BÁO CÁO THỰC HÀNH BÀI 3

Môn học: IT007- Mã lớp: **Q.14**

Giảng viên hướng dẫn thực hành: **Phạm Minh Quân**

Thông tin sinh viên	Mã số sinh viên: 24522016 Họ và tên: Ngô Hồng Vinh
Link các tài liệu tham khảo *(nếu có)*	
Đánh giá của giảng viên: *+ Nhận xét* *+ Các lỗi trong chương trình* *+ Gợi ý*	

Câu 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: test_fork.c

Mã nguồn:

```
/*########################################
# University of Information Technology
# IT007 Operating System
#
# <Your name>, <your Student ID>
# File: test_fork.c
#
########################################*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
int main(int argc, char *argv[])
{
 __pid_t pid;
 pid = fork();
 if (pid > 0)
 {
 printf("PARENTS | PID = %ld | PPID = %ld\n",
(long)getpid(), (long)getppid());
 if (argc > 2)
 printf("PARENTS | There are %d arguments\n",
argc - 1);
 wait(NULL);
 }
 if (pid == 0)
 {
 printf("CHILDREN | PID = %ld | PPID = %ld\n",
(long)getpid(), (long)getppid());
 printf("CHILDREN | List of arguments: \n");
 for (int i = 1; i < argc; i++)
 { printf("%s\n", argv[i]);
 }
 }
 exit(0);
}
```

Giải thích code:

1. Khởi tạo biến pid mang kiểu dữ liệu __pid_t là kiểu số nguyên để lưu trữ process ID (pid)
2. Khởi tạo tiến trình con, lúc này tiến trình cha có pid = tiến trình con, và tiến trình con có pid = 0
3. Dựa trên pid của cha và con sẽ đi vào 2 câu lệnh if với cha là câu lệnh if đầu và con là câu lệnh if sau
4. Thực hiện xuất ra kết quả, nếu có tham số thì in kèm tham số
5. Tiến trình cha dùng wait(NULL) để đợi tiến trình con kết thúc, đảm bảo cha luôn in xong sau khi con hoàn thành.

Bổ sung: PPID là parent process ID, chú ý thấy ppid của tiến trình con chính là pid của tiến trình cha.

Kết quả:
Truyền vào không tham số:

```
PARENTS | PID = 2144 | PPID = 420
CHILDREN | PID = 2145 | PPID = 2144
CHILDREN | List of arguments:
```

Truyền vào có tham số:

```
PARENTS | PID = 2181 | PPID = 420
PARENTS | There are 3 arguments
CHILDREN | PID = 2182 | PPID = 2181
CHILDREN | List of arguments:
ThamSo1
ThamSo2
ThamSo3
```

Ví dụ 3-2: test_exec1.c và count.sh
Mã nguồn:

```
/*########################################
# University of Information Technology
# IT007 Operating System
#
# <Your name>, <your Student ID>
# 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);
}
```

```
#!/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
```

Giải thích code:

1. Đoạn khai báo biến và fork giống, sau đó, thực thi tiến trình cha giống với ví dụ 3-1.
2. Đến với tiến trình con, chương trình thực hiện execl, ghi đè tiến trình lên tiến trình con, thực hiện count.sh, nhận tham số là 10. Nhận thấy tiến trình con không còn thông qua việc không thực hiện câu lệnh in Children
3. Thực hiện chương trình count.sh.
4. Trong chương trình count.sh thực hiện ghi ra file count.txt ghi số từ 1 đến 10.
5. Chú ý: Lệnh ps -ef | grep count.sh sẽ in mọi tiến trình đang chạy có chứa "count.sh" trong tên lệnh. Ở output ta sẽ thấy có:
    a. ngohong+ 2483 2482 0 08:33 pts/0 00:00:00 /bin/bash ./count.sh 10
    b. ngohong+ 2485 2483 0 08:33 pts/0 00:00:00 grep count.sh

   Tức là nó đã in ra: Chính tiến trình shell ./count.sh đang chạy và tiến trình grep count.sh

Kết quả:

Truyền vào không tham số:

```
PARENTS | PID = 2452 | PPID = 420
Implementing: ./count.sh
PPID of count.sh:
ngohong+    2453    2452  0 08:31 pts/0    00:00:00 /bin/bash ./count.sh 10
ngohong+    2455    2453  0 08:31 pts/0    00:00:00 grep count.sh
```

Truyền vào có tham số:

```
PARENTS | PID = 2482 | PPID = 420
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
ngohong+    2483    2482  0 08:33 pts/0    00:00:00 /bin/bash ./count.sh 10
ngohong+    2485    2483  0 08:33 pts/0    00:00:00 grep count.sh
```

Count.txt chứa kết quả của 2 lần truyền:

count.txt

```
1    1
2    2
3    3
4    4
5    5
6    6
7    7
8    8
9    9
10   10
11   1
12   2
13   3
14   4
15   5
16   6
17   7
18   8
19   9
20   10
```

Ví dụ 3-3: test_system.c và count.sh
Mã nguồn:

```
/*########################################
# University of Information Technology
# 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);
}
```

Giải thích code:

1. Khi bắt đầu, thực hiện in ra PID và PPID của tiến trình gốc. In ra số lượng tham số nếu số tham số lớn hơn 2.
2. Tạo mới 1 tiến trình, ta sẽ nhận thấy nó ở phần output có thêm phần:
    a. ngohong+ 2857 2856 0 08:33 pts/0 00:00:00 sh -c ./count.sh 10
3. Thực hiện tiếp chương trình count.sh. Sau đó in ra lần lượt các tham số của tiến trình gốc.

Test case:

Truyền vào không tham số:

```
PARENTS | PID = 2856 | PPID = 420
Implementing: ./count.sh
PPID of count.sh:
ngohong+    2857    2856  0 09:06 pts/0    00:00:00 sh -c ./count.sh 10
ngohong+    2858    2857  0 09:06 pts/0    00:00:00 /bin/bash ./count.sh 10
ngohong+    2860    2858  0 09:06 pts/0    00:00:00 grep count.sh
PARENTS | List of arguments:
```

Truyền vào có tham số:

```
PARENTS | PID = 2898 | PPID = 420
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
ngohong+    2899    2898  0 09:07 pts/0    00:00:00 sh -c ./count.sh 10
ngohong+    2900    2899  0 09:07 pts/0    00:00:00 /bin/bash ./count.sh 10
ngohong+    2902    2900  0 09:07 pts/0    00:00:00 grep count.sh
PARENTS | List of arguments:
ThamSo1
ThamSo2
ThamSo3
```

Count.txt chứa kết quả của 2 lần truyền:

count.txt

```
1   1
2   2
3   3
4   4
5   5
6   6
7   7
8   8
9   9
10  10
11  1
12  2
13  3
14  4
15  5
16  6
17  7
18  8
19  9
20  10
```

Ví dụ 3-4: test_shm_A.sh và test_shm_B.sh

Mã nguồn:

```
/*#######################################
# University of Information Technology
# IT007 Operating System
#
# <Your name>, <your Student ID>
# 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");
 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;
}
```

Process B:

```
/*#######################################
```

```
# University of Information Technology
# IT007 Operating System
#
# <Your name>, <your Student ID>
# File: test_shm_B.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_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;
}
```

Giải thích code:

Chương trình A:

1. Khai báo các biến cần thiết:
   SIZE = 4096 là kích thước vùng nhớ dùng chung.
   name = "OS" là tên vùng nhớ.
   fd là file descriptor, ptr là con trỏ trỏ đến vùng nhớ chia sẻ.
2. Tạo vùng nhớ chia sẻ bằng shm_open(name, O_CREAT | O_RDWR, 0666).
3. Thiết lập kích thước vùng nhớ với ftruncate(fd, SIZE).

4. Ánh xạ vùng nhớ chia sẻ vào không gian tiến trình bằng mmap(). Sau đó, con trỏ ptr có thể đọc/ghi trực tiếp lên vùng nhớ.
5. Ghi chuỗi "Hello Process B" vào vùng nhớ bằng strcpy(ptr, ...).
6. Dùng vòng lặp while() để kiểm tra xem nội dung vùng nhớ có thay đổi không.
   Nếu nội dung vẫn là "Hello Process B", tiến trình A tiếp tục chờ bằng cách in ra Waiting Process B update shared memory
   Khi Process B ghi đè dữ liệu khác, vòng lặp kết thúc.
7. In ra nội dung mới trong vùng nhớ sau khi Process B cập nhật.
8. Giải phóng tài nguyên bằng munmap(ptr, SIZE) và close(fd).

Chương trình B:

1. Khai báo các biến tương tự Process A để truy cập cùng vùng nhớ "OS".
2. Mở lại vùng nhớ đã được tạo bởi Process A bằng shm_open(name, O_RDWR, 0666).
3. Ánh xạ vùng nhớ vào không gian tiến trình bằng mmap().
4. Đọc và in nội dung hiện tại trong vùng nhớ — chính là dữ liệu Process A đã ghi.
5. Ghi đè nội dung mới "Hello Process A" vào cùng vị trí bộ nhớ bằng strcpy().
6. In lại nội dung sau khi đã ghi đè để xác nhận quá trình cập nhật.
7. Dừng 5 giây bằng sleep(5) để quan sát kết quả.
8. Giải phóng tài nguyên:
   munmap() hủy ánh xạ vùng nhớ.
   close() đóng file descriptor.
   shm_unlink(name) xóa vùng nhớ chia sẻ khỏi hệ thống.

Test case:

Thực hiện A:

```
ngohongvinh-24522016@Dmig:/mnt/d/IT007$ ./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
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
```

Thực hiện B:

```
Read shared memory: Hello Process B
Shared memory updated: Hello Process A
```

Chú ý thấy khi thực thi B xong thì A không còn waiting nữa.

Câu 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.

Mã nguồn: time.c

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

```
#include <sys/time.h>
#include <string.h>

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <command> [args...]\n", argv[0]);
        return 1;
    }

    struct timeval start, end;
    gettimeofday(&start, NULL);

    pid_t pid = fork();
    if (pid < 0) {
        perror("fork");
        return 1;
    }

    if (pid == 0) {

        execvp(argv[1], &argv[1]);
        perror("execvp");
        exit(1);
    } else {
        int status;
        waitpid(pid, &status, 0);
        gettimeofday(&end, NULL);

        long seconds = end.tv_sec - start.tv_sec;
        long useconds = end.tv_usec - start.tv_usec;
        if (useconds < 0) {
            seconds--;
            useconds += 1000000;
        }

        printf("Elapsed time: %ld.%06ld seconds\n", seconds, useconds);
    }

    return 0;
}
```

Ý tưởng: Đo thời gian bắt đầu trước khi fork và thời gian kết thúc sau khi tiến trình con kết thúc. Từ đó, tính được thời gian thực thi. Ở đây, em đã dùng execvp, tương đối giống với câu lệnh parser.add_argument('command', nargs='+', help='Command and args to run') có thêm tham số nargs trong python là cho phép truyền vào nhiều tham số.

Test case:

```
ngohongvinh-24522016@Dmig:/mnt/d/IT007$ ./time ls
count.sh      test_fork     test_shm_B     time
count.txt     test_fork.c   test_shm_B.c   time.c
test_exec1    test_shm_A    test_system
test_exec1.c  test_shm_A.c  test_system.c
Elapsed time: 0.003690 seconds
```

Lưu ý: Chương trình đã thực hiện lệnh ls sau đó in ra thời gian thực thi.

Câu 3. Viết một chương trình làm bốn công việc sau theo thứ tự

Mã nguồn: stop.c

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

pid_t child_pid = -1;

void handle_sigint(int sig) {
    if (child_pid > 0) {
        kill(child_pid, SIGINT);
        printf("\ncount.sh has stopped\n");
    }
}

int main() {
    printf("Welcome to IT007, I am 24522016!\n");

    signal(SIGINT, handle_sigint);

    child_pid = fork();
    if (child_pid < 0) {
        perror("fork");
        return 1;
    }

    if (child_pid == 0) {

        execl("./count.sh", "count.sh", "120", NULL);
        perror("execl");
        exit(1);
    } else {

        int status;
        waitpid(child_pid, &status, 0);
    }

    return 0;
}
```

Ý tưởng:
Xử lý việc dừng bằng tín hiệu Ctrl+C (SIGINT). Ở đây, em đã thiết lập 1 hàm handle_sigint để khi nhận tín hiệu thì chỉ kill mỗi tiến trình con tức là count.sh.

Test case:

```
Welcome to IT007, I am 24522016!
Implementing: ./count.sh
PPID of count.sh:
ngohong+    1456    1455  0 10:33 pts/2    00:00:00 /bin/bash ./count.sh 120
ngohong+    1458    1456  0 10:33 pts/2    00:00:00 grep count.sh
^C
count.sh has stopped
```

Lưu ý: Ở đây ^C là dấu hiệu của ctrl + C.

Câu 4. Viết chương trình mô phỏng bài toán Producer - Consumer

Mã nguồn: producer_consumer.c

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

#define SIZE 10
#define SHM_NAME "/buffer_shm"

typedef unsigned char int1byte;

typedef struct {
    int1byte buffer[SIZE];
    int count;
    int total;
} shared_data;

int main() {
    srand(time(NULL));

    int fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (fd == -1) {
        perror("shm_open");
        return 1;
    }

    if (ftruncate(fd, sizeof(shared_data)) == -1) {
        perror("ftruncate");
        return 1;
    }
```

```
    shared_data *shm = mmap(NULL, sizeof(shared_data),
                            PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (shm == MAP_FAILED) {
        perror("mmap");
        return 1;
    }

    shm->count = 0;
    shm->total = 0;

    pid_t pid = fork();

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

    if (pid == 0) {
        // ========================
        // Consumer
        // ========================
        while (1) {
            if (shm->count > 0) {
                int val = shm->buffer[0];

                // Dịch mảng sau khi đọc
                for (int i = 1; i < shm->count; i++)
                    shm->buffer[i - 1] = shm->buffer[i];

                shm->count--;
                shm->total += val;

                printf("Consumer read: %d, total=%d\n", val, shm->total);

                if (shm->total > 100)
                    break;
            } else {
                usleep(100000);
            }
        }

        munmap(shm, sizeof(shared_data));
        close(fd);
        shm_unlink(SHM_NAME);
        exit(0);
    } else {
        // ========================
        // Producer
        // ========================
        while (1) {
            if (shm->count < SIZE) {
                int num = rand() % 11 + 10;  // [10,20]
```

```
            shm->buffer[shm->count++] = (int1byte)num;
            printf("Producer wrote: %d\n", num);
            usleep(100000);
        }

        if (shm->total > 100)
            break;
    }

    wait(NULL);
    munmap(shm, sizeof(shared_data));
    close(fd);
  }

  return 0;
}
```

Ý tưởng:

- Tạo một vùng nhớ dùng chung (shared memory) tên /buffer_shm có kích thước bằng cấu trúc shared_data. Vùng nhớ này chứa một buffer 10 bytes (Định nghĩa kiểu dữ liệu số nguyên 1 byte), biến count lưu số phần tử hiện có và total lưu tổng các giá trị tiêu thụ được.
- **Producer (tiến trình cha)**
  Sinh ngẫu nhiên các số trong khoảng 10–20, ghi lần lượt vào buffer nếu chưa đầy. Mỗi số được ghi chiếm đúng 1 byte.
- **Consumer (tiến trình con)**
  Đọc phần tử đầu tiên trong buffer, dịch mảng sang trái, cộng giá trị vào total và in ra màn hình.
- Khi vượt quá 100, cả Producer và Consumer dừng hoạt động.
- Giải phóng tài nguyên
  Sau khi kết thúc, chương trình unmap và xóa vùng nhớ chia sẻ bằng shm_unlink()

Test case:

```
Producer wrote: 10
Consumer read: 10, total=10
Producer wrote: 17
Producer wrote: 18
Consumer read: 17, total=27
Consumer read: 18, total=45
Producer wrote: 14
Consumer read: 14, total=59
Producer wrote: 16
Consumer read: 16, total=75
Producer wrote: 11
Consumer read: 11, total=86
Producer wrote: 11
Consumer read: 11, total=97
Producer wrote: 16
Consumer read: 16, total=113
```