|  |
| --- |
| Họ và tên: Trần Đồng Trúc Lam  Mã số sinh viên: 22520746  Lớp: IT007.O21.CNVN |

HỆ ĐIỀU HÀNH  
BÁO CÁO LAB 3

**CHECKLIST**

**3.5. BÀI TẬP THỰC HÀNH**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **BT 1** | **BT 2** | **BT 3** | **BT 4** |
| **Trình bày cách làm** |  |  |  |  |
| **Chụp hình minh chứng** |  |  |  |  |
| **Giải thích kết quả** |  |  |  |  |

**3.6. BÀI TẬP ÔN TẬP**

|  |  |
| --- | --- |
|  | **BT 1** |
| **Trình bày cách làm** |  |
| **Chụp hình minh chứng** |  |
| **Giải thích kết quả** |  |

**Tự chấm điểm:** 9.5

*\*Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp:* ***<MSSV>\_LAB3.pdf***

**2.5. BÀI TẬP THỰC HÀNH**

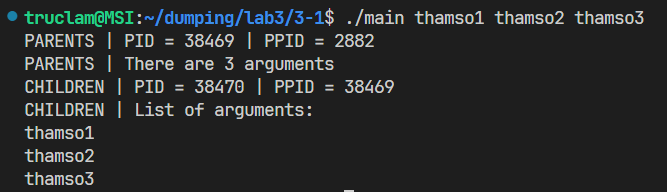
# Thực hiện Ví dụ 3-1, Ví dụ 3-2, Ví dụ 3-3, Ví dụ 3-4 giải thích code và kết quả nhận được?

Trả lời:

**Ví dụ 3-1:**

A screen shot of a computer program

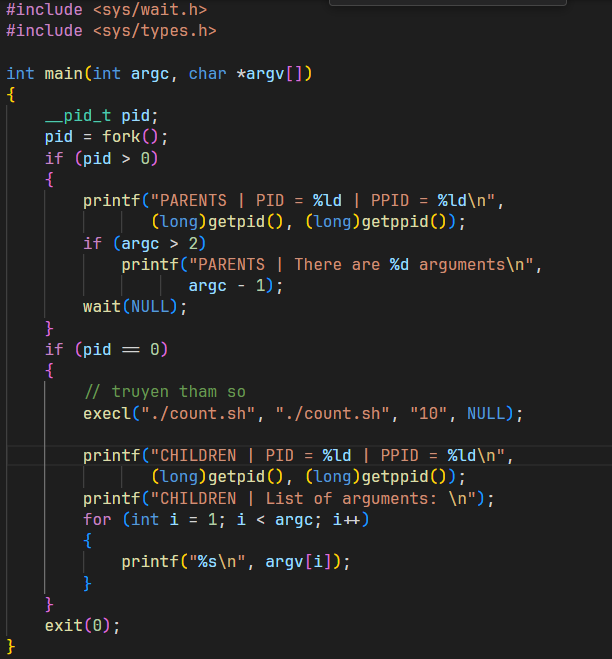
Description automatically generated



* Giải thích code:

|  |
| --- |
| * **Hàm main:** Hàm main thực hiện các bước sau:   + Gọi fork() để tạo ra một tiến trình con.   + Kiểm tra giá trị trả về của fork():     - Nếu giá trị trả về > 0: **Tiến trình cha**       * In ra thông tin về PID và PPID của tiến trình cha.       * Kiểm tra số lượng tham số dòng lệnh.       * In ra số lượng tham số dòng lệnh.       * Gọi wait() để chờ tiến trình con kết thúc.     - Nếu giá trị trả về == 0: **Tiến trình con**       * In ra thông tin về PID và PPID của tiến trình con.       * In ra danh sách các tham số dòng lệnh. * **Hàm getpid():** Hàm này lấy PID của tiến trình hiện tại. * **Hàm getppid():** Hàm này lấy PPID (PID của tiến trình cha) của tiến trình hiện tại. * **Hàm wait():** Hàm này khiến tiến trình cha chờ cho đến khi tiến trình con kết thúc. |

**Ví dụ 3.2:**



A screenshot of a computer

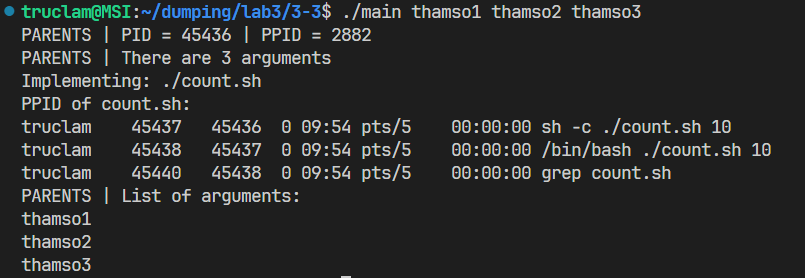
Description automatically generated

|  |
| --- |
| **Ví dụ 3-2:**   * **Chức năng:** Đếm biến i từ 1 đến $1 (được truyền vào từ dòng lệnh) và ghi kết quả vào file count.txt. Mỗi lần đếm cách nhau 1 giây. * **Giải thích:**   + **Hàm main():**     - Tạo ra một tiến trình con bằng fork().     - **Tiến trình cha:**       * In ra thông tin về PID và PPID.       * Chờ tiến trình con kết thúc.     - **Tiến trình con:**       * Sử dụng execl() để thực thi file ./count.sh với tham số 10.       * In ra thông tin về PID và PPID.       * In ra danh sách các tham số. |

**Ví dụ 3-3:**

A computer screen shot of text

Description automatically generated

A number on a black background

Description automatically generated

|  |
| --- |
| **Hàm main():**   * In ra thông tin về PID và PPID của tiến trình cha.   + Gọi system() để thực thi file ./count.sh với tham số 10.   + In ra danh sách các tham số. * **Lưu ý:**   + File count.sh cần được thực thi được.   + Thay đổi giá trị 10 nếu muốn thay đổi số lần đếm.   **So sánh với Ví dụ 3-2:**   * Đoạn code này sử dụng hàm system() để thực thi file count.sh trong một tiến trình con mới. * Ví dụ 3-2 sử dụng fork() và execl() để tạo ra một tiến trình con mới và thực thi file count.sh.   **Ưu điểm của việc sử dụng hàm** system()**:**   * Đơn giản và dễ sử dụng. * Không cần thao tác fork/exec.   **Nhược điểm:**   * Ít linh hoạt hơn so với sử dụng fork() và exec(). * Khó kiểm soát hơn đối với tiến trình con. |

**Ví dụ 3-4:**

A computer screen shot of code

Description automatically generated

A black background with white text and numbers

Description automatically generated

|  |
| --- |
| **Giải thích: Hàm** exit()**:** Dùng để kết thúc tiến trình và hoàn trả lại tài nguyên.   * Khi một tiến trình con kết thúc, một tín hiệu sẽ được gửi tới tiến trình cha và nó sẽ được đặt trong một trạng thái zombie đặc biệt để biểu diễn các tiến trình đã kết thúc. * Tiến trình cha sẽ gọi hàm wait() hoặc waitpid() để thu hồi tài nguyên của tiến trình con. |

# 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

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 gettimeifday() một lần nữa để lấy mốc thời gian sau khi thực thi lệnh shell và tính toán.**

Trả lời

A screenshot of a computer code

Description automatically generated

A screen shot of a computer

Description automatically generated

|  |
| --- |
| * **Chức năng:** Đo thời gian thực thi của một lệnh shell. * **Cách thức hoạt động:**   1. Lấy thời gian bắt đầu (start) bằng gettimeofday.   2. Tạo tiến trình con bằng fork.   3. **Tiến trình con:**      + Thực thi lệnh shell được truyền vào như tham số thứ hai (argv[1]) bằng execl.   4. **Tiến trình cha:**      + Đợi tiến trình con kết thúc bằng wait.      + Lấy thời gian kết thúc (end) bằng gettimeofday.      + Tính toán thời gian thực thi bằng cách trừ thời gian bắt đầu khỏi thời gian kết thúc.      + In ra thời gian thực thi. |

# 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 stoppped”

Trả lời...

A screen shot of a computer program

Description automatically generatedA screen shot of a computer code

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a number

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA number on a black background

Description automatically generated

|  |
| --- |
| Giải thích kết quả: Thực thi thành công script `count.sh` và in ra thông báo chào mừng. Sau đó, khi nhấn Ctrl+C, chương trình đã bắt tín hiệu `SIGINT` và gọi hàm `on\_sigint`. Trong hàm này, thông báo "Ctrl+C is pressed! Exiting..." được in ra và biến `loop\_forever` được gán giá trị 0, dừng vòng lặp trong hàm `main()`. Điều này dẫn đến kết thúc chương trình.  Trong quá trình chạy, script `count.sh` được thực thi và in ra thông báo **"Implementing: ./count.sh"** cùng với thông tin về các tiến trình chạy. `grep count.sh` là lệnh `ps` để tìm kiếm tiến trình có tên là `count.sh`. Trong trường hợp này, nó tìm thấy lệnh `count.sh` và in ra thông tin về tiến trình đó.  Kết quả của `ps -ef | grep count.sh` cho thấy:  - `85207`: Là PID của tiến trình bash đang thực thi script `count.sh`.  - `85209`: Là PID của tiến trình `grep count.sh` đang chạy để tìm kiếm tiến trình `count.sh`.  Sau khi nhấn Ctrl+C, tiến trình chính (`main`) nhận tín hiệu và thoát khỏi vòng lặp, kết thúc chương trình. |

# 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

Trả lời...

A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated

|  |
| --- |
| ***Đoạn code:***   * Trong hàm producer, tiến trình cha (Producer) tạo ra một số ngẫu nhiên trong khoảng từ 10 đến 20 và ghi chúng vào buffer. Sau đó, nó kiểm tra tổng của tất cả các số đã ghi vào buffer. Nếu tổng lớn hơn 100, tiến trình dừng lại. * Trong hàm consumer, tiến trình con (Consumer) lấy dữ liệu từ buffer, in ra màn hình và tính tổng các số đã lấy. Nếu tổng lớn hơn 100, tiến trình dừng lại. * Trong hàm main, chương trình tạo một segment shared-memory bằng cách sử dụng shmget. Sau đó, nó gắn segment này vào không gian địa chỉ của tiến trình bằng cách sử dụng shmat. Chương trình sau đó khởi tạo chỉ số của buffer và tạo một tiến trình con bằng cách sử dụng fork. * Tiến trình cha (Producer) sẽ gọi hàm producer và tiến trình con (Consumer) sẽ gọi hàm consumer. Mỗi tiến trình sẽ làm công việc của mình song song với nhau, sử dụng dữ liệu chia sẻ trong buffer. * Sau khi cả hai tiến trình đã hoàn thành công việc của mình, tiến trình cha sẽ chờ tiến trình con kết thúc bằng hàm wait, sau đó gỡ bỏ segment shared-memory bằng cách sử dụng shmdt và shmctl.   ***Giải thích kết quả:***  A screen shot of a computer  Description automatically generated   * Ban đầu, Producer tạo ra số 18 và ghi vào buffer. Tiếp theo, Consumer lấy số 18 từ buffer và in ra màn hình. Sau đó, Producer tiếp tục tạo ra số 15 và 10, nhưng Consumer không kịp lấy các số này, nên khi nó thử lấy dữ liệu từ buffer thì nhận được giá trị 0. * Tiếp theo, Producer tạo ra số 19 và 15, nhưng Consumer cũng không kịp lấy các số này nên cũng nhận được giá trị 0 khi thử lấy dữ liệu từ buffer. * Producer tiếp tục tạo ra số 14 và ghi vào buffer. Lần này, Consumer đã kịp lấy số 18 từ buffer và in ra màn hình. * Producer tiếp tục tạo ra số 20 và ghi vào buffer. Consumer không kịp lấy số này nên cũng nhận được giá trị 0 khi thử lấy dữ liệu từ buffer. * Consumer tiếp tục lấy các số còn lại từ buffer (15, 10, 19, 15, 14), sau đó tính tổng của các số này và in ra màn hình. Tổng cuối cùng là 109. * Kết quả này phản ánh quá trình sản xuất và tiêu thụ dữ liệu trong buffer giữa Producer và Consumer, và kết thúc khi tổng của các số đã tiêu thụ vượt qua giá trị 100, như đã mô tả trong chương trình.   + - Consumer không kịp lấy dữ liệu được tạo ra bởi Producer vì các hoạt động của hai tiến trình này đang xảy ra song song và không được đồng bộ hóa.     - Khi Consumer cố gắng lấy dữ liệu từ buffer, nó cần phải kiểm tra xem buffer có chứa dữ liệu hay không trước khi lấy. Tuy nhiên, trong trường hợp này, Consumer không thực hiện kiểm tra này mà ngay lập tức cố gắng lấy dữ liệu từ buffer. Khi buffer chưa được Producer ghi dữ liệu, Consumer sẽ nhận được giá trị 0 hoặc giá trị ngẫu nhiên từ buffer.     - Để giải quyết vấn đề này, chúng ta cần thêm đồng bộ hóa giữa Producer và Consumer bằng cách sử dụng semaphore hoặc các phương pháp khác để đảm bảo rằng Consumer chỉ lấy dữ liệu từ buffer khi buffer thực sự chứa dữ liệu mới được tạo ra bởi Producer. |