ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO THỰC HÀNH HỆ ĐIỀU HÀNH

| CHỦ ĐỀ |

THREAD SEMAPHORE

| GIẢNG VIÊN HƯỚNG DẪN |

Cô Vũ Thị Mỹ Hằng

Thầy Lê Quốc Hòa

| SINH VIÊN THỰC HIỆN |

21120302 – Huỳnh Trí Nhân

Thành phố Hồ Chí Minh – 2023

MỤC LỤC

1 ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

1.1 Đề tài

Viết chương trình tạo ra 2 tiểu trình con. Tiểu trình 1 ghi vào file text.txt với nội dung MSSV-Họ Tên, tiểu trình 2 đọc nội dung file text.txt và xuất ra màn hình. Sử dụng Semaphore để đảm bảo tại một thời điểm chỉ có duy nhất một tiểu trình đọc hoặc ghi file và việc ghi file phải được thực hiện trước đọc file.

1.2 Đánh giá mức độ hoàn thành

Yêu cầu	Tự đánh giá
Tạo 2 tiểu trình.	100%
Tiến trình thực hiện ghi file.	100%
Tiến trình thực hiện đọc file.	100%
Tại một thời điểm chỉ có một tiến trình thực hiện.	100%
Tiến trình ghi file phải xảy ra trước tiến trình đọc file.	100%
Thực hiện chương trình trên GCC của hệ điều hành Linux	100%

2 HƯỚNG GIẢI QUYẾT PHẦN LẬP TRÌNH

2.1 Thuật toán đề xuất

```
Semaphore sem = 0, mutex = 1;

Write_file{
        down(mutex);

        [Critical Section]

        up(mutex);
        up(sem);
}

Read_file{
        down(sem);
        down(mutex);

        [Critical Section]

        up(mutex);
        up(sem);
}
```

- Ban đầu biến **mutex = 1**, bất cứ tiến trình nào muốn vào miền găng thì phải down được biến mutex. Nếu không down được biến mutex, thì tức là đang có một tiến trình khác đang sử dụng để đi vào thực thi miền gằng, do đó chương trình không down được sẽ bị block cho đến khi nào tiến trình khác trả lại biến mutex về giá trị 1.
- Ban đầu biến **sem = 0**, tức là chưa một tiến trình viết file nào xảy ra. Do đó ở tiến trình readfile không thể nào down được biến sem. Biến Sem chỉ được tăng lên thành số dương khi và chỉ khi thực hiện xong tiến trình ghi file nào đó. Từ đó tiến trình đọc file mới có thể thực thi.
- Biến mutex chỉ bao bọc miền găng ở mỗi tiến trình và em thiết kế tăng và giảm sem ở bên ngoài phạm vi hoạt động của biến mutex. Để đảm bảo nếu tiến trình đọc file vô trước nếu không down được biến sem thì cũng không giữ luên biến mutex (trường hợp down(mutex) trước down(sem)) sẽ gây ra tình trạng deadlock hoặc lỗi.

```c
sem_t sem, mutex;
char *file_name = "text.txt";
char *content = "21120302-HuynhTriNhan";

typedef struct
{
    char *filename;
    char *content;
} thread_args;
```

Hình 1 Các biến cục bộ và cấu trúc tham số đầu vào

```c
void *write_file(void *args)
{
    printf("Writing thread...\n");
    thread_args *a = (thread_args *)args;
    sem_wait(&mutex);
    printf("\nThe writing thread is keeping mutex semaphore\n");
    // Test case: Xem xet neu tien trinh doc vao thi co chay duoc khong
    usleep(2000);
    FILE *f = fopen(a->filename, "w");
    fprintf(f, "%s", a->content);
    fclose(f);

    sem_post(&mutex);
    printf("\nThe writing thread gave back mutex semaphore\n");
    // Test case: Da tra mutex nhung chua tang sem len thi co chay duoc khong
    usleep(2000);
    sem_post(&sem);
    printf("\nWriting done! 'sem' variable semaphore increased\n");
    return NULL;
}
```

Hình 2 ĐInh nghĩa tiểu trình Viết vào file

```c
void *read_file(void *args)
{
    printf("Reading thread...\n");
    thread_args *a = (thread_args *)args;
    usleep(2000);
    while (sem_trywait(&sem) != 0)
    {
        printf("Thread reading is waiting for thread writing ...\n");
    }
    printf("\nThe reading thread got a signal that The writing thread completed \n");

    sem_wait(&mutex);
    printf("\nThe reading thread is keeping mutex semaphore\n");
    FILE *f = fopen(a->filename, "r");
    char buffer[256];
    fscanf(f, "%s", buffer);
    printf("\nNoi dung doc duoc la: %s\n", buffer);
    fclose(f);

    sem_post(&mutex);
    printf("\nThe reading thread gave back mutex semaphore\n");
    sem_post(&sem);
    printf("\nReading done! 'sem' variable semaphore increased\n");
    return NULL;
}
```

Hình 3 Định nghĩa tiểu trình Đọc từ file

- Ở hàm read_file, trong vòng while thay vì dùng sem_wait(&sem) thì em dùng sem_trywait(&sem) để có gắng xem thử có thể down được biến sem hay không.

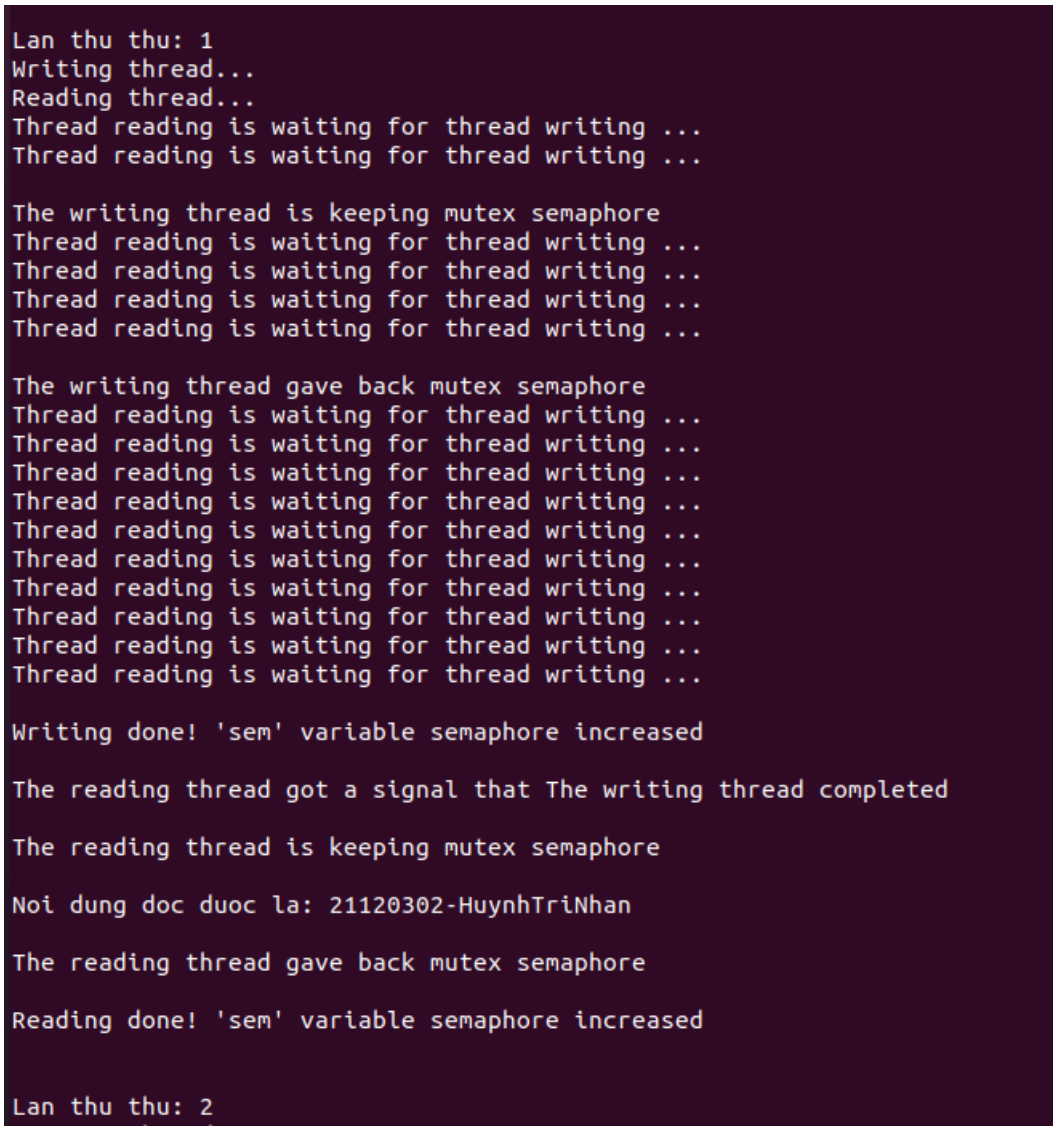
- Ở những gia đoạn luôn thông báo tiến độ chạy chương trình.

```c
int main()
{
    for (int i = 0; i < 3; i++)
    {
        printf("\n\nLan thu thu: %d\n", i + 1);
        sem_init(&mutex, 0, 1);
        sem_init(&sem, 0, 0);
        pthread_t pth_write, pth_read;

        thread_args args = {file_name, content};

        pthread_create(&pth_write, NULL, write_file, &args);
        pthread_create(&pth_read, NULL, read_file, &args);

        pthread_join(pth_write, NULL);
        pthread_join(pth_read, NULL);

        sem_destroy(&mutex);
        sem_destroy(&sem);
    }

    return 0;
}
```

Hình 4 Hàm main của chương trình

- Ở hàm main, em tạo 3 vòng lặp để xem các trường hợp có thể xảy ra và có sai sót gì không.

2.2 Kết quả chạy chương trình



Hình 5 Kết quả chương trình

Đây là kết quả vòng lặp đầu tiên của chương trình

- **Bước 1:** Ban đầu tiểu trình **write vào trước**, sau đó tiểu trình **read cũng vào theo** sau nhưng không thể thực hiện và in ra dòng "Thread reading is waiting for thread writing ..." do không thể down được biến sem **(sem =0)**
- **Bước 2:** Sau đó **tiểu trình write down biến mutex** để đi vào miền găng (điều này có nghĩa nếu có tiến trình ghi file khác cùng thực hiện thì sẽ bị blocked) **mutex = 0**.
- **Bước 3:** Sau khi thực hiện xong quá trình ghi file ở miền găng thì thông báo "The writing thread is keeping mutex semaphore" có nghĩa là tiến trình ghi file đã xong và trả về biến mutex

về giá trị 1(**mutex = 1**). **Nhưng tiểu trình đọc vẫn chưa thực hiện được** do chưa tăng biến sem (**sem = 0**)(điều này giải thích cho tại sao khi trả về mutex mà tiểu trình read vẫn bị blocked)

- **Bước 4:** Sau khi được thông báo "**Writing done! 'sem' variable semaphore increased" (sem = 1**) thì có thể down được biến sem ở tiểu trình read (**sem =0**)nên có thông báo **"The reading thread got a signal that The writing thread completed"** và tiểu trình read được diễn ra bình thường như: **Giảm mutex đi vào miền găng -> In kết quả -> trả về mutex -> trả về biến sem (sem = 1, mutex = 1).** Biến sem dược trả về lại giá trị dương (**sem = 1 > 0**) do biến sem semaphore chỉ đảm bảo cho tiểu trình write diễn ra trước tiểu trình read (Giá trị dương biểu thị cho đã có tiểu trình write thực hiện và có thể down được ở tiểu trình đọc nhưng phải trả lại sau khi kết thúc tiểu trình).

Tương tự ở những lần chạy tiếp theo vẫn nhận được kết quả tương tự.

```
Lan thu thu: 2
Writing thread...
Reading thread...
Thread reading is waiting for thread writing ...
Thread reading is waiting for thread writing ...
Thread reading is waiting for thread writing ...
Thread reading is waiting for thread writing ...

The writing thread is keeping mutex semaphore
Thread reading is waiting for thread writing ...
Thread reading is waiting for thread writing ...
Thread reading is waiting for thread writing ...
Thread reading is waiting for thread writing ...

The writing thread gave back mutex semaphore
Thread reading is waiting for thread writing ...
Thread reading is waiting for thread writing ...
Thread reading is waiting for thread writing ...
Thread reading is waiting for thread writing ...

Writing done! 'sem' variable semaphore increased

The reading thread got a signal that The writing thread completed

The reading thread is keeping mutex semaphore

Noi dung doc duoc la: 21120302-HuynhTriNhan

The reading thread gave back mutex semaphore

Reading done! 'sem' variable semaphore increased
```

PHỤ LỤC

Thư mục 21120302

- File **21120302.c** : chứa mã nguồn chương trình em đã cài đặt.
- File **21120302.pdf** chứa báo cáo bài thực hành.
- Kết quả chạy chương trình chạy trên **hệ điều hành Linux của Ubuntu bản 20.04.6.**
- Sử dụng editor VSCode có sẵn trên Ubuntu.

TÀI LIỆU THAM KHẢO

- Giáo trình Hệ Điều Hành HCMUS
- Tài liệu thực hành lớp Hệ Điều Hành 21_4