上海交通大學

《操作系统》课程

学生实验报告

实验名称: Project5

姓　　名：洪瑄锐

学　　号：517030910227

班　　级：F1703302

手　　机：15248246044

任课老师：吴晨涛

2019 年 5 月 26 日

目录

一、实验要求

2-1 Designing a Thread Pool

设计一个线程池，使用线程池时，任务将提交到池中，并由池中的线程执行。使用队列将工作提交到线程池，并且可用线程从队列中删除工作。如果没有可用的线程，则工作保持排队，直到有一个线程可用。如果没有工作，则线程等待通知，直到有任务存在。

2-2 The Producer-Consumer Problem

设计有界缓冲问题的解决方案。使用三个信号量，empty，full 以及互斥量，分别用来计算缓冲区中空的和满的槽的数量，保护实际的插入或者删除。

二、程序设计思想及代码解释

2-1 Designing a Thread Pool

• pool_init()

创造线程，初始化信号量和互斥量，对工作队列的 push 位置和 pop 位置初始化。sem_notice 用于通知工作队列中是否有任务存在，因此初始化为 0。sem_mutex 用于保护插入和删除，因此初始化为 1。

```
void pool_init(void)
{
    sem_init(&mutex_notice, 0, 0);
    sem_init(&mutex_race, 0, 1);

    work_head = 0;
    work_tail = 0;
    queue_num = 0;

    pthread_create(&bee,NULL,worker,NULL);
}
```

• enqueue(task k)

采用静态数组作为工作队列，queue_head 存储 pop 位置，初始化为 0，queue_tail 存储 push 位置，初始化为 0，queue_num 存储工作队列中的任务数量，初始化为 0。如果 queue_num 与工作队列大小相等，说明工作队列已满，插入失败，返回 1，否则插入，返回 0.

```
int enqueue(task t)
{
    if (queue_num==QUEUE_SIZE)
    {
        return 1;
    }
    queue[work_tail].function = t.function;
    queue[work_tail].data = t.data;
    work_tail = (work_tail + 1) % QUEUE_SIZE;
    queue_num++;
    return 0;
}
```

- dequeue()

从 queue_head 位置 pop 出一个任务，将存在的任务数量减一即可。

```
task dequeue()
{
    worktodo.function = queue[work_head].function;
    worktodo.data = queue[work_head].data;
    work_head = (work_head + 1) % QUEUE_SIZE;
    queue_num--;
    return worktodo;
}
```

- pool_submit()

注意信号量和互斥量的使用。在尝试插入之前需要 sem_wait(&sem_mutex),使其在不为 0 的情况下自减一变为 0，这样其他线程就无法使其自减一从而不可以插入。在尝试插入之后 sem_post(&sem_mutex),将其恢复。如果插入成功，要将 sem_notice 自增 1.

```
int pool_submit(void (*somefunction)(void *p), void *p)
{
    worktodo.function = somefunction;
    worktodo.data = p;
    sem_wait(&mutex_race);
    int flag;
    flag = enqueue(worktodo);
    sem_post(&mutex_race);
    if (flag)
        printf("submit failure.\n");
    else
    {
        sem_post(&mutex_notice);
        printf("submit successfully.\n");
    }

    return 0;
}
```

- execute(void (*somefunction)(void *p), void *p)

  输出执行成功

```
void execute(void (*somefunction)(void *p), void *p)
{
    printf("Execute successfully.\n");
    (*somefunction)(p);
}
```

- worker()

  循环操作，在 while 中首先 sem_wait(&sem_notice)，即当 sem_notice 不为 0 时才能自减一执行下面的内容。在进行 dequeue()之前和之后分别调用 sem_wait(&sem_mutex)和 sem_post(&sem_mutex),保护只有这一个线程在对工作队列进行操作.

```
void *worker(void *param)
{
    while (1)
    {
        sem_wait(&mutex_notice);
        sem_wait(&mutex_race);
        worktodo = dequeue();
        sem_post(&mutex_race);
        // execute the task
        execute(worktodo.function, worktodo.data);
    }
    pthread_exit(0);
}
```

- pool_shutdown()

  该函数将取消每个工作线程然后通过调用 pthread_join()等待每个线程终止.

```
void pool_shutdown(void)
{
    pthread_cancel(bee);
    pthread_join(bee,NULL);
    printf("exit.\n");
}
```

2-2 The Producer-Consumer Problem

- insert_item(buffer_item item)

  head 记录 buffer 中数据应被 pop 的位置，tail 记录 buffer 中数据应被 push 的位置，buffer_num 记录 buffer 中的数据数目（与 2-1 相同）。如果 buffer_num 等于 buffer 容量，那么插入失败，返回-1，否则插入，返回 0.

```c
int insert_item(buffer_item item)
{
    if (buffer_num == BUFFER_SIZE)
        return -1;
    buffer[tail] = item;
    tail = (tail + 1) % BUFFER_SIZE;
    buffer_num++;
    return 0;
}
```

• remove_item()

如果 buffer_num=0，证明 buffer 为空，移除失败，输出错误信息，返回 -1。否则移除并将 buffer_num-1,返回 0.

```c
int remove_item(buffer_item *item)
{
    if (buffer_num == 0)
        return -1;
    *item = buffer[head];
    head = (head + 1) % BUFFER_SIZE;
    buffer_num--;
    return 0;
}
```

• producer()

注意信号量和互斥量的使用。在生产之前需要 sem_wait(&empty),意味着当槽满的时候即 empty 为 0 时生产者无法进行生产。如果 empty 不为 0，则自减 1 进行生产。利用 sem_race 信号量保证此时只有一个线程在操作槽。生产结束后恢复 sem_race，并将 full 自增 1.

```c
void *producer(void *param)
{
    srand((unsigned)time(NULL));
    buffer_item item;
    while (1)
    {
        int time = rand()%5+1;
        sleep(time);
        item = rand();
        sem_wait(&empty);
        sem_wait(&race);
        if (insert_item(item))
            printf("Buffer is full.\n");
        else
            printf("producer produced %d.\n", item);
        sem_post(&race);
        sem_post(&full);
    }
}
```

• consumer()

注意信号量和互斥量的使用。在生产之前需要 sem_wait(&full),意味着当槽空的时候即 full 为 0 时消费者无法进行消费。如果 full 不为 0，则自减 1 进行消费。利用 sem_race 信号量保证此时只有一个线程在操作槽。消费结束后恢复 sem_race，并将 empty 自增 1.

```
void *consumer(void *param)
{
    buffer_item item;
    srand((unsigned)time(NULL));

    while (1)
    {
        int time = rand()%5+1;
        sleep(time);
        sem_wait(&full);
        sem_wait(&race);
        if (remove_item(&item))
            printf("Buffer is empty.\n");
        else
            printf("consumer consumed %d.\n",item);
        sem_post(&race);
        sem_post(&empty);
    }
}
```

• main()

1. 得到命令行参数 argv[1].argv[2].argv[3]
2. 初始化 empty=buffer_size,full=0,race=1
3. 创建生产者线程
4. 创建消费者线程
5. 睡眠
6. 结束

```c
/*Get conmmand line arguments*/

int t, p_num, c_num;

t = atoi(argv[1]);
p_num = atoi(argv[2]);
c_num = atoi(argv[3]);

/*Initialize buffer*/
sem_init(&race, 0, 1);
sem_init(&empty, 0, BUFFER_SIZE);
sem_init(&full, 0, 0);
/*Greate producer thread(s) and consumer thread(s)*/
pthread_t pro_thread[100];
pthread_t con_thread[100];

for(int i = 0; i < p_num; i++)
    pthread_create(&pro_thread[i], NULL, producer, NULL);
for (int i = 0; i < c_num; i++)
    pthread_create(&con_thread[i], NULL, consumer, NULL);

sleep(t);

for (int i = 0; i < p_num; i++)
{
    pthread_cancel(pro_thread[i]);
    pthread_join(pro_thread[i], NULL);
}

for (int i = 0; i < c_num; i++)
{
    pthread_cancel(con_thread[i]);
    pthread_join(con_thread[i], NULL);
}

return 0;
```

三、运行结果截图

1. threadpool

```
osc@ubuntu:~/final-src-osc10e/ch7/project-1/posix$ ./example
submit successfully.
Execute successfully.
exit.
```

2. producer_consumer

```
osc@ubuntu:~/final-src-osc10e/ch7/project-1/posix$ ./producer_consumer 5 5 5
producer produced 1593658036.
consumer consumed 1593658036.
producer produced 1903207756.
consumer consumed 1903207756.
producer produced 147496509.
producer produced 1761097782.
consumer consumed 147496509.
producer produced 579041104.
consumer consumed 1761097782.
consumer consumed 579041104.
producer produced 1692235249.
```