TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

MÔN HỆ ĐIỀU HÀNH

BÁO CÁO THỰC HÀNH LAB 3

Lớp IT007.P11

23520552 – PHAN CẢNH ĐĂNG HUÂN

NỘI DUNG BÁO CÁO

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?

a) Ví dụ 3-1

vd3_1.c

lab3 > C vd3_1.c > main(int, char * [])

```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);
}
```

```
┌──(kali㉿kali)-[~/Desktop/HDH/lab3]
└─$ ./vd3_1 1 2 3
PARENTS | PID = 4402 | PPID = 4068
PARENTS | There are 3 arguments
CHILDREN | PID = 4403 | PPID = 4402
CHILDREN | List of arguments:
1
2
3
```

Giải thích ví dụ 3-1 : Chương trình này tạo ra một tiến trình con từ tiến trình cha. Tiến trình cha sẽ in thông tin về chính nó và nếu có nhiều hơn hai tham số đầu vào, in ra số lượng tham số. Tiến trình con sẽ in ra thông tin của nó
cùng danh sách các tham số đầu vào được truyền vào khi chạy chương trình.

b) Ví dụ 3-2

```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);
}
```

```
lab3 > $ 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
```

```
┌──(kali㉿kali)-[~/Desktop/HDH/lab3]
└─$ ./vd3_2 1 2 3
PARENTS | PID = 8669 | PPID = 4068
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
kali        8670    8669  0 01:48 pts/0    00:00:00 /bin/bash ./count.sh 10
kali        8672    8670  0 01:48 pts/0    00:00:00 grep count.sh
```

Giải thích ví dụ 3-2 :
Tiến trình cha: In ra thông tin và đợi tiến trình con hoàn thành.
Tiến trình con: Chạy tập lệnh count.sh, thực hiện đếm từ 1 đến 10, ghi vào count.txt, và dừng lại 1 giây
mỗi lần đếm.

c) Ví dụ 3-3 :

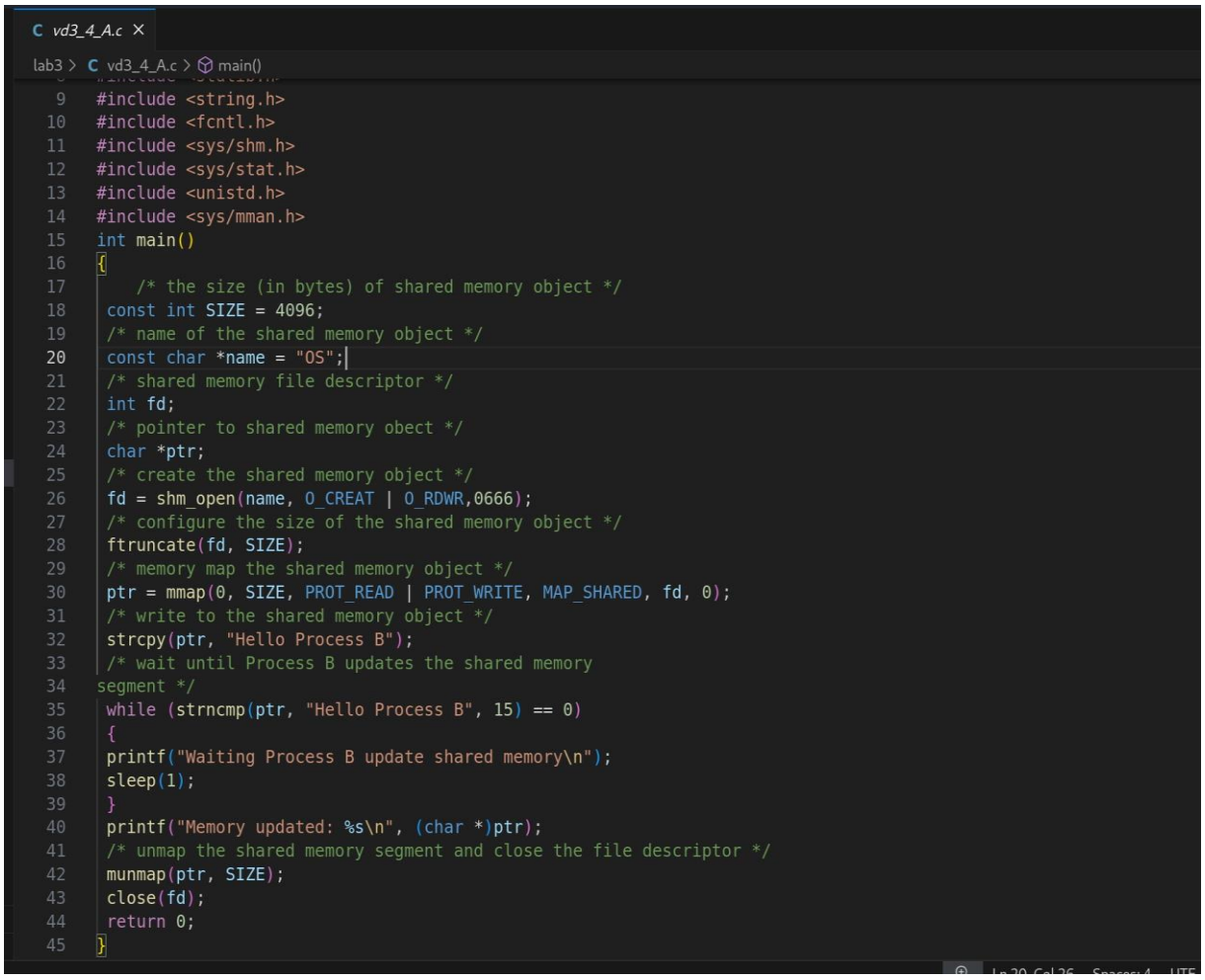
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
int main(int argc, char* argv[])
{
 printf("PARENTS | PID = %ld | PPID = %ld\n",
(long)getpid(), (long)getppid());
 if (argc > 2)
 printf("PARENTS | There are %d arguments\n", argc
- 1);

 system("./count.sh 10");
 printf("PARENTS | List of arguments: \n");
 for (int i = 1; i < argc; i++)
 {
 printf("%s\n", argv[i]);
 }
 exit(0);
}
```

```
┌──(kali㉿kali)-[~/Desktop/HDH/lab3]
└─$ ./vd3_3 1 2 3
PARENTS | PID = 10540 | PPID = 4068
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
kali       10541   10540  0 01:51 pts/0    00:00:00 sh -c -- ./count.sh 10
kali       10542   10541  0 01:51 pts/0    00:00:00 /bin/bash ./count.sh 10
kali       10544   10542  0 01:51 pts/0    00:00:00 grep count.sh
PARENTS | List of arguments:
1
2
3
```

Giải thích ví dụ 3-3 : Chương trình C này in ra thông tin của tiến trình cha, kiểm tra và hiển thị số lượng đối số nếu có.
Chạy tập lệnh count.sh với tham số 10, sau đó in danh sách các đối số.
Cuối cùng, chương trình kết thúc với mã trạng thái 0.

d) Ví dụ 3-4 :


vd3_4_A.c
lab3 > vd3_4_A.c > main()
#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;
}

vd3_4_B.c

lab3 > vd3_4_B.c > main()

```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 thefile descriptor
 munmap(ptr, SIZE);
 close(fd);
 // remove the shared memory segment
 shm_unlink(name);
 return 0;
}
```

Ln 29, Col 42

```
┌──(kali㉿kali)-[~/Desktop/HDH/lab3]
└─$ ./vd3_4_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
^C

┌──(kali㉿kali)-[~/Desktop/HDH/lab3]
└─$ ./vd3_4_B
Read shared memory: Hello Process B
Shared memory updated: Hello Process A
```

Giải thích ví dụ 3-4 :
Process A khởi tạo bộ nhớ chia sẻ, ghi vào bộ nhớ "Hello Process B", sau đó chờ bộ nhớ được cập nhật bởi Process B.
Proces B truy cập bộ nhớ chia sẻ, đọc dữ liệu do Process A ghi vào, sau đó cập nhật bộ nhớ với chuỗi "Hello Process A.

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 " với là lệnh shell muốn đo thời gian thực thi.

lab3 > C time.c > main(int, char * [])

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <sys/time.h>
int main(int argc, char* argv[])
{
if (argc < 2) {
fprintf(stderr, "Usage: %s <command>\n", argv[0]);
exit(1);
}
struct timeval start, end;
pid_t pid;
// Lấy thời gian bắt đầu
gettimeofday(&start, NULL);
pid = fork();
if (pid < 0) {
perror("fork");
exit(1);
}
else if (pid == 0) {
// Tiến trình con thực thi lệnh shell
execl("/bin/sh", "sh", "-c", argv[1], (char *)NULL);
perror("execl");
exit(1);
}
else {
// Tiến trình cha đợi tiến trình con hoàn thành
wait(NULL);
// Lấy thời gian kết thúc
gettimeofday(&end, NULL);
// Tính toán thời gian thực thi
double elapsed_time = (end.tv_sec - start.tv_sec) +
(end.tv_usec - start.tv_usec) / 1000000.0;
printf("Thời gian thực thi: %.5f giây\n", elapsed_time);
}
return 0;
}
```

Kết quả:

```
┌──(kali㉿kali)-[~/Desktop/HDH/lab3]
└─$ ./time "ls"
count.sh  count.txt  time  time.c  vd3_1  vd3_1.c  vd3_2  vd3_2.c  vd3_3  vd3_3.c  vd3_4_A  vd3_4_A.c  vd3_4_B  vd3_4_B.c
Thời gian thực thi: 0.00120 giây
```

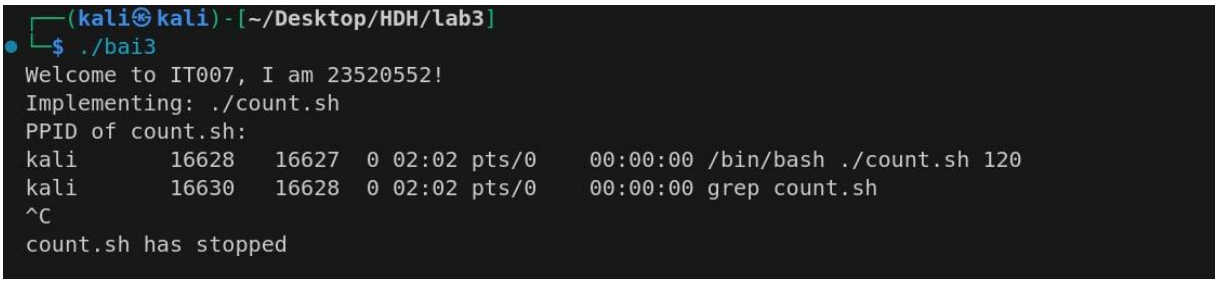
3) Viết một chương trình làm bốn công việc sau theo thứ tự:

- In ra dòng chữ: "Welcome to IT007, I am !"
- 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"

lab3 > C bai3.c > main()

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
pid_t child_pid;
// Hàm xử lý tín hiệu SIGINT (CTRL+C)
void handle_sigint(int sig) {
if (child_pid > 0) {
kill(child_pid, SIGKILL);
printf("\ncount.sh has stopped\n");
}
exit(0);
}
int main() {
printf("Welcome to IT007, I am 23520552!\n");
signal(SIGINT, handle_sigint);
child_pid = fork();
if (child_pid == 0) {
execl("./count.sh", "./count.sh", "120", (char *)NULL);
perror("execl");
exit(1);
}
else if (child_pid > 0) {
wait(NULL);
}
else {
perror("fork");
exit(1);
}
return 0;
}
```

Kết quả:



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

lab3 > C bai4.c > main()

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#define BUFFER_SIZE 10
typedef struct {
int buffer[BUFFER_SIZE];
int in;
int out;
int sum;
} SharedMemory;
int main() {
int shm_id;
SharedMemory *shared_mem;
// Tạo vùng nhớ chia sẻ
shm_id = shmget(IPC_PRIVATE, sizeof(SharedMemory), IPC_CREAT | 0666);
if (shm_id < 0) {
perror("shmget failed");
exit(1);
}
shared_mem = (SharedMemory *)shmat(shm_id, NULL, 0);
if (shared_mem == (SharedMemory *)-1) {
perror("shmat failed");
exit(1);
}
// Khởi tạo buffer và các chỉ số´
shared_mem->in = 0;
shared_mem->out = 0;
shared_mem->sum = 0;
pid_t pid = fork();
if (pid < 0) {
perror("fork failed");
exit(1);
```

```
exit(1);
} else if (pid == 0) { // Tiến trình con - Consumer
while (1) {
// Kiểm tra xem có dữ liệu trong buffer không
if (shared_mem->in != shared_mem->out) {
int item = shared_mem->buffer[shared_mem->out];
shared_mem->out = (shared_mem->out + 1) % BUFFER_SIZE;
// Tính tổng và in ra màn hình
shared_mem->sum += item;
printf("Consumer consumed: %d, Current Sum: %d\n", item, shared_mem->sum);
// Dừng nếu tổng vượt quá 100
if (shared_mem->sum > 100) {
break;
}
}
usleep(100000); // Đợi để tránh busy-waiting
}
shmdt(shared_mem);
exit(0);
} else { // Tiến trình cha - Producer
srand(time(NULL));
while (1) {
// Tạo số ngẫu nhiên từ 10 đến 20
int item = rand() % 11 + 10;
// Kiểm tra nếu buffer không đầy
if ((shared_mem->in + 1) % BUFFER_SIZE != shared_mem->out) {
shared_mem->buffer[shared_mem->in] = item;
shared_mem->in = (shared_mem->in + 1) % BUFFER_SIZE;
printf("Producer produced: %d\n", item);
}
// Kiểm tra nếu tổng đã vượt quá 100 để dừng
if (shared_mem->sum > 100) {
break;
}
usleep(200000); // Đợi để tránh busy-waiting
```

Ln 81, Col 2 Spaces: 4

```
71  usleep(200000); // Đợi để tránh busy-waiting
72  }
73  // Đợi tiến trình con hoàn thành
74  wait(NULL);
75  // Hủy vùng nhớ chia sẻ
76  shmdt(shared_mem);
77  shmctl(shm_id, IPC_RMID, NULL);
78  printf("Producer and Consumer have stopped.\n");
79  }
80  return 0;
81  }
```

Kết quả:

```
┌──(kali㉿kali)-[~/Desktop/HDH/lab3]
└─$ ./bai4
Producer produced: 16
Consumer consumed: 16, Current Sum: 16
Producer produced: 17
Consumer consumed: 17, Current Sum: 33
Producer produced: 12
Consumer consumed: 12, Current Sum: 45
Producer produced: 15
Consumer consumed: 15, Current Sum: 60
Producer produced: 15
Consumer consumed: 15, Current Sum: 75
Producer produced: 17
Consumer consumed: 17, Current Sum: 92
Producer produced: 18
Consumer consumed: 18, Current Sum: 110
Producer produced: 14
Producer and Consumer have stopped.
```