Lab 5: Lập trình song song trên Linux

Thực hành

Exercise 1:

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define START 1
#define END 10000000
#define THREADS 4

int total = 0;
pthread_mutex_t lock;

void* count_odds(void* arg) {
    int id = *(int*)arg;
    int range = (END - START + 1) / THREADS;
    int local_start = START + id * range;
    int local_end = (id == THREADS - 1) ? END : local_start + range - 1;

    int local_count = 0;
    for (int i = local_start; i <= local_end; ++i) {
        if (i % 2 != 0)
            local_count++;
    }

    pthread_mutex_lock(&lock);
    total += local_count;
    pthread_mutex_unlock(&lock);

    return NULL;
}

int main() {
    pthread_t tid[THREADS];
    int ids[THREADS];
    pthread_mutex_init(&lock, NULL);

    for (int i = 0; i < THREADS; ++i) {
        ids[i] = i;
        pthread_create(&tid[i], NULL, count_odds, &ids[i]);
    }

    for (int i = 0; i < THREADS; ++i) {
        pthread_join(tid[i], NULL);
    }

    printf("Total odd numbers = %d\n", total);

    pthread_mutex_destroy(&lock);
    return 0;
}
```

Exercise 2:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define THREADS 4
#define RANGE 256

int* data;
int data_size = 1000000;

int histogram[RANGE] = {0};
pthread_mutex_t lock;

void* compute_histogram(void* arg) {
    int id = *(int*)arg;
    int chunk = data_size / THREADS;
    int start = id * chunk;
    int end;
    if (id == THREADS - 1) {
        end = data_size;
    } else {
        end = start + chunk;
    }

    int local[RANGE] = {0};
    for (int i = start; i < end; i++) {
        int v = data[i];
        local[v]++;
    }

    pthread_mutex_lock(&lock);
    for (int i = 0; i < RANGE; i++) {
        histogram[i] += local[i];
    }
    pthread_mutex_unlock(&lock);

    return NULL;
}
```

```
int main() {
    data = malloc(sizeof(int) * data_size);
    for (int i = 0; i < data_size; i++) {
        data[i] = rand() % RANGE;
    }

    pthread_t t[THREADS];
    int id[THREADS];
    pthread_mutex_init(&lock, NULL);

    for (int i = 0; i < THREADS; i++) {
        id[i] = i;
        pthread_create(&t[i], NULL, compute_histogram, &id[i]);
    }

    for (int i = 0; i < THREADS; i++) {
        pthread_join(t[i], NULL);
    }

    for (int i = 0; i < RANGE; i++) {
        if (histogram[i] > 0) {
            printf("%d %d\n", i, histogram[i]);
        }
    }

    pthread_mutex_destroy(&lock);
    free(data);
    return 0;
}
```

Exercise 3:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>

#define THREADS 4
#define SIZE 100000
#define TARGET "hello"

char** array;
int found = 0;
int found_index = -1;

pthread_mutex_t lock;

void* search(void* arg) {
    int id = *(int*)arg;
    int chunk = SIZE / THREADS;
    int start = id * chunk;
    int end;
    if (id == THREADS - 1) {
        end = SIZE;
    } else {
        end = start + chunk;
    }

    for (int i = start; i < end; i++) {
        pthread_mutex_lock(&lock);
        if (found) {
            pthread_mutex_unlock(&lock);
            break;
        }
        pthread_mutex_unlock(&lock);

        if (strcmp(array[i], TARGET) == 0) {
            pthread_mutex_lock(&lock);
            if (!found) {
                found = 1;
                found_index = i;
            }
            pthread_mutex_unlock(&lock);
            break;
        }
    }

    return NULL;
}

int main() {
    array = malloc(sizeof(char*) * SIZE);
    for (int i = 0; i < SIZE; i++) {
        array[i] = malloc(6);
        strcpy(array[i], "test");
    }
    strcpy(array[SIZE - 123], TARGET);

    pthread_t t[THREADS];
    int id[THREADS];
    pthread_mutex_init(&lock, NULL);

    for (int i = 0; i < THREADS; i++) {
        id[i] = i;
        pthread_create(&t[i], NULL, search, &id[i]);
    }

    for (int i = 0; i < THREADS; i++) {
        pthread_join(t[i], NULL);
    }
    if (found) {
        printf("Found at %d\n", found_index);
    } else {
        printf("Not found\n");
    }

    for (int i = 0; i < SIZE; i++) {
        free(array[i]);
    }
    free(array);
    pthread_mutex_destroy(&lock);
    return 0;
}
```

Exercise 4:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define N 4

int A[N][N], B[N][N], C[N][N];

void* multiply_row(void* arg) {
    int row = *(int*)arg;
    for (int j = 0; j < N; j++) {
        C[row][j] = 0;
        for (int k = 0; k < N; k++) {
            C[row][j] += A[row][k] * B[k][j];
        }
    }
    return NULL;
}

int main() {
    pthread_t t[N];
    int id[N];

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            A[i][j] = i + j;
            B[i][j] = i - j;
        }
    }

    for (int i = 0; i < N; i++) {
        id[i] = i;
        pthread_create(&t[i], NULL, multiply_row, &id[i]);
    }

    for (int i = 0; i < N; i++) {
        pthread_join(t[i], NULL);
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%4d ", C[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Exercise 5:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define THRESHOLD 1000

typedef struct {
    int* arr;
    int left;
    int right;
} Args;

void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int partition(int* arr, int left, int right) {
    int pivot = arr[right];
    int i = left - 1;
    for (int j = left; j < right; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i+1], &arr[right]);
    return i + 1;
}

void* quicksort(void* a) {
    Args* args = (Args*)a;
    int left = args->left;
    int right = args->right;
    int* arr = args->arr;

    if (left >= right) return NULL;

    if (right - left + 1 <= THRESHOLD) {
        for (int i = left; i < right; i++) {
            for (int j = i + 1; j <= right; j++) {
                if (arr[i] > arr[j]) {
                    swap(&arr[i], &arr[j]);
                }
            }
        }
        return NULL;
    }

    int p = partition(arr, left, right);

    Args* a1 = malloc(sizeof(Args));
    Args* a2 = malloc(sizeof(Args));

    a1->arr = arr; a1->left = left; a1->right = p - 1;
    a2->arr = arr; a2->left = p + 1; a2->right = right;

    pthread_t t1, t2;

    pthread_create(&t1, NULL, quicksort, a1);
    pthread_create(&t2, NULL, quicksort, a2);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    free(a1);
    free(a2);

    return NULL;
}

int main() {
    int size = 10000;
    int* arr = malloc(sizeof(int) * size);
    for (int i = 0; i < size; i++) {
        arr[i] = rand() % 100000;
    }

    Args arg = {arr, 0, size - 1};
    quicksort(&arg);
    for (int i = 0; i < 20; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    free(arr);
    return 0;
}
```