

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

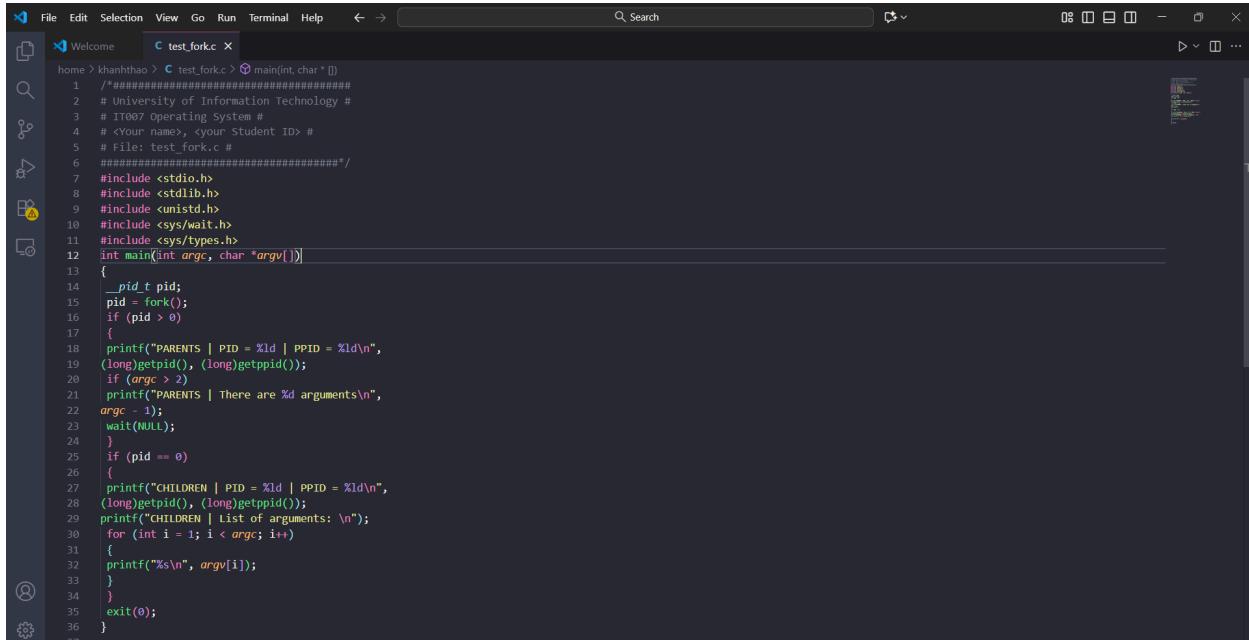
MSSV: 23521469

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:



```
File Edit Selection View Go Run Terminal Help ← → Search
home > khanhhao > c test_fork.c main(int, char *[])
1  ##### University of Information Technology #
2  # IT007 Operating System #
3  # <Your name>, <your Student ID> #
4  # File: test_fork.c #
5  #####
6  #####
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <unistd.h>
10 #include <sys/wait.h>
11 #include <sys/types.h>
12 int main(int argc, char *argv[])
13 {
14     _pid_t pid;
15     pid = fork();
16     if (pid > 0)
17     {
18         printf("PARENTS | PID = %ld | PPID = %ld\n",
19             (long)getpid(), (long)getppid());
20         if (argc > 2)
21             printf("PARENTS | There are %d arguments\n",
22                 argc - 1);
23         wait(NULL);
24     }
25     if (pid == 0)
26     {
27         printf("CHILDREN | PID = %ld | PPID = %ld\n",
28             (long)getpid(), (long)getppid());
29         printf("CHILDREN | List of arguments: \n");
30         for (int i = 1; i < argc; i++)
31         {
32             printf("%s\n", argv[i]);
33         }
34         exit(0);
35     }
36 }
```

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:**

```

khanhtao@khanhtao-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

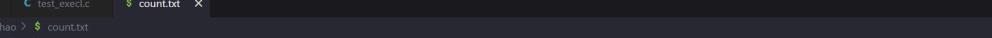
```

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:



```
home > hanhthao > $ count.txt
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
```

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

```
home > khanhthao > C test_execlc > $ count.txt
1 //#####
2 # University of Information Technology #
3 # IT007 Operating System #
4 # <Lê Vũ Khánh Thành>, <23521469> #
5 # File: test_execlc.c #
6 #####
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <unistd.h>
10 #include <sys/wait.h>
11 #include <sys/types.h>
12 int main(int argc, char* argv[])
13 {
14     _pid_t pid;
15     pid = fork();
16     if (pid > 0)
17     {
18         printf("PARENTS | PID = %ld | PPID = %ld\n",
19             (long)getpid(), (long)getppid());
20         if (argc > 2)
21             printf("PARENTS | There are %d arguments\n",
22                 argc - 1);
23         wait(NULL);
24     }
25     if (pid == 0)
26     {
27         execl("./count.sh", "./count.sh", "10", NULL);
28
29         printf("CHILDREN | PID = %ld | PPID = %ld\n",
30             (long)getpid(), (long)getppid());
31         printf("CHILDREN | List of arguments: \n");
32         for (int i = 1; i < argc; i++)
33         {
34             printf("%s\n", argv[i]);
35         }
36     }
37     exit(0);
}
```

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 exec*

```

#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:*

```

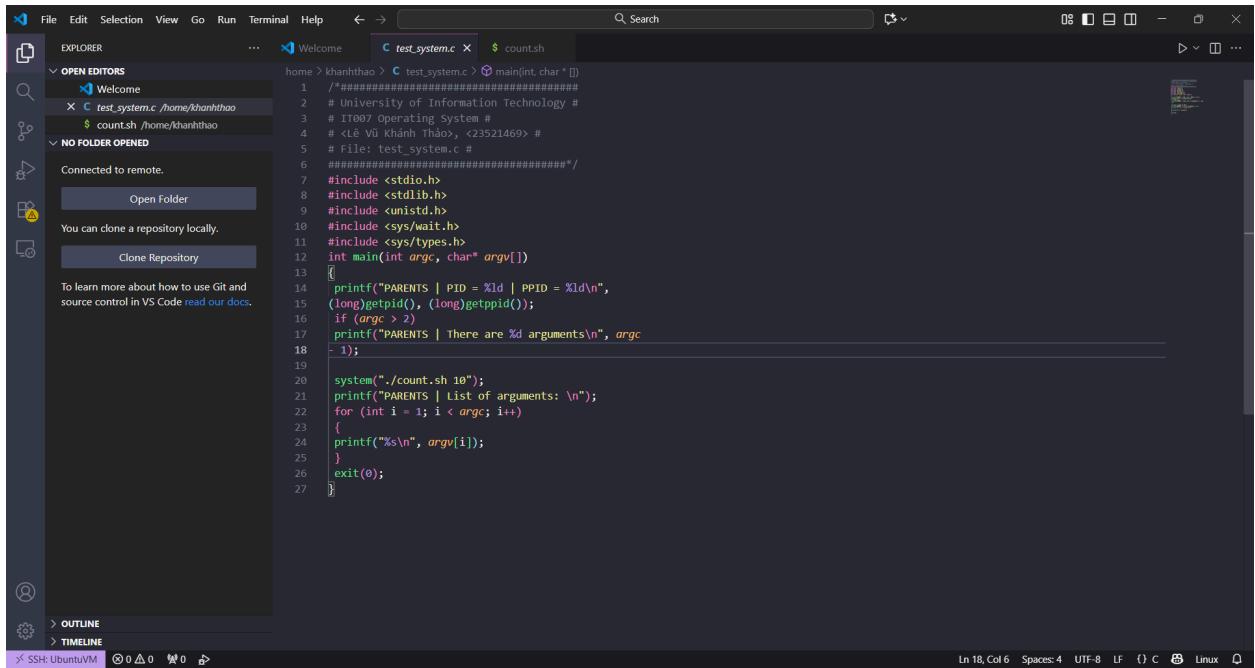
khanhtao@khanhtao-VirtualBox:~/Documents$ ./test_exec1 1 2 3
PARENTS | PID = 9911 | PPID = 9544
PARENTS | There are 3 arguments
CHILDREN | PID = 9912 | PPID = 9911
CHILDREN | List of arguments:
1
2
3

```

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:



```
File Edit Selection View Go Run Terminal Help < > Search
OPEN EDITORS
  Welcome
  test_system.c /home/khanhthao
  count.sh /home/khanhthao
NO FOLDER OPENED
Connected to remote.
Open Folder
Clone Repository
To learn more about how to use Git and source control in VS Code read our docs.

home > khanhthao > C test_system.c > main(int, char **)
1  /*#####
2   # University of Information Technology #
3   # IT07 Operating System #
4   # <Lê Vũ Khanh Thảo>, <23521469> #
5   # File: test_system.c #
6   #####*/
7   #include <stdio.h>
8   #include <stdlib.h>
9   #include <unistd.h>
10  #include <sys/wait.h>
11  #include <sys/types.h>
12  int main(int argc, char* argv[])
13  {
14      printf("PARENTS | PID = %ld | PPID = %ld\n",
15      (long)getpid(), (long)getppid());
16      if (argc > 2)
17          printf("PARENTS | There are %d arguments\n", argc
18          - 1);
19
20      system("./count.sh 10");
21      printf("PARENTS | List of arguments: \n");
22      for (int i = 1; i < argc; i++)
23      {
24          printf("%s\n", argv[i]);
25      }
26  }
27

In 18, Col 6  Spaces: 4  UTF-8  LF  ⚡ C  🔍 Linux  🌐
```

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:*

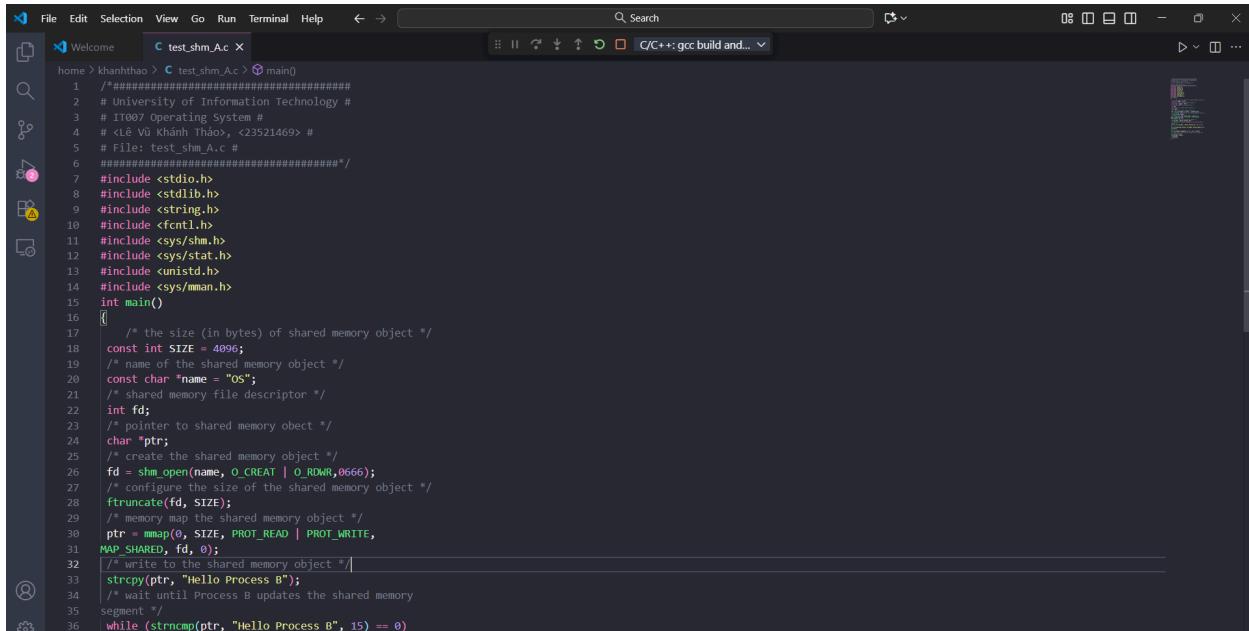
```

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:
khanhthao 12658 12657 25 00:55 pts/5 00:00:00 sh -c -- ./count.sh 10
khanhthao 12659 12658 0 00:55 pts/5 00:00:00 /bin/bash ./count.sh 10
khanhthao 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:



```

File Edit Selection View Go Run Terminal Help ← → Search
Welcome C test_shm_A.c main()
1 //#####
2 # University of Information Technology #
3 # IT07 Operating system #
4 # <Lê Vũ Khánh Thảo>, <23521469> #
5 # File: test_shm_A.c #
6 #####
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <string.h>
10 #include <fcntl.h>
11 #include <sys/shm.h>
12 #include <sys/stat.h>
13 #include <unistd.h>
14 #include <sys/mman.h>
15 int main()
16 {
17     /* the size (in bytes) of shared memory object */
18     const int SIZE = 4096;
19     /* name of the shared memory object */
20     const char *name = "OS";
21     /* shared memory file descriptor */
22     int fd;
23     /* pointer to shared memory object */
24     char *ptr;
25     /* create the shared memory object */
26     fd = shm_open(name, O_CREAT | O_RDWR, 0666);
27     /* configure the size of the shared memory object */
28     ftruncate(fd, SIZE);
29     /* memory map the shared memory object */
30     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE,
31     MAP_SHARED, fd, 0);
32     /* write to the shared memory object */
33     strcpy(ptr, "Hello Process A");
34     /* wait until Process B updates the shared memory
35     segment */
36     while (strcmp(ptr, "Hello Process A", 15) == 0)
37     {
38         printf("Waiting Process B update shared memory\n");
39         sleep(1);
40     }
41     printf("Memory updated: %s\n", (char *)ptr);
42     /* unmap the shared memory segment and close the
43     file descriptor */
44     munmap(ptr, SIZE);
45     close(fd);
46     return 0;
47 }

```

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

```

1  #####Universität für Informationstechnik#####
2  # University of Information Technology #
3  # IT007 Operating System #
4  # <Lê Vũ Khánh Thảo>, <23521469> #
5  # File: test_shm_B.c #
6  #####
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10 #include <fcntl.h>
11 #include <sys/shm.h>
12 #include <sys/stat.h>
13 #include <unistd.h>
14 #include <sys/mman.h>
15 int main()
16 {
17     /* the size (in bytes) of shared memory object */
18     const int SIZE = 4096;
19     /* name of the shared memory object */
20     const char *name = "OS";
21     /* shared memory file descriptor */
22     int fd;
23     /* pointer to shared memory object */
24     char *ptr;
25     /* create the shared memory object */
26     fd = shm_open(name, O_RDWR, 0666);
27     /* memory map the shared memory object */
28     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE,
29                MAP_SHARED, fd, 0);
30     /* read from the shared memory object */
31     printf("Read shared memory: ");
32     printf("%s\n", (char *)ptr);
33     /* update the shared memory object */
34     strcpy(ptr, "Hello Process A");
35     printf("Shared memory updated: %s\n", ptr);
36     sleep(5);
37     // unmap the shared memory segment and close the
38     munmap(ptr, SIZE);
39     close(fd);
40     // remove the shared memory segment
41     shm_unlink(name);
42     return 0;
43 }

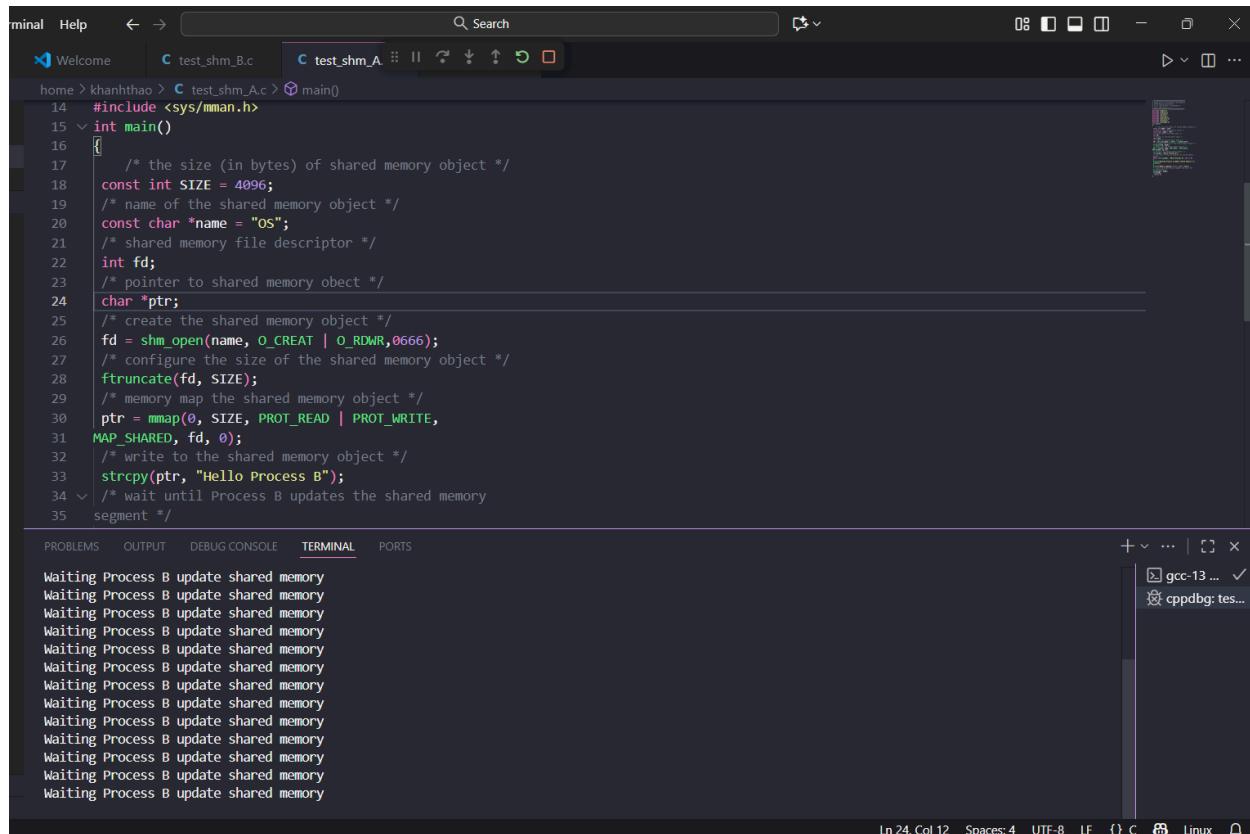
```

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.
- + 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 nhận được sau khi thực thi Process A:**

Chạy cho đến khi ./test_shm_B cập nhật.



The screenshot shows a terminal window with the title bar "terminal Help" and the current tab "test_shm_A". The code in the editor is for Process A, which creates a shared memory object named "os" of size 4096 bytes and maps it at address 0. It then writes "Hello Process B" to the memory. The terminal output shows repeated messages: "Waiting Process B update shared memory", indicating that Process A is waiting for updates from Process B. The status bar at the bottom right shows "Ln 24, Col 12" and other terminal settings.

```
#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 object */
    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 */
}
```

Waiting Process B update shared memory
Waiting Process B update shared memory

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`:

```
home > khanhthao > C test_shm_B.c > main()
1  /*#####
2  # University of Information Technology #
3  # IT007 Operating System #
4  # <Lê Vũ Khánh Thảo>, <23521469> #
5  # File: test_shm_B.c #
6 ####*/
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <string.h>
10 #include <fcntl.h>
11 #include <sys/stat.h>
12 #include <sys/mman.h>
13 #include <unistd.h>
14
15 int main() {
16     // Kích thước vùng nhớ dùng chung (bytes)
17     const int SIZE = 4096;
18
19     // Tên vùng nhớ dùng chung (phải khớp với tiến trình A)
20     const char *name = "/OS";
21
22     // 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`:

The screenshot shows a terminal window with the following output:

```

Waiting Process B update shared memory
Memory updated: Hello Process A

```

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 muôn đo thời gian thực thi.

Ví dụ:

\$./time ls

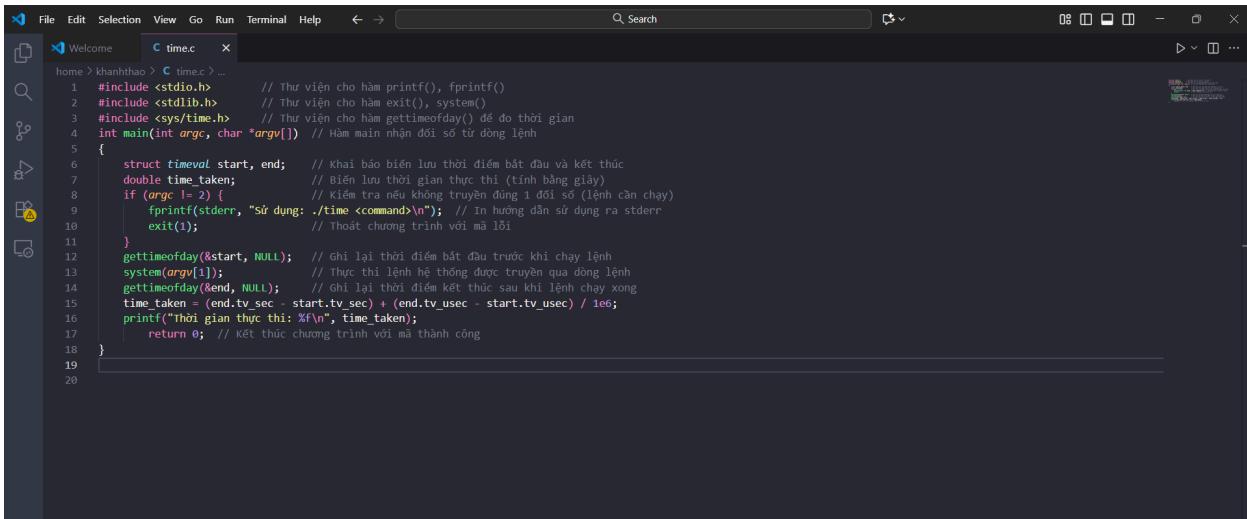
time.c 44

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:*



```
home > khanhthao > C time.c > ...
1 #include <stdio.h>           // Thư viện cho hàm printf(), fprintf()
2 #include <stdlib.h>          // Thư viện cho hàm exit(), system()
3 #include <sys/time.h>         // Thư viện cho hàm gettimeofday() để đo thời gian
4 int main(int argc, char *argv[]) // Hàm main nhận dối số từ dòng lệnh
{
    struct timeval start, end;   // Khai báo biến lưu thời điểm bắt đầu và kết thúc
    double time_taken;          // Biến lưu thời gian thực thi (tính bằng giây)
    if (argc != 2) {             // Kiểm tra nếu không truyền đúng 1 đối số (lệnh cần chạy)
        fprintf(stderr, "Sử dụng: ./time <command>\n"); // In hướng dẫn sử dụng ra stderr
        exit(1);                // Thoát chương trình với mã lỗi
    }
    gettimeofday(&start, NULL); // Ghi lại thời điểm bắt đầu trước khi chạy lệnh
    system(argv[1]);           // Thực thi lệnh hệ thống được truyền qua dòng lệnh
    gettimeofday(&end, NULL);  // Ghi lại thời điểm kết thúc sau khi lệnh chạy xong
    time_taken = (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec) / 1e6;
    printf("Thời gian thực thi: %f\n", time_taken);
    return 0;                  // Kết thúc chương trình với mã thành công
}
```

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

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

```
khanhthao@khanhthao-VirtualBox:~$ gcc time.c -o time
```

Hình 16: Chương trình biên dịch File (gcc)

- *Chạy thử một số lệnh:*

- + *ls:*

Sử dụng: ./time <command>
[1] + Done " /usr/bin/gdb" --interpreter=mi --tty=\${DbgTe
rm} <"/tmp/Microsoft-MIEngine-In-1wyt3bfe.nju" 1>/tmp/Microsoft-MIEngine-Out
-vlkah55.rla"
khanhthao@khanhthao-VirtualBox:~\$ gcc time.c -o time
khanhthao@khanhthao-VirtualBox:~\$./time ls
Bai3.sh Lab2.sh test_fork.c
Bai4.sh LevUkhanhThao test_shm_A
case1.sh LevUkhanhThao test_shm_A.c
case1.sh.save monhoc.txt test_shm_B
case2.sh monhoc.txt.save test_shm_B.c
case3.sh Music test_system
count.sh Myweb test_system.c
count.txt OS_LAB2_IMG test.txt
Desktop password.sh 'ThamSo1 ThamSo2 ThamSo3'
Documents Pictures 'ThamSo1 ThamSo2 ThamSo3.c'
Downloads PNG time
elif_control2.sh Public time.c
elif_control.sh scripts.sh try_variables.sh
for_loop2.sh snap until_user.sh
for_loop.sh tao_thumuc.sh variables.sh
hello Templates Videos
hello.c test_execl while_for.sh
if_control.sh test_execl.c
JPG test_fork
Thời gian thực thi: 0.028140 3f
khanhthao@khanhthao-VirtualBox:~\$

Hình 17: Chạy thử lệnh ls

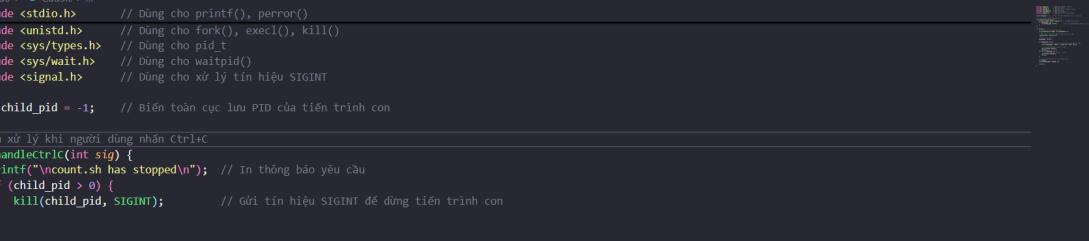
+ Lệnh mkdir:

```
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”.



```
File Edit Selection View Go Run Terminal Help ← → Search

Welcome $ count.sh C Cau3.c x

home > khanhtao > C Cau3.c > ...
1 #include <stdio.h> // Dùng cho printf(), perror()
2 #include <unistd.h> // Dùng cho fork(), exec(), kill()
3 #include <sys/types.h> // Dùng cho pid_t
4 #include <sys/wait.h> // Dùng cho waitpid()
5 #include <signal.h> // Dùng cho xử lý tín hiệu SIGINT
6
7 pid_t child_pid = -1; // Biến toàn cục lưu PID của tiến trình con
8
9 // Hàm xử lý khi người dùng nhấn Ctrl+C
10 void handleCtrlC(int sig) {
11     printf("ncount.sh has stopped\n"); // In thông báo yêu cầu
12     if (child_pid > 0) {
13         kill(child_pid, SIGINT); // Gửi tín hiệu SIGINT để dừng tiến trình con
14     }
15 }
16
17
18 int main() {
19     // In lời chào với mã số sinh viên
20     printf("Welcome to IT007, I am 23521469!\n");
21
22     // Gán hàm xử lý Ctrl+C trước khi tạo tiến trình con
23     signal(SIGINT, handleCtrlC);
24
25     // Tạo tiến trình con
26     child_pid = fork();
27
28     if (child_pid == 0) {
29         // Tiến trình con: thực thi script count.sh với đối số "120"
30         exec("./bin/bash", "bash", "./count.sh", "120", NULL);
31
32         // Nếu exec thất bại, in lỗi và thoát
33         perror("exec failed");
34         exit(1);
35     } else if (child_pid < 0) {
36         // Nếu fork thất bại, in lỗi và thoát
37         perror("fork failed");
38         exit(1);
39 }

In 10, Col 20 Spaces: 4 UTE-8 LF {} C Linux
```

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

A screenshot of a terminal window within a code editor interface. The terminal title bar says '\$ count.sh'. The code editor sidebar shows icons for file operations like Open, Save, Find, and Copy/Paste. The main pane displays the content of the file 'count.sh'.

```
home > khanhthao > $ count.sh
1  #!/bin/bash
2  echo "Implementing: $0"
3  echo "PPID of count.sh:"
4  ps -ef | grep count.sh
5
6  trap "echo 'Script bị dừng'; exit" SIGINT # Xử lý Ctrl+C nếu cần
7
8  i=1
9  while [ $i -le $1 ]
10 do
11     echo "Count: $i"          # In ra từng dòng kiểu Count: 1, Count: 2, ...
12     i=$((i + 1))
13     sleep 1
14 done
15
16 exit 0
17
```

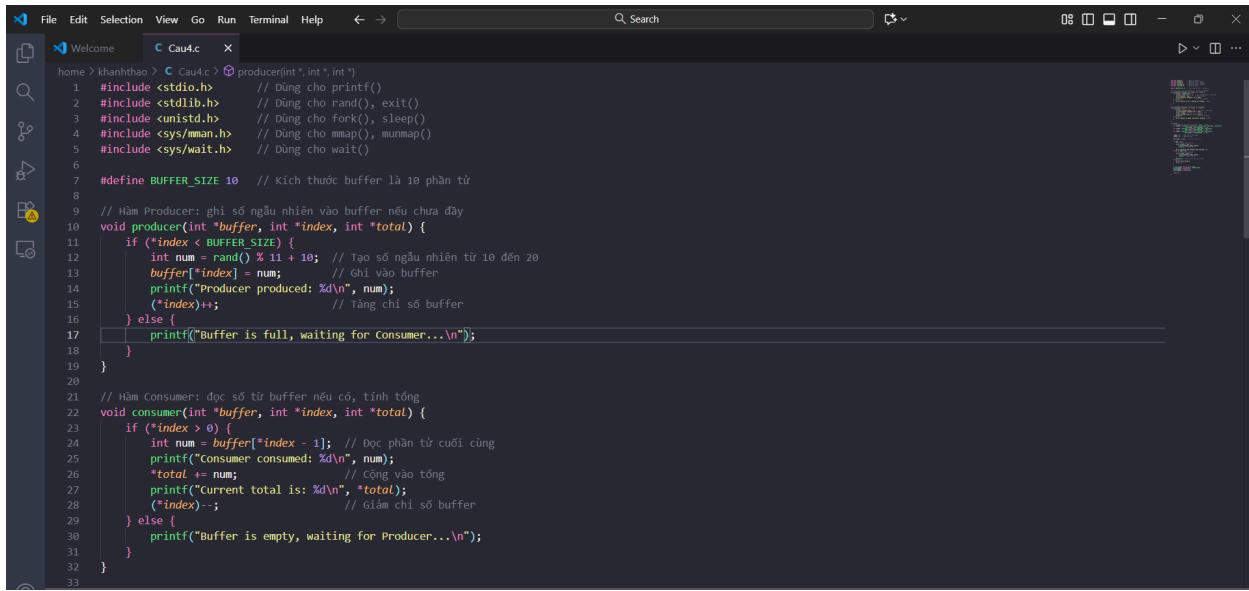
Hình 20 : File script count.sh với số lần đếm là 120.

```
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:
khanhthao    10144    10143    0 22:53 pts/0    00:00:00 bash ./count.sh 120
khanhthao    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.
- Tiền trình cha đóng vai trò là Producer, tạo một số ngẫu nhiên trong khoảng [10, 20] và ghi dữ liệu vào buffer.
- Tiền 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



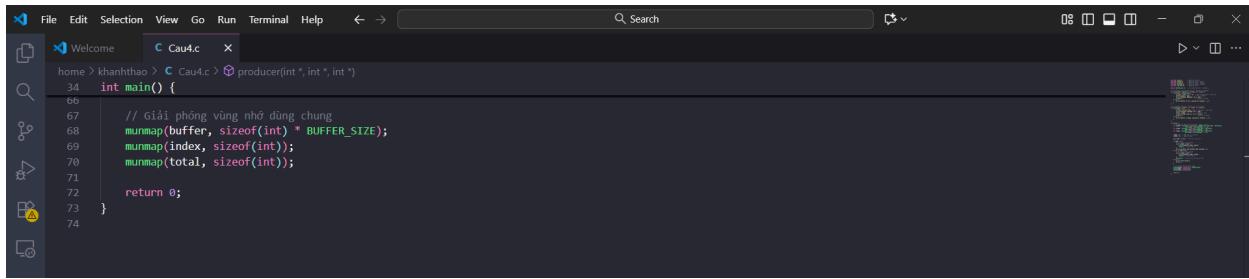
```
File Edit Selection View Go Run Terminal Help ← → Search

home > khanhtkho > Cau4.c > producer(int *, int *, int *)
1 #include <stdio.h> // Dùng cho printf()
2 #include <stdlib.h> // Dùng cho rand(), exit()
3 #include <unistd.h> // Dùng cho fork(), sleep()
4 #include <sys/mman.h> // Dùng cho mmap(), munmap()
5 #include <sys/wait.h> // Dùng cho wait()
6
7 #define BUFFER_SIZE 10 // Kích thước buffer là 10 phần tử
8
9 // Hàm Producer: ghi số ngẫu nhiên vào buffer nếu chưa đầy
10 void producer(int *buffer, int *index, int *total) {
11     if (*index < BUFFER_SIZE) {
12         int num = rand() % 11 + 10; // Tạo số ngẫu nhiên từ 10 đến 20
13         buffer[*index] = num; // Ghi vào buffer
14         printf("Producer produced: %d\n", num);
15         (*index)++; // Tăng chỉ số buffer
16     } else {
17         printf("Buffer is full, waiting for Consumer...\n");
18     }
19 }
20
21 // Hàm Consumer: đọc số từ buffer nếu có, tính tổng
22 void consumer(int *buffer, int *index, int *total) {
23     if (*index > 0) {
24         int num = buffer[*index - 1]; // Đọc phần tử cuối cùng
25         printf("Consumer consumed: %d\n", num);
26         *total += num; // Cộng vào tổng
27         printf("Current total is: %d\n", *total);
28         (*index)--; // Giảm chỉ số buffer
29     } else {
30         printf("Buffer is empty, waiting for Producer..\n");
31     }
32 }
33 }
```

```

34 int main() {
35     // Tạo vùng nhớ dùng chung cho buffer, index và total
36     int *buffer = mmap(NULL, sizeof(int) * BUFFER_SIZE, PROT_READ | PROT_WRITE,
37                         MAP_SHARED | MAP_ANONYMOUS, -1, 0);
38     int *index = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
39                         MAP_SHARED | MAP_ANONYMOUS, -1, 0);
40     int *total = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
41                         MAP_SHARED | MAP_ANONYMOUS, -1, 0);
42
43     *index = 0; // Khởi tạo chỉ số buffer
44     *total = 0; // Khởi tạo tổng
45
46     pid_t pid = fork(); // Tạo tiến trình con
47
48     if (pid == 0) {
49         // Tiến trình con: Producer
50         while (*total < 100) {
51             producer(buffer, index, total);
52             sleep(1); // Nghỉ 1 giây
53         }
54         printf("Producer and Consumer have finished.\n");
55     } else if (pid > 0) {
56         // Tiến trình cha: Consumer
57         while (*total < 100) {
58             consumer(buffer, index, total);
59             sleep(1); // Nghỉ 1 giây
60         }
61         wait(NULL); // Chờ tiến trình con kết thúc
62     } else {
63         perror("fork failed");
64     }
65 }

```



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