OPERATING SYSTEM

LAB 05 | ĐỒNG BỘ TIẾN TRÌNH/TIỂU TRÌNH

Tìm hiểu về lập trình đa tiểu trình, đồng thời sử dụng hai giải pháp đồng bộ phổ biến bao gồm Mutex Locks và Semaphore để giải quyết các bài toán đồng bộ

MỤC TIÊU

Sau LAB này, sinh viên sẽ có thể:

1. Thực hành lập trình đa tiểu trình bao gồm các thao tác: tạo tiểu trình, huỷ tiểu trình, hợp tiểu trình
2. Ứng dụng mutex locks trong việc đảm bảo loại trừ tương hỗ
3. Ứng dụng semaphore trong việc giải quyết các bài toán động bộ

NỘI DUNG

THỰC HÀNH

1. Lập trình đa tiểu trình
2. Sử dụng Mutex locks
3. Sử dụng Semaphores

LẬP TRÌNH ĐA TIỂU TRÌNH

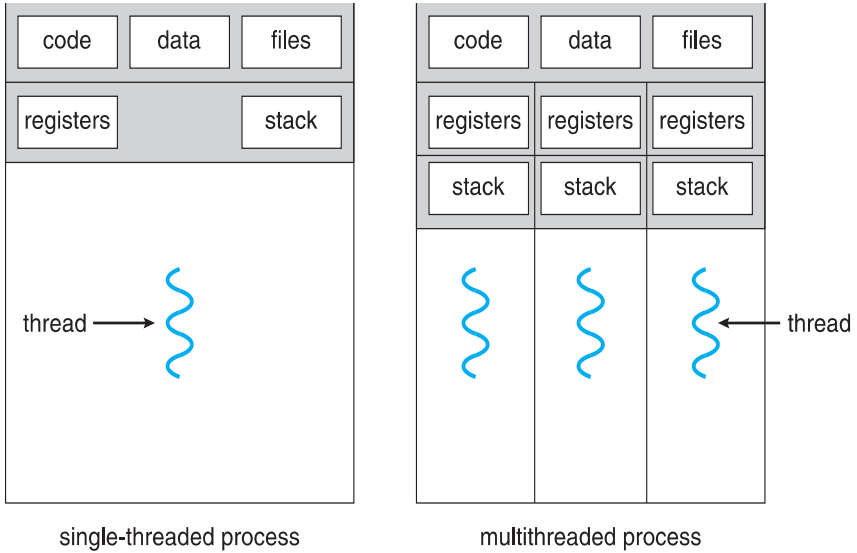
01.

1. Lập trình đa tiểu trình

- Tạo ra nhiều luồng xử lý bên trong một tiến trình → Tiến trình đa tiểu trình
- Mỗi luồng xử lý có thể phụ trách một công việc khác nhau

1. Lập trình đa tiểu trình

- **Các tiểu trình chia sẻ**:
  - Code → Code chung của tiến trình
  - **Data → Biến toàn cục**
  - Files



- Các tiểu trình chia sẻ dữ liệu với nhau rất **dễ dàng** và **nhanh chóng**

LẬP TRÌNH ĐA TIỂU TRÌNH

1.1. Tạo tiểu trình

01.

1.1. Tạo tiểu trình

Phần: **1.1. Tạo tiểu trình**

- Sử dụng thư viện pthread.h
- Bao gồm **3 bước**:
  - Định nghĩa công việc của tiểu trình thông qua con trỏ hàm
  - Khai báo tiểu trình
  - Khởi tạo tiểu trình

1.1. Tạo tiểu trình

Định nghĩa công việc của tiểu trình thông qua **con trỏ hàm**

```
void *job_name(void *message)
{
    while(1){
        // Thread's job
    }
}
```

- Nếu tiểu trình chạy liên tục thì cần đặt trong vòng lặp while(1)

1.1. Tạo tiểu trình

Khai báo tiểu trình

```
pthread_t thread_name;
```

Khởi tạo tiểu trình

```
int pthread_create
(
      pthread_t * thread,            pthread_attr_t *
attr,
      void * (*start_routine)(void *),
      void * arg
);
```

Ví dụ:

```
pthread_create
(
      &thread_name,
      NULL,
      &thread_print,
      NULL
)
```

LẬP TRÌNH ĐA TIỂU TRÌNH

1.2. Huỷ tiểu trình

01.

1.2. Huỷ tiến trình

- Để dừng một pthread có thể sử dụng hàm **pthread_exit()**
  - Gọi ngoài hàm main(): dừng pthread gọi hàm này;
  - Gọi trong main(): đợi các pthread trong nó dừng rồi nó mới dừng.

Phần: **1.2. Huỷ tiểu trình**

LẬP TRÌNH ĐA TIỂU TRÌNH

1.3. Hợp/gỡ tiểu trình

01.

1.3. Hợp/gỡ tiểu trình

- Để kết hợp các pthread, có thể sử dụng hàm **pthread_join(threadid, status)**
- pthread_join() sẽ ngưng pthread đang gọi tới khi threadid kết thúc.
- Khi pthread kết thúc, pthread_join() sẽ trả về giá trị 0.

Phần: **1.3. Hợp/gỡ tiểu trình**

LẬP TRÌNH ĐA TIỂU TRÌNH

1.4. Truyền tham số cho tiểu trình

01.

1.4. Truyền tham số cho tiểu trình

- Đối số cuối cùng của hàm pthread_create() là một con trỏ đối số cho thủ tục mà tiểu trình được tạo ra sẽ thực thi.

```
long tID = 0;
pthread_create(
        &threads[tID],
        NULL,
        &thread_print,
        (void *)tID
);
```

```
void *thread_print(void *threadid)
{
        long tid;
        tid = (long)threadid;
        printf("Hello IT007! I'm Thread #%ld!\n",
        tid);
        pthread_exit(NULL);
}
```

1.4. Truyền tham số cho tiểu trình

- Để truyền các đối số phức tạp, ta có thể sử dụng struct.
  - Tạo struct tham số bao gồm các trường dữ liệu muốn gửi
  - Ép kiểu struct thành void* để gửi cho tiểu trình
  - Khi tiểu trình nhận được, ép kiểu void* thành struct và lấy dữ liệu

Phần: **1.4. Truyền tham số cho tiểu trình**

MUTEX LOCKS

Đọc và thực thi *Phần 2: Mutex Locks*

02.

SEMAPHORES

Đọc và thực thi *Phần 3: Semaphores*

03.

Bài tập

04.

4. Bài tập

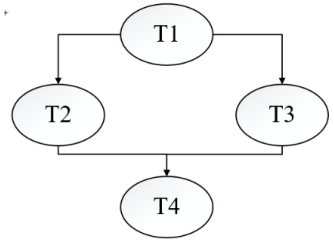
Bài tập: 01

Hiện thực lại bài toán bán hàng với điều kiện

products <= sells <= products + <4 số cuối MSSV>

Bài tập: 02

Cho 4 tiểu trình T1, T2, T3, T4, hãy thực hiện đồng bộ 4 tiểu trình thực thi theo đúng thư tự trong hình bên, biết rằng công việc của mỗi tiểu trình là in ra tên của tiểu trình.



4. Bài tập

Bài tập: 03

Cho một mảng a được khai báo như một mảng số nguyên có thể chứa n phần tử, a được khai báo như một biến toàn cục. Viết chương trình bao gồm 2 thread chạy song song:

- Một thread làm nhiệm vụ sinh ra một số nguyên ngẫu nhiên sau đó bỏ vào a. Sau đó đếm và xuất ra số phần tử của a có được ngay sau khi thêm vào.

4. Bài tập

Bài tập: 03 (tt)

- Thread còn lại lấy ra một phần tử trong a (phần tử bất kỳ, phụ thuộc vào người lập trình). Sau đó đếm và xuất ra số phần tử của a có được ngay sau khi lấy ra, nếu không có phần tử nào trong a thì xuất ra màn hình “Nothing in array a”.

Chạy thử và tìm ra lỗi khi chạy chương trình trên khi chưa được đồng bộ.

Thực hiện đồng bộ hóa với semaphore và mutex.

4. Bài tập

Bonus:

Biến ans được tính từ các biến x1, x2, x3, x4, x5, x6 như sau:

w = x1 * x2; (a)

v = x3 * x4; (b)

y = v * x5; (c)

z = v * x6; (d)

y = w * y; (e)

z = w * z; (f)

ans = y + z; (g)

4. Bài tập

Bonus (tt):

Giả sử các lệnh từ (a) → (g) nằm trên các thread chạy song song với nhau. Hãy lập trình mô phỏng và đồng bộ trên C trong hệ điều hành Linux theo thứ tự sau:

- (c), (d) chỉ được thực hiện sau khi v được tính
- (e) chỉ được thực hiện sau khi w và y được tính
- (g) chỉ được thực hiện sau khi y và z được tính