ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

HỆ ĐIỀU HÀNH

BÁO CÁO THỰC HÀNH LAB 3

Giảng viên hướng dẫn: Nguyễn Thanh Nam

Lớp: IT007.Q11.2

Sinh viên thực hiện:

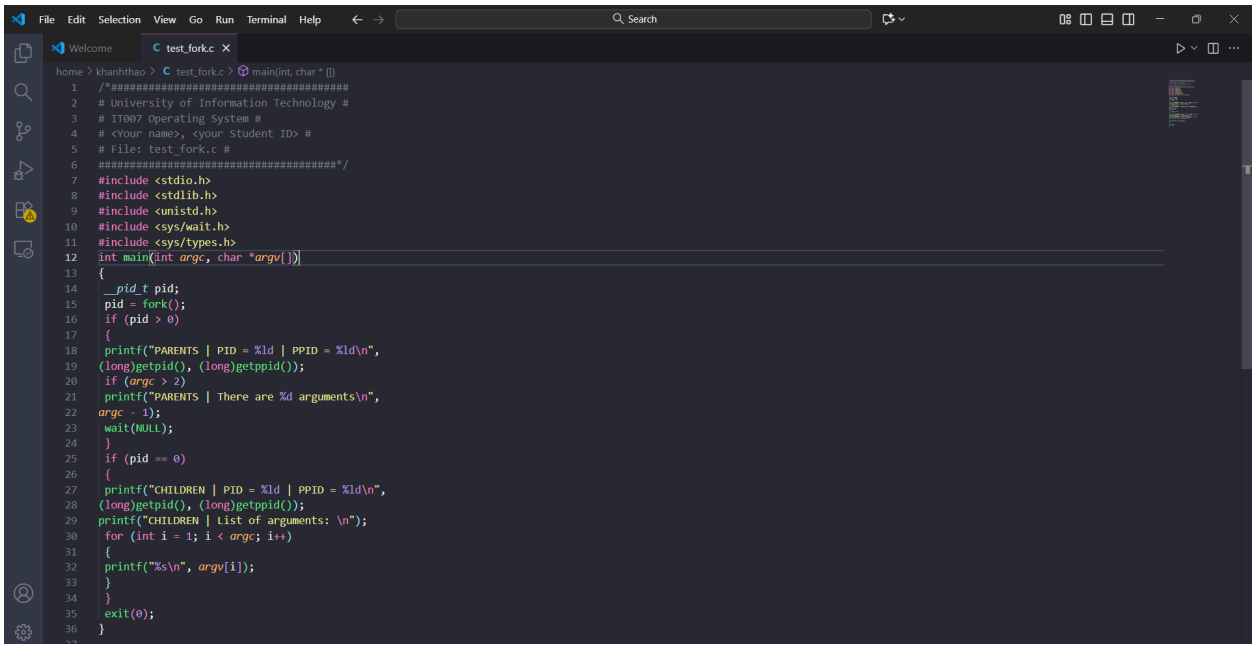
Lê Vũ Khánh Thảo MSSV: 23521469

TP. Hồ Chí Minh – 25/10/2025

I. 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?

a. Ví dụ 3-1:



Hình 1: Tái hiện lại code của Ví dụ 1

- ***Giải thích code:***

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h> // Này là khai báo các thư viện cần thiết.
int main(int argc, char *argv[])
//Hàm main nhận đối số từ dòng lệnh:

+ argc: số lượng đối số
+ argv[]: mảng chứa các đối số

pid_t pid;
pid = fork();
//Tạo tiến trình con bằng fork():

+ pid > 0: tiến trình cha
+ pid == 0: tiến trình con
+ pid < 0: lỗi khi tạo tiến trình

if (pid > 0) {
printf("PARENT : PID = %ld ; PPID = %ld\n", (long)getpid(), (long)getppid()); // In ra PID của tiến trình cha và PID của cha nó (thường là terminal hoặc shell).
if (argc > 2)
printf("PARENT : There are %d arguments\n", argc);
// Nếu có nhiều hơn 2 đối số, in ra số lượng đối số.
wait(NULL);}// Chờ tiến trình con kết thúc trước khi tiến trình cha tiếp tục.
if (pid == 0) {
printf("CHILDREN : PID = %ld ; PPID = %ld\n", (long)getpid(), (long)getppid());
// In ra PID của tiến trình con và PID của cha nó (chính là tiến trình cha).
printf("CHILDREN : list of arguments : \n");
for (int i = 1; i < argc; i++)
printf("%s\n", argv[i]);// In ra danh sách các đối số truyền vào chương trình (bỏ qua argv[]là tên chương trình).
exit(0);}// Kết thúc tiến trình con.

- ***Kết quả nhận được:***

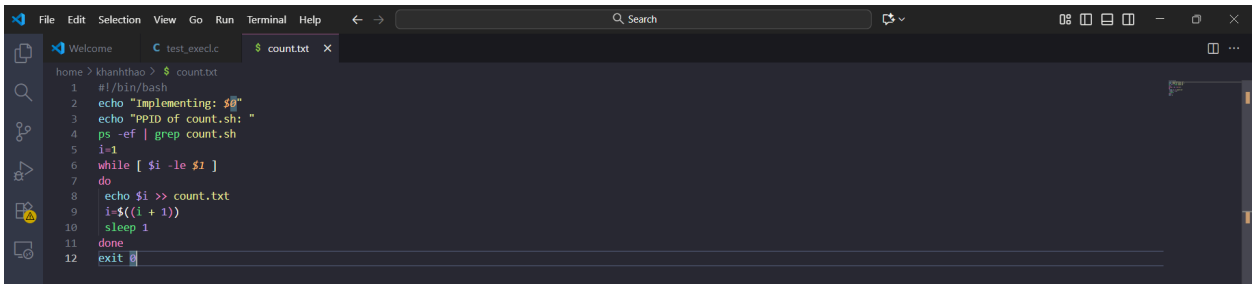
```
khanhthao@khanhthao-VirtualBox:~$ ./test_fork 5 10 15
PARENTS | PID = 9208 | PPID = 8808
CHILDREN | PID = 9209 | PPID = 9208
CHILDREN | List of arguments:
5
10
15
PARENTS | There are 3 arguments
```

Hình 2: Kết quả nhận được của Ví dụ 1

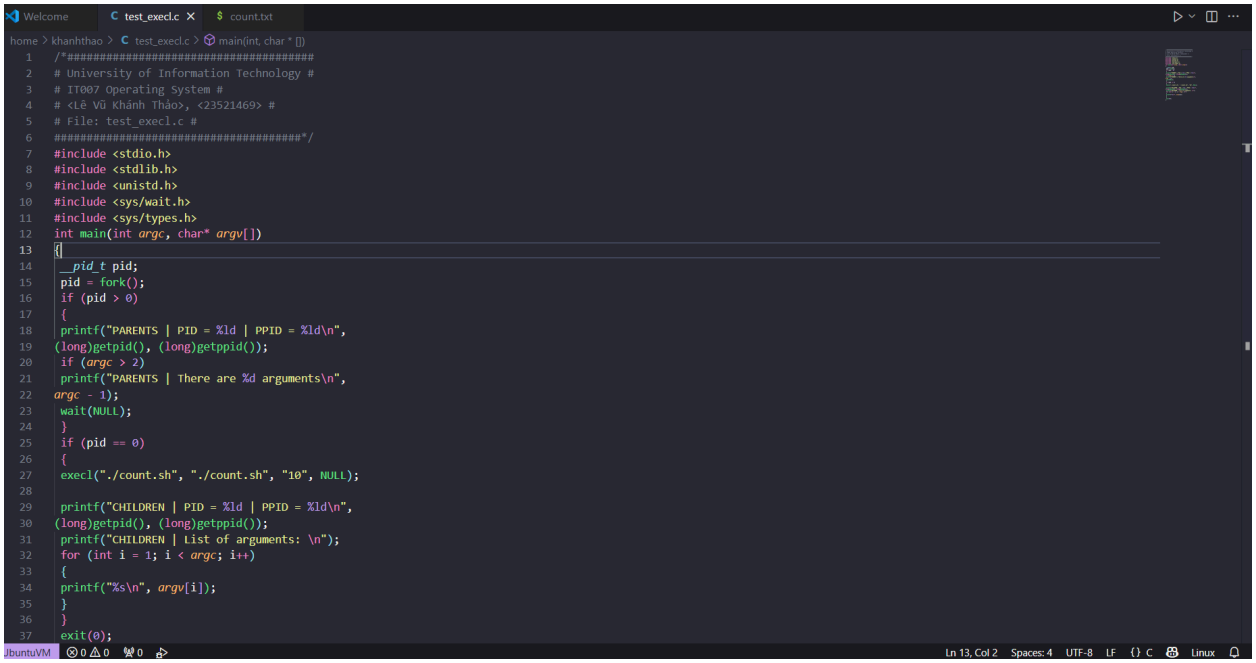
- **PARENTS | PID = 9208 | PPID = 8808**: Dòng này in ra thông tin về tiến trình cha. PID = 9208 là ID của tiến trình cha và PPID = 8808 là ID của tiến trình cha của tiến trình cha.
- **PARENTS | There are 3 arguments**: Dòng này in ra số lượng đối số được truyền vào chương trình từ dòng lệnh. Trong trường hợp này, có 3 đối số là 5, 10, 15.

- **CHILDREN | PID = 9209 | PPID = 9208**: Dòng này in ra thông tin về tiến trình con. PID = 9209 là ID của tiến trình con và PPID = 9208 là ID của tiến trình cha của tiến trình con.
- **CHILDREN | List of arguments: 5 10 15**: Dòng này in ra danh sách các đối số được truyền vào chương trình từ dòng lệnh.

b. Ví dụ 3-2:



Hình 3: File text count.txt của ví dụ 3.2



Hình 4: Chương trình execl() của ví dụ 3.2

- ***Giải thích code của file text count.txt:***

#!/bin/bash // Khai báo đây là một script Bash

echo "Implementing: $0" // In ra tên file đang chạy ($0 là tên script)

echo "PPID of count.sh:"

ps -ef | grep count.sh //In ra thông tin liên quan đến count.sh , bao gồm PID và PPID

i=1 // Khởi tạo biến đếm i bắt đầu từ 1

while [$i -le $1] // Vòng lặp chạy từ 1 đến $1 (đối số truyền vào từ dòng lệnh).
do
  echo $i >> count.txt //Ghi số i vào file count.txt (thêm vào cuối file)
i=$(($i + 1)) // Tăng i lên 1
sleep 1 // Dừng 1 giây giữa mỗi lần ghi
done
exit 0 // Kết thúc vòng lặp và thoát chương trình với mã 0 (thành công).

- ***Giải thích code của chương trình dùng execl***

```
#include <stdio.h>       // Thư viện cho hàm printf()
#include <stdlib.h>      // Thư viện cho hàm exit()
#include <unistd.h>      // Thư viện cho fork(), execl(), getpid(), getppid()
#include <sys/wait.h>    // Thư viện cho hàm wait()
#include <sys/types.h>   // Thư viện định nghĩa kiểu pid_t
int main(int argc, char* argv[])  // Hàm main nhận đối số từ dòng lệnh
{
   __pid_t pid;              // Khai báo biến pid kiểu __pid_t (tương đương pid_t)
  pid = fork();              // Tạo tiến trình con. fork() trả về:
                        // > 0: tiến trình cha
                        // = 0: tiến trình con
                        // < 0: lỗi
   if (pid > 0)              // Nếu là tiến trình cha
   {
      printf("PARENTS | PID = %ld | PPID = %ld\n",
           (long)getpid(), (long)getppid());  // In PID của cha và PID của cha nó
(thường là terminal)
      if (argc > 2)          // Nếu có nhiều hơn 1 đối số (ngoài tên chương trình)
         printf("PARENTS | There are %d arguments\n",
              argc - 1);    // In số lượng đối số thực sự (trừ argv[0])
      wait(NULL);            // Tiến trình cha chờ tiến trình con kết thúc
   }
   if (pid == 0)             // Nếu là tiến trình con
   {
      execl("./count.sh", "./count.sh", "10", NULL);
                             // Thực thi file count.sh với đối số "10"
```

```
                    // Sau dòng này, nếu thành công thì tiến trình con bị thay thế hoàn toàn
                    // Nếu thất bại, chương trình tiếp tục chạy các dòng bên dưới
        printf("CHILDREN | PID = %ld | PPID = %ld\n",
            (long)getpid(), (long)getppid());  // In PID và PPID của tiến trình con

        printf("CHILDREN | List of arguments: \n");
        for (int i = 1; i < argc; i++)          // Duyệt qua các đối số từ argv[1] trở đi
        {
            printf("%s\n", argv[i]);            // In từng đối số
        }
    }
    exit(0);                // Kết thúc chương trình với mã thoát 0 (thành công)
}
```

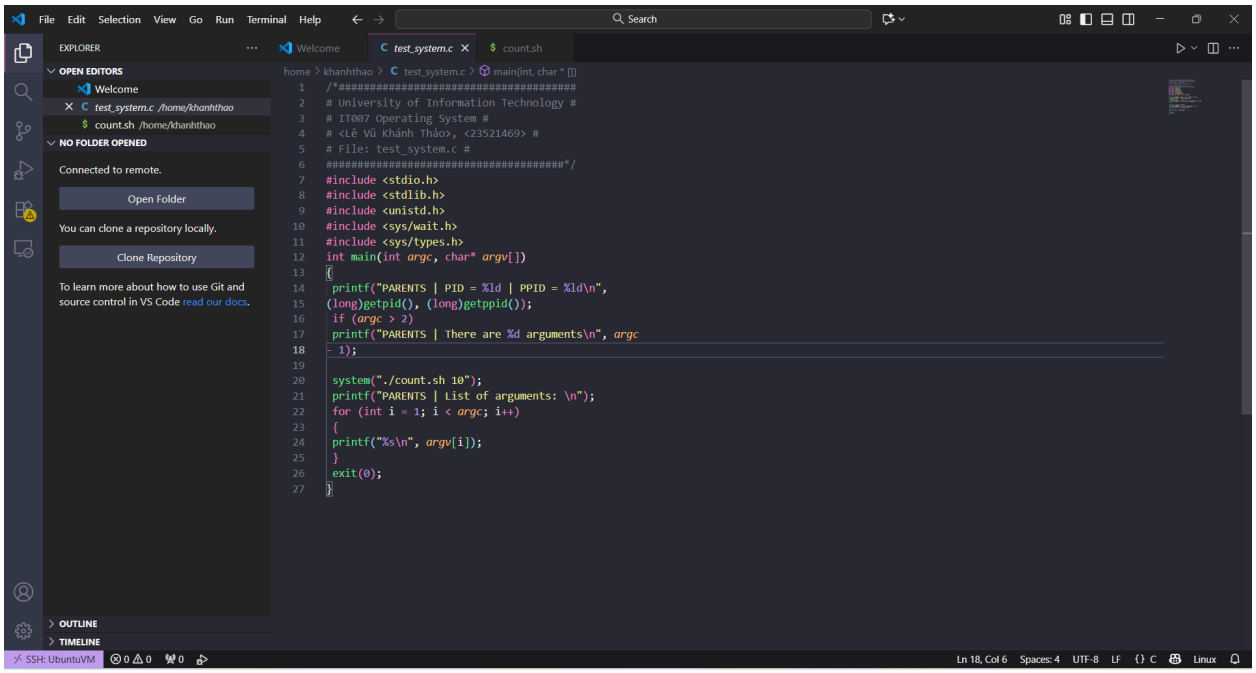
- ***Kết quả nhận được:***

```
khanhthao@khanhthao-VirtualBox:~$ ./test_execl 1 2 3
PARENTS | PID = 9911 | PPID = 9544
PARENTS | There are 3 arguments
CHILDREN | PID = 9912 | PPID = 9911
CHILDREN | List of arguments:
1
2
3
```

Hình 5: Kết quả nhận được của Ví dụ 2

- **PARENTS | PID = 9911 | PPID = 9544**: là tiến trình cha. PID của cha là 9911, PPID là 9544.
- **PARENTS | There are 3 arguments**: Đã truyền 3 đối số 1, 2, 3 -> argc = 4 (bao gồm tên chương trình), nên argc - 1 = 3.
- **CHILDREN | PID = 9912 | PPID = 9911**: là tiến trình con, PID của con là 9912, PPID là 9911 là PID của tiến trình cha.
- **CHILDREN | List of arguments: 1 2 3**: tiến trình con in ra các đối số từ argv[1] đến argv[argc-1].

c. Ví dụ 3.3:



Hình 6: Chương trình system() của ví dụ 3.3

- ***Giải thích code của chương trình system():***

#include <stdio.h> // Thư viện cho hàm printf()

#include <stdlib.h> // Thư viện cho hàm exit(), system()

#include <unistd.h> // Thư viện cho getpid(), getppid()

#include <sys/wait.h> // Thư viện cho hàm wait() (dù không dùng ở đây)

#include <sys/types.h> // Thư viện định nghĩa kiểu pid_t

int main(int argc, char* argv[]) // Hàm main nhận đối số từ dòng lệnh

{

// In ra PID của tiến trình hiện tại (cha) và PID của tiến trình cha nó (thường là terminal)

printf("PARENTS | PID = %ld | PPID = %ld\n",

(long)getpid(), (long)getppid());

// Nếu có nhiều hơn 2 đối số (tức là từ argv[1] trở đi), in ra số lượng đối số thực sự

if (argc > 2)

printf("PARENTS | There are %d arguments\n", argc - 1); // Trừ argv[0] là tên chương trình

// Gọi script count.sh và truyền đối số "10" cho nó

// Script sẽ chạy như một tiến trình con và ghi số từ 1 đến 10 vào file count.txt

system("./count.sh 10");

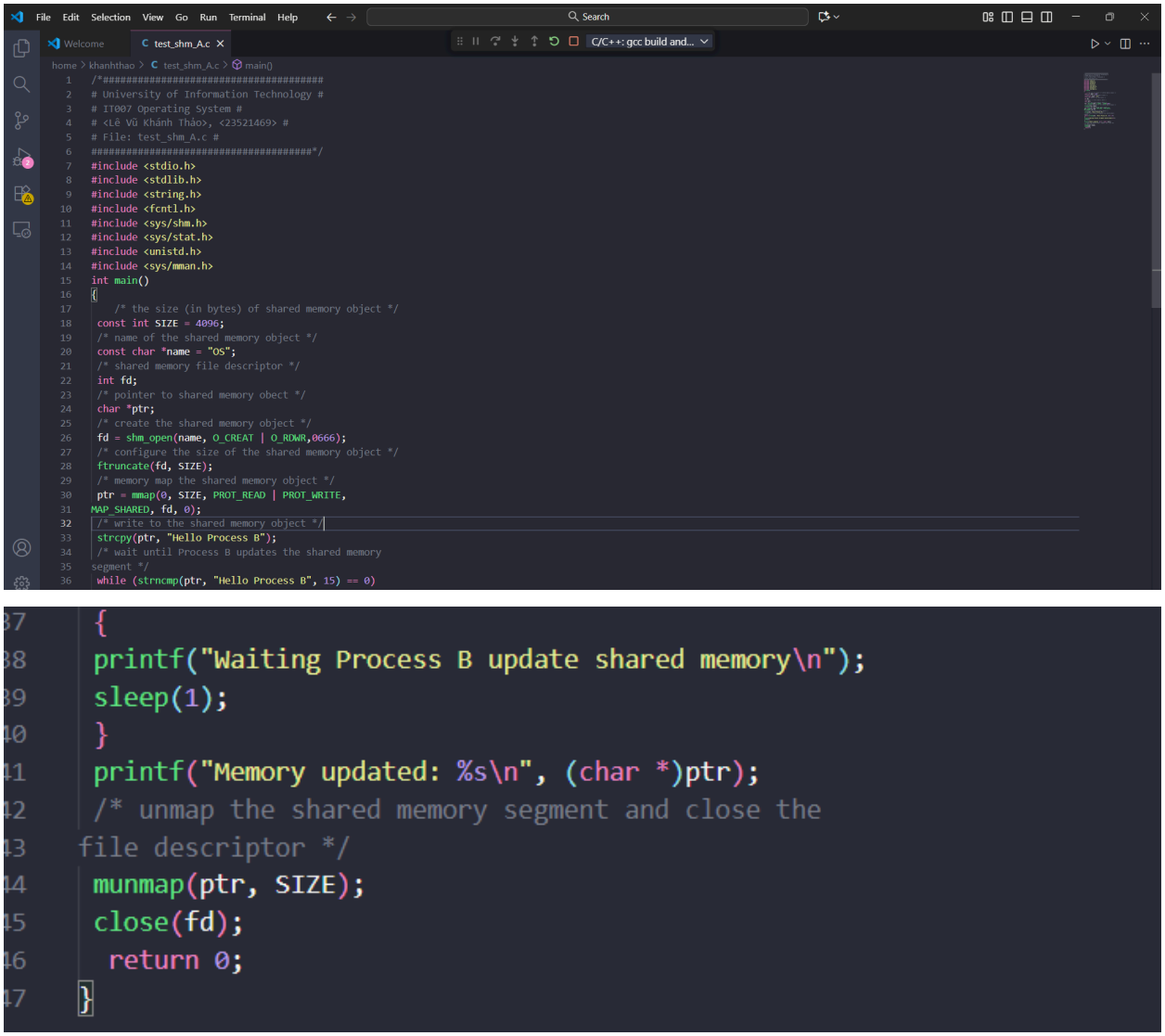
```
    // In tiêu đề cho danh sách đối số dòng lệnh
    printf("PARENTS | List of arguments: \n");
    // Duyệt qua các đối số từ argv[1] đến argv[argc - 1] và in ra từng cái
    for (int i = 1; i < argc; i++)
    {
        printf("%s\n", argv[i]);  // In từng đối số, ví dụ: 3, 4, 5
    }
    // Kết thúc chương trình với mã thoát 0 (thành công)
    exit(0);
}
```

- ***Kết quả nhận được:***

```
khanhthao@khanhthao-VirtualBox:~$ ./test_system 3 4 5
PARENTS | PID = 12657 | PPID = 10397
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
khanhth+   12658   12657 25 00:55 pts/5    00:00:00 sh -c -- ./count.sh 10
khanhth+   12659   12658  0 00:55 pts/5    00:00:00 /bin/bash ./count.sh 10
khanhth+   12661   12659  0 00:55 pts/5    00:00:00 grep count.sh
PARENTS | List of arguments:
3
4
5
```

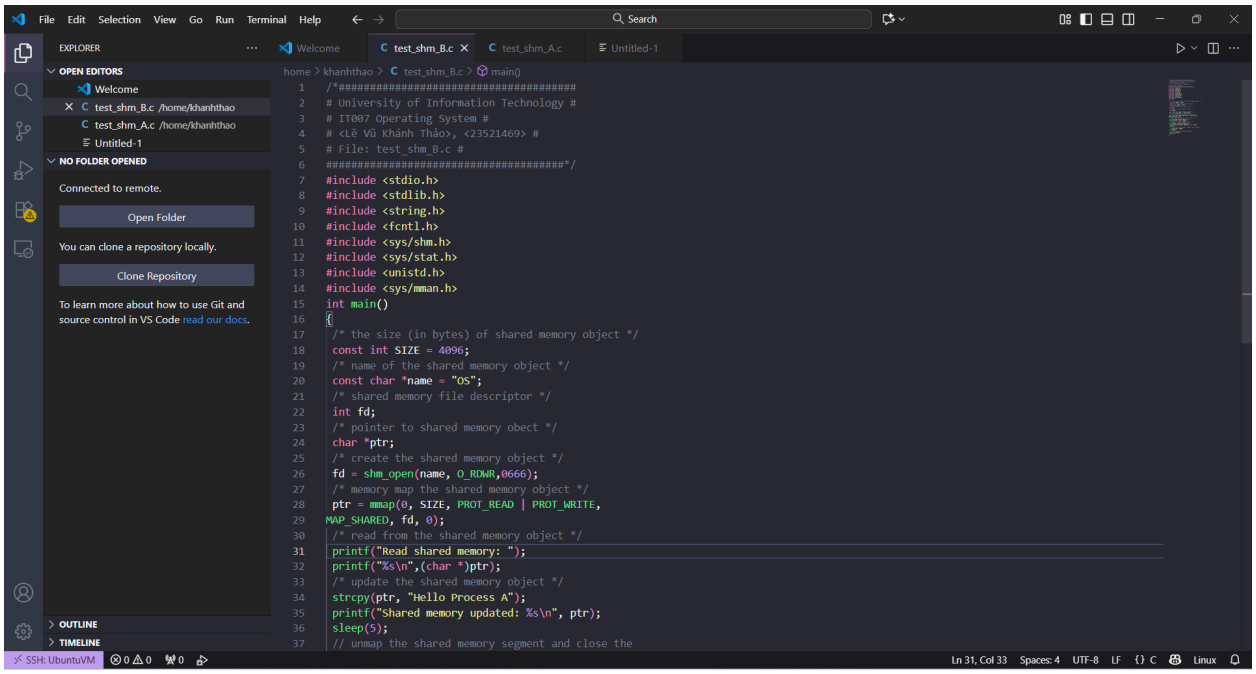
Hình 7: Kết quả nhận được của Ví dụ 3

d. Ví dụ 3.4:



```c
/*########################################
# University of Information Technology #
# IT007 Operating System #
# <Lê Vũ Khánh Thảo>, <23521469> #
# 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;
}
```

Hình 8,9 : Chương trình process A của ví dụ 3.4



```
// 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;
}
```

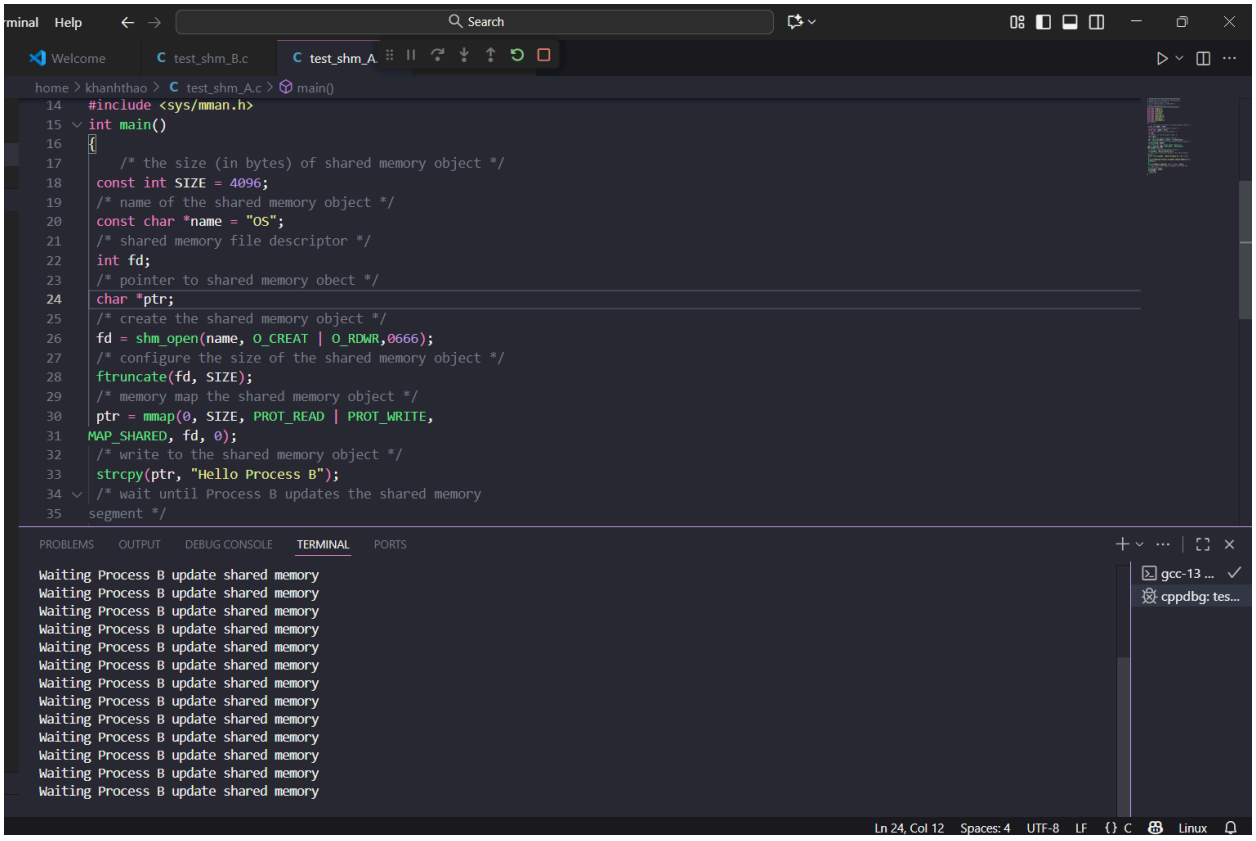
Hình 10, 11 : Chương trình process B của ví dụ 3.4

- ***Giải thích đoạn code chương trình process A:***

+ fd = shm_open(name, O_RDWR,0666): Mở một đối tượng bộ nhớ chia sẻ hiện có với tên là “OS” đã được tạo trước đó.
+ ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd,0): Ánh xạ đối tượng bộ nhớ chia sẻ vào không gian địa chỉ của tiến trình. Kết quả của hàm mmap() là một con trỏ (ptr) trỏ đến bắt đầu của đối tượng bộ nhớ chia sẻ.
+ printf(“%s\n”, (char*)ptr): Đọc và in ra chuỗi từ đối tượng bộ nhớ chia sẻ.
+ strepy(ptr, “Hello Process A”): Cập nhật chuỗi trong đối tượng bộ nhớ chia sẻ thành “Hello Process A”.
+ munmap(ptr, SIZE); close(fd): Hủy ánh xạ bộ nhớ và đóng file description của đối tượng bộ nhớ chia sẻ.
+ shm_unlink(name): Xóa đối tượng bộ nhớ chía sẻ.

- ***Kết quả nhận được sau khi thực thi Process A:***

  Chạy cho đến khi ./test_shm_B cập nhật.



Hình 12 : Kết quả nhận được sau khi thực thi Process A

- ***Giải thích đoạn code của chương trình B:***

+ fd = shm_open(name, O_RDWR,0666): Mở một đối tượng bộ nhớ chia sẻ hiện có với tên là “OS” đã được tạo trước đó.
+ ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0): Ánh xạ đối tượng bộ nhớ chia sẻ vào không gian địa chỉ của tiến trình. Kết quả của hàm mmap() là một con trỏ (ptr) trỏ đến bắt đầu của đối tượng bộ nhớ chia sẻ.
+ printf(“%\n”, (char*)ptr): Đọc và in ra chuỗi từ đối tượng bộ nhớ chia sẻ.
+ strcpy(ptr, “Hello Process A”): Cập nhật chuỗi trong đối tượng bộ nhớ chia sẻ thành “Hello Process A”.
+ munmap(ptr, SIZE); close(fd): Hủy ánh xạ bộ nhớ và đóng file description của đối tượng bộ nhớ chia sẻ.
+ shm_unlink(name): Xóa đối tượng bộ nhớ chia sẻ.

- ***Kết quả thực thi ./test_shm_B:***

Welcome | test_shm_B.c | test_shm_A.c | Untitled-1

home > khanhthao > test_shm_B.c > main()

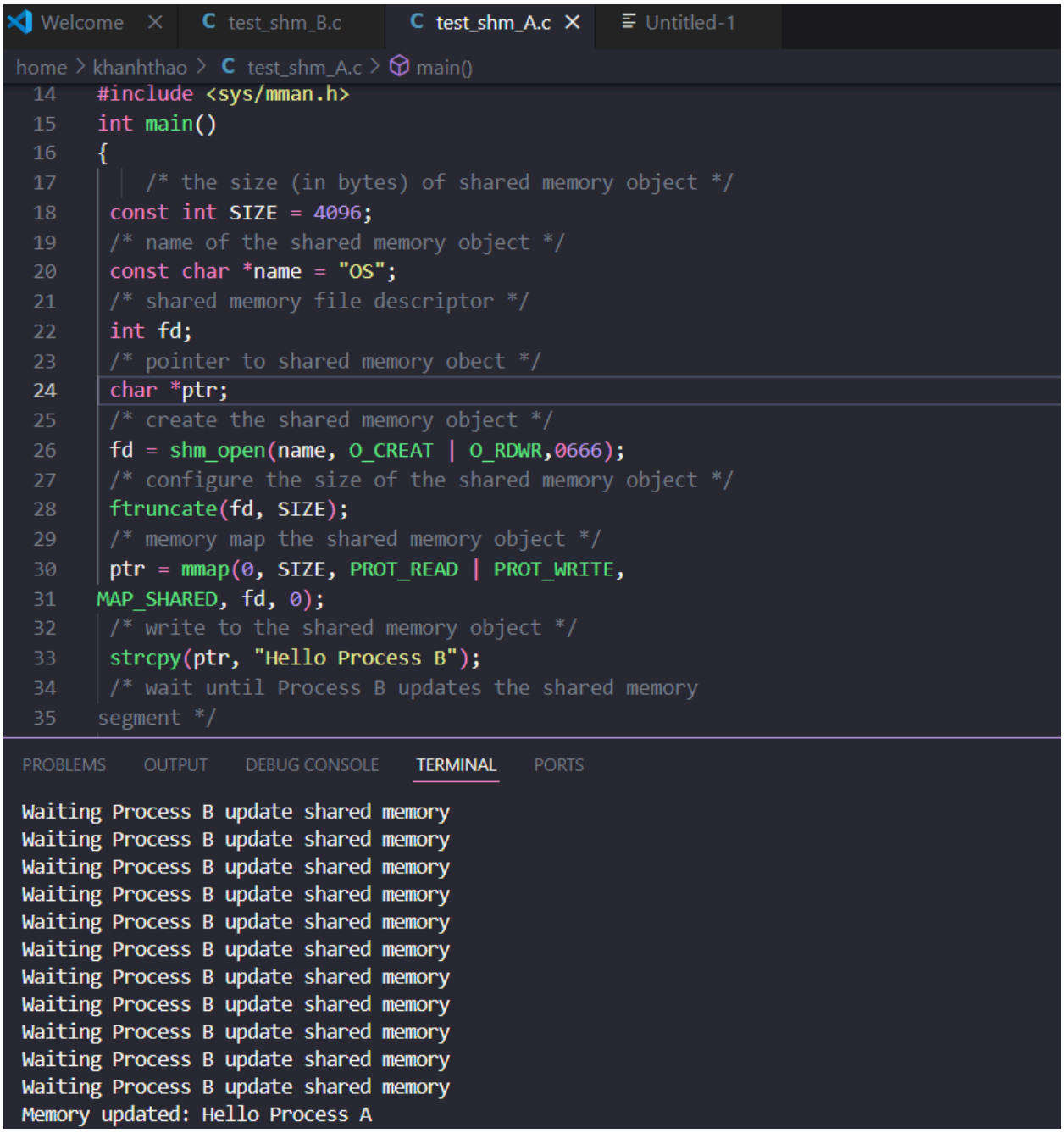
```c
/*########################################
# University of Information Technology #
# IT007 Operating System #
# <Lê Vũ Khánh Thảo>, <23521469> #
# File: test_shm_B.c #
########################################*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <unistd.h>

int main() {
    // Kích thước vùng nhớ dùng chung (bytes)
    const int SIZE = 4096;

    // Tên vùng nhớ dùng chung (phải khớp với tiến trình A)
    const char *name = "/OS";

    // File descriptor cho vùng nhớ
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Shared memory read: Hello Process B
Shared memory updated: Hello Process A
```

Hình 13 : Kết quả thực thi ./test_shm_B

- ***Kết quả ./test_shm A sau khi thực thi ./test_shm_B:***



Hình 14 : Kết quả ./test_shm A sau khi thực thi ./test_shm_B

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

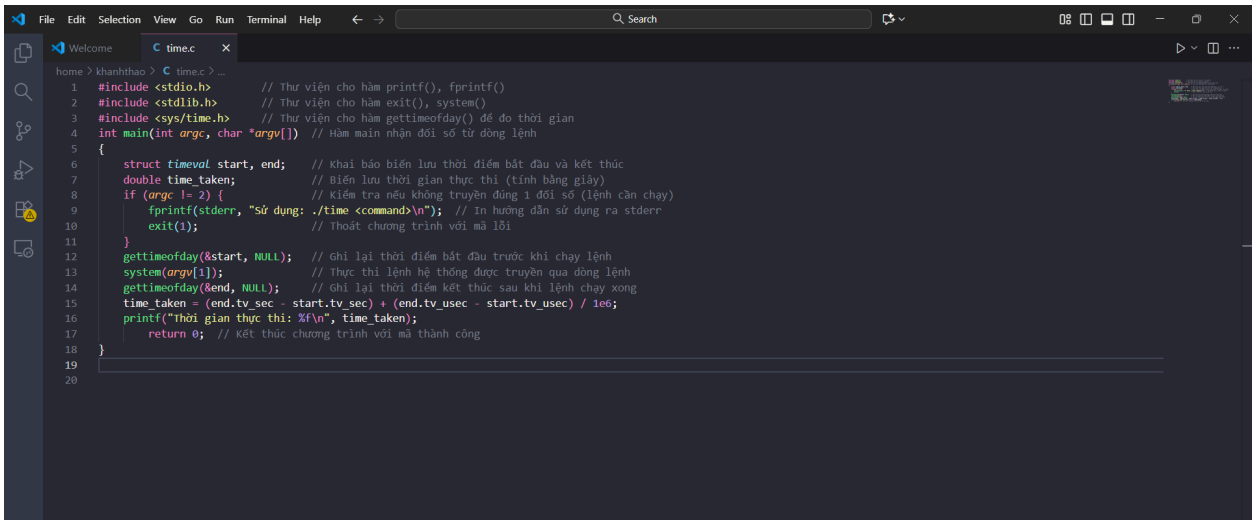
time.c 44

time

Thời gian thực thi: 0.25422

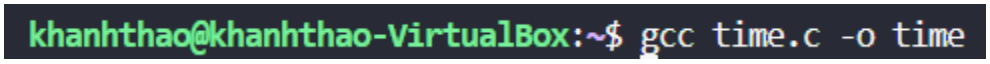
Gợi ý: Tiến trình cha gọi hàm fork() tạo ra tiến trình con rồi wait(). Tiến trình con gọi hàm gettimeofday() để lấy mốc thời gian trước khi thực thi lệnh shell, sau đó sử dụng hàm execl() để thực thi lệnh. Sau khi tiến trình con kết thúc, tiến trình cha tiếp tục gọi hàm gettimeofday() một lần nữa để lấy mốc thời gian sau khi thực thi lệnh shell và tính toán.

- ***Chương trình thực hiện yêu cầu:***



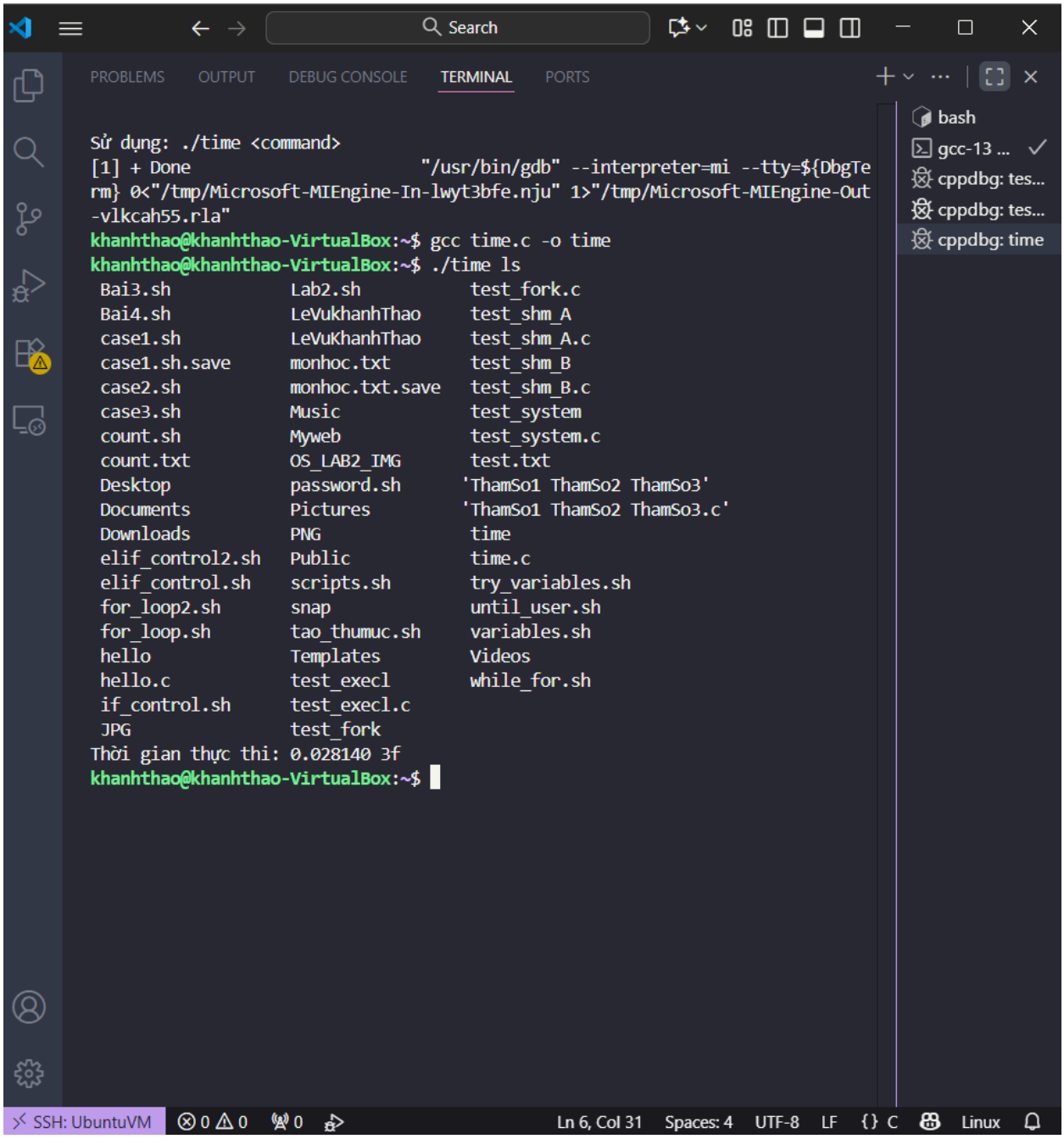
Hình 15 : Chương trình thực thi yêu cầu và giải thích

- ***Chương trình biên dịch File (gcc):***



Hình 16: Chương trình biên dịch File (gcc)

- ***Chạy thử một số lệnh:***
  - ***ls:***



Hình 17: Chạy thử lệnh ls

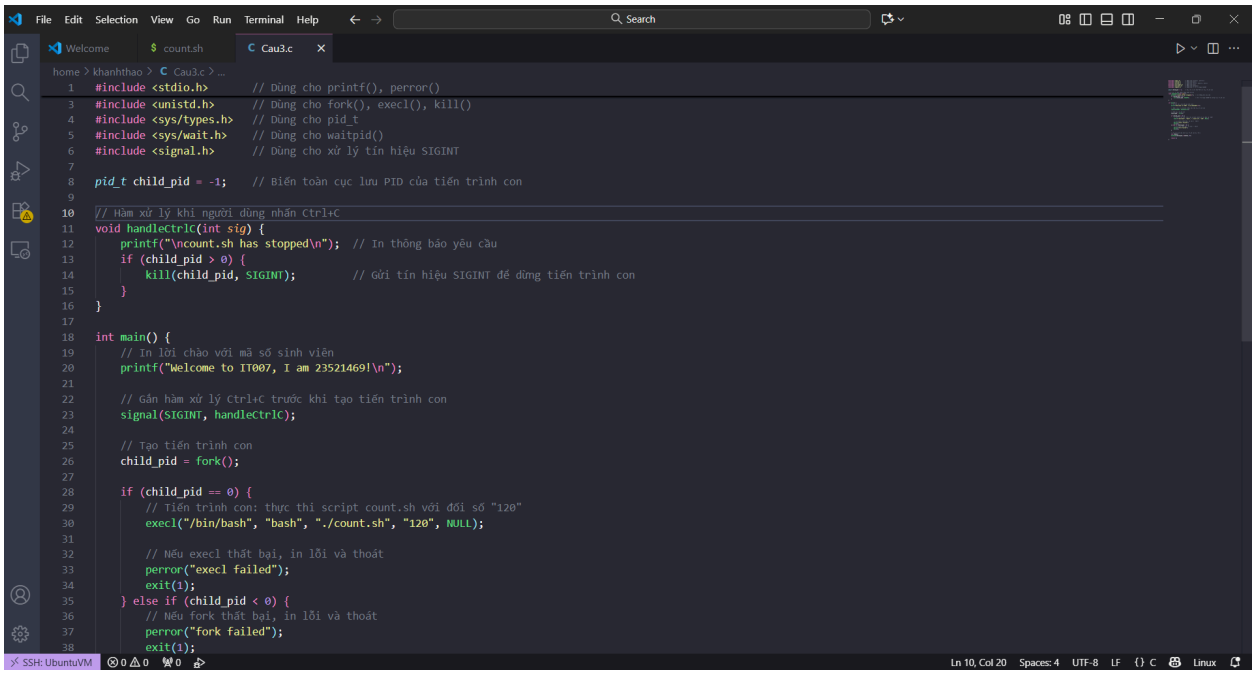
+ ***Lệnh mkdir:***

```
khanhthao@khanhthao-VirtualBox:~$ ./time mkdir
mkdir: missing operand
Try 'mkdir --help' for more information.
Thời gian thực thi: 0.024470 3f
```

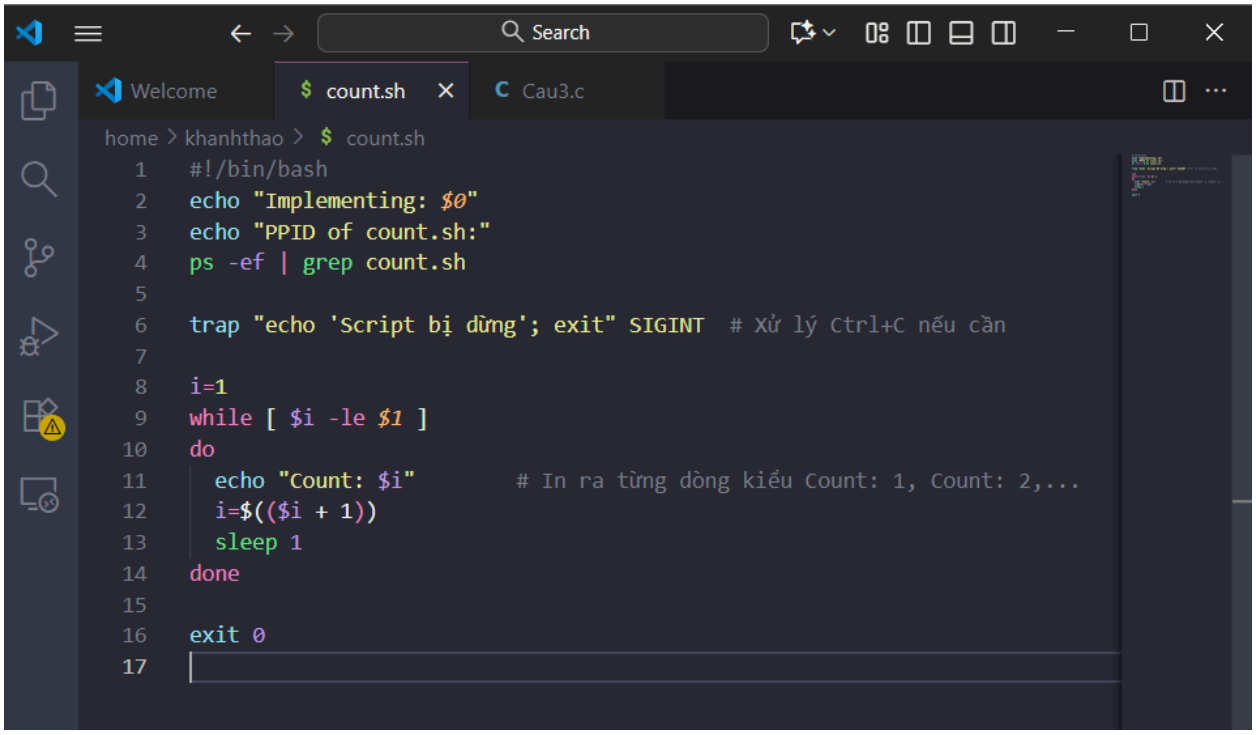
Hình 18: Chạy thử lệnh mkdir

3. Viết một chương trình làm 4 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".**



Hình 19 : Chương trình thực thi yêu cầu và giải thích



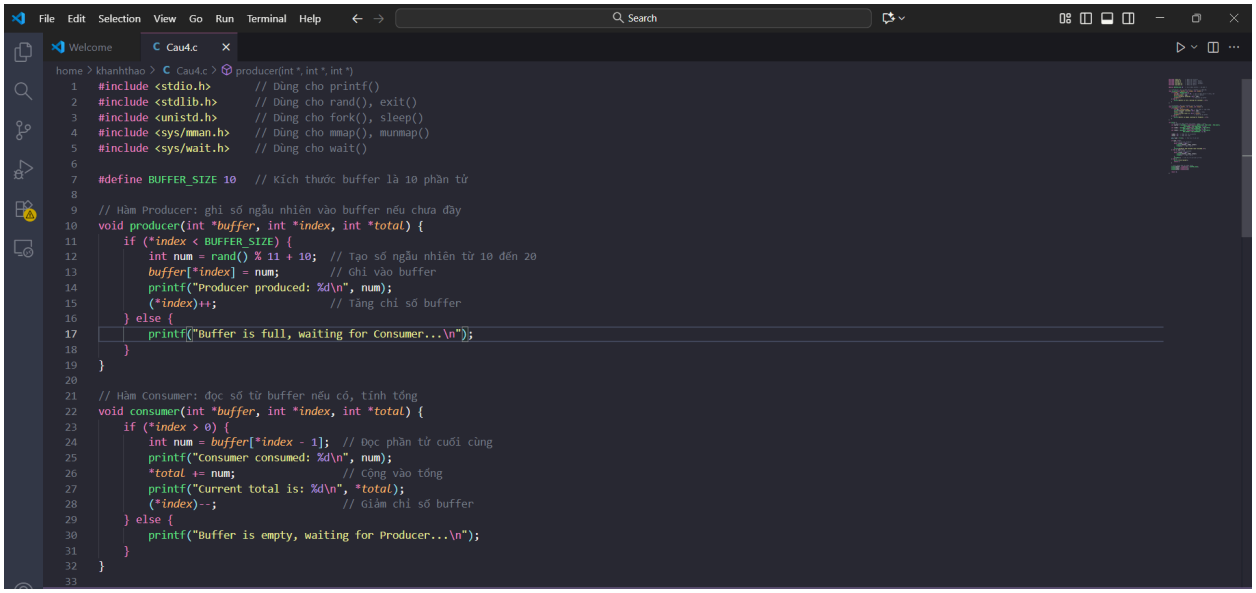
Hình 20 : File script count.sh với số lần đếm là 120.

```
khanhthao@khanhthao-VirtualBox:~$ chmod +x count.sh
khanhthao@khanhthao-VirtualBox:~$ gcc Cau3.c -o main
khanhthao@khanhthao-VirtualBox:~$ ./Cau3
Welcome to IT007, I am 23521469!
Implementing: ./count.sh
PPID of count.sh:
khanhth+   10144   10143  0 22:53 pts/0    00:00:00 bash ./count.sh 120
khanhth+   10146   10144  0 22:53 pts/0    00:00:00 grep count.sh
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
Count: 6
Count: 7
Count: 8
Count: 9
^C
count.sh has stopped
Script bị dừng
```

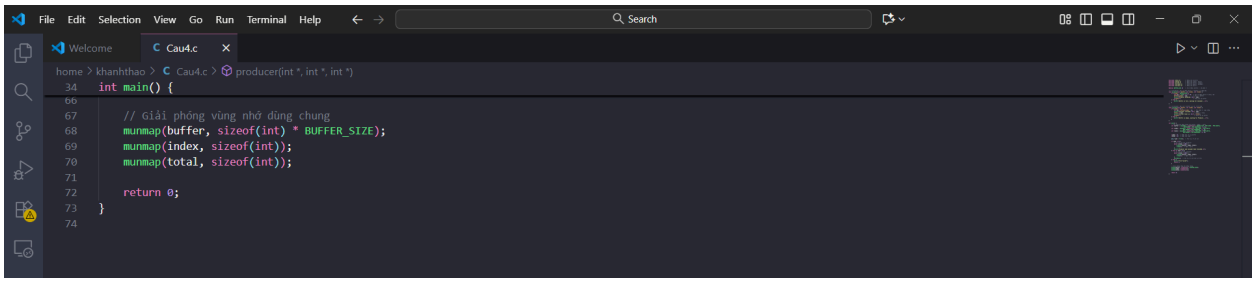
Hình 21: Kết quả của Câu 3

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**



```c
int main() {
    // Tạo vùng nhớ dùng chung cho buffer, index và total
    int *buffer = mmap(NULL, sizeof(int) * BUFFER_SIZE, PROT_READ | PROT_WRITE,
                       MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    int *index = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
                      MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    int *total = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
                      MAP_SHARED | MAP_ANONYMOUS, -1, 0);

    *index = 0;  // Khởi tạo chỉ số buffer
    *total = 0;  // Khởi tạo tổng

    pid_t pid = fork();  // Tạo tiến trình con

    if (pid == 0) {
        // Tiến trình con: Producer
        while (*total < 100) {
            producer(buffer, index, total);
            sleep(1);  // Nghỉ 1 giây
        }
        printf("Producer and Consumer have finished.\n");
    } else if (pid > 0) {
        // Tiến trình cha: Consumer
        while (*total < 100) {
            consumer(buffer, index, total);
            sleep(1);  // Nghỉ 1 giây
        }
        wait(NULL);  // Chờ tiến trình con kết thúc
    } else {
        perror("fork failed");
        return 1;
    }
```



Hình 22, 23, 24: Đoạn code thực thi Câu 4

```
khanhthao@khanhthao-VirtualBox:~$ gcc Cau4.c -o pc
khanhthao@khanhthao-VirtualBox:~$ ./pc
Buffer is empty, waiting for Producer...
Producer produced: 16
Consumer consumed: 16
Current total is: 16
Producer produced: 20
Producer produced: 16
Consumer consumed: 16
Current total is: 32
Producer produced: 12
Consumer consumed: 12
Current total is: 44
Consumer consumed: 20
Producer produced: 11
Current total is: 64
Consumer consumed: 20
Current total is: 84
Producer produced: 14
Producer produced: 10
Consumer consumed: 10
Current total is: 94
Producer produced: 16
Consumer consumed: 16
Current total is: 110
Producer and Consumer have finished.
```

Hình 25: Kết quả của Câu 4