 huy-22520550 1069 Oct 23 10:12 test_shm.c
-rwxrwxr-x 1 huy-22520550 huy-22520550 16184 Oct 23 09:40 test_system
-rw-rw-r-- 1 huy-22520550 huy-22520550 427 Oct 23 09:40 test_system.c
-rwxrwxr-x 1 huy-22520550 huy-22520550 16360 Nov 4 22:29 time
-rw-rw-r-- 1 huy-22520550 huy-22520550 965 Nov 4 22:05 time.c
Thời gian thực thi: 0.001803 giây
huy-22520550@huy-22520550-VirtualBox:~/AnhHuy$
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/time.h>
4  #include <sys/wait.h>
5  #include <unistd.h>
6  int main(int argc, char *argv[]) {
7      if (argc < 2) {
8          fprintf(stderr, "Usage: ./time <command>\n");
9          return 1;
10     }
11     struct timeval start, end;
12     gettimeofday(&start, NULL);
13     pid_t child_pid = fork();
14     if (child_pid == 0) {
15         execvp(argv[1], &argv[1]);
16         perror("execvp");
17         exit(1);
18     }
19     else if (child_pid < 0) {
20         perror("fork");
21         exit(1);
22     } else {
23         int status;
24         waitpid(child_pid, &status, 0);
25         gettimeofday(&end, NULL);
26
27         if (WIFEXITED(status)) {
28             double elapsed_time = (end.tv_sec - start.tv_sec) +
29                                   (end.tv_usec - start.tv_usec) / 1e6;
30             printf("Thời gian thực thi: %f giây\n", elapsed_time);
31         } else {
32             fprintf(stderr, "Có lỗi xảy ra!\n");
33         }
34     }
35     return 0;
36 }
```

Giải thích:

- Khai báo các thư viện cần thiết để thực hiện yêu cầu bài toán
- Ở câu lệnh if đầu tiên ta sẽ kiểm tra số tham số trên dòng lệnh có ít hơn 2 không, nếu có thì xuất ra cú pháp đúng để thực thi và thoát chương trình. Còn nếu đúng thì ta sẽ tiếp tục thực thi để tính thời gian. Giả sử ta có hai câu lệnh là ./time và ./time ls thì câu lệnh đầu tiên chỉ có tên chương trình là time nên hệ thống mặc định số lượng tham số là 1 và không hợp lệ, còn câu lệnh sau có 2 tham số là time và ls nên ta có thể thực thi nó
- Tiếp đến ta khai báo hai biến start và end có kiểu là 'struct timeval' dùng để lưu thời gian bắt đầu và kết thúc thực thi câu lệnh
- Ta sử dụng gettimeofday(&start, NULL) để lấy thời gian hiện tại và lưu vào biến start
- Tiếp đến ta tạo tiến trình con bằng hàm fork()
- Nếu như child\_pid == 0 tức đang trong tiến trình con thì ta thực hiện các lệnh bên trong nó. Lệnh execvp(argv[1], &argv[1]), ta sử dụng hàm execvp để thực thi lệnh shell truyền qua tham số dòng lệnh và lệnh shell này được lưu trong argv[1]. Nếu lệnh shell thất bại thì chương trình con in ra thông báo lỗi và thoát
- Nếu fork thất bại (child\_pid < 0) thì tiến trình cha in ra thông báo lỗi và thoát
- Đối với trường hợp cuối cùng, tức là tiến trình cha, ta thực hiện các lệnh trong khối else như sau: Sử dụng waitpid để đợi tiến trình con kết thúc và lưu trạng thái của tiến trình con vào biến là status, lấy thời gian kết thúc thực thi lệnh bằng gettimeofday(&end, NULL)
- Ở câu lệnh if tiếp theo ta kiểm tra xem tiến trình con kết thúc có lỗi hay không, nếu không thì ta thực hiện việc tính và xuất ra thời gian, do thời gian thực thi câu lệnh có thể rất nhanh nên ta cần phải lấy thêm phần micro giây phía sau để đảm bảo độ chính xác mà thời gian được xuất ra. Vậy ta đã hoàn tất yêu cầu bài toán

### **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"

### **Giải thích cách làm:**

Trước tiên khai báo đầy đủ thư viện, sau đó tạo hàm handleCtrlC để ngắt việc đếm khi gõ tổ hợp phím Ctrl+C

Hàm main:

Sử dụng lệnh printf để in ra câu dòng chữ “Welcome to IT007, I am <your\_Student\_ID>!”

Sử dụng câu lệnh signal(SIGINT, handleCtrlC) để đăng ký xử lý tín hiệu khi nhấn Ctrl+C. Khi tín hiệu SIGINT xuất hiện (người dùng nhấn CTRL+C), chương trình sẽ thực hiện hàm handleCtrlC

Sau đó sử dụng lệnh system("./count.sh 120") để đếm đến 120 trong file count.sh, file count.sh mình giữ nguyên từ file hướng dẫn thực hành Lab3.

### **Code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
void handleCtrlC(int signal)
{
    printf("\ncount.sh has stopped\n");
    exit(0);
}
int main()
{
    char* studentID = "22520521, 22520550, 22520884, 22520967";

    printf("Welcome to IT007, I am %s!\n", studentID);

    signal(SIGINT, handleCtrlC);

    system("./count.sh 120");

    return 0;
}
```

count.sh:

```
#!/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
```

**Khi chạy chương trình, bấm ctrl c giữa chừng để dừng đếm:**

```
donghungpham@22520521-Hung:~/.ssh$ ./Bai03Lab03
Welcome to IT007, I am 22520521, 22520550, 22520884, 22520967!
Implementing: ./count.sh
PPID of count.sh:
donghun+   5907    5906  0 21:30 pts/3    00:00:00 sh -c ./count.sh 120
donghun+   5908    5907  0 21:30 pts/3    00:00:00 /bin/bash ./count.sh 120
donghun+   5910    5908  0 21:30 pts/3    00:00:00 grep count.sh
^Cdonghungpham@22520521-Hung:~/.ssh$
```

**Kết quả:**

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

**Khi để chương trình tiếp tục chạy mà không bấm ctrl C:**

```
donghun+ 5510 5508 0 21:30 pts/3 00:00:00 grep count.sh
● ^Cdonghungpham@22520521-Hung:~/ssh$ ./Bai03Lab03
Welcome to IT007, I am 22520521, 22520550, 22520884, 22520967!
Implementing: ./count.sh
PPID of count.sh:
donghun+ 6015 6014 0 21:32 pts/3 00:00:00 sh -c ./count.sh 120
donghun+ 6016 6015 0 21:32 pts/3 00:00:00 /bin/bash ./count.sh 120
donghun+ 6018 6016 0 21:32 pts/3 00:00:00 grep count.sh
○ donghungpham@22520521-Hung:~/ssh$
```

**Kết quả:**

|     |     |
|-----|-----|
| 108 | 93  |
| 109 | 94  |
| 110 | 95  |
| 111 | 96  |
| 112 | 97  |
| 113 | 98  |
| 114 | 99  |
| 115 | 100 |
| 116 | 101 |
| 117 | 102 |
| 118 | 103 |
| 119 | 104 |
| 120 | 105 |
| 121 | 106 |
| 122 | 107 |
| 123 | 108 |
| 124 | 109 |
| 125 | 110 |
| 126 | 111 |
| 127 | 112 |
| 128 | 113 |
| 129 | 114 |
| 130 | 115 |
| 131 | 116 |
| 132 | 117 |
| 133 | 118 |
| 134 | 119 |
| 135 | 120 |

- Số 120 ở dòng 135 vì khi chạy đoạn chương trình bấm ctrl C giữa chừng đã chạy đến dòng thứ 15 rồi

### **Giải thích kết quả:**

Đối với chương trình bấm Ctrl+C để dừng thì mới vào

+Với câu lệnh printf thì chương trình sẽ in "Welcome IT007, I am <MSSV>"

+Khi đó thì chương trình sẽ in ra dần dần các giá trị bắt đầu từ 1. Khi nhấn Ctrl+C thì chương trình lập tức dừng do câu lệnh signal(SIGINT, handleCtrlC).

Còn đối với chương trình không bấm Ctrl+C thì chương trình sẽ đếm đến 120 rồi mới dừng

#### 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

Giải thích cách làm:

1. Dòng **#include** được sử dụng để bao gồm các thư viện cần thiết cho chương trình.
2. **BUFFER\_SIZE** là kích thước của bộ đệm.
3. Phương thức **main()** là hàm chính của chương trình.
4. **key** được tạo từ tên tệp và một số nguyên (trong trường hợp này là 65) bằng cách sử dụng hàm **ftok** để tạo khóa duy nhất cho bộ nhớ dùng chung.
5. **shmget** được sử dụng để tạo một vùng nhớ dùng chung mới hoặc truy cập vào một vùng nhớ dùng chung hiện có.
6. **shmat** được sử dụng để gắn vùng nhớ dùng chung vào không gian địa chỉ của quá trình hiện tại.
7. **fork** được sử dụng để tạo một quá trình con mới.
8. Vòng lặp **while** trong **if (pid == 0)** được sử dụng để tiêu thụ dữ liệu từ bộ nhớ dùng chung và tính tổng của dữ liệu tiêu thụ.
9. Vòng lặp **while** trong **else** được sử dụng để tạo dữ liệu ngẫu nhiên và đưa chúng vào bộ nhớ dùng chung, tính tổng dữ liệu được tạo ra.
10. **shmdt** được sử dụng để tách vùng nhớ dùng chung khỏi không gian địa chỉ của quá trình.
11. **shmctl** được sử dụng để kiểm soát hoặc giải phóng vùng nhớ dùng chung.

Code:



```
C bai4.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/ipc.h>
6  #include <sys/shm.h>
7  #include <sys/wait.h>
8
9  #define BUFFER_SIZE 10
10
11 int main() {
12     int shmid;
13     key_t key = ftok("shmfile", 65);
14     shmid = shmget(key, BUFFER_SIZE, 0666|IPC_CREAT);
15
16     int *buffer = (int *)shmat(shmid, (void *)0, 0);
17     *buffer = 0;
18     int sum = 0;
19
20     int pid = fork();
21
22     if (pid < 0) {
23         fprintf(stderr, "Fork failed");
24         return 1;
25     } else if (pid == 0) {
26         while (1) {
27             if (*buffer > 0) {
28                 printf("Consumed: %d\n", *buffer);
29                 sum += *buffer;
30                 *buffer = 0;
31                 if (sum >= 100) {
32                     break;
33                 }
34             }
35             sleep(1);
36         }
37     } else {
38         while (1) {
39             if (sum >= 100) {
40                 break;
41             }
42             int random_number = (rand() % (20 - 10 + 1)) + 10;
43             *buffer = random_number;
44             printf("Produced: %d\n", random_number);
45             sum += random_number;
46             sleep(1);
47         }
48         printf("Sum: %d\n", sum);
49     }
50 }
51
52 shmdt(buffer);
53 shmctl(shmid, IPC_RMID, NULL);
54
55 return 0;
56 }
57
```

Kết quả:

```
nguyen-22520967@nguyen22520967-VirtualBox:~$ gcc bai4.c -o bai4
nguyen-22520967@nguyen22520967-VirtualBox:~$ ./bai4
Produced: 16
Consumed: 16
Produced: 20
Consumed: 20
Produced: 16
Consumed: 16
Produced: 12
Consumed: 12
Produced: 11
Consumed: 11
Produced: 14
Consumed: 14
Produced: 10
Consumed: 10
Produced: 16
Consumed: 16
Sum: 115
nguyen-22520967@nguyen22520967-VirtualBox:~$
```

Giải thích kết quả là:

1. Mã bắt đầu bằng việc bao gồm các thư viện cần thiết và định nghĩa các hằng số cần dùng.
2. Sau đó, mã tạo một vùng nhớ dùng chung với kích thước `BUFFER_SIZE` và lấy khóa duy nhất từ tệp `"shmfile"` và số 65 bằng cách sử dụng hàm `ftok`.
3. Sau đó, mã sử dụng `shmat` để gắn vùng nhớ dùng chung vào không gian địa chỉ của tiến trình hiện tại.
4. Một quá trình con mới được tạo bằng `fork()`.
5. Trong quá trình con, vòng lặp `while` kiểm tra nếu giá trị trong bộ nhớ dùng chung lớn hơn 0, nó sẽ tiêu thụ giá trị đó và tính tổng của các giá trị đã tiêu thụ.
6. Trong quá trình cha, vòng lặp `while` tạo ra một số ngẫu nhiên và đưa giá trị đó vào bộ nhớ dùng chung, tính tổng của tất cả các giá trị đã tạo.
7. Khi tổng đạt được giá trị lớn hơn hoặc bằng 100, cả hai tiến trình đều dừng lại và thông báo kết quả.

=> Kết quả đầu ra của chương trình sẽ hiển thị các dòng chứa thông tin về quá trình sản xuất và tiêu thụ dữ liệu. Cụ thể, nó sẽ in ra các dòng như "Produced: X" hoặc "Consumed: X" để biểu thị giá trị dữ liệu được tạo ra hoặc tiêu thụ. Khi tổng của dữ liệu sản xuất và tiêu thụ đạt giá trị lớn hơn hoặc bằng 100, chương trình sẽ in ra dòng "Sum: X" để hiển thị tổng của tất cả các giá trị.

## 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:

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

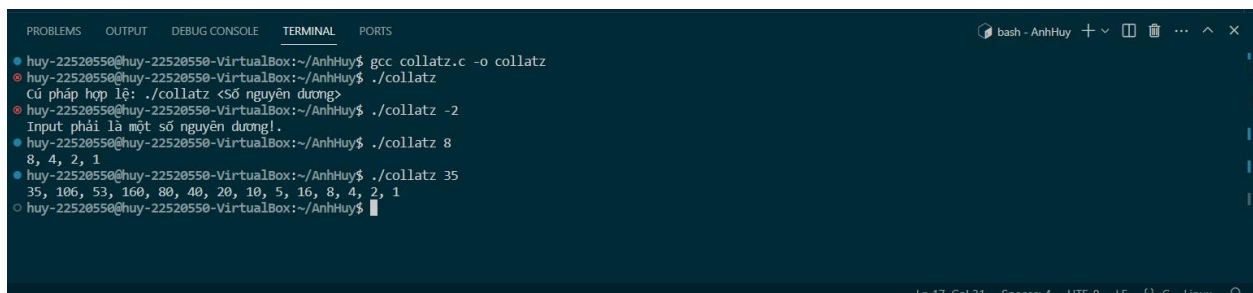
Phỏng đoán phát biểu rằng khi thuật toán này được áp dụng liên tục, tất cả số nguyên dương đều sẽ tiến đến 1. Ví dụ, với  $n = 35$ , ta sẽ có chuỗi kết quả như sau:

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.

Cách làm: Đầu tiên ta cho người dùng nhập vào một số và kiểm tra xem có phải số nguyên dương không để tiếp tục. Sau đó tạo tiến trình con, tiến trình cha và con chia sẻ một bộ nhớ để truyền kết quả, tiến trình con sẽ tính toán chuỗi collatz và lưu kết quả vào bộ nhớ chung đó, tiến trình cha đợi tiến trình con hoàn thành, sau đó đọc kết quả từ bộ nhớ.

Hình ảnh:



```
huy-22520550@huy-22520550-VirtualBox:~/AnhHuy$ gcc collatz.c -o collatz
huy-22520550@huy-22520550-VirtualBox:~/AnhHuy$ ./collatz
Cú pháp hợp lệ: ./collatz <Số nguyên dương>
huy-22520550@huy-22520550-VirtualBox:~/AnhHuy$ ./collatz -2
Input phải là một số nguyên dương!
huy-22520550@huy-22520550-VirtualBox:~/AnhHuy$ ./collatz 8
8, 4, 2, 1
huy-22520550@huy-22520550-VirtualBox:~/AnhHuy$ ./collatz 35
35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1
huy-22520550@huy-22520550-VirtualBox:~/AnhHuy$
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6 #include <sys/ipc.h>
7 #include <sys/shm.h>
8 int collatz(int n) {
9     if (n % 2 == 0) {
10         return n / 2;
11     } else {
12         return 3 * n + 1;
13     }
14 }
15 int main(int argc, char *argv[]) {
16     if (argc != 2) {
17         printf("Cú pháp hợp lệ: %s <Số nguyên dương>\n", argv[0]);
18         return 1;
19     }
20     int n = atoi(argv[1]);
21     if (n <= 0) {
22         printf("Input phải là một số nguyên dương!.\n");
23         return 1;
24     }
25     int shmid = shmget(IPC_PRIVATE, 4096, IPC_CREAT | 0666);
26     if (shmid == -1) {
27         perror("shmget");
28         return 1;
29     }
30     int *buffer = (int *)shmat(shmid, NULL, 0);
31     if (buffer == (int *)(-1)) {
32         perror("shmat");
33         return 1;
34     }
35     buffer[0] = n;
36     pid_t child_pid = fork();
37     if (child_pid == -1) {
38         perror("fork");
39         return 1;
40     }
41     if (child_pid == 0) {
42         int index = 1;
43         while (n != 1) {
44             n = collatz(n);
45             buffer[index++] = n;
46         }
47         exit(0);
48     } else {
49         wait(NULL);
50         int index = 0;
51         while (buffer[index] != 0) {
52             printf("%d", buffer[index]);
53             if (buffer[index + 1] != 0) {
54                 printf(", ");
55             }
56             index++;
57         }
58         printf("\n");
59         shmdt(buffer);
60         shmctl(shmid, IPC_RMID, NULL);
61     }
62     return 0;
63 }
64
```

Giải thích:

- Khai báo các thư viện cần thiết để thực hiện yêu cầu của bài toán và quan trọng nhất là sử dụng bộ nhớ chia sẻ bằng thư viện <sys/shm.h> và sử dụng hệ thống key IPC bằng thư viện <sys/ipc.h>
- Ta định nghĩa hàm collatz để tính toán chuỗi collatz cho một số nguyên n
- Vì input là ./collatz <số nguyên dương> nên ta sẽ kiểm tra điều kiện nhập với argc != 2, tức là nếu số lượng tham số đầu vào khác 2 thì ta sẽ in ra cú pháp đúng cho người dùng và thoát chương trình, nếu đúng cú pháp thì ta tiếp tục
- Tiếp đến là kiểm tra kiểu số mà người dùng nhập, nếu không phải là số nguyên dương thì in ra thông báo và thoát chương trình, nếu đúng là số nguyên dương thì ta tiếp tục
- int shmids = shmget(IPC\_PRIVATE, 4096, IPC\_CREAT | 0666); dùng để tạo một shared memory segment và trả về một ID (shmids) để xác định nó. Dùng 'IPC\_PRIVATE' để tạo một key duy nhất cho segment
- int \*buffer = (int \*)shmat(shmids, NULL, 0): kết nối tiến trình hiện tại đến share memory segment bằng cách sử dụng shmids. Con trỏ buffer trỏ đến share memory này
- Lưu giá trị n vào share memory bằng buffer[0] = n và bắt đầu quá trình tính toán chuỗi Collatz
- Tạo tiến trình con và kiểm tra các trường hợp, nếu tạo không thành công thì báo lỗi và kết thúc chương trình
- Trong tiến trình con (child\_pid == 0), ta tính toán chuỗi Collatz và lưu kết quả vào share memory
- while (buffer[index] != 0) { ... } ta duyệt qua mảng kết quả trong share memory và in ra từng phần tử, vì mỗi phần tử cách nhau bởi dấu phẩy nên ta kiểm tra phần tử tiếp theo có tồn tại hay không để quyết định có cần in dấu phẩy hay không
- shmdt(buffer) dùng để ngắt kết nối tiến trình cha với share memory và shmctl(shmids, IPC\_RMID, NULL) để xóa share memory segment
- Vậy ta đã hoàn thành yêu cầu bài toán