山东大学 计算机科学与技术 学院

操作系统 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202200130119 | 姓名：于斐 | | 班级：学堂计机22 |
| 实验题目： 进程同步 | | | |
| 实验学时： 4 | | 实验日期： 2024年3月25日 | |
| 实验目的：  加深对并发协作进程同步与互斥概念的理解，观察和体验并发进程同步与互斥操作的效果，分析与研究经典进程同步与互斥问题的实际解决方案。同时，了解 Linux 系统中 IPC 进程同步工具的用法，并通过练习编程与调试技术来加强对并发协作进程的同步与互斥操作的理解与掌握。 | | | |
| 实验过程中遇到和解决的问题：  实验中遇到“同一窗口不方便展示多个进程”的问题，最终学习使用 tmux 解决。 | | | |
| 实验步骤与内容：  实验环境：Arch Linux x86\_64 with GCC 13.2.1-5  实验步骤：   1. 新建 manager.c producer.c smoker.c 文件，分别写入附录中的代码内容。    1. manager.c 是一个管理器程序。init\_sem 函数负责创建并初始化一系列信号量，包括表示空槽数量的信号量、表示每种物品数量的信号量以及互斥访问信号量，在 main 函数中，程序调用 init\_sem 进行信号量的初始化，并输出一些提示信息，然后进入一个无限循环等待信号。    2. producer.c是一个实现生产者逻辑的程序。在 main 函数中，程序首先解析命令行参数以获取生产者的ID，然后调用 init\_sem 函数进行信号量的初始化。接着，程序进入一个无限循环，其中每次循环表示生产一个物品。在每次循环中，生产者会等待获取对应物品的需求信号量，并等待有空槽可用的信号量，然后获取互斥信号量以确保对共享资源的互斥访问。之后，生产者会生产物品，并释放互斥信号量和对应物品的信号量，然后循环等待。    3. smoker.c 负责实现吸烟者的逻辑。在 main 函数中，程序首先解析命令行参数以获取吸烟者的ID，然后调用 init\_sem 函数进行信号量的初始化。接着，程序输出初始化完成的提示信息，并释放对应吸烟者所需物品的需求信号量。随后，程序进入一个无限循环，其中每次循环表示吸烟一次。在每次循环中，吸烟者会等待获取所需物品的信号量，并等待获取互斥信号量以确保对共享资源的互斥访问。然后，吸烟者会模拟吸烟的过程，并释放互斥信号量和对应吸烟者所需物品的信号量，然后循环等待。 2. 新建 Makefile 文件，在执行 make 时做编译工作，make clean 时清除所有 .o 及无扩展名程序文件。 3. 在程序目录下分别执行 make clean; make && ./manager，./producer [1-2]，./smoker [1-3]，观察实验结果。   实验结果如图，为方便展示，假定生产者五秒制造一套资源，吸烟者十秒吸完一根香烟。    初始时，制造者等待吸烟者 1 发送需求    吸烟者 1 启动后，制造者 1 收到请求制造 1 号材料，之后被吸烟者 1 得到。    生产者 1 抢到吸烟者 2 的需求，占据 empty 位置，生产 2 号材料。生产者 2 收到吸烟者 1 的请求，但是在等待 empty 位置，因此无法制造。    生产者 1 接连抢到吸烟者 2，3 的需求并完成制造，之后由于轮空，生产者 2 终于抢到 empty 位置，制造生产者 1 的请求。    一段时间后体系正常运行。    在吸烟者 2 停止后，生产者 1, 2 均在等待相关需求，其他吸烟者无法得到材料。 | | | |
| 实验总结：  通过本次实验，学习和使用 Linux 系统中的 IPC 机制和信号量，初步掌握在实际开发中如何利用系统提供的工具来实现进程之间的同步和互斥，加深对并发编程中进程同步与互斥的理解。  生产者/消费者问题和抽烟者问题都是并发计算中的经典问题，它们涉及到进程同步和资源共享的核心原理。生产者/消费者问题主要关注的是生产者和消费者之间的协调，确保生产者生产的物品能够被消费者及时消费，同时避免资源竞争和死锁等问题。而抽烟者问题则关注于多个抽烟者之间对烟的共享和同步，保证抽烟行为的合理性和安全性。  在真实操作系统中：   1. 信号量是一种用于控制多个线程对共享资源访问的同步机制。在生产者/消费者问题中，可以使用信号量来实现对缓冲区的同步访问。生产者生产一个物品后，会释放（增加）一个信号量，表示缓冲区中有一个空位；消费者消费一个物品前，会等待（减少）一个信号量，表示需要获取一个物品。这样，通过信号量的操作，可以确保生产者和消费者之间的协调同步。 2. 互斥锁是另一种常用的同步机制，用于保证同一时间只有一个线程可以访问某个共享资源。在抽烟者问题中，可以使用互斥锁来保护烟的共享状态。当某个抽烟者想要抽烟时，需要先获得互斥锁，然后检查烟的状态，如果烟存在则进行抽烟操作，并更新烟的状态；操作完成后释放互斥锁，以便其他抽烟者可以访问。   信号量机制是操作系统中用于解决进程同步和互斥问题的一种重要手段。它基于信号量的概念，通过控制对共享资源的访问来协调多个进程或线程的执行，通过初始值为 1 的信号量的 PV 操作实现互斥，通过初始值为 0 的信号量的 PV 实现同步。  在上面的例子中，信号量初值往往表示“XXX 剩余多少”。如，empty 表示当前空位剩余几个，items 表示某套材料剩余几个，need 表示某个需求剩余几个等等。信号量的值变化会反映对应资源数量的变化。  通过相关测试，程序可以在不同数量、不同时延下完成任务。 | | | |

附录：程序关键源代码

**Manager.c:**

#include "ipc.h"

#include "main.h"

void init\_sem(int\* empty, int\* items, int\* need, int\* mutex) {

    \*empty = semget(EMPTY, 1, IPC\_CREAT | 0666);

    if (\*empty == -1) {

        perror("semget");

        exit(EXIT\_FAILURE);

    }

    sem\_setval(\*empty, size\_of\_shared\_memory);

    for (int i = 0; i < ITEM\_SIZE; i++) {

        items[i] = semget(ITEMS[i], 1, IPC\_CREAT | 0666);

        if (items[i] == -1) {

            perror("semget");

            exit(EXIT\_FAILURE);

        }

        sem\_setval(items[i], 0);

    }

    for (int i = 0; i < ITEM\_SIZE; i++) {

        need[i] = semget(NEED[i], 1, IPC\_CREAT | 0666);

        if (need[i] == -1) {

            perror("semget");

            exit(EXIT\_FAILURE);

        }

        sem\_setval(need[i], 0);

    }

    \*mutex = semget(MUTEX, 1, IPC\_CREAT | 0666);

    if (\*mutex == -1) {

        perror("semget");

        exit(EXIT\_FAILURE);

    }

    sem\_setval(\*mutex, 1);

}

void sig\_handler() {

    fprintf(stderr, "Manager: Received SIGINT. Stopping...\n");

    sem\_delete(EMPTY, semget(EMPTY, 1, IPC\_CREAT | 0666));

    for (int i = 0; i < ITEM\_SIZE; i++) {

        sem\_delete(ITEMS[i], semget(ITEMS[i], 1, IPC\_CREAT | 0666));

    }

    for (int i = 0; i < ITEM\_SIZE; i++) {

        sem\_delete(NEED[i], semget(NEED[i], 1, IPC\_CREAT | 0666));

    }

    sem\_delete(MUTEX, semget(MUTEX, 1, IPC\_CREAT | 0666));

    exit(0);

}

int main(int argc, char\* argv[]) {

    int empty, items[ITEM\_SIZE], need[ITEM\_SIZE], mutex;

    init\_sem(&empty, items, need, &mutex);

    fprintf(stderr, "Manager: Finished.\n");

    fprintf(stderr, "Smoker 1 will need tobacco and paper.\n");

    fprintf(stderr, "Smoker 2 will need tobacco and glue.\n");

    fprintf(stderr, "Smoker 3 will need paper and glue.\n");

    fprintf(stderr, "Producer will produce Smoker 1, 2, 3, 1, 2, 3, ...'s needs successively.\n");

    signal(SIGINT, sig\_handler);

    for (; ; ) {

        sleep(1);

    }

}

**Producer.c:**

#include "ipc.h"

#include "main.h"

void init\_sem(int\* empty, int\* items, int\* need, int\* mutex) {

    \*empty = semget(EMPTY, 1, IPC\_CREAT | 0666);

    if (\*empty == -1) {

        perror("semget");

        exit(EXIT\_FAILURE);

    }

    for (int i = 0; i < ITEM\_SIZE; i++) {

        items[i] = semget(ITEMS[i], 1, IPC\_CREAT | 0666);

        if (items[i] == -1) {

            perror("semget");

            exit(EXIT\_FAILURE);

        }

    }

    for (int i = 0; i < ITEM\_SIZE; i++) {

        need[i] = semget(NEED[i], 1, IPC\_CREAT | 0666);

        if (need[i] == -1) {

            perror("semget");

            exit(EXIT\_FAILURE);

        }

    }

    \*mutex = semget(MUTEX, 1, IPC\_CREAT | 0666);

    if (\*mutex == -1) {

        perror("semget");

        exit(EXIT\_FAILURE);

    }

}

int main(int argc, char\* argv[]) {

    int id = atoi(argv[1]);

    int empty, items[ITEM\_SIZE], need[ITEM\_SIZE], mutex;

    init\_sem(&empty, items, need, &mutex);

    fprintf(stderr, "Producer %d: Initilized.\n", id);

    int i = 0;

    for (; ; ) {

        fprintf(stderr, "Producer %d: Waiting for need of %d-th items.\n", id, i + 1);

        P(need[i]);

        fprintf(stderr, "Producer %d: Got the need of %d-th items.\n", id, i + 1);

        fprintf(stderr, "Producer %d: Waiting for the space.\n", id);

        P(empty);

        fprintf(stderr, "Producer %d: Got the space.\n", id);

        P(mutex);

        sleep(5); // for better visualization

        fprintf(stderr, "Producer %d: Produced %d-th items.\n", id, i + 1);

        V(mutex);

        V(items[i]);

        i = (i + 1) % ITEM\_SIZE;

        sleep(1); // for better visualization

    }

}

**Smoker.c:**

#include "ipc.h"

#include "main.h"

void init\_sem(int\* empty, int\* items, int\* need, int\* mutex) {

    \*empty = semget(EMPTY, 1, IPC\_CREAT | 0666);

    if (\*empty == -1) {

        perror("semget");

        exit(EXIT\_FAILURE);

    }

    for (int i = 0; i < ITEM\_SIZE; i++) {

        items[i] = semget(ITEMS[i], 1, IPC\_CREAT | 0666);

        if (items[i] == -1) {

            perror("semget");

            exit(EXIT\_FAILURE);

        }

    }

    for (int i = 0; i < ITEM\_SIZE; i++) {

        need[i] = semget(NEED[i], 1, IPC\_CREAT | 0666);

        if (need[i] == -1) {

            perror("semget");

            exit(EXIT\_FAILURE);

        }

    }

    \*mutex = semget(MUTEX, 1, IPC\_CREAT | 0666);

    if (\*mutex == -1) {

        perror("semget");

        exit(EXIT\_FAILURE);

    }

}

int main(int argc, char\* argv[]) {

    int id = atoi(argv[1]);

    int empty, items[ITEM\_SIZE], need[ITEM\_SIZE], mutex;

    init\_sem(&empty, items, need, &mutex);

    fprintf(stderr, "Smoker %d: Initilized.\n", id);

    V(need[id - 1]);

    for (; ; ) {

        fprintf(stderr, "Smoker %d: Waiting for the items.\n", id);

        P(items[id - 1]);

        P(mutex);

        fprintf(stderr, "Smoker %d: Got the items. Smoking...\n", id);

        sleep(5); // for better visualization

        V(mutex);

        V(need[id - 1]);

        V(empty);

        fprintf(stderr, "Smoker %d: Smoked.\n", id);

        sleep(1); // for better visualization

    }

}