

Họ và tên: Đồng Quốc Thắng

Mã số sinh viên: 23521421

Lớp: IT007.P11.CTTN

HỆ ĐIỀU HÀNH BÁO CÁO LAB 3

CHECKLIST (Đánh dấu x khi hoàn thành)

Lưu ý mỗi câu phải làm đủ 3 yêu cầu

I. Bài Tập Thực hành

	BT 1	BT 2	BT 3	BT 4
Trình bày cách làm	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Chụp hình minh chứng	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Giải thích kết quả	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

II. Bài Tập Ôn Tập

	a
Trình bày cách làm	<input type="checkbox"/>
Chụp hình minh chứng	<input type="checkbox"/>
Giải thích kết quả	<input type="checkbox"/>

Tự chấm điểm: 10

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?

Ví dụ 3-1:

```
vim test_f
5 # Dong Quoc Thang, 23521421
6 # File: test_fork.c
7 #
8 #####*/
9 #include <stdio.h>
10 #include <stdlib.h>
11 #include <unistd.h>
12 #include <sys/wait.h>
13 #include <sys/types.h>
14 int main(int argc, char *argv[])
15 {
16     __pid_t pid;
17     pid = fork();
18     if (pid > 0)
19     {
20         printf("PARENTS | PID = %ld | PPID = %ld\n",
21             (long)getpid(), (long)getppid());
22         if (argc > 2)
23             printf("PARENTS | There are %d arguments\n",
24                 argc - 1);
25         wait(NULL);
26     }
27     if (pid == 0){
28         printf("CHILDREN | PID = %ld | PPID = %ld\n",
29             (long)getpid(), (long)getppid());
30         printf("CHILDREN | List of arguments: \n");
31         for (int i = 1; i < argc; i++)
32         {
33             printf("%s\n", argv[i]);
34         }
35     }
36     exit(0);
```

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Thân Thế Tùng.

Ta có pid là giá trị return của hàm fork, nếu pid > 0 thì tiến trình đó là tiến trình cha nên sẽ print ra PARENTS và PID của parent, parent PID của tiến trình cha. Sử dụng wait(NULL) để tiến trình con kết thúc rồi tiến trình cha mới kết thúc.

Ở tiến trình con, sẽ in ra CHILDREN và các command line argument ta nhập vào lúc chạy code này.

```
(~/IT007/Lab3) —————  
[ (08:28:59 on main * *) —> ./test_fork arg1 arg2 arg3 arg4  
PARENTS | PID = 23357 | PPID = 14666  
PARENTS | There are 4 arguments  
CHILDREN | PID = 23358 | PPID = 23357  
CHILDREN | List of arguments:  
arg1  
arg2  
arg3  
arg4  
  
(~/IT007/Lab3) —————  
[ (08:29:08 on main * *) —> |
```

Ta có thể thấy parent process cũng có 4 argument và child cũng có 4 arg là các arg được truyền từ command line => children process cũng copy các command line argument của parent process.

Ví dụ 3-2:

```
vim test_exec
5 # Dong Quoc Thang, 23521421
6 # File: test_execl.c
7 #
8 #####*/
9 #include <stdio.h>
10 #include <stdlib.h>
11 #include <unistd.h>
12 #include <sys/wait.h>
13 #include <sys/types.h>
14 int main(int argc, char* argv[])
15 {
16     __pid_t pid;
17     pid = fork();
18     if (pid > 0)
19     {
20         printf("PARENTS | PID = %ld | PPID = %ld\n",
21             (long) getpid(), (long) getppid());
22         if (argc > 2)
23             printf("PARENTS | There are %d arguments\n",
24                 argc - 1);
25         wait(NULL);
26     }
27     if (pid == 0)
28     {
29         execl("./count.sh", "./count.sh", "10", NULL);
30         printf("CHILDREN | PID = %ld | PPID = %ld\n",
31             (long) getpid(), (long) getppid());
32         printf("CHILDREN | List of arguments: \n");
33         for (int i = 1; i < argc; i++)
34         {
35             printf("%s\n", argv[i]);
36         }
37     }
38 }
```

Đoạn code này chỉ khác ở đoạn ở đoạn code trong ví dụ 3.1 ở dòng 29. Ở đây khi tiến trình con chạy lệnh execl, thì tiến trình con sẽ chỉ chạy file count.sh và sẽ không chạy tiếp những dòng 30, 31,... trong file test_exec này

```
turtle@EvilBrewHausen:~/IT007/Lab3 | 147x
[~/IT007/Lab3]
(08:39:47 on main * *)-> ./test_exec1 arg1 arg2 arg3
PARENTS | PID = 26877 | PPID = 14666
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
turtle      26878    26877    0 08:39 pts/1    00:00:00 /bin/bash ./count.sh 10
turtle      26880    26878    0 08:39 pts/1    00:00:00 grep count.sh
[~/IT007/Lab3]
(08:39:59 on main * *)-> |
```

Output ở trên cho thấy sau khi tiến trình cha chạy xong thì sẽ đến tiến trình con, tiến trình con chỉ chạy file count.sh và không chạy các lệnh printf ở phía dưới.

Ví dụ 3-3

```
vim test
1 /*#####
2 # University of Information Technology
3 # IT007 Operating System
4 #
5 # Dong Quoc Thang 23521421
6 # File: test_system.c
7 #
8 #####*/
9 #include <stdio.h>
10 #include <stdlib.h>
11 #include <unistd.h>
12 #include <sys/wait.h>
13 #include <sys/types.h>
14 int main(int argc, char* argv[])
15 {
16     printf("PARENTS | PID = %ld | PPID = %ld\n",
17           (long)getpid(), (long)getppid());
18     if (argc > 2)
19         printf("PARENTS | There are %d arguments\n", argc
20               - 1);
21
22     system("./count.sh 10");
23     printf("PARENTS | List of arguments: \n");
24     for (int i = 1; i < argc; i++)
25     {
26         printf("%s\n", argv[i]);
27     }
28     exit(0);
29 }
```

Ở đây khi ta dùng lệnh system, thì sẽ tạo ra hẳn một tiến trình mới để chạy, và sẽ chương trình sẽ tiếp tục thực hiện các dòng lệnh sau lệnh system


```
(~/IT007/Lab3) —
(08:46:16 on main * *)—> ./test_system arg1 arg2 arg3
PARENTS | PID = 28824 | PPID = 14666
PARENTS | There are 3 arguments
Implementing: ./count.sh
PPID of count.sh:
turtle      28825      28824  0 08:46 pts/1      00:00:00 /bin/bash ./count.sh 10
turtle      28827      28825  0 08:46 pts/1      00:00:00 grep count.sh
PARENTS | List of arguments:
arg1
arg2
arg3
(~/IT007/Lab3) —
(08:46:33 on main * *)—> |
```

Sau khi chạy file count.sh xong thì vẫn in ra các argument arg1 arg2 arg3.

Ví dụ 3-4:

```
(~/IT007/Lab3) —
(08:59:19 on main * *)—> ./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
(~/IT007/Lab3) —
(08:59:28 on main * *)—>
```

```
(~/IT007/Lab3) —
(08:59:18 on main * *)—> ./test_shm_B
Read shared memory: Hello Process B
Shared memory updated: Hello Process A
(~/IT007/Lab3) —
(08:59:32 on main * *)—> |
```

```
vim te
17
18     /* the size (in bytes) of shared memory object */
19     const int SIZE = 4096;
20     /* name of the shared memory object */
21     const char *name = "OA";
22     /* shared memory file descriptor */
23     int fd;
24     /* pointer to shared memory object */
25     char *ptr;
26     /* create the shared memory object */
27     fd = shm_open(name, O_CREAT | O_RDWR, 0666);
28     /* configure the size of the shared memory object */
29     ftruncate(fd, SIZE);
30     /* memory map the shared memory object */
31     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE,
32               MAP_SHARED, fd, 0);
33     /* write to the shared memory object */
34     strcpy(ptr, "Hello Process B");
35     /* wait until Process B updates the shared memory
36        segment */
37     while (strcmp(ptr, "Hello Process B", 15) == 0)
38     {
39         printf("Waiting Process B update shared memory\n");
40         sleep(1);
41     }
42     printf("Memory updated: %s\n", (char *)ptr);
43     /* unmap the shared memory segment and close the
44        file descriptor */
45     munmap(ptr, SIZE);
46     close(fd);
47     return 0;
48 }
```



```
vim test_shm_B.c
13 #include <sys/stat.h>
14 #include <unistd.h>
15 #include <sys/mman.h>
16 int main()
17
18     /* the size (in bytes) of shared memory object */
19     const int SIZE = 4096;
20     /* name of the shared memory object */
21     const char *name = "0A";
22     /* shared memory file descriptor */
23     int fd;
24     /* pointer to shared memory object */
25     char *ptr;
26     /* create the shared memory object */
27     fd = shm_open(name, O_RDWR, 0666);
28     /* memory map the shared memory object */
29     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE,
30               MAP_SHARED, fd, 0);
31     /* read from the shared memory object */
32     printf("Read shared memory: ");
33     printf("%s\n", (char *)ptr);
34     /* update the shared memory object */
35     strcpy(ptr, "Hello Process A");
36     printf("Shared memory updated: %s\n", ptr);
37     sleep(5);
38     // unmap the shared memory segment and close the file descriptor
39     munmap(ptr, SIZE);
40     close(fd);
41     // remove the shared memory segment
42     shm_unlink(name);
43     return 0;
44
```

Ở file test_shm_A, thực hiện các lệnh shm_open để tạo shared memory, sau đó dùng ftruncate để chỉ số bộ nhớ cần dùng, và tạo con trỏ ptr để chỉ đến vùng nhớ đó(dùng hàm mmap). Ở file test_shm_B, ta không cần tạo shared memory nữa mà chỉ cần truy cập vào shared memory mà A đã tạo, và viết vào Hello process A, ở process A chờ shared memory được thay đổi, sau khi thay đổi thì đóng lại các con trỏ và vùng nhớ đã tạo. Còn ở process B thì còn phải unlink shared memory để bỏ shared memory segment.

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.

```
vim measure_time.c
1 #include<stdio.h>
2 #include<time.h>
3 #include<stdlib.h>
4 #include <sys/time.h>
5
6 int main(int argc, char* argv[])
7 {
8     if (argc != 2) {
9         printf("chon mot command muon thuc hien\n");
10        printf("Cach dung: ./measure_time \"<command> <argument> <argument>\"\n");
11        exit(0);
12    }
13
14    struct timeval tval_before, tval_after, tval_result;
15
16    gettimeofday(&tval_before, NULL);
17    system(argv[1]);
18    gettimeofday(&tval_after, NULL);
19
20    timersub(&tval_after, &tval_before, &tval_result);
21
22    printf("Thoi gian thuc hien: %ld.%06ld\n", (long int)tval_result.tv_sec, (long int)tval_result.tv_usec);
23    return 0;
24 }
```

```
(~/IT007/Lab3)
(09:34:12 on main * *) -> ./measure_time "ls -l && echo banana"
total 128
-rwxr-xr-x 1 turtle turtle 169 08:57 17 Thg 10 count.sh
-rw-r--r-- 1 turtle turtle 63 08:46 22 Thg 10 count.txt
-rwxr-xr-x 1 turtle turtle 15712 09:34 22 Thg 10 measure_time
-rw-r--r-- 1 turtle turtle 586 09:34 22 Thg 10 measure_time.c
-rwxr-xr-x 1 turtle turtle 15800 08:58 17 Thg 10 test_exec1
-rw-r--r-- 1 turtle turtle 849 08:34 22 Thg 10 test_exec1.c
-rwxr-xr-x 1 turtle turtle 15744 08:45 17 Thg 10 test_fork
-rw-r--r-- 1 turtle turtle 797 08:35 22 Thg 10 test_fork.c
-rwxr-xr-x 1 turtle turtle 15856 09:46 17 Thg 10 test_shm_A
-rw-r--r-- 1 turtle turtle 1307 09:02 22 Thg 10 test_shm_A.c
-rwxr-xr-x 1 turtle turtle 15800 09:46 17 Thg 10 test_shm_B
-rw-r--r-- 1 turtle turtle 1199 09:46 17 Thg 10 test_shm_B.c
-rwxr-xr-x 1 turtle turtle 15704 09:00 17 Thg 10 test_system
-rw-r--r-- 1 turtle turtle 646 08:45 22 Thg 10 test_system.c
banana
Thoi gian thuc hien: 0.006407
(~/IT007/Lab3)
(09:34:22 on main * *) -> |
```

Cách làm: chương trình sử dụng thư viện sys/time.h và các hàm gettimeofday để lấy thời gian. Sau đó sử dụng hàm timer_sub để trừ 2 thời gian.

Giải thích: Ta sử dụng hàm system để gọi hàm muốn thực hiện, argument cho hàm system là một command mà ta muốn shell thực hiện. Ta đưa argument cho hàm system bằng cách lấy argument từ command line argument. Ta sử dụng hàm system thay vì hàm exec vì ta muốn lấy thời gian ở phía sau command và trừ để lấy thời gian thực hiện.

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 stopped”

```
(~/IT007/Lab3)
(10:03:03 on main * *)→ ./signal
Welcome to IT007, I am 23521421!
Implementing: ./count.sh
PPID of count.sh:
turtle      53488    53487    0 10:03 pts/1    00:00:00 /bin/bash ./count.sh 120 NULL
turtle      53490    53488    0 10:03 pts/1    00:00:00 grep count.sh
^C
count.sh has stopped
(~/IT007/Lab3)
(10:03:04 on main * *)→ |
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <signal.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/wait.h>
7
8 pid_t pid;
9
10 void handle_sigint(int sig) {
11     printf("\ncount.sh has stopped\n");
12     kill(pid, SIGINT);
13     exit(0);
14 }
15
16 int main() {
17     printf("Welcome to IT007, I am 23521421!\n");
18
19     pid = fork();
20
21     if (pid == 0) {
22         execl("./count.sh", "./count.sh", "120", "NULL");
23         exit(1);
24     } else {
25         signal(SIGINT, handle_sigint);
26
27         wait(NULL);
28     }
29     return 0;
30 }
31
```

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Thân Thế Tùng.

Cách làm: sử dụng hàm fork để tạo ra tiến trình cha và tiến trình con, tiến trình con sẽ thực hiện lệnh count.sh 120, còn tiến trình cha sẽ chờ và handle SIGINT

Giải thích: Tiến trình cha sẽ dùng hàm signal để chờ SIGINT và khi có SIGINT, thì sẽ gọi hàm handle_sigint, trong lúc đó, tiến trình con sẽ thực hiện count.sh. Khi có tín hiệu SIGINT, thì hàm handle_sigint sẽ chạy để in ra dòng count.sh has stopped và kill tiến trình con đi(kill count.sh để không chạy nữa).

4. Viết chương trình mô phỏng bài toán Producer - Consumer như sau:

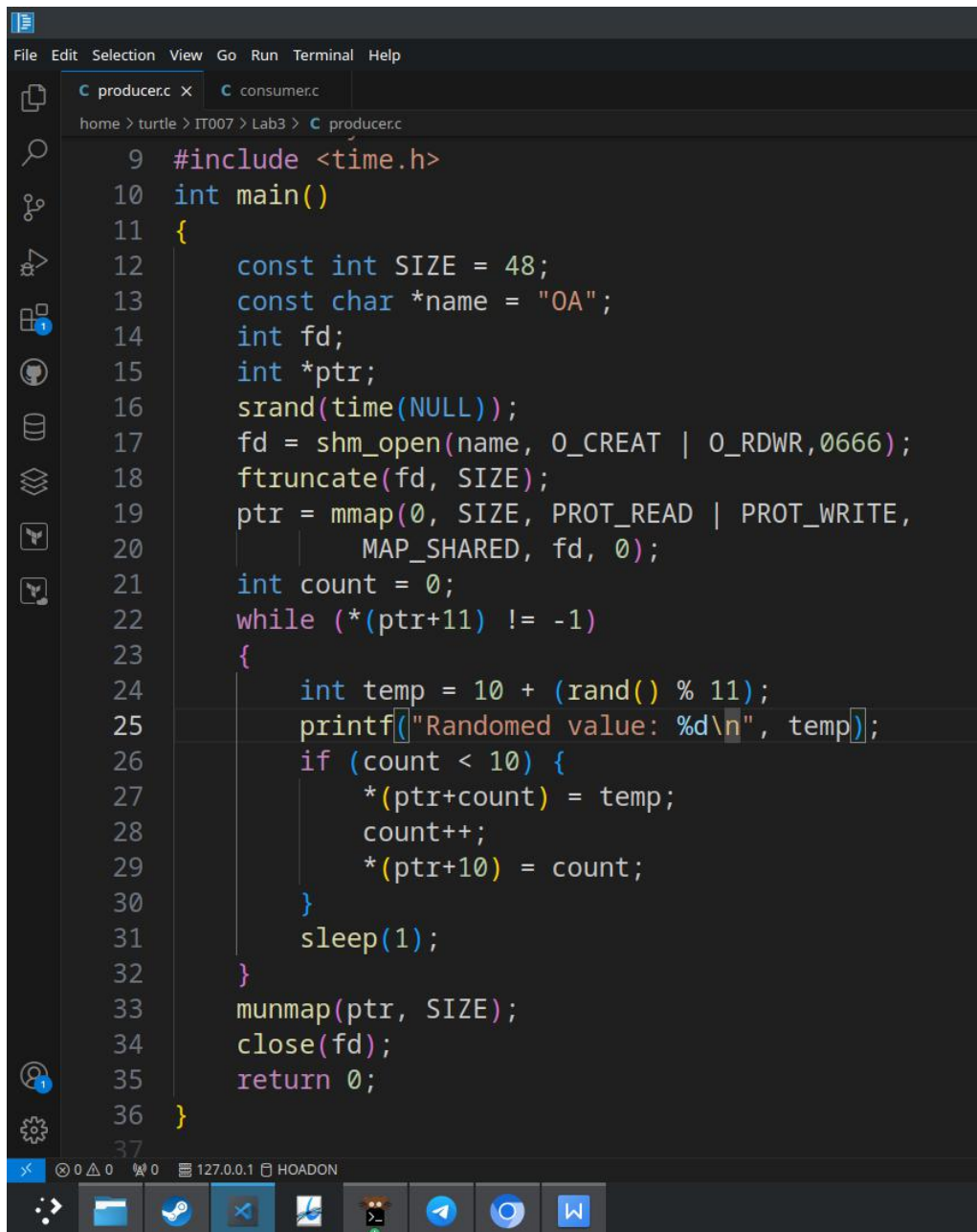
+) Sử dụng kỹ thuật shared-memory để tạo một bounded-buffer có độ lớn là 10 bytes.

+) 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

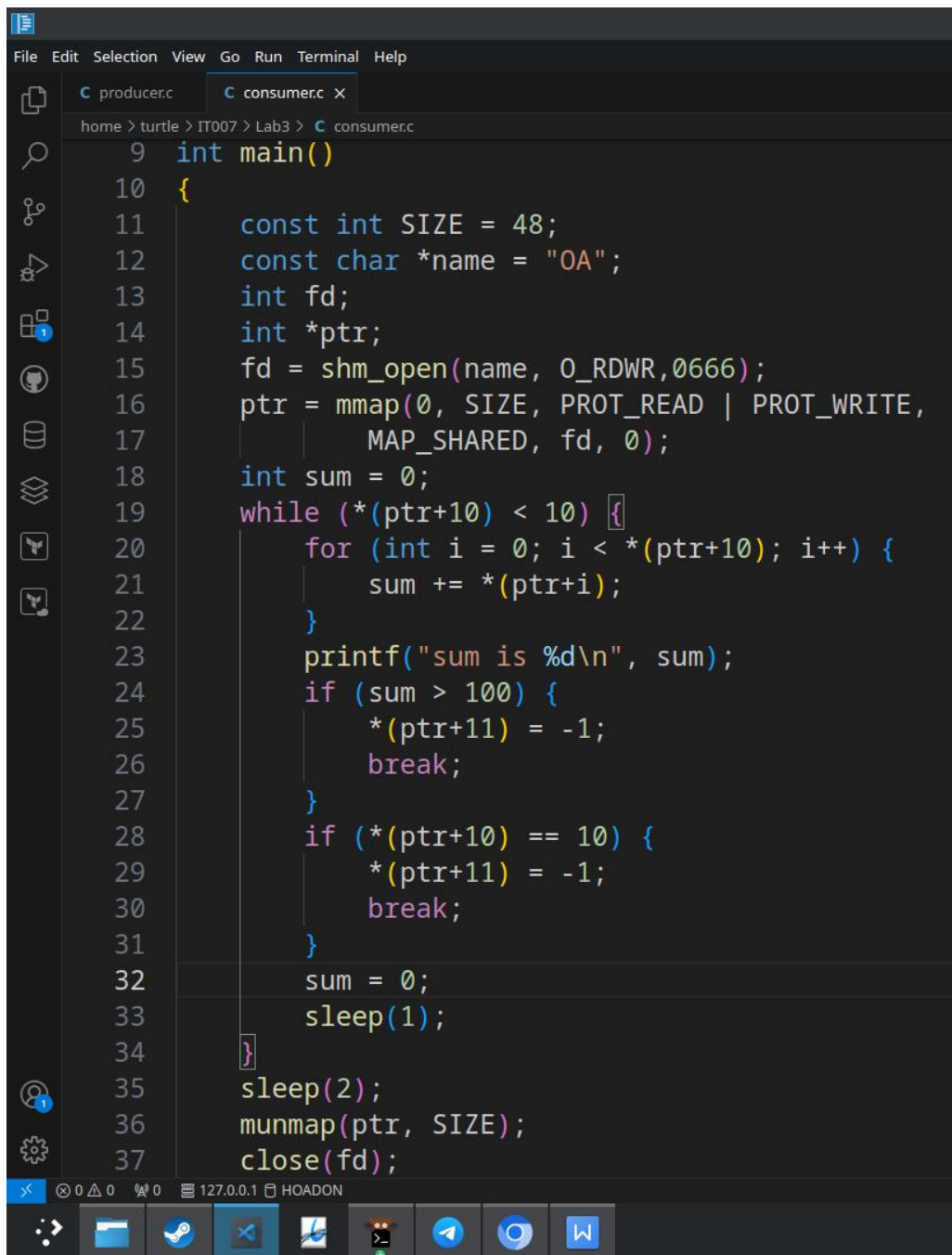
+) Khi tổng lớn hơn 100 thì cả 2 dừng lại

Code của producer:



```
9  #include <time.h>
10 int main()
11 {
12     const int SIZE = 48;
13     const char *name = "OA";
14     int fd;
15     int *ptr;
16     srand(time(NULL));
17     fd = shm_open(name, O_CREAT | O_RDWR, 0666);
18     ftruncate(fd, SIZE);
19     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE,
20              MAP_SHARED, fd, 0);
21     int count = 0;
22     while (*(ptr+11) != -1)
23     {
24         int temp = 10 + (rand() % 11);
25         printf("Randomed value: %d\n", temp);
26         if (count < 10) {
27             *(ptr+count) = temp;
28             count++;
29             *(ptr+10) = count;
30         }
31         sleep(1);
32     }
33     munmap(ptr, SIZE);
34     close(fd);
35     return 0;
36 }
```

Code của consumer:



```
9 int main()
10 {
11     const int SIZE = 48;
12     const char *name = "OA";
13     int fd;
14     int *ptr;
15     fd = shm_open(name, O_RDWR, 0666);
16     ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE,
17               MAP_SHARED, fd, 0);
18     int sum = 0;
19     while (*(ptr+10) < 10) {
20         for (int i = 0; i < *(ptr+10); i++) {
21             sum += *(ptr+i);
22         }
23         printf("sum is %d\n", sum);
24         if (sum > 100) {
25             *(ptr+11) = -1;
26             break;
27         }
28         if (*(ptr+10) == 10) {
29             *(ptr+11) = -1;
30             break;
31         }
32         sum = 0;
33         sleep(1);
34     }
35     sleep(2);
36     munmap(ptr, SIZE);
37     close(fd);
}
```

Sau khi chạy:


```
(~/IT007/Lab3)
(10:44:35 on main * *) -> ./producer
Randomed value: 16
Randomed value: 19
Randomed value: 18
Randomed value: 11
Randomed value: 19
Randomed value: 10
Randomed value: 20
(~/IT007/Lab3)
(10:44:46 on main * *) -> |
```

```
(~/IT007/Lab3)
(10:44:35 on main * *) -> ./consumer
sum is 16
sum is 35
sum is 53
sum is 64
sum is 83
sum is 93
sum is 113
(~/IT007/Lab3)
(10:44:48 on main * *) -> |
```

Cách làm: sử dụng shared memory, allocate 40 bytes đầu tiên cho các giá trị random từ 10 đến 20, 2 giá trị cuối để lưu số giá trị đang có trong mảng này và trạng thái(-1 có nghĩa là đã đầy mảng, nên dừng tiến trình)

Giải thích: tạo producer cho random các giá trị từ 10 đến 20 bằng hàm rand() và lấy modulo 10(sau đó +10), lưu các giá trị vào trong mảng, và update giá trị count tương ứng. Chương trình sẽ dừng khi giá trị sum vượt quá 100. trạng thái của *(ptr+11) là -1 thì producer sẽ dừng và không produce nữa. Hoặc là khi count = 10, trạng thái của *(ptr+11) cũng sẽ được set là -1 và sẽ không được chạy nữa.

II. Bài tập ôn tập

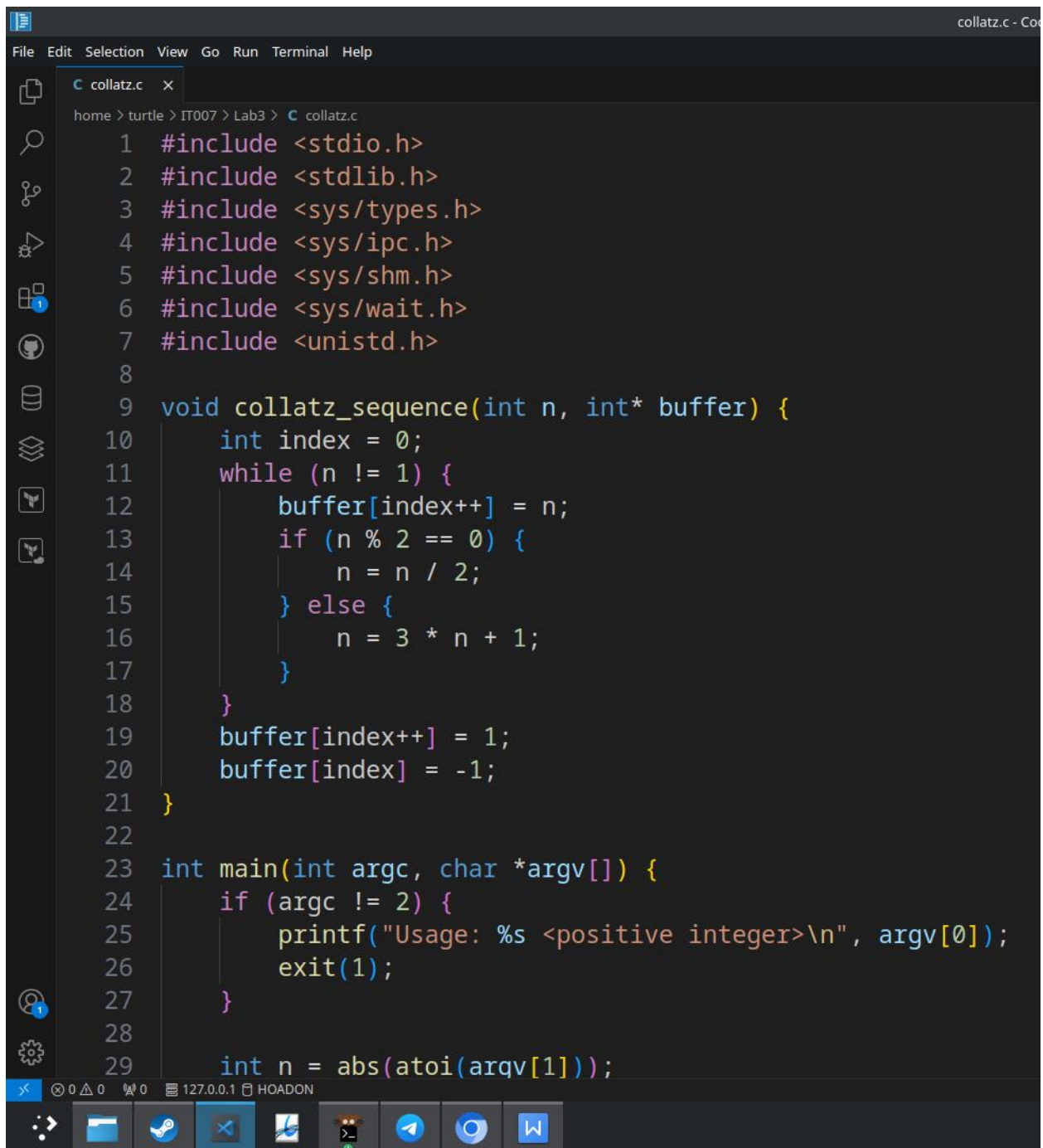
1. Phỏng đoán Collatz xem xét chuyện gì sẽ xảy ra nếu ta lấy một số nguyên dương bất kỳ và áp dụng theo thuật toán sau đây:

$$n = \begin{cases} n/2 & \text{nếu } n \text{ là số chẵn} \\ 3 * n + 1 & \text{nếu } n \text{ là số lẻ} \end{cases}$$

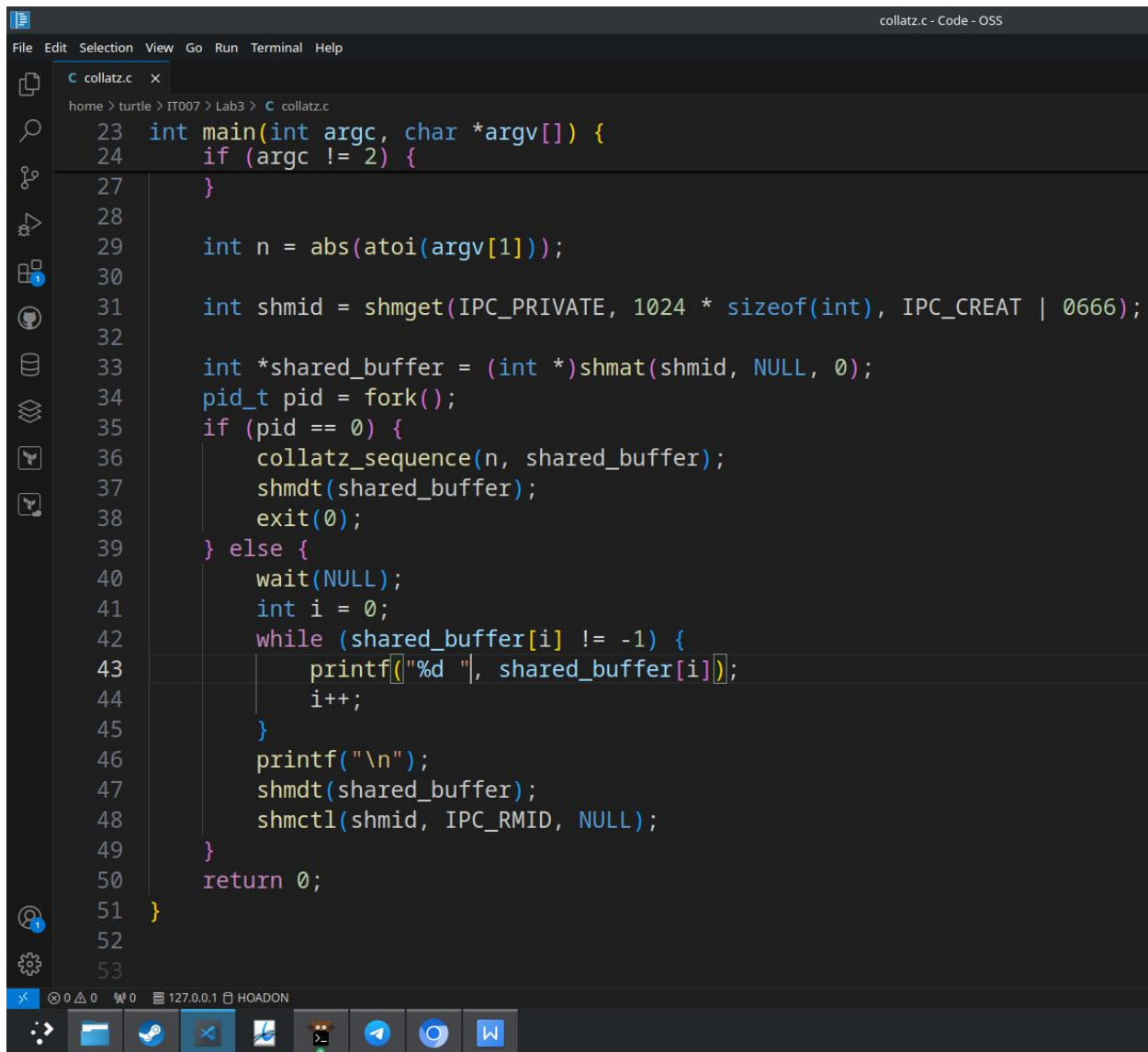
Phỏng đoán phát biểu rằng khi thuật toán này được áp dụng liên tục, tất cả số nguyên dương đều sẽ tiến đến 1. Ví dụ, với $n = 35$, ta sẽ có chuỗi kết quả như sau:

35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

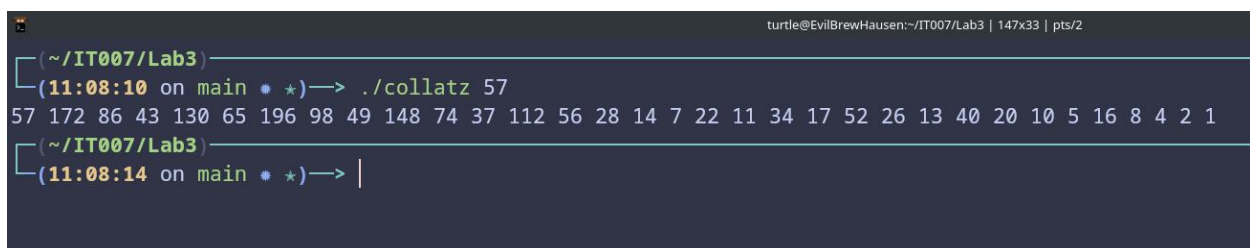
Viết chương trình C sử dụng hàm fork() để tạo ra chuỗi này trong tiến trình con. Số bắt đầu sẽ được truyền từ dòng lệnh. Ví dụ lệnh thực thi ./collatz 8 sẽ chạy thuật toán trên $n = 8$ và chuỗi kết quả sẽ ra là 8, 4, 2, 1. Khi thực hiện, tiến trình cha và tiến trình con chia sẻ một buffer, sử dụng phương pháp bộ nhớ chia sẻ, hãy tính toán chuỗi trên tiến trình con, ghi kết quả vào buffer và dùng tiến trình cha để in kết quả ra màn hình. Lưu ý, hãy nhớ thực hiện các thao tác để kiểm tra input là số nguyên dương.



```
collatz.c - Co
File Edit Selection View Go Run Terminal Help
C collatz.c x
home > turtle > IT007 > Lab3 > C collatz.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/ipc.h>
5 #include <sys/shm.h>
6 #include <sys/wait.h>
7 #include <unistd.h>
8
9 void collatz_sequence(int n, int* buffer) {
10     int index = 0;
11     while (n != 1) {
12         buffer[index++] = n;
13         if (n % 2 == 0) {
14             n = n / 2;
15         } else {
16             n = 3 * n + 1;
17         }
18     }
19     buffer[index++] = 1;
20     buffer[index] = -1;
21 }
22
23 int main(int argc, char *argv[]) {
24     if (argc != 2) {
25         printf("Usage: %s <positive integer>\n", argv[0]);
26         exit(1);
27     }
28
29     int n = abs(atoi(argv[1]));
```



```
collatz.c - Code - OSS
File Edit Selection View Go Run Terminal Help
C collatz.c x
home > turtle > IT007 > Lab3 > C collatz.c
23 int main(int argc, char *argv[]) {
24     if (argc != 2) {
27     }
28
29     int n = abs(atoi(argv[1]));
30
31     int shmid = shmget(IPC_PRIVATE, 1024 * sizeof(int), IPC_CREAT | 0666);
32
33     int *shared_buffer = (int *)shmat(shmid, NULL, 0);
34     pid_t pid = fork();
35     if (pid == 0) {
36         collatz_sequence(n, shared_buffer);
37         shmdt(shared_buffer);
38         exit(0);
39     } else {
40         wait(NULL);
41         int i = 0;
42         while (shared_buffer[i] != -1) {
43             printf("%d ", shared_buffer[i]);
44             i++;
45         }
46         printf("\n");
47         shmdt(shared_buffer);
48         shmctl(shmid, IPC_RMID, NULL);
49     }
50     return 0;
51 }
52
53
```



```
turtle@EvilBrewHausen:~/IT007/Lab3 | 147x33 | pts/2
~/IT007/Lab3
(11:08:10 on main * *) -> ./collatz 57
57 172 86 43 130 65 196 98 49 148 74 37 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
~/IT007/Lab3
(11:08:14 on main * *) -> |
```

Cách làm: sử dụng shmget để lấy một vùng nhớ chia sẻ, sau đó dùng shmat để lấy vị trí trong vùng nhớ chia sẻ mình muốn lấy và đưa nó cho con trỏ(return value của function).

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Thân Thế Tùng.

Sau đó dùng hàm fork để tạo ra 1 tiến trình con, chạy collatz sequence trên tiến trình con và lưu vào shared memory. Ở tiến trình cha ta dùng wait(NULL) để đợi tiến trình con chạy xong trước, sau đó in ra các phần tử trong shared memory. Sau đó đóng lại shared memory.

Giải thích kết quả: khi gọi collatz 57 thì số 57 sẽ được đưa cho tiến trình con và chạy hàm collatz_sequence, sau đó thì các giá trị đều được lần lượt lưu vào shared memory và tiến trình cha in ra tất cả các giá trị được tính.