Linux 线程竞争范围及线程调度

1. 实验目的

> 1. 了解和掌握 Linux 系统的线程竞争范围
> 2. 了解和掌握 Linux 系统提供的线程调度算法

2. 实验内容

> 1. 编译运行 thread_scope.c 程序，确认 Linux 系统提供的线程竞争范围，利用代码加以分析
> 2. 编译运行 thread_sched.c 程序，确认 Linux 系统提供的线程调度算法，利用代码加以分析

3. 例子代码

3.1. 线程竞争范围代码（thread_scope.c）

```c
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
void * runner(void * param);

int main (int argc, char * argv[]){
    int i, scope;
    pthread_t tid[NUM_THREADS];
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    if (pthread_attr_getscope(&attr, &scope) != 0)
        fprintf(stderr, "unable to ge scheduling scope\n");
    else {
        if (scope == PTHREAD_SCOPE_PROCESS)
            printf("PTHREAD_SCOPE_PROCESS\n");
        else if (scope == PTHREAD_SCOPE_SYSTEM)
            printf("PTHREAD_SCOPE_SYSTEM\n");
        else
            fprintf(stderr, "Illegal scope value.\n");
    }
    pthread_attr_setscope(&attr, PTHREAD_SCOPE_PROCESS);

    if (pthread_attr_getscope(&attr, &scope) != 0)
        fprintf(stderr, "unable to ge scheduling scope\n");
    else {
        if (scope == PTHREAD_SCOPE_PROCESS)
            printf("PTHREAD_SCOPE_PROCESS\n");
        else if (scope == PTHREAD_SCOPE_SYSTEM)
            printf("PTHREAD_SCOPE_SYSTEM\n");
        else
            fprintf(stderr, "Illegal scope value.\n");
    }

    for (i = 0; i < NUM_THREADS; i++ )
        pthread_create(&tid[i], &attr, runner, NULL);
    for (i = 0; i < NUM_THREADS; i++ )
        pthread_join(tid[i], NULL);
}

void * runner(void * param)
{
    pthread_exit(0);
}
```

3.2. 线程调度确认源码（thread_sched.c）

```c
#include <unistd.h>
#include <pthread.h>
#include <sched.h>
#include <stdio.h>

static int get_thread_policy(pthread_attr_t attr)
{
    int policy;
    pthread_attr_getschedpolicy(&attr, &policy);
    switch(policy)
    {
        case SCHED_FIFO:
            printf(" policy = SCHED_FIFO\n");
            break;
        case SCHED_RR:
            printf(" policy = SCHED_RR\n");
            break;
        case SCHED_OTHER:
            printf(" policy = SCHED_OTHER\n");
            break;
        default:
            printf(" policy = UNKOWN\n");
            break;
    }
    return policy;
}


static void set_thread_policy(pthread_attr_t attr, int policy)
{
    pthread_attr_setschedpolicy(&attr, policy);
    get_thread_policy(attr);
}

static void show_thread_priority(pthread_attr_t attr, int policy)
{
    int priority = sched_get_priority_max(policy);
    printf(" max priority = %d, ", priority);
    priority = sched_get_priority_min(policy);
    printf(" min priority = %d\n", priority);
}

static int get_thread_priority(pthread_attr_t attr)
{
    struct sched_param param;
    pthread_attr_getschedparam(&attr, &param);
    printf(" priority = %d\n", param.sched_priority);
    return param.sched_priority;
}

int main (void)
{
    pthread_attr_t attr;
    struct sched_param sched;
    pthread_attr_init(&attr);

    printf("- Show Current Policy:");
    int policy = get_thread_policy(attr);
    printf("- Show current configuration of priority:");
    show_thread_priority(attr, policy);
    printf("- Show SCHED_FIFO of priority:");
    show_thread_priority(attr, SCHED_FIFO);
    printf("- Show SCHED_RR of priority:");
    show_thread_priority(attr, SCHED_RR);

    printf("- Show priority of current thread:");
    int priority = get_thread_priority(attr);
    printf("\n -SET THREAD POLICY\n");
    printf("- SET SCHED_FIFO policy:");
    set_thread_policy(attr, SCHED_FIFO);
    printf("- SET SCHED_RR policy:");
    set_thread_policy(attr, SCHED_RR);
    printf("- Restore current policy:");
    set_thread_policy(attr, policy);
    pthread_attr_destroy(&attr);
    return 0;
}
```