Name: Đỗ Hoàng Anh

ID: 22520041

Class: IT007.O212.1

OPERATING SYSTEM
LAB 3 REPORT

SUMMARY

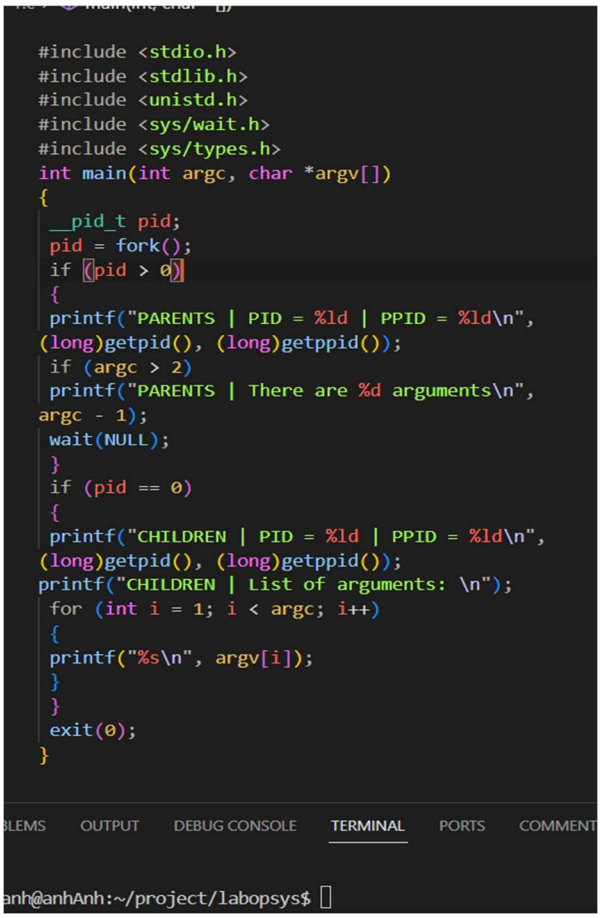
Self-scrores: 10/10

Note: Export file to **PDF and name the file by following format:*
LAB X – <Student ID>.pdf

3.5 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:



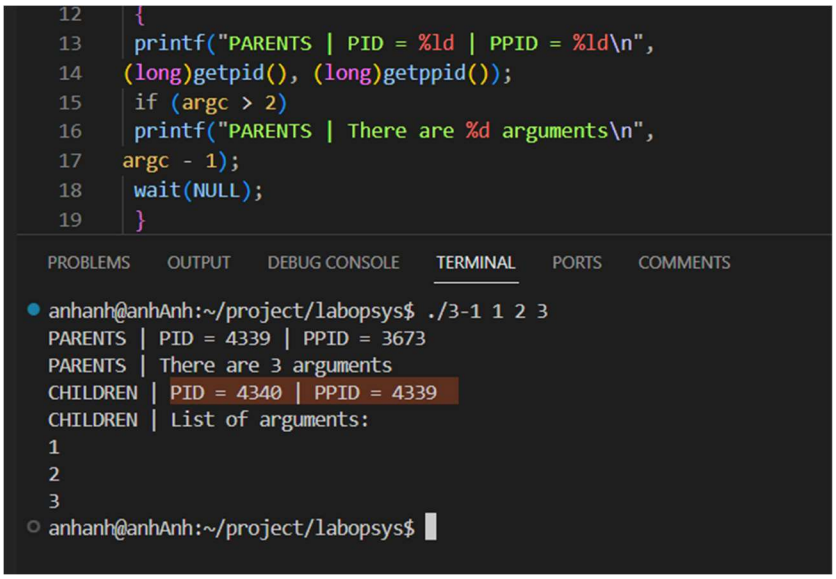
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
int main(int argc, char *argv[])
{
 __pid_t pid;
 pid = fork();
 if (pid > 0)
 {
 printf("PARENTS | PID = %ld | PPID = %ld\n",
(long)getpid(), (long)getppid());
 if (argc > 2)
 printf("PARENTS | 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);
}
```

```
BLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENT

anh@anhAnh:~/project/labopsys$
```

➔ Giải thích code:

- Đầu tiên trong hàm main pid được khởi tạo theo kiểu dữ liệu __pid_t dùng để lưu trử id của tiến trình.
- Hàm fork() được gọi để tạo ra tiến trình mới. Sau khi được gọi thành công 2 tiến trình cha và con sẽ được tạo ra.
- Pid sẽ có những giá trị khác nhau với tiến trình cha và con, nếu pid = 0 thì nó là tiến trình con và ngược lại là tiến trình cha.
- Trong câu lệnh if dùng để kiểm tra xem liệu nó là tiến trình cha hay con dựa vào pid nếu nó là tiến trình cha (pid > 0) thì sẽ in ra thông tin pid của tiến trình và ppid của tiến trình cha. Và đồng thời cũng kiểm tra xem nếu có nhiều hơn 2 argc thì sẽ in ra tổng argc. Và sau đó nó chờ tiến trình con kết thúc bằng việc sử dụng wait().

- Tương tự nếu là tiến trình con (pid = 0) thì sẽ in ra thông tin về pid của tiến trình và ppid(tiến trình cha của nó). Sau đó sẽ in ra thông tin từng argument được đưa vào.



➔ Giải thích kết quả khi xuất:

- Đầu tiên ta chạy chương trình và đưa vào 3 argument là (1,2,3).
- Sau khi chạy hàm fork() sẽ khởi tạo tiến trình cha và con và sẽ dùng pid để kiểm tra xem thì trong trường hợp này đầu tiên pid > 0 nên nó sẽ là tiến trình cha và sẽ in ra pid = 4339 và ppid=3673 của tiến trình này sau nó sẽ kiểm tra nếu số lượng argument > 2 thì sẽ xuất ra, trong trường hợp này có 3 argument và dùng lệnh wait() để chờ tới khi tiến trình con kết thúc.
- Nếu pid = 0 thì nó là tiến trình con và lúc này sẽ in ra thông tin pid = 4340 và ppid = 4339 ta thấy tiến trình con có ppid=pid(của tiến trình cha). Và sau đó sẽ xuất ra từng giá trị được bỏ và của arguments nến lúc này sẽ xuất ra 1, 2 và 3.

Ví dụ 3-2:

```c
/*####################################
# University of Information Technology #
# IT007 Operating System #
# Do Hoang Anh, 22520041 #
# 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();
 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);
```

➔ Giải thích code:

- **__pid_t pid:** Khai báo biến pid kiểu __pid_t. __pid_t là kiểu dữ liệu dùng để lưu trữ Process ID (PID).
- **pid = fork():** Gọi hàm fork() để tạo một bản sao của tiến trình hiện tại. Sau khi gọi fork(), hai tiến trình sẽ tiếp tục thực thi code từ dòng này, nhưng trong hai không gian bộ nhớ khác nhau (tiến trình cha và tiến trình con).
- **if (pid > 0) { ... }:** Nếu pid lớn hơn 0, đây là tiến trình cha. Tiến trình cha sẽ in ra thông tin về PID của nó và PID của tiến trình cha của nó (PPID), sau đó kiểm tra xem có bao nhiêu đối số được truyền vào chương trình và in ra số lượng đối số.
- **wait(NULL):** Tiến trình cha sẽ chờ cho đến khi tiến trình con kết thúc (bằng cách gọi hàm wait() mà không truyền vào tham số), tránh việc tiến trình cha kết thúc trước khi tiến trình con kết thúc.

- **if (pid == 0) { ... }:** Nếu pid bằng 0, đây là tiến trình con. Tiến trình con sẽ thực hiện gọi hàm execl() để thực thi một script shell có tên là "count.sh" với đối số là "10".
- **execl("./count.sh", "./count.sh", "10", NULL):** Thực thi script shell "count.sh" với tham số "10". Đối số cuối cùng của execl() phải là NULL.
- **exit(0)**: Sau khi tiến trình con kết thúc, tiến trình con sẽ thoát.

➔ Giải thích kết quả khi xuất:

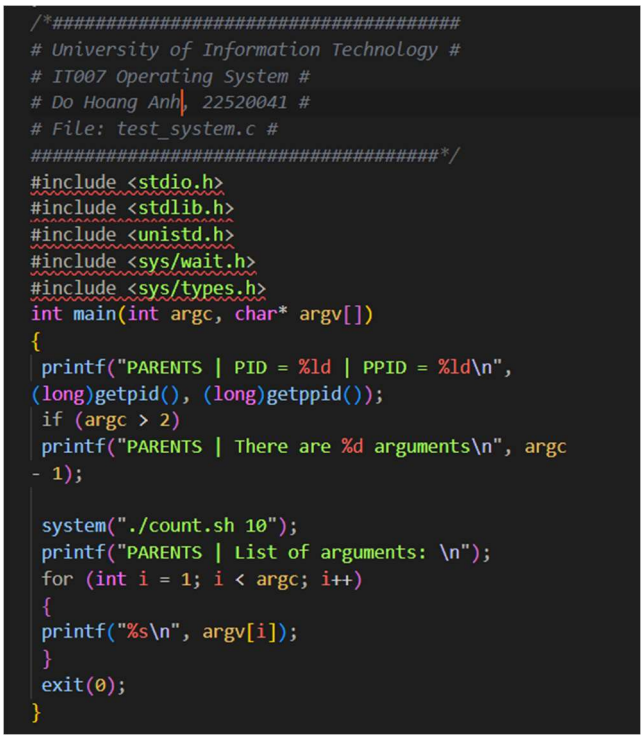
```
1  /*#######################################
2    # University of Information Technology #
3    # IT007 Operating System #
4    # Do Hoang Anh, 22520041 #
5    # File: test_execl.c #
6    #######################################*/
7  #include <stdio.h>
```

```
PROBLEMS 6   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMMENTS

anhanh@anhAnh:~/lab3_hdh$ ./test_excel 1 2 3
PARENTS | PID = 2640 | PPID = 1540
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
anhanh    2641  2640  0 12:19 pts/7    00:00:00 /bin/bash ./count.sh 10
anhanh    2643  2641  0 12:19 pts/7    00:00:00 grep count.sh
anhanh@anhAnh:~/lab3_hdh$
```

- **PARENTS | PID = 2640:** PID của tiến trình cha là 2640.
- **PPID = 1540:** PPID (Parent Process ID) của tiến trình cha là 1540.
- **There are 3 arguments:** Có 3 đối số được truyền vào chương trình.
- **anhanh 2641**: Tên người dùng là "anhanh", PID của script shell là 2641. 2640: PPID của script shell là 2640, tức là PID của tiến trình cha.Và tiến trình con thực thi script shell "count.sh" với tham số "10". Khi thực thi, script shell in ra PID của chính nó và PPID.
- **grep count.sh** đã được thực thi và đang chạy trong tiến trình có PID là 2643, PPID là 2641 (nghĩa là tiến trình cha của nó là script shell có PID là 2641), và đang chạy trên terminal pts/7.

Ví dụ 3-3:

➔ Giải thích code:



```c
/*###################################
# University of Information Technology #
# IT007 Operating System #
# Do Hoang Anh, 22520041 #
# 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 = %ld | PPID = %ld\n", (long)getpid(), (long)getppid()):** Dòng này in ra thông tin về PID (Process ID) của tiến trình hiện tại và PPID (Parent Process ID) của tiến trình cha của nó.
- **if (argc > 2) printf("PARENTS | There are %d arguments\n", argc - 1):** Điều kiện này kiểm tra xem có ít nhất 2 đối số được truyền vào chương trình không. Nếu có, nó in ra số lượng đối số trừ đi 1, bởi vì đối số đầu tiên thường là tên của chương trình.
- **system("./count.sh 10"):** Dòng này gọi hàm system() để thực thi lệnh shell "count.sh 10". system() là một hàm trong C cho phép bạn thực thi các lệnh shell từ bên trong chương trình C của mình.
- **printf("PARENTS | List of arguments: \n"):** Dòng này in ra dòng chữ "List of arguments: ".
- **for (int i = 1; i < argc; i++) { printf("%s\n", argv[i]); }:** Vòng lặp này in ra các đối số được truyền vào chương trình, bắt đầu từ đối số thứ hai (vì đối số đầu tiên là tên của chương trình).
- **exit(0):** Dòng này thoát khỏi chương trình với mã trạng thái 0, chỉ ra rằng chương trình đã thực hiện xong mà không gặp phải lỗi.

➔ Giải thích kết quả khi xuất:

```c
/*###################################
# University of Information Technology #
# IT007 Operating System #
# Do Hoang Anh, 22520041 #
# 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());
```

```
PROBLEMS 6  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

anhanh@anhAnh:~/lab3_hdh$ ./test_system 1 2 3
PARENTS | PID = 4484 | PPID = 1540
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
anhanh    4485  4484  0 12:37 pts/7    00:00:00 sh -c ./count.sh 10
anhanh    4486  4485  0 12:37 pts/7    00:00:00 /bin/bash ./count.sh 10
anhanh    4488  4486  0 12:37 pts/7    00:00:00 grep count.sh
PARENTS | List of arguments:
1
2
3
```

Dòng đầu tiên in ra thông tin về PID của tiến trình cha (4484) và PPID của nó (1540).

Dòng thứ hai in ra rằng có 3 đối số được truyền vào chương trình.

Tiến trình cha sau đó thực thi hàm system("./count.sh 10");. Hàm này thực thi lệnh shell "count.sh 10" từ trong chương trình C. Kết quả của việc thực thi lệnh shell "count.sh 10" được in ra:

- Dòng thứ nhất in ra thông tin về tiến trình shell (sh) được tạo ra để thực thi lệnh "count.sh 10". PPID của tiến trình shell này là 4484 (PID của tiến trình cha).
- Dòng thứ hai in ra thông tin về tiến trình bash được tạo ra bởi tiến trình shell ở trên để thực thi tệp tin "count.sh" với đối số là "10". PPID của tiến trình bash này là 4485 (PID của tiến trình shell).
- Dòng thứ ba là kết quả của lệnh grep count.sh, có thể là kết quả từ việc bạn đang sử dụng một lệnh grep trong script "count.sh" để tìm kiếm các dòng chứa "count.sh", hoặc là một lệnh khác không liên quan.
- Cuối cùng, các đối số được truyền vào chương trình (1, 2, 3) được in ra

Ví dụ 3-4:

➔ Giải thích code:

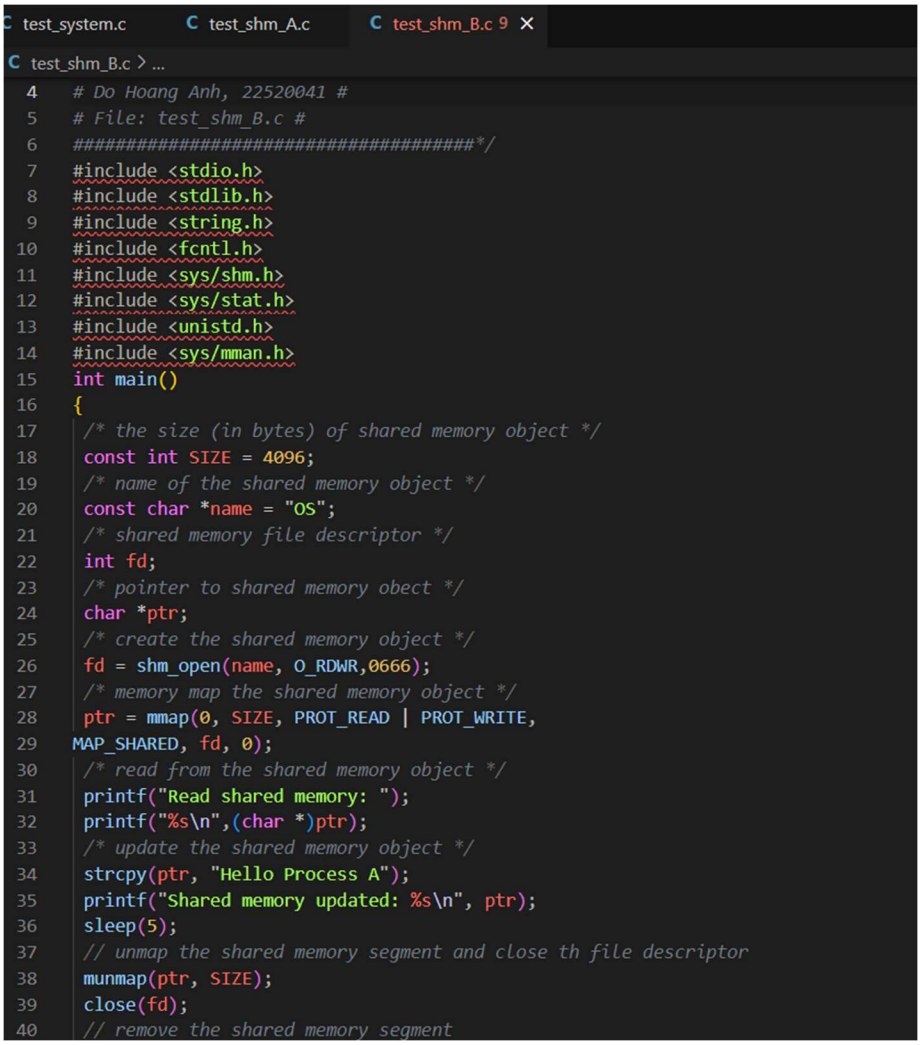
Process A:

```c
# University of Information Technology #
# IT007 Operating System #
# Do Hoang Anh, 22520041 #
# File: test_shm_A.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>
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");
```

- **const int SIZE = 4096:** Định nghĩa kích thước của vùng shared memory (4KB trong trường hợp này).
- **const char *name = "OS":** Đặt tên cho vùng shared memory. Tên này cần phải được chia sẻ giữa các quá trình muốn truy cập shared memory.
- **int fd:** File descriptor sẽ được sử dụng để tham chiếu đến vùng shared memory.
- **char *ptr:** Con trỏ sẽ trỏ đến vùng shared memory.
- fd = shm_open(name, O_CREAT | O_RDWR,0666): Tạo hoặc mở shared memory object.
- **Hàm shm_open()** tạo ra một shared memory object mới hoặc mở một shared memory object đã tồn tại. Trong trường hợp này, chế độ O_CREAT được sử dụng để tạo một shared memory object mới và O_RDWR được sử dụng để mở shared memory object cho việc đọc và ghi. Tham số cuối cùng là quyền truy cập (0666 cho quyền đọc và ghi cho tất cả các người dùng).

- **ftruncate(fd, SIZE):** Cấu hình kích thước của shared memory object. Hàm này cắt shared memory object đến kích thước đã chỉ định.
- **ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0):** Ánh xạ shared memory object vào vùng nhớ của quá trình. Hàm này trả về một con trỏ (ptr) trỏ đến vùng nhớ của shared memory object. Quyền truy cập được đặt là đọc và ghi (PROT_READ | PROT_WRITE) và các thay đổi sẽ được phản ánh ngay lập tức đến shared memory object (MAP_SHARED).
- **strcpy(ptr, "Hello Process B"):** Ghi dữ liệu vào shared memory object. Trong trường hợp này, chuỗi "Hello Process B" được sao chép vào vùng nhớ của shared memory object.
- **while (strncmp(ptr, "Hello Process B", 15) == 0) { ... }:** Vòng lặp này đợi cho đến khi quá trình B cập nhật shared memory object. Nó kiểm tra xem liệu chuỗi "Hello Process B" đã được ghi vào shared memory object hay không. Nếu chuỗi chưa được thay đổi, vòng lặp tiếp tục chạy, với mỗi lần chờ 1 giây trước khi kiểm tra lại.
- **printf("Memory updated: %s\n", (char *)ptr):** Sau khi shared memory object đã được cập nhật bởi quá trình B, thông điệp này in ra nội dung của shared memory object.
- **munmap(ptr, SIZE):** Hủy bỏ ánh xạ của shared memory object khỏi vùng nhớ của quá trình.
- **close(fd):** Đóng file descriptor của shared memory object.

Process B:

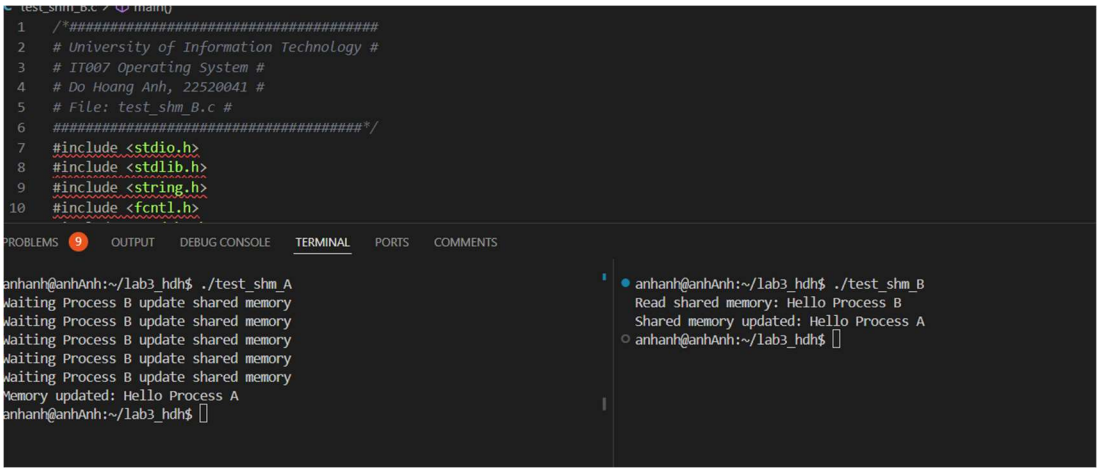


- **const int SIZE = 4096:** Định nghĩa kích thước của vùng shared memory (trong trường hợp này là 4096 bytes hoặc 4KB).
- **const char *name = "OS":** Đặt tên cho vùng shared memory. Tên này phải được chia sẻ giữa các quá trình muốn truy cập vào shared memory.
- **int fd:** File descriptor sẽ được sử dụng để tham chiếu đến vùng shared memory.
- **char *ptr:** Con trỏ sẽ trỏ đến vùng shared memory.
- **fd = shm_open(name, O_RDWR, 0666):** Mở shared memory object đã được tạo trước đó. Hàm shm_open() sẽ mở một shared memory object đã tồn tại với quyền truy cập đọc và ghi (O_RDWR). Tham số cuối cùng là quyền truy cập (0666 cho quyền đọc và ghi cho tất cả các người dùng).
- **ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0):** Ánh xạ shared memory object vào vùng nhớ của quá trình. Hàm này trả về một con trỏ (ptr) trỏ đến vùng nhớ của shared

memory object. Quyền truy cập được đặt là đọc và ghi (PROT_READ | PROT_WRITE) và các thay đổi sẽ được phản ánh ngay lập tức đến shared memory object (MAP_SHARED).

- **printf("Read shared memory: "):** In ra một thông điệp để chỉ ra rằng dữ liệu được đọc từ shared memory object.
- **printf("%s\n",(char *)ptr):** In ra nội dung của shared memory object.
- **strcpy(ptr, "Hello Process A"):** Ghi dữ liệu vào shared memory object. Trong trường hợp này, chuỗi "Hello Process A" được sao chép vào vùng nhớ của shared memory object.
- **printf("Shared memory updated: %s\n", ptr):** In ra một thông điệp để chỉ ra rằng shared memory object đã được cập nhật với dữ liệu mới.
- **sleep(5):** Tiến trình tạm ngừng trong 5 giây.
- munmap(ptr, SIZE)**:** Hủy bỏ ánh xạ của shared memory object khỏi vùng nhớ của quá trình. close(fd): Đóng file descriptor của shared memory object.
- **shm_unlink(name):** Xóa shared memory object khỏi hệ thống. Điều này sẽ làm cho shared memory object không còn tồn tại sau khi tất cả các tiến trình đã kết thúc sử dụng nó.

➔ Giải thích kết quả khi chạy



- Khi chạy test_shm_A:

  Chương trình hiển thị thông điệp "Waiting Process B update shared memory" mỗi giây, cho đến khi dữ liệu trên shared memory object được cập nhật bởi quá trình B.

  Sau khi quá trình B cập nhật shared memory object, chương trình hiển thị thông điệp "Memory updated: Hello Process A" và kết thúc.

- Khi chạy test_shm_B:

  Chương trình hiển thị thông điệp "Read shared memory: Hello Process B", đọc dữ liệu từ shared memory object và in ra màn hình.

Sau đó, chương trình cập nhật shared memory object với dữ liệu mới là "Hello Process A" và hiển thị thông điệp "Shared memory updated: Hello Process A".

Chương trình kết thúc.

- Giải thích:

Quá trình A và quá trình B đang chia sẻ một shared memory object có tên "OS".

Quá trình A chờ đợi cho đến khi quá trình B cập nhật shared memory object.

Trong khi đợi, nó in ra thông điệp "Waiting Process B update shared memory".

Quá trình B đọc dữ liệu từ shared memory object và in ra màn hình. Sau đó, nó cập nhật shared memory object với dữ liệu mới.

Khi quá trình B cập nhật shared memory object, quá trình A nhận biết và tiếp tục thực thi.

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 44 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.

➔ Đầu tiên ta sẽ tạo 1 file với tên là time.c và thiết lập mã nguồn theo yêu cầu

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

int main(int argc, char *argv[]) {
    // Kiểm tra số lượng đối số
    if (argc < 2) {
        fprintf(stderr, "Sử dụng: %s <command>\n", argv[0]);
        return 1;
    }

    // Khai báo biến
    struct timeval start, end;
    pid_t pid;
    int status;

    // Lấy thời gian bắt đầu
    gettimeofday(&start, NULL);

    // Fork một tiến trình con
    pid = fork();

    if (pid < 0) {
        // Fork không thành công
        perror("fork");
        return 1;
    } else if (pid == 0) {
        // Tiến trình con: thực thi lệnh shell
        execl("/bin/sh", "sh", "-c", argv[1], (char *)NULL);
        // Nếu exec lỗi, in ra thông báo và thoát
        perror("exec");
```

```
        exit(1);
    } else {
        // Tiến trình cha: đợi tiến trình con kết thúc
        waitpid(pid, &status, 0);

        int gettimeofday()  thúc
        gettimeofday(&end, NULL);

        // Tính thời gian thực thi
        double exec_time = (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\n", exec_time);
    }

    return 0;
}
```

- Chương trình này nhận lệnh shell cần đo thời gian thực thi thông qua đối số dòng lệnh.
- Sử dụng hàm gettimeofday() để lấy thời gian trước và sau khi thực thi lệnh shell.
- Sử dụng hàm fork() để tạo ra một tiến trình con để thực thi lệnh shell.
- Tiến trình con sử dụng hàm execl() để thực thi lệnh shell.
- Tiến trình cha sử dụng hàm waitpid() để chờ tiến trình con kết thúc.
- Sau khi tiến trình con kết thúc, tính toán thời gian thực thi và in ra kết quả.

- Chạy chương trình:

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 stopped”

➔ Đầu tiên ta tạo 1 file có tên là bai3.c với mã nguồn theo thực thi theo yêu cầu.

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

void handler(int signum) {
    if (signum == SIGINT) {
        printf("count.sh has stopped\n");
        exit(0);
    }
}

int main() {
    // In ra dòng chữ chào mừng
    printf("Welcome to IT007, I am 22520041!\n");

    // Đăng ký signal handler để bắt SIGINT (CTRL+C)
    signal(SIGINT, handler);

    // Thực thi count.sh với số lần đếm là 120
    system("./count.sh 120");

    return 0;
}
```

- Chương trình bắt đầu bằng việc in ra dòng chữ chào mừng.
- Tiếp theo, nó đăng ký một signal handler bằng hàm signal() để bắt tín hiệu SIGINT (CTRL+C).
- Sau đó, nó thực thi script count.sh với số lần đếm là 120 bằng hàm system().
- Khi người dùng nhấn CTRL+C, signal handler sẽ được gọi, in ra dòng chữ thông báo và kết thúc chương trình bằng lệnh exit(0).

➔ Thêm dòng lệnh “trap 'echo "count.sh has stopped"; exit' SIGINT” vào “count.sh” để xử lý tín hiệu SIGINT và sau đó gọi hàm exit() nếu tín hiệu được nhận.

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

➔ Kết quả khi chạy.

```
PROBLEMS 5  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

anhanh@anhAnh:~/lab3_hdh$ ./bai3
Welcome to IT007, I am 22520041!
Implementing: ./count.sh
PPID of count.sh:
anhanh   17165 17164  0 14:12 pts/4    00:00:00 sh -c ./count.sh 120
anhanh   17166 17165  0 14:12 pts/4    00:00:00 /bin/bash ./count.sh 120
anhanh   17168 17166  0 14:12 pts/4    00:00:00 grep count.sh
^Ccount.sh has stopped
anhanh@anhAnh:~/lab3_hdh$
```

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. 45
- 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

➔ Ta tạo 1 file có tên là bai4.c với mã nguồn thực hiện theo yêu cầu bài toán.

bai4.c > main()

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

#define BUFFER_SIZE 10

// Cấu trúc dữ liệu của buffer
typedef struct {
    int data[BUFFER_SIZE];
    int count;
    int sum;
} Buffer;

int main() {
    // Tạo key để tạo shared memory
    key_t key = ftok("buffer", 65);

    // Tạo segment của shared memory
    int shmid = shmget(key, sizeof(Buffer), IPC_CREAT | 0666);

    // Attach segment vào vùng nhớ
    Buffer *buffer = (Buffer *)shmat(shmid, NULL, 0);

    // Khởi tạo buffer
    buffer->count = 0;
    buffer->sum = 0;

    // Fork để tạo tiến trình con
    pid_t pid = fork();

    if (pid < 0) {
        fprintf(stderr, "Fork failed\n");
        return 1;
    }
```

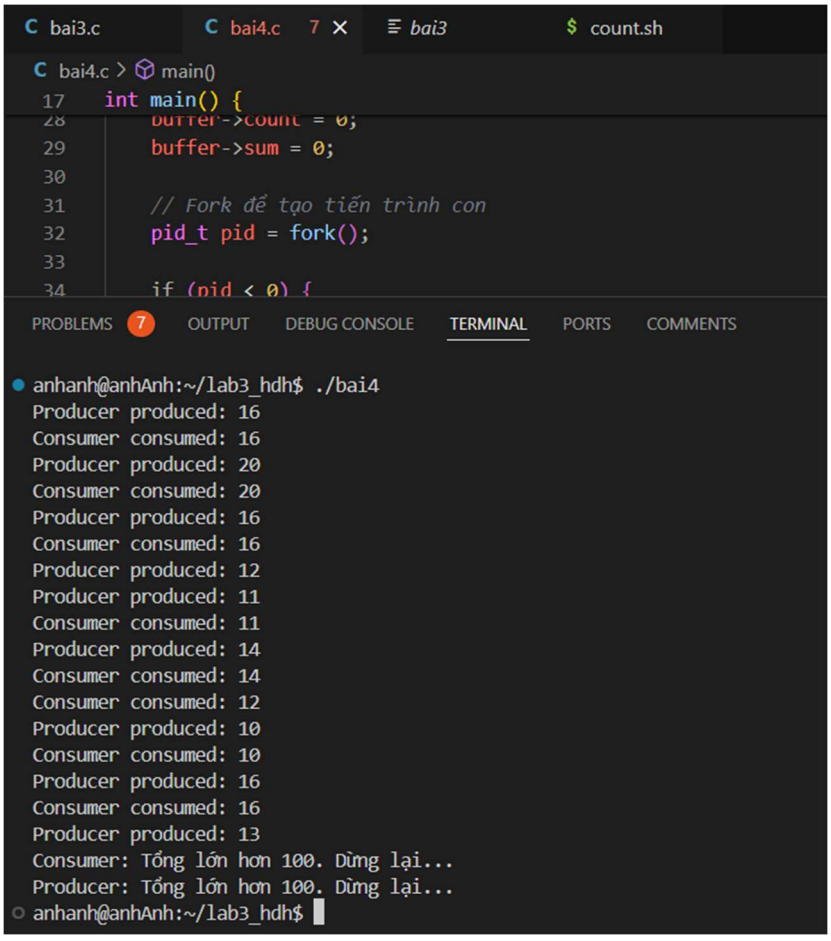
```
int main() {
    if (pid == 0) {
        // Tiến trình con (Consumer)
        while (1) {
            // Kiểm tra điều kiện dừng
            if (buffer->sum > 100) {
                printf("Consumer: Tổng lớn hơn 100. Dừng lại...\n");
                break;
            }

            // Kiểm tra buffer trống
            if (buffer->count > 0) {
                // Đọc dữ liệu từ buffer
                int num = buffer->data[buffer->count - 1];
                buffer->count--;
                printf("Consumer consumed: %d\n", num);

                // Tính tổng
                buffer->sum += num;
            }

            // Ngừng một chút trước khi đọc dữ liệu mới từ buffer
            usleep(500000);
        }
    } else {
        // Tiến trình cha (Producer)
        while (1) {
            // Kiểm tra điều kiện dừng
            if (buffer->sum > 100) {
                printf("Producer: Tổng lớn hơn 100. Dừng lại...\n");
                break;
            }

            // Kiểm tra buffer đầy
            if (buffer->count < BUFFER_SIZE) {
                // Tạo số ngẫu nhiên từ 10 đến 20
                int num = rand() % 11 + 10;
```

```
                // Ghi dữ liệu vào buffer
                buffer->data[buffer->count] = num;
                buffer->count++;
                printf("Producer produced: %d\n", num);
            }

            // Ngừng một chút trước khi thêm số mới vào buffer
            usleep(500000);
        }

        // Chờ tiến trình con kết thúc
        wait(NULL);

        // Detach và xóa shared memory
        shmdt(buffer);
        shmctl(shmid, IPC_RMID, NULL);
    }

    return 0;
}
```

- Đầu tiên, nó tạo một key từ tên "buffer" để sử dụng cho việc tạo shared memory.

- Sau đó, nó tạo một segment của shared memory với kích thước là sizeof(Buffer) và quyền truy cập IPC_CREAT | 0666. Tiến trình cha (Producer) và tiến trình con (Consumer) được tạo ra bằng cách sử dụng fork().
- Trong tiến trình con, nếu tổng vượt quá 100, nó in ra thông báo và kết thúc vòng lặp.
- Trong tiến trình con, nếu buffer không rỗng, nó lấy phần tử cuối cùng ra khỏi buffer, in ra màn hình và cập nhật tổng.
- Trong tiến trình cha, nếu tổng vượt quá 100, nó cũng in ra thông báo và kết thúc vòng lặp.
- Trong tiến trình cha, nếu buffer chưa đầy, nó tạo một số ngẫu nhiên từ 10 đến 20, ghi vào buffer và in ra màn hình.
- Sau khi tiến trình cha kết thúc, nó chờ tiến trình con kết thúc, sau đó detach và xóa shared memory.

➔ Kết quả khi chạy:



3.6 Bài tập ôn tập

Phỏng đoán Collatz xem xét chuyện gì sẽ xảy ra nếu ta lấy một số nguyên dương bất kỳ và áp dụng theo thuật toán sau đây:

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

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

➔ Đầu tiên ta tạo 1 file với tên là collatz.c và mã nguồn theo yêu cầu:

collatz.c 7 ×

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

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

#define BUFFER_SIZE 100

// Cấu trúc dữ liệu của buffer
typedef struct {
    int data[BUFFER_SIZE];
    int count;
} Buffer;

// Kiểm tra xem số nguyên nhập vào có hợp lệ không
int isValidInput(char *input) {
    while (*input != '\0') {
        if (*input < '0' || *input > '9')
            return 0;
        input++;
    }
    return 1;
}
```

```c
// Tính toán chuỗi Collatz và ghi vào buffer
void collatzSequence(int n, Buffer *buffer) {
    buffer->count = 0;
    while (n != 1) {
        buffer->data[buffer->count++] = n;
        if (n % 2 == 0)
            n = n / 2;
        else
            n = 3 * n + 1;
    }
    buffer->data[buffer->count++] = 1; // Đặc biệt, kết thúc chuỗi

}

int main(int argc, char *argv[]) {
    // Kiểm tra số lượng đối số dòng lệnh
    if (argc != 2 || !isValidInput(argv[1])) {
        printf("Input phải là số nguyên dương\n");
        return 1;
    }

    // Chuyển đổi đối số dòng lệnh thành số nguyên
    int startNumber = atoi(argv[1]);
    if (startNumber <= 0) {
        printf("Input phải là số nguyên dương\n");
        return 1;
    }

    // Tạo key để tạo shared memory
    key_t key = ftok("buffer", 65);

    // Tạo segment của shared memory
    int shmid = shmget(key, sizeof(Buffer), IPC_CREAT | 0666);
    if (shmid == -1) {
        perror("shmget");
        return 1;
    }

    // Attach segment vào vùng nhớ
    Buffer *buffer = (Buffer *)shmat(shmid, NULL, 0);
    if (buffer == (void *)-1) {
        perror("shmat");
        return 1;
    }

    // Tạo tiến trình con
    pid_t pid = fork();
    if (pid == -1) {
```

```
        perror("fork");
        return 1;
    }

    if (pid == 0) {
        // Tiến trình con: tính toán chuỗi Collatz
        collatzSequence(startNumber, buffer);
        shmdt(buffer); // Detach shared memory
    } else {
        // Tiến trình cha: đợi tiến trình con kết thúc
        wait(NULL);

        // In kết quả từ buffer ra màn hình
        for (int i = 0; i < buffer->count; ++i) {
            printf("%d ", buffer->data[i]);
        }
        printf("\n");

        // Xóa shared memory
        shmctl(shmid, IPC_RMID, NULL);
    }

    return 0;
```

Trong chương trình này:

- Hàm isValidInput được sử dụng để kiểm tra xem đối số dòng lệnh có phải là một số nguyên dương hợp lệ hay không.
- Hàm collatzSequence tính toán chuỗi Collatz cho một số nguyên dương đầu vào và ghi kết quả vào buffer.
- Tiến trình cha chờ tiến trình con tính toán xong chuỗi Collatz và in kết quả từ buffer ra màn hình.

➔ Kết quả khi chạy:

```
PROBLEMS 7   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMMENTS

anhanh@anhAnh:~/lab3_hdh$ ./collatz 35
35 106 53 160 80 40 20 10 5 16 8 4 2 1
anhanh@anhAnh:~/lab3_hdh$ ./collatz 8
8 4 2 1
anhanh@anhAnh:~/lab3_hdh$ ./collatz 100
100 50 25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
anhanh@anhAnh:~/lab3_hdh$ ./collatz 0
Input phải là số nguyên dương
anhanh@anhAnh:~/lab3_hdh$ ./collatz -1
Input phải là số nguyên dương
anhanh@anhAnh:~/lab3_hdh$
```