重庆邮电大学

学生作业报告

作 业 3报 告

学年学期：2024-2025 学年 ☑春□秋学期

课程名称：操作系统

学生学院：国际学院

专业班级：34082201

学生学号：2022214961

学生姓名：周明宇

联系电话：13329148059

重庆邮电大学教务处制

课程名称	操作系统	课程编号	A2130330
作业名称	使用信号量的多线程编程		

一、作业目的

1. 掌握 Linux 环境下使用 pthread 库创建多线程程序的方法
2. 理解信号量(semaphore)的线程同步机制
3. 实现特定执行顺序的线程间同步控制
4. 分析多线程程序的执行流程和竞态条件

二、作业内容

修改提供的 shopping.c 程序，使用信号量实现以下同步要求：

1. "Salad"必须早于"Butter"打印
2. "Milk"必须早于"Apples"打印
3. 确保线程安全，避免竞态条件

三、实现步骤

① 总体实现步骤

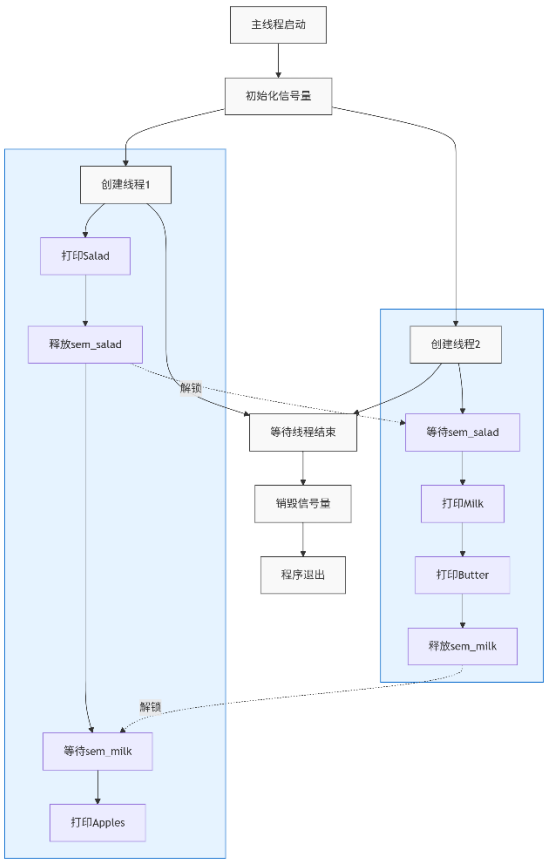

图 1 系统流程图

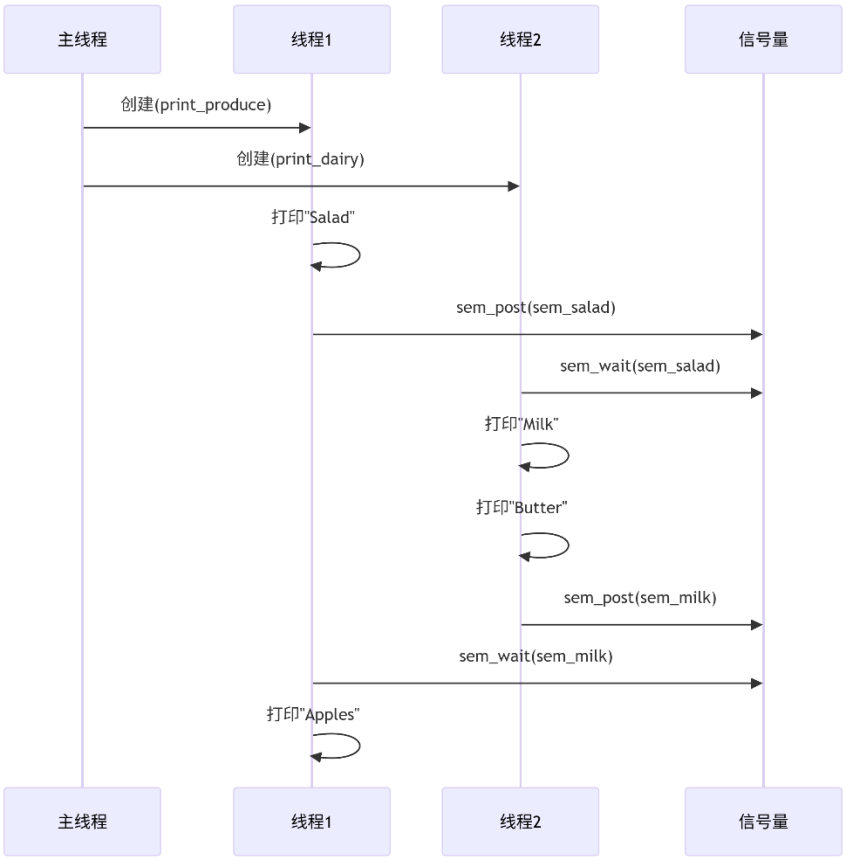

图 2 同步关系图

② 全局信号量声明

```
sem_t sem_salad, sem_milk;
```

定义两个信号量变量，用于线程间同步：

- **sem_salad**：控制 Salad 和 Butter 的打印顺序
- **sem_milk**：控制 Milk 和 Apples 的打印顺序

③ print_produce函数实现

```
void *print_produce(void *items)
{
    char** array = (char**)items;

    printf("got %s\n", array[0]); // 1. 先打印 Salad
    sem_post(&sem_salad);         // 2. 释放 sem_salad 信号量

    sem_wait(&sem_milk);          // 5. 等待 sem_milk 信号量
    printf("got %s\n", array[1]); // 6. 最后打印 Apples

    return NULL;
}
```

1. 立即打印 Salad（无阻塞）
2. 通过 **sem_post** 通知 dairy 线程可以打印 Butter

3. 等待 milk 信号量确保 Milk 已打印
4. 最后打印 Apples

④ print_dairy函数实现

```
void *print_dairy(void *items)
{
    char** array = (char**)items;

    sem_wait(&sem_salad);         // 3. 等待 sem_salad 信号量
    printf("got %s\n", array[0]); // 4a. 打印 Milk
    printf("got %s\n", array[1]); // 4b. 打印 Butter
    sem_post(&sem_milk);          // 4c. 释放 sem_milk 信号量

    return NULL;
}
```

1. 首先等待 Salad 打印完成的信号
2. 连续打印 Milk 和 Butter
3. 通过 sem_post 通知 produce 线程可以打印 Apples

⑤ main函数实现

```
int main()
{
    // 初始化信号量（初始值为 0）
    sem_init(&sem_salad, 0, 0);
    sem_init(&sem_milk, 0, 0);

    // 商品数组
    char *produce[] = { "Salad", "Apples", NULL };
    char *dairy[] = { "Milk", "Butter", NULL };

    // 创建线程
    pthread_t th1, th2;
    pthread_create(&th1, NULL, print_produce, (void*)produce);
    pthread_create(&th2, NULL, print_dairy, (void*)dairy);

    // 等待线程结束
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    // 清理信号量资源
    sem_destroy(&sem_salad);
    sem_destroy(&sem_milk);

    return 0;
}
```

1. 信号量初始化为 0，表示初始时不可用
2. 创建两个线程分别处理 produce 和 dairy 商品
3. 使用 pthread_join 确保主线程等待所有子线程完成
4. 最后销毁信号量释放资源

四、分析与结果

① 编译运行

```
(base) root@LAPTOP-0CUR2SRA:/home/ICSI412/Assignment4# gcc shopping.c -o shopping -lpthread
(base) root@LAPTOP-0CUR2SRA:/home/ICSI412/Assignment4# ./shopping
```

图 3 编译运行

② 运行结果

```
(base) root@LAPTOP-0CUR2SRA:/home/ICSI412/Assignment4# ./shopping
got Salad
got Milk
got Butter
got Apples
```

图 4 结果

经过多次运行，结果中均保证 Salad 先于出现 Butter，Milk 先于 Apples 出现。

五、心得体会

通过本次线程同步作业，我对多线程编程有了更深刻的认识。最初运行时出现的随机输出顺序让我直观理解了线程竞争的不可预测性，而通过信号量的引入，我不仅掌握了 sem_init、sem_wait、sem_post 等关键函数的使用技巧，更重要的是领悟到同步机制的本质是线程间的约定与协作。实验过程中，当出现"Butter"意外先于"Salad"打印时，通过添加调试语句逐步排查，最终发现是信号量释放时机不当所致，这个调试经历让我深刻认识到多线程编程中时序控制的精密性。特别是在分析两种合法输出序列（Salad-Milk-Butter-Apples 与 Salad-Milk-Apples-Butter）时，我理解了同步约束下仍存在的合理并发多样性。这次实验不仅锻炼了我的代码能力，更培养了我对并发程序设计的系统性思维，意识到在资源竞争环境下，良好的同步机制就像交通信号灯，既能防止冲突又能保证效率，这为今后开发更复杂的并发系统打下了坚实基础。

源代码

```c
/* shopping.c - 修改后的版本 */
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

// 声明两个信号量
sem_t sem_salad, sem_milk;
```

```c
void *print_produce(void *);
void *print_dairy(void *);

void *print_dairy(void *items)
{
    char** array = (char**)items;

    // 等待 Salad 先打印
    sem_wait(&sem_salad);
    printf("got %s\n", array[0]); // 打印 Milk
    printf("got %s\n", array[1]); // 打印 Butter

    // 释放信号量，允许 Apples 打印
    sem_post(&sem_milk);
    return NULL;
}

void *print_produce(void *items)
{
    char** array = (char**)items;

    printf("got %s\n", array[0]); // 打印 Salad
    // 释放信号量，允许 Butter 打印
    sem_post(&sem_salad);

    // 等待 Milk 先打印
    sem_wait(&sem_milk);
    printf("got %s\n", array[1]); // 打印 Apples

    return NULL;
}

int main()
{
    // 初始化信号量
    sem_init(&sem_salad, 0, 0);
    sem_init(&sem_milk, 0, 0);

    char *produce[] = { "Salad", "Apples", NULL };
    char *dairy[] = { "Milk", "Butter", NULL };

    pthread_t th1, th2;
    pthread_create(&th1, NULL, print_produce, (void*)produce);
    pthread_create(&th2, NULL, print_dairy, (void*)dairy);

    pthread_join(th1, NULL);
```

```
    pthread_join(th2, NULL);

    // 销毁信号量
    sem_destroy(&sem_salad);
    sem_destroy(&sem_milk);

    return 0;
}
```