Lab 03 – Group 5

Class : it007.019.1 && IT007.019.2

Date: 23/10/2023

**Members:**

**1.** Hà Huy Hoàng | 22520460

2. Nguyễn Duy Hoàng | 22520467

3. Nguyễn Hoàng Hiệp | 22520452

4. Nguyễn Hoàng Phúc | 22521129

**SUMMARY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | | **Status** | **Members** |
| Thực hành | 1 | Hoàn thành | 1,2,3,4 |
| 2 | Hoàn thành | 2 |
| 3 | Hoàn thành | 4 |
| 4 | Hoàn thành | 3 |
| Ôn tập | 1 | Hoàn thành | 1 |

**A. Bài tập thực hành**

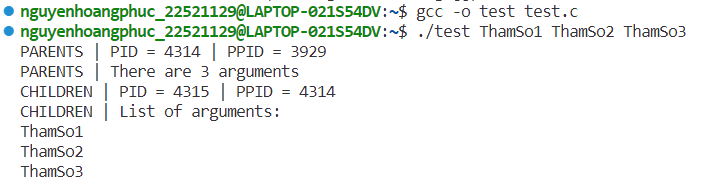
1. **Thực hiện Ví dụ 3-1, Ví dụ 3-2, Ví dụ 3-3, Ví dụ 3-4 giải thích code và kết quả nhận được?**

* Ví dụ 3-1:

*File test.c:*

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  //Sử dụng hàm fork()  #include <sys/wait.h>  #include <sys/types.h>  int main(int argc, char \*argv[])   // argc là số lượng đối số được truyền vào  {      // Tạo một tiến trình con      \_\_pid\_t pid;      pid = fork();      // Tiến trình cha      if (pid > 0)      {          printf("PARENTS | PID = %ld | PPID = %ld\n", (long)getpid(), (long)getppid()); //in ra pid và ppid của tiến trình hiện tại          if (argc > 2)              printf("PARENTS | There are %d arguments\n", argc - 1);          wait(NULL); // Đợi tiến trình con kết thúc      }      // Tiến trình con      if (pid == 0)   //thông tin về quy trình con và các đối số được in ra màn hình      {          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]); // In ra các đối số từ dòng lệnh          }      }      exit(0);   //Kết thúc tiến trình.  } |

* 1. Đây là thông tin về tiến trình cha (parent process). PID = 4314 là ID của tiến trình cha và PPID = 3929 là ID của tiến trình ông nội (grandparent process).
  2. Đây cho biết rằng bạn đã chạy chương trình với 3 đối số đầu vào, là ThamSo1, ThamSo2, và ThamSo3.
  3. Đây là thông tin về tiến trình con (child process). PID = 4315 là ID của tiến trình con và PPID = 4314 là ID của tiến trình cha.
  4. Đây là danh sách các đối số đầu vào mà bạn đã truyền vào chương trình. Trong trường hợp này, các đối số là ThamSo1, ThamSo2, và ThamSo3.



1.

2.

3.

4.

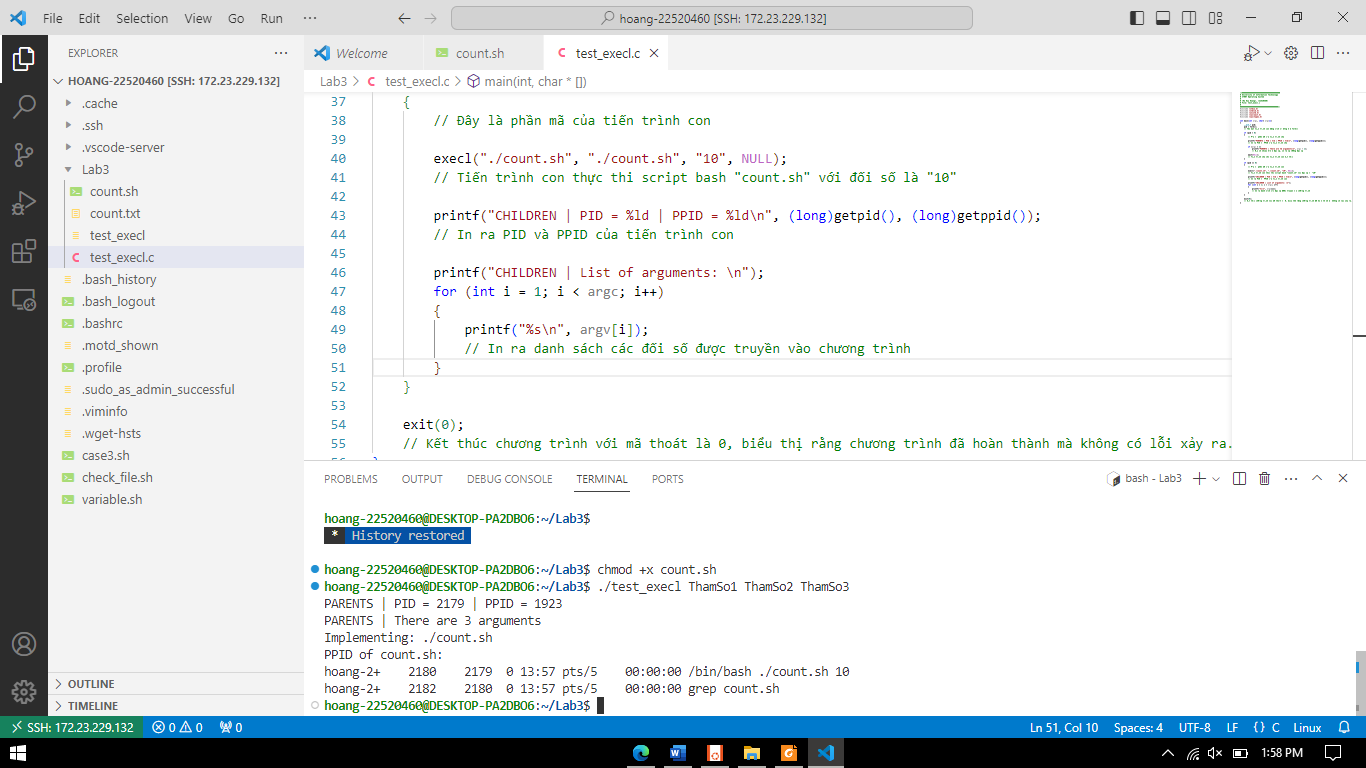
* Ví dụ 3-2:

*File count.sh*

|  |
| --- |
| #!/bin/bash  # Đây là shebang, nó chỉ định rằng script này sẽ được thực thi bằng /bin/bash  echo "Implementing: $0"  # In ra tên của script hiện tại  echo "PPID of count.sh: "  # In ra chuỗi "PPID of count.sh: "  ps -ef | grep count.sh  # Lệnh 'ps -ef' sẽ liệt kê tất cả các tiến trình đang chạy.  # Kết quả sau đó được chuyển tới lệnh 'grep count.sh',  # lọc ra những dòng có chứa chuỗi 'count.sh'  i=1  # Khởi tạo biến i với giá trị là 1  while [ $i -le $1 ]  # Vòng lặp while sẽ chạy miễn là giá trị của i nhỏ hơn hoặc bằng tham số đầu tiên được truyền vào script ($1)  do      echo $i >> count.txt      # In giá trị của i vào file count.txt. Dấu '>>' có nghĩa là nối vào cuối file nếu file đã tồn tại.      i=$((i + 1))      # Tăng giá trị của i lên 1      sleep 1      # Dừng script trong 1 giây  done  # Kết thúc vòng lặp while  exit 0  # Kết thúc script với mã thoát là 0, biểu thị rằng script đã hoàn thành mà không có lỗi xảy ra. |

*File test\_execl.c:*

|  |
| --- |
| /\*######################################  # University of Information Technology  # IT007 Operating System  #  # <Ha Huy Hoang>, <22520460>  # File: test\_execl.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();      // Tạo một tiến trình con bằng cách sử dụng hàm fork()      if (pid > 0)      {          // Đây là phần mã của tiến trình cha          printf("PARENTS | PID = %ld | PPID = %ld\n", (long)getpid(), (long)getppid());          // In ra PID và PPID của tiến trình cha          if (argc > 2)              printf("PARENTS | There are %d arguments\n", argc - 1);              // Nếu có nhiều hơn 2 đối số, in ra số lượng đối số          wait(NULL);          // Tiến trình cha chờ tiến trình con kết thúc      }      if (pid == 0)      {          // Đây là phần mã của tiến trình con          execl("./count.sh", "./count.sh", "10", NULL);          // Tiến trình con thực thi script bash "count.sh" với đối số là "10"          printf("CHILDREN | PID = %ld | PPID = %ld\n", (long)getpid(), (long)getppid());          // In ra PID và PPID của tiến trình con          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 được truyền vào chương trình          }      }      exit(0);      // Kết thúc chương trình với mã thoát là 0, biểu thị rằng chương trình đã hoàn thành mà không có lỗi xảy ra.  } |



1.

2.

3.

4.

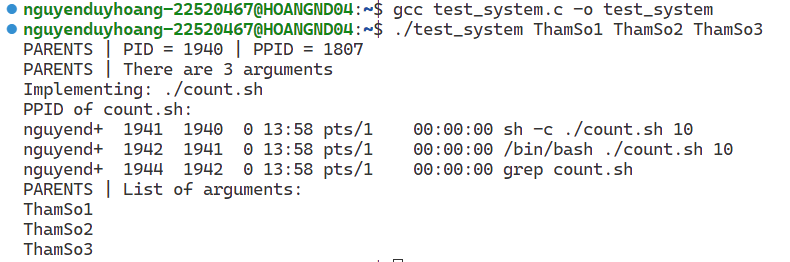
5.

6.

* 1. Đây là output từ tiến trình cha trong chương trình test\_execl.c. Nó in ra PID (Process ID) của tiến trình cha là 2179 và PPID (Parent Process ID) của nó là 1923.
  2. Đây là output từ tiến trình cha, nó in ra số lượng đối số được truyền vào chương trình test\_execl.c (không tính tên chương trình). Trong trường hợp này, bạn đã truyền vào 3 đối số: “ThamSo1”, “ThamSo2”, và “ThamSo3”.
  3. Đây là output từ script bash count.sh, nó in ra tên của script hiện tại.
  4. Tiếp tục là output từ script bash count.sh, nó in ra chuỗi "PPID of count.sh: ".
  5. Đây là output từ lệnh ps -ef | grep count.sh trong script bash count.sh. Nó in ra thông tin về tiến trình đang chạy script count.sh. Trong đó, PID của tiến trình này là 2180 và PPID của nó (là tiến trình cha đã tạo ra nó) là 2179.
  6. Đây cũng là output từ lệnh ps -ef | grep count.sh, nhưng nó liên quan đến tiến trình grep mà lệnh này đã tạo ra để lọc kết quả. PID của tiến trình grep này là 2182 và PPID của nó (là tiến trình đã tạo ra nó) là 2180.
* Ví dụ 3-3:

*File test\_system.c:*

|  |
| --- |
| /\*######################################  # University of Information Technology  # IT007 Operating System  #  # Nguyen Duy Hoang, 22520467  # File: test\_system.c  ######################################\*/  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/wait.h>  #include <sys/types.h>  int main(int argc, char\* argv[]) {      // In ra ID của tiến trình hiện tại và ID của tiến trình cha (nếu có)      printf("PARENTS | PID = %ld | PPID = %ld\n", (long)getpid(), (long)getppid());      // Nếu có nhiều hơn 2 tham số đầu vào, in ra số lượng tham số      if (argc > 2) {          printf("PARENTS | There are %d arguments\n", argc - 1);      }      // Thực thi script shell count.sh với tham số là 10 trong một tiến trình con      system("./count.sh 10");      // In ra danh sách các tham số đầu vào      printf("PARENTS | List of arguments: \n");      for (int i = 1; i < argc; i++) {          printf("%s\n", argv[i]);      }      // Kết thúc chương trình với mã lỗi 0 (tức là không có lỗi xảy ra)      exit(0);  } |



1.

2.

3.

4.

5.

Đây là thông tin về tiến trình cha (parent process). PID = 1940 là ID của tiến trình cha và PPID = 1807 là ID của tiến trình ông nội (grandparent process).

Đây cho biết rằng bạn đã chạy chương trình với 3 đối số đầu vào, là ThamSo1, ThamSo2, và ThamSo3.

Đây cho biết rằng chương trình đang thực thi script shell count.sh.

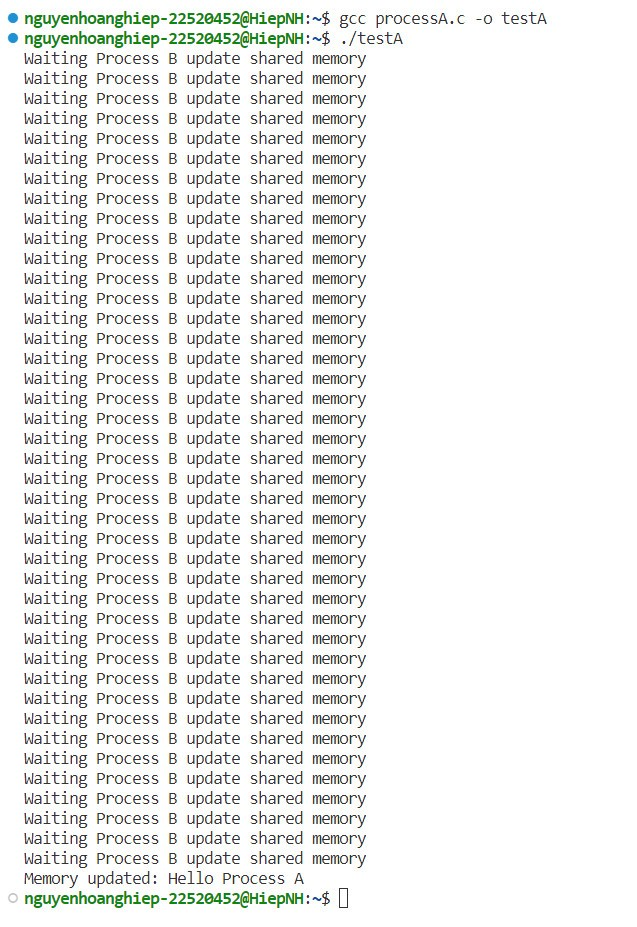
Đây là thông tin về các tiến trình liên quan đến việc thực thi script shell count.sh. Cụ thể, sh -c ./count.sh 10 là tiến trình shell được tạo ra bởi hàm system() để thực thi script, và /bin/bash ./count.sh 10 là tiến trình của script shell count.sh đang chạy.

Đây là danh sách các đối số đầu vào mà bạn đã truyền vào chương trình.

* Ví dụ 3-4:
  + 1. *Process A:*

|  |
| --- |
| #include <stdio.h>  // Thư viện chuẩn C cho I/O  #include <stdlib.h> // Thư viện chuẩn C cho các hàm như malloc, free, exit...  #include <string.h> // Thư viện chuẩn C cho các hàm xử lý chuỗi  #include <fcntl.h>  // Thư viện POSIX cho file control  #include <sys/shm.h> // Thư viện POSIX cho shared memory  #include <sys/stat.h> // Thư viện POSIX cho các hàm xử lý file status  #include <unistd.h>  // Thư viện POSIX cho các hàm như close, write, read...    #include <sys/mman.h>  // Thư viện POSIX cho memory management    int main()  {      /\* the size (in bytes) of shared memory object \*/      const int SIZE = 4096;  // Kích thước (theo byte) của đối tượng bộ nhớ chia sẻ      /\* name of the shared memory object \*/      const char \*name = "OS";  // Tên của đối tượng bộ nhớ chia sẻ      /\* shared memory file descriptor \*/      int fd;  // File descriptor cho bộ nhớ chia sẻ      /\* pointer to shared memory obect \*/      char \*ptr;  // Con trỏ đến đối tượng bộ nhớ chia sẻ      /\* create the shared memory object \*/      fd = shm\_open(name, O\_CREAT | O\_RDWR,0666);  // Tạo đối tượng bộ nhớ chia sẻ      /\* configure the size of the shared memory object \*/      ftruncate(fd, SIZE);  // Cấu hình kích thước của đối tượng bộ nhớ chia sẻ      /\* memory map the shared memory object \*/      ptr = mmap(0, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  // Ánh xạ bộ nhớ của đối tượng bộ nhớ chia sẻ      /\* write to the shared memory object \*/      strcpy(ptr, "Hello Process B");  // Ghi vào đối tượng bộ nhớ chia sẻ         /\* wait until Process B updates the shared memory segment \*/      while (strncmp(ptr, "Hello Process B", 15) == 0)  // Chờ cho đến khi Process B cập nhật phân đoạn bộ nhớ chia sẻ      {          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);  // Hủy ánh xạ phân đoạn bộ nhớ chia sẻ và đóng file descriptor      close(fd);         return 0;  } |

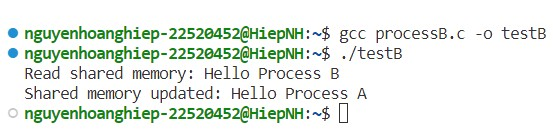
Chương trình A sẽ tạo ra một đối tượng bộ nhớ chia sẻ và ghi chuỗi "Hello Process B" vào đó, sau đó liên tục in ra thông báo "Waiting Process B update shared memory” cho đến khi chương trình B cập nhật bộ nhớ chia sẻ.



* + 1. *Process B:*

|  |
| --- |
| #include <stdio.h>  // Thư viện chuẩn C cho I/O  #include <stdlib.h> // Thư viện chuẩn C cho các hàm như malloc, free, exit...  #include <string.h> // Thư viện chuẩn C cho các hàm xử lý chuỗi  #include <fcntl.h>  // Thư viện POSIX cho file control  #include <sys/shm.h> // Thư viện POSIX cho shared memory  #include <sys/stat.h> // Thư viện POSIX cho các hàm xử lý file status  #include <unistd.h>  // Thư viện POSIX cho các hàm như close, write, read...    #include <sys/mman.h>  // Thư viện POSIX cho memory management    int main()  {      /\* the size (in bytes) of shared memory object \*/      const int SIZE = 4096;  // Kích thước (theo byte) của đối tượng bộ nhớ chia sẻ      /\* name of the shared memory object \*/      const char \*name = "OS";  // Tên của đối tượng bộ nhớ chia sẻ      /\* shared memory file descriptor \*/      int fd;  // File descriptor cho bộ nhớ chia sẻ      /\* pointer to shared memory obect \*/      char \*ptr;  // Con trỏ đến đối tượng bộ nhớ chia sẻ      /\* create the shared memory object \*/      fd = shm\_open(name, O\_RDWR,0666);  // Mở đối tượng bộ nhớ chia sẻ đã tồn tại      /\* memory map the shared memory object \*/      ptr = mmap(0, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  // Ánh xạ bộ nhớ của đối tượng bộ nhớ chia sẻ      /\* read from the shared memory object \*/      printf("Read shared memory: ");      printf("%s\n",(char \*)ptr);  // Đọc từ đối tượng bộ nhớ chia sẻ      /\* update the shared memory object \*/      strcpy(ptr, "Hello Process A");  // Cập nhật đối tượng bộ nhớ chia sẻ      printf("Shared memory updated: %s\n", ptr);      sleep(5);      // unmap the shared memory segment and close the file descriptor      munmap(ptr, SIZE);  // Hủy ánh xạ phân đoạn bộ nhớ chia sẻ và đóng file descriptor      close(fd);     // remove the shared memory segment      shm\_unlink(name);  // Gỡ bỏ đối tượng bộ nhớ chia sẻ      return 0;  } |

Chương trình B chia chạy sẽ đọc chuỗi từ bộ nhớ chia sẻ ("Hello Process B"), sau đó in ra chuỗi đã đoc và cập nhật bộ nhớ chia sẻ với chuỗi “Hello Process A” và in ra chuỗi mới.



1. **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.**

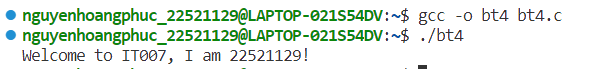
|  |
| --- |
| #include <stdlib.h>  #include <stdio.h>  #include <unistd.h>  #include <sys/time.h>  #include <sys/wait.h>  int main(int argc, char \*argv[]) {      // Kiểm tra xem có đúng một đối số được truyền vào không      if (argc != 2) {          printf("Cách sử dụng: ./time <lệnh>\n");          return 1;      }      struct timeval start, end;      // Lấy thời gian hiện tại và lưu vào biến start      gettimeofday(&start, NULL);      // Tạo một tiến trình con      pid\_t pid = fork();      // Nếu đây là tiến trình con      if (pid == 0) {          // Thực thi lệnh được truyền vào từ dòng lệnh          if (execl("/bin/sh", "/bin/sh", "-c", argv[1], NULL) == -1) {              printf("Lỗi: không thể thực thi lệnh\n");              return 1;          }      }      // Nếu đây là tiến trình cha      else if (pid > 0) {          // Chờ tiến trình con kết thúc          wait(NULL);          // Lấy thời gian hiện tại và lưu vào biến end          gettimeofday(&end, NULL);          // Tính toán thời gian thực thi bằng cách lấy thời gian kết thúc trừ đi thời gian bắt đầu          double ans = (end.tv\_sec - start.tv\_sec) + ((end.tv\_usec - start.tv\_usec)/1000000.0);          // In ra thời gian thực thi          printf("Thời gian thực thi: %.5f giây\n", ans);      }      // Nếu có lỗi xảy ra khi tạo tiến trình con      else {          printf("Lỗi: không thể tạo tiến trình con\n");          return 1;      }      return 0;  } |

****

1. **Viết một chương trình làm bốn công việc sau theo thứ tự:**

* **In ra dòng chữ: “Welcome to IT007, I am <your\_Student\_ID>!”**

|  |
| --- |
| #include <stdio.h>  int main() {      char studentID[] = "22521129";      printf("Welcome to IT007, I am %s!\n", studentID);      return 0;  } |

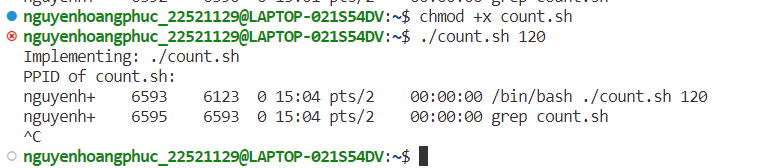


* **Thực thi file script count.sh với số lần đếm là 120**

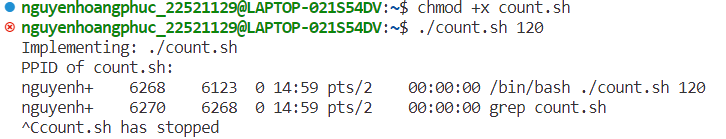
|  |
| --- |
| #!/bin/bash  echo "Implementing: $0"  echo "PPID of count.sh: "  ps -ef | grep count.sh  i=1  while [ $i -le $1 ]  do  echo $i >> count.txt  i=$((i + 1))  sleep 1  done  exit 0 |

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, hàng

Mô tả được tạo tự động

* **Trước khi count.sh đếm đến 120, bấm CTRL+C để dừng tiến trình này**
* **File count.sh in ra "count.sh has stopped" khi có tín hiệu dừng**

|  |
| --- |
| #!/bin/bash  echo "Implementing: $0"  echo "PPID of count.sh: "  ps -ef | grep count.sh  cleanup() {      echo "count.sh has stopped"      exit 1  }  trap cleanup INT  i=1  while [ $i -le $1 ]  do      echo $i >> count.txt      i=$((i + 1))      sleep 1  done |

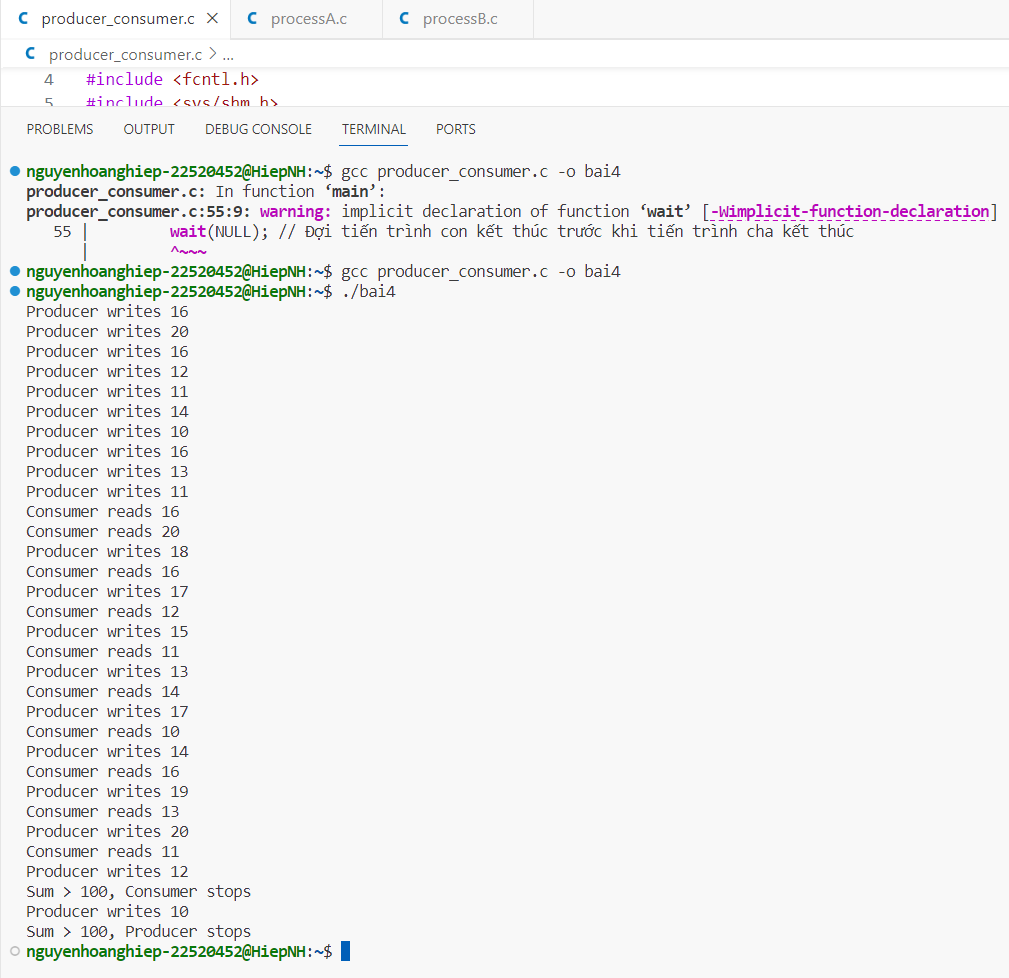


1. **Viết chương trình mô phỏng bài toán Producer - Consumer**

**như sau:**

* **Sử dụng kỹ thuật shared-memory để tạo một boundedbuffer 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**

|  |
| --- |
| #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>  #include <sys/wait.h>  int main() {      const int BUFFER\_SIZE = 10; // Định nghĩa kích thước buffer      const char \*name = "OS"; // Tên của shared memory object      // Tạo shared memory object      int shm\_fd = shm\_open(name, O\_CREAT | O\_RDWR, 0666);      // Cấu hình kích thước của shared memory object      ftruncate(shm\_fd, sizeof(int) \* (BUFFER\_SIZE + 1));      // Mapping shared memory object      int \*buffer = mmap(0, sizeof(int) \* (BUFFER\_SIZE + 1), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);      // Kiểm tra xem mmap có thành công không      if (buffer == MAP\_FAILED) {          perror("mmap");          return 1;      }      int \*sum = buffer + BUFFER\_SIZE; // Lưu tổng vào cuối buffer      \*sum = 0; // Khởi tạo tổng bằng 0      pid\_t pid = fork(); // Tạo một tiến trình con      if (pid == 0) { // Tiến trình con (Consumer)          while (\*sum <= 100) { // Chạy cho đến khi tổng lớn hơn 100              for (int i = 0; i < BUFFER\_SIZE; i++) { // Duyệt qua từng phần tử trong buffer                  if (buffer[i] != 0) { // Nếu phần tử không rỗng                      printf("Consumer reads %d\n", buffer[i]); // In ra giá trị phần tử                      \*sum += buffer[i]; // Cộng giá trị phần tử vào tổng                      buffer[i] = 0; // Đặt lại giá trị phần tử thành 0                  }              }          }          printf("Sum > 100, Consumer stops\n"); // In ra thông báo khi tổng lớn hơn 100      } else if (pid > 0) { // Tiến trình cha (Producer)          while (\*sum <= 100) { // Chạy cho đến khi tổng lớn hơn 100              for (int i = 0; i < BUFFER\_SIZE; i++) { // Duyệt qua từng phần tử trong buffer                  if (buffer[i] == 0) { // Nếu phần tử rỗng                      buffer[i] = rand() % 11 + 10; // Tạo một số ngẫu nhiên trong khoảng [10,20] và gán vào phần tử                      printf("Producer writes %d\n", buffer[i]); // In ra giá trị đã ghi vào phần tử                  }              }          }          printf("Sum > 100, Producer stops\n"); // In ra thông báo khi tổng lớn hơn 100          wait(NULL); // Đợi tiến trình con kết thúc trước khi tiến trình cha kết thúc          // Xóa shared memory object          shm\_unlink(name);      } else { // fork failed          perror("fork");          return 1;      }      return 0;  } |

****

**B. 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:**

**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**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/wait.h>  #include <sys/mman.h>  int main(int argc, char \*argv[])  {      if (argc != 2)      {          printf("Vui long nhap mot so duong");          return 1;      }      int n = atoi(argv[1]);      if (n <= 0)      {          printf("Vui long nhap mot so duong\n");          return 1;      }      pid\_t pid;      int \*buffer;      // Tạo bộ nhớ chia sẻ      buffer = mmap(NULL, sizeof(int) \* (n + 1), PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);      pid = fork();      if (pid == 0)      { // Tiến trình con          int i = 0;          while (n != 1)          {              buffer[i++] = n;              if (n % 2 == 0)                  n /= 2;              else                  n = 3 \* n + 1;          }          buffer[i] = n; // Thêm số cuối cùng (1) vào chuỗi          buffer[i + 1] = -1; // Đánh dấu kết thúc chuỗi      }      else      { // Tiến trình cha          wait(NULL); // Đợi tiến trình con kết thúc          // In chuỗi          for (int i = 0; buffer[i] != -1; i++)              printf("%d ", buffer[i]);          printf("\n");          // Dọn dẹp bộ nhớ chia sẻ          munmap(buffer, sizeof(int) \* (n + 1));      }      return 0;  } |

