《操作系统原理》实验报告

实验名称	多线程编程	实验序号	3
实验日期	2023/4/11	实验人	盖乐

一、实验题目

编写一个多线程排序程序，其工作方式如下：一个整数列表被分成两个大小相等的较小列表。两个单独的线程使用合适的排序算法对每个子列表进行排序。这两个子列表随后由第三个线程 a merging thread 将两个子列表合并为一个排序列表。

二、相关原理与知识

多线程编程是一种在计算机程序中同时执行多个线程（或称为子任务、子进程）的编程技术。多线程编程可以充分利用多核处理器的优势，提高程序的执行效率，使程序能够同时处理多个任务，从而改善系统的响应性和并发性。

1.线程：线程是程序的基本执行单位，是进程中的一个独立的控制流。一个进程可以包含多个线程，它们共享进程的内存空间和资源。线程之间可以并发执行，相较于多个进程之间的通信和同步开销较小。

2.并发与并行：并发是指多个任务在同一时间段内执行，但不一定同时执行；而并行是指多个任务在同一时刻同时执行。多线程编程可以实现并发和并行的效果，提高系统的整体性能。

pthread（POSIX threads）是一套用于多线程编程的标准接口，定义了一组用于创建、管理和同步线程的函数和数据类型。本次使用到的 pthread_create 和 pthread_join 是 POSIX 线程库中常用的两个函数，用于创建线程和等待线程结束。

1.pthread_create 函数用于创建一个新的线程。它接受四个参数：

- thread: 用于存储新线程标识符的指针。
- attr: 用于定义线程属性的指针。如果为 NULL，则使用默认属性。
- start_routine: 新线程要执行的函数指针。
- arg: 要传递给新线程的参数。

调用 pthread_create 成功后，会创建一个新的线程，并将其标识符存储在 thread 指针所指向的位置。

2.pthread_join 函数用于等待一个线程结束。它接受两个参数：

- thread: 要等待的线程标识符。
- retval: 用于存储被等待线程返回值的指针。

调用 pthread_join 会阻塞当前线程，直到指定的线程结束。如果被等待的线程已经结束，那么调用 pthread_join 将立即返回，并将被等待线程的返回值存储在 retval 指针所指向的位置。

三、实验过程

编程思路如下：

首先将原始整数列表分成两个大小相等的较小列表。之后创建两个 sorting threads，每个线程处理一个子列表并让 sorting threads 并行执行归并排序操作。然后创建一个 merging thread，然后将两个排好序的子列表传递给它。merging thread 将两个子列表合并为一个排序列表，并返回结果。

1. 整数列表的初始化

首先定义了一个长度为 SIZE 的整型数组 list 和一个长度为 SIZE 的结果数组 result。其中 list 存放待排序的数组，result 用于存放归并排序后的结果。

```
#define SIZE 10

int list[SIZE] = {9, 4, 3, 10, 5, 8, 7, 2, 1, 6};
int result[SIZE];
```

若未给出指定数量的整数将自动初始化为 0。

2. 函数 merge 用于合并两个已经排好序的子数组，即将 list 中下标从 low 到 mid 和下标从 mid+1 到 high 之间的元素进行合并,并将结果存储在 result 数组中。

```
//归并排序
void merge(int low, int mid, int high) {
    int l1, l2, i;
    for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {
        if(list[l1] <= list[l2])
            result[i] = list[l1++];
        else
            result[i] = list[l2++];
    }

    while(l1 <= mid)
        result[i++] = list[l1++];

    while(l2 <= high)
        result[i++] = list[l2++];

    for(i = low; i <= high; i++)
        list[i] = result[i];
}
```

3. 函数 merge_sort 用于递归地执行归并排序。它接收一个包含两个整数的指针参数 arg，表示要排序的子数组的下标范围。在具体实现中，程序将原始数组的起始索引和结束索引作为参数传递给函数 merge_sort。函数将这些值解析为左右子数组的起始和结束索引，并计算出中间索引。然后它使用 malloc 动态分配内存来创建两个包含左右子数组的新参数数组。接下来，函数使用 pthread_create 函数创建两个新线程，将左右子数组的索引作为参数传递给它们。这些线程将同时运行，处理各自的子数组。然后使用 pthread_join 函数等待左右线程结束。最后，函数调用 merge 函数将两个已经排序的子数组合并为一个有序的数组。

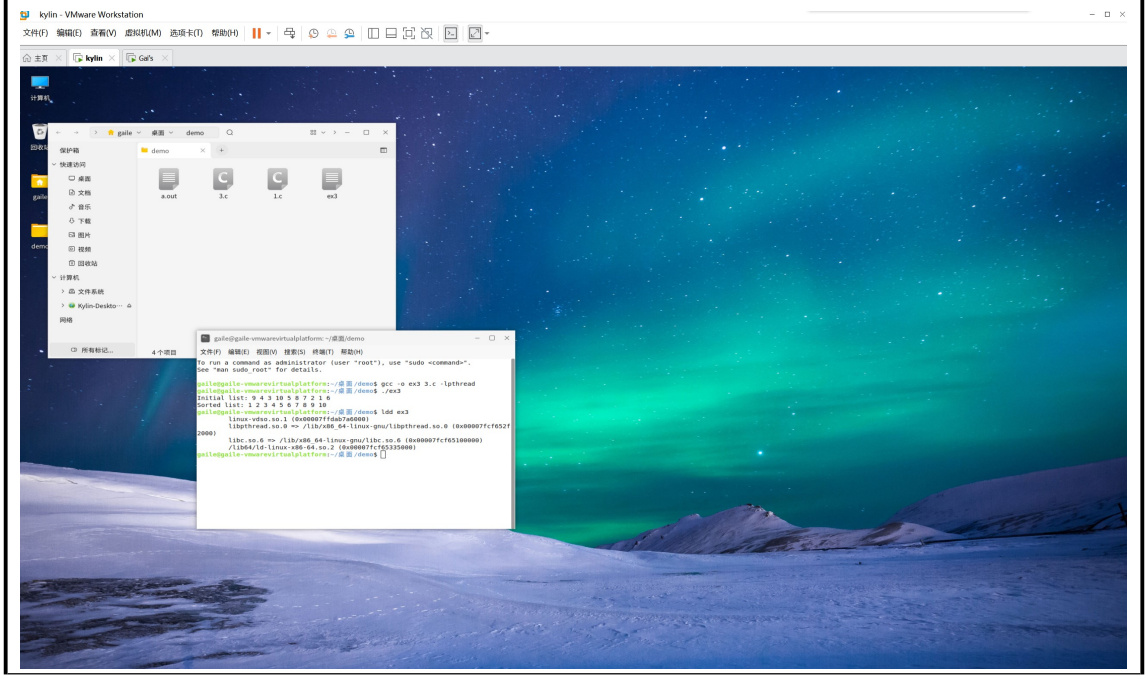
```c
void *merge_sort(void *arg) {
    int *args = (int *)arg;
    int low = args[0];
    int high = args[1];
    int mid = low + (high - low) / 2;
    int *left_args = malloc(2 * sizeof(int));
    int *right_args = malloc(2 * sizeof(int));
    pthread_t left_thread, right_thread;    //声明线程ID

    if(low < high) {
        left_args[0] = low;
        left_args[1] = mid;
        right_args[0] = mid + 1;
        right_args[1] = high;

        //pthread_create 函数用于创建一个新的线程。
        pthread_create(&left_thread, NULL, merge_sort, left_args);
        pthread_create(&right_thread, NULL, merge_sort, right_args);

        //pthread_join 函数用于等待一个线程结束。
        pthread_join(left_thread, NULL);
        pthread_join(right_thread, NULL);

        merge(low, mid, high);
    }

    free(left_args);
    free(right_args);
    pthread_exit(NULL);
}
```

4. 在 main 函数中，首先打印出原始数组 list，然后创建一个新的线程，将归并排序的任务交给该线程处理。等待线程结束后，打印出排序后的结果。

```c
int main() {
    int i;
    int *args = malloc(2 * sizeof(int));
    args[0] = 0;
    args[1] = SIZE - 1;
    printf("Initial list: ");
    for(i = 0; i < SIZE; i++)
        printf("%d ", list[i]);
    printf("\n");
    pthread_t thread;

    pthread_create(&thread, NULL, merge_sort, args);
    pthread_join(thread, NULL);

    printf("Sorted list: ");
    for(i = 0; i < SIZE; i++)
        printf("%d ", list[i]);
    printf("\n");

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

四、实验结果与分析

在初始化的整数列表 list 为{9, 4, 3, 10, 5, 8, 7, 2, 1, 6}的情况下：



五、问题总结

1.原始列表分成的子列表大小不一致

解决方案：可以使用 C 语言中的数组和指针来实现。通过计算原始数组中元素的数量和每个子数组中元素的数量，可以得出需要划分数组的位置，并使用指针将原始数组分为两个子数组。

2 会出现两个 sorting threads 线程不同时结束

解决方案：可以使用 POSIX 线程库中的 pthread_join() 函数，在主函数中调用该函数等待所有线程执行完毕。

3.如何将两个子列表合并为一个排序列表

解决方案：可以使用归并排序算法将两个已排好序的子列表合并为一个排序列表。需要创建一个 merging thread 线程来执行此任务。该线程通过指针访问两个子数组，并将它们合并为一个排序列表。

4.程序终止时如何确保所有线程都完成其任务后再终止

解决方案：可以使用 POSIX 线程库中的 pthread_barrier_init() 和 pthread_barrier_wait() 函数来同步所有线程的工作。使用 pthread_barrier_init() 函数初始化一个 Barrier 对象，并在每个线程完成任务后使用 pthread_barrier_wait() 函数通知主函数继续执行。

六、源代码

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

#define SIZE 10

int list[SIZE] = {9, 4, 3, 10, 5, 8, 7, 2, 1, 6};
int result[SIZE];

//归并排序
void merge(int low, int mid, int high) {
    int l1, l2, i;
    for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {
        if(list[l1] <= list[l2])
            result[i] = list[l1++];
        else
```

```
                result[i] = list[l2++];
        }

        while(l1 <= mid)
                result[i++] = list[l1++];

        while(l2 <= high)
                result[i++] = list[l2++];

        for(i = low; i <= high; i++)
                list[i] = result[i];
}

void *merge_sort(void *arg) {
        int *args = (int *)arg;
        int low = args[0];
        int high = args[1];
        int mid = low + (high - low) / 2;
        int *left_args = malloc(2 * sizeof(int));
        int *right_args = malloc(2 * sizeof(int));
        pthread_t left_thread, right_thread;//声明线程 ID

        if(low < high) {
                left_args[0] = low;
                left_args[1] = mid;
                right_args[0] = mid + 1;
                right_args[1] = high;

               //pthread_create  函数用于创建一个新的线程。
                pthread_create(&left_thread, NULL, merge_sort, left_args);
                pthread_create(&right_thread, NULL, merge_sort, right_args);
```

```
            //pthread_join  函数用于等待一个线程结束。
            pthread_join(left_thread, NULL);
            pthread_join(right_thread, NULL);

            merge(low, mid, high);
        }

        free(left_args);
        free(right_args);
        pthread_exit(NULL);
}

int main() {
        int i;
        int *args = malloc(2 * sizeof(int));
        args[0] = 0;
        args[1] = SIZE - 1;
        printf("Initial list: ");
        for(i = 0; i < SIZE; i++)
                printf("%d ", list[i]);
        printf("\n");
        pthread_t thread;

        pthread_create(&thread, NULL, merge_sort, args);
        pthread_join(thread, NULL);

        printf("Sorted list: ");
        for(i = 0; i < SIZE; i++)
                printf("%d ", list[i]);
        printf("\n");

        free(args);
```

```
    return 0;
}
```