

Name: Nguyễn Huỳnh Nhân

ID: 23521080

Class:IT007.P28.2

OPERATING SYSTEM LAB X'S REPORT

SUMMARY

Task		Status	Page
Section 3.5	Ex 1	Hoàn thành	2
	Ex 2	Hoàn thành	8
	Ex 3	Hoàn thành	10
	EX 4	Hoàn thành	11
...	...		
	...		

Self-scores: 10/10

Section 3.5

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

+ Giải thích

```
test_fork.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     __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 }
31
32
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(kali@nhan23521080) - [~/HDH]
$ ./test_fork 1 2 3
PARENTS | PID = 83773 | PPID = 81086
PARENTS | There are 3 arguments
CHILDREN | PID = 83774 | PPID = 83773
CHILDREN | List of arguments:
1
2
3
```

+ Giải thích:

- Hàm `fork()` tạo ra bản sao của tiến trình gọi nó. Trong trường hợp trên nó tạo ra một tiến trình con y hệt cha

- Nếu $pid > 0$ bắt đầu xử lý tiến trình cha, chương trình sẽ in ra PID và PPID của tiến trình cha. Nếu số lượng tham số nhập vào lớn hơn 2 thì sẽ in ra số lượng tham số. Sau đó tiến trình cha chờ tiến trình con kết thúc bằng lệnh `wait(NULL)`;
- Nếu $pid=0$ bắt đầu xử lý tiến trình con, chương trình sẽ in ra pid và ppid của tiến trình con, sau đó in ra các tham số đã truyền vào

Ví dụ 3-2

```
C test_execl.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     __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        printf("CHILDREN | PID = %ld | PPID = %ld\n",
23              (long)getpid(), (long)getppid());
24        printf("CHILDREN | List of arguments: \n");
25        for (int i = 1; i < argc; i++)
26        {
27            printf("%s\n", argv[i]);
28        }
29    }
30    exit(0);
31 }
32
33
```

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS
kali	86051	86049	0 03:32 pts/2	00:00:00 grep count.sh

```

(kali@nhan23521080) - [~/HDH]
$ ./test_execl 1 2 3
PARENTS | PID = 86301 | PPID = 81086
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
kali      86302  86301  0 03:32 pts/2    00:00:00 /bin/bash ./count.sh 10
kali      86304  86302  0 03:32 pts/2    00:00:00 grep count.sh

```

```

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

```

Giải thích:

- Sử dụng hàm `fork()` để tạo tiến trình con.
- Xử lý trong tiến trình cha: Nếu `pid` lớn hơn 0, tức là đang ở trong tiến trình cha, in ra thông tin về `PID` và `PPID` của tiến trình cha. Nếu có hơn 2 tham số dòng lệnh, in ra số lượng tham số.
- Xử lý trong tiến trình con: Nếu `pid` bằng 0, tức là đang ở trong tiến trình con, sử dụng hàm `execl()` để thực thi tập lệnh `count.sh` với tham số là "10" và ghi đè lên tiến trình con đang chạy bên dưới. In ra thông tin về `PID` và `PPID` của tiến trình con. In ra các tham số dòng lệnh truyền vào.
- Hàm `coun.sh` dùng để in ra từ 1->\$1 sau mỗi 1s

```
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
10 10
11
```

Ví dụ 3-3

```
C 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
12         - 1);
13
14     system("./count.sh 10");printf("PARENTS | List of arguments: \n");
15     for (int i = 1; i < argc; i++)
16     {
17         printf("%s\n", argv[i]);
18     }
19     exit(0);
20 }
21
22
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(kali@nhan23521080)~[/HDH]
$ ./test_system 1 2 3
PARENTS | PID = 88064 | PPID = 81086
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
kali      88065    88064    0 03:35 pts/2    00:00:00 sh -c -- ./count.sh 10
kali      88066    88065    0 03:35 pts/2    00:00:00 /bin/bash ./count.sh 10
kali      88068    88066    0 03:35 pts/2    00:00:00 grep count.sh
PARENTS | List of arguments:
1
2
3
```

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

Giải thích

- Xử lý trong tiến trình cha: In ra thông tin về PID và PPID của tiến trình cha. Nếu có hơn 2 tham số dòng lệnh, in ra số lượng tham số.
- Thực thi lệnh với hàm system(): Sử dụng hàm system() để thực thi lệnh ./count.sh 10. Điều này sẽ gọi script count.sh với tham số là 10 và sẽ tạo mới hoàn toàn tiến trình

Ví dụ 3-4

```
C test_shm_b.c
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_RDWR, 0666);
21     /* memory map the shared memory object */
22     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
23     /* read from the shared memory object */
24     printf("Read shared memory: ");
25     printf("%s\n", (char *)ptr);
26     /* update the shared memory object */
27     strcpy(ptr, "Hello Process A");
28     printf("Shared memory updated: %s\n", ptr);
29     sleep(5);
30     // unmap the shared memory segment and close the
31     munmap(ptr, SIZE);
32     close(fd);
33     // remove the shared memory segment
34     shm_unlink(name);
35     return 0;
36 }
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
li@nh ~$ ./test_shm_b
an23521080) Read shared memory: Hello Process B
[~/HDH] Shared memory updated: Hello Process A
(kali@nhan23521080) - [~/HDH]
li@nh ~$ ./test_shm_b
an23521080) Read shared memory: Hello Process B
[~/HDH] Shared memory updated: Hello Process A
(kali@nhan23521080) - [~/HDH]
li$
```

```
C test_shm_a.c
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 = "0S";
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 */
27     strcpy(ptr, "Hello Process B");
28     /* wait until Process B updates the shared memory
29     segment */
30     while (strcmp(ptr, "Hello Process B", 15) == 0)
31     {
32         printf("Waiting Process B update shared memory\n");
33         sleep(1);
34     }
35     printf("Memory updated: %s\n", (char *)ptr);
36     /* unmap the shared memory segment and close the
37     file descriptor */
38     munmap(ptr, SIZE);
39     close(fd);
40     return 0;
41 }
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

Memory updated: Hello Process A

```
(kali@nhan23521080) - [~/HDH]
$ ./test_shm_a
Waiting Process B update shared memory
Waiting Process B update shared memory
Waiting Process B update shared memory
Memory updated: Hello Process A
$
```

Giải thích:

- Process A: tạo một đối tượng shared memory, ghi dữ liệu vào và sau đó chờ tiến trình B cập nhật dữ liệu vào đối tượng shared memory, khi B cập nhật xong A sẽ đọc dữ liệu và in dữ liệu ra
- Tạo một đối tượng mới bằng shm_open().
- Cấu hình kích thước bằng ftruncate().
- Ánh xạ đối tượng shared memory vào không gian địa chỉ của quy trình bằng mmap().
- Ghi dữ liệu vào share memory bằng cách sao chép chuỗi "Hello Process B" vào con trỏ ptr

- Chờ đợi B cập nhật dữ liệu trong vòng lặp While
- Khi dữ liệu đã được cập nhật, đọc và in ra dữ liệu đã được cập nhật.
- Hủy ánh xạ và đóng file descriptor bằng munmap() và close().

Process B

- Process B mở đối tượng shared memory được tạo bởi A đọc dữ liệu và thay đổi đối tượng shared memory sau đó chờ một thời gian để kết thúc
- Mở đối tượng shared memory bằng hàm shm_open() với quyền đọc và ghi.
- Ánh xạ đối tượng shared memory vào không gian địa chỉ của quy trình bằng mmap().
- Đọc dữ liệu từ đối tượng shared memory và in ra màn hình.
- Cập nhật dữ liệu trong đối tượng shared memory bằng cách sao chép chuỗi "Hello Process A" vào con trỏ ptr.
- Chờ một khoảng thời gian bằng hàm sleep().
- Hủy ánh xạ và đóng file bằng munmap() và close().
- Xóa đối tượng shared memory bằng hàm shm_unlink().

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.


```

C time.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5  #include <sys/time.h>
6
7  int main(int argc, char *argv[]) {
8      if (argc < 2) {
9          printf("Usage: %s <command>\n", argv[0]);
10         return 1;
11     }
12
13     struct timeval start, end;
14     pid_t pid;
15
16     gettimeofday(&start, NULL); // Lấy thời gian bắt đầu
17
18     pid = fork();
19     if (pid < 0) {
20         perror("fork failed");
21         return 1;
22     }
23
24     if (pid == 0) {
25         // Tiến trình con: thực thi lệnh shell
26         execlp(argv[1], argv[1], NULL);
27         // Nếu execlp thất bại
28         perror("execlp failed");
29         exit(1);
30     } else {
31         // Tiến trình cha: đợi con kết thúc
32         wait(NULL);
33         gettimeofday(&end, NULL); // Lấy thời gian kết thúc
34     }
35 }

```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```

(kali@nhan23521080) - [~/HDH]
$ ./time ls
3-1.png 3-3.png count.sh count-txt.png hello.c main.c test_execl test_fork test_system time
3-2.png count.png count.txt hello hello.h Makefile test_execl.c test_fork.c test_system.c time.c
Thời gian thực thi: 0.00511 giây

```

```

double elapsed = (end.tv_sec - start.tv_sec) +
(end.tv_usec - start.tv_usec) / 1000000.0;
printf("Thời gian thực thi: %.5f giây\n", elapsed);

```

Giải thích

- Nếu tham số nhận vào < 2 thì sẽ cảnh báo và thoát
- Gọi struct start và end có kiểu dữ liệu timeval để lấy thời gian bắt đầu và kết thúc, gettimeofday(&start,null) để lấy thời gian ban đầu
- Tạo tiến trình con bằng fork(), fork<0 tạo tiến trình con thất bại và báo lỗi, fork()==0 tiến trình con đang chạy, sử dụng execlp để lấy tham số truyền vào
- Fork>0 tiến trình cha đang chạy tiến trình cha chờ tiến trình con chạy xong rồi mới gọi hàm gettimeofday(&end,null) để tính thời gian kết thúc
- Thời gian chạy bằng thời gian kết thúc - thời gian ban đầu

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”s

```
C bai3.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <signal.h>
5 #include <sys/wait.h>
6
7 pid_t pid;
8 void handler(int sig)
9 {
10     if (pid > 0) {
11         kill(pid, SIGKILL);
12         printf("count.sh has stopped\n");
13     }
14     exit(0);
15 }
16
17 int main() {
18     printf("Welcome to IT007, I am 23521080\n");
19     signal(SIGINT, handler);
20     pid=fork();
21     if (pid==0){
22         execl("./count.sh", "./count.sh", "120", NULL);
23         exit(1);
24     }
25     else{
26         wait(NULL);
27     }
28     return 0;
29 }
```

```
(kali@nhan23521080) ~[HDH]
$ ./bai3
Welcome to IT007, I am 23521080
Implementing: ./count.sh
PPID of count.sh:
kali 54816 0 08:23 pts/2 00:00:00 /bin/bash ./count.sh 120
kali 54819 54817 0 08:23 pts/2 00:00:00 grep count.sh
^Ccount.sh has stopped
(kali@nhan23521080) ~[HDH]
$
```

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

Giải thích

- Tạo hàm handler để bắt sự kiện khi nhấn ctrl+C: kết thúc tiến trình và thông báo ra màn hình
- Signal(SIGINT, handler): hàm gọi sự kiện khi nhấn ctrl+C thì sẽ gọi đến hàm handler

- In ra dòng chữ chào mừng với Student ID.
- Thực thi file script count.sh với số lần đếm là 120 bằng cách sử dụng hàm execl().
- Nếu không có lỗi xảy ra trong quá trình thực thi file script, tiến trình con sẽ kết thúc và tiến trình cha sẽ đợi cho đến khi tiến trình con kết thúc.
- Nếu có lỗi xảy ra trong quá trình thực thi file script, tiến trình con sẽ in ra thông báo lỗi và thoát với mã lỗi 1.
- Khi tiến trình cha nhận được tín hiệu SIGINT (CTRL+C), nó sẽ kích hoạt handler sigint_handler(), in ra thông báo "count.sh has stopped" và thoát chương trình.

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

Process Produce.c

- Share.h dùng để tạo các biến dùng chung cho cả 2 tiến trình producer và consumer
- Tiến trình produce khởi tạo vùng nhớ dùng chung PC_SHARED
- Share->in : vị trí hiện tại của producer
- Share-out vị trí hiện tại của consumer
- Share-done: biến cho biết tiến trình con đã hoàn thành
- Share->sum: biến tính tổng
- Srand(time(NULL)): dùng để tạo các số ngẫu nhiên không trùng lặp
- Kiểm tra nếu tiến trình con chưa hoàn thành thì thực hiện while
- Gán biến val có giá trị từ 10-20
- Next->in: vị trí tiếp theo của producer
- Kiểm tra nếu vị trí tiếp theo và vị trí hiện tại của consumer khác nhau (còn chỗ) thì thêm giá trị val vào buffer hiện tại sau đó tăng vị trí hiện tại của producer lên 1 ngược lại thì chờ

Process Consumer.c

- Khởi tạo biến dùng chung PC_SHARED
- Cộng dồn giá trị của sum cho đến khi >100 thì gán share->done=1(kết thúc tiến trình)
- Giải phóng và hủy bỏ tiến trình
- Dọn dẹp bộ nhớ dùng chung

...