生产者与消费者

> 2021240205班 严晏来 2021902228

一、代码理解

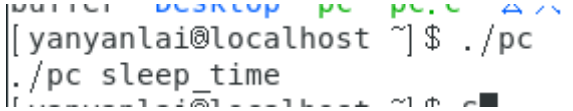
这段代码实现了一个基于 System V 信号量的生产者-消费者问题解决方案。生产者和消费者是两个独立的进程，通过信号量来同步它们对一个共享资源的访问。主要功能包括：

1. **进程创建与信号量初始化：** 根据命令行参数判断当前进程是生产者还是消费者，然后创建或获取一个包含两个信号量的信号量集。如果是生产者，初始化信号量集的值。
2. **生产者功能：** 生成随机数，等待共享资源可用，将生成的数字写入本地文件，通知消费者资源已被生产。生产者在一个循环中执行这个过程，每次生成一个数字。
3. **消费者功能：** 等待共享资源可读，读取本地文件中的数字，通知生产者资源已被读取。消费者在一个无限循环中执行这个过程，每次读取一个数字。
4. **清理：** 生产者在完成一定次数的生成后，等待一段时间后删除信号量集。程序输出相应的信息，表示生产者已经完成生产并删除了信号量集。

总体来说，这段代码演示了通过 System V 信号量实现的进程间同步和共享资源的基本机制。

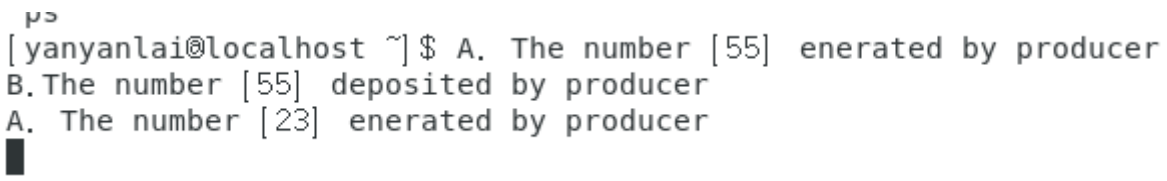
二、执行结果

2.1直接执行



2.2执行命令 ./pc 2&pc 0

第一次：



第二次：

```
[yanyanlai@localhost ~]$ C.The number [55] obtained by consumer   2759
B.The number [23] deposited by producer
C.The number [23] obtained by consumer   2759
A. The number [ 3] enerated by producer
B.The number [ 3] deposited by producer
C.The number [ 3] obtained by consumer   2759
A. The number [36] enerated by producer
B.The number [36] deposited by producer
C.The number [36] obtained by consumer   2759
A. The number [51] enerated by producer
B.The number [51] deposited by producer
[yanyanlai@localhost ~]$ C.The number [51] obtained by consumer   2759
A. The number [51] enerated by producer
B.The number [51] deposited by producer
C.The number [51] obtained by consumer   2759
A. The number [25] enerated by producer
B.The number [25] deposited by producer
C.The number [25] obtained by consumer   2759
A. The number [22] enerated by producer
B.The number [22] deposited by producer
C.The number [22] obtained by consumer   2759
A. The number [48] enerated by producer
B.The number [48] deposited by producer
C.The number [48] obtained by consumer   2759
A. The number [73] enerated by producer
B.The number [73] deposited by producer
```

三、多个缓冲文件的生产者与消费者问题

为了实现多个缓冲文件的生产者与消费者问题，采用以下方法：

1. 为每个缓冲文件创建独立的信号量集，以确保每个缓冲文件都有自己的读取和生产信号量。
2. 修改代码以支持多个缓冲文件的操作，包括文件路径的动态生成和处理多个信号量集。

以下是修改后的代码：

```
cCopy code#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#define NUM_BUFFERS 3
#define BASE_KEY 1234

union semun {
    int val;
    struct semid_ds *buf;
    ushort *array;
};

main(int argc, char *argv[]) {
    FILE *fptr;
    static struct sembuf acquire = {0, -1, SEM_UNDO}, release = {0, 1,
SEM_UNDO};
    pid_t c_pid;
    key_t ipc_key;
    static ushort start_val[2] = {1, 0};
```

```
    int semid, producer = 0, i, n, p_sleep, c_sleep;
    union semun arg;
    enum {READ, MADE};

    if (argc != 2) {
        fprintf(stderr, "%s sleep_time\n", argv[0]);
        exit(-1);
    }

    ipc_key = ftok(".", BASE_KEY);

    for (int buffer_index = 0; buffer_index < NUM_BUFFERS; buffer_index++) {
        ipc_key++; // Generate a new key for each buffer

        if ((semid = semget(ipc_key, 2, IPC_CREAT | IPC_EXCL | 0666)) != -1) {
            producer = 1;
            arg.array = start_val;

            if (semctl(semid, 0, SETALL, arg) == -1) {
                perror("semctl -- producer -- initialization");
                exit(1);
            }
        } else if ((semid = semget(ipc_key, 2, 0)) == -1) {
            perror("semget");
            exit(2);
        }

        switch (fork()) {
            case -1:
                perror("fork");
                exit(3);
            case 0:
                // Child process (either producer or consumer)
                p_sleep = atoi(argv[1]);
                c_sleep = atoi(argv[1]);
                srand((unsigned)getpid());

                if (producer) {
                    // Producer process
                    // ... (same as before)
                } else {
                    // Consumer process
                    c_pid = getpid();
                    // ... (same as before)
                }

                exit(0);
        }
    }

    // Parent process
    // ... (same as before, maybe wait for child processes)

    exit(0);
}
```

这个修改后的代码通过增加 `NUM_BUFFERS` 宏定义来支持多个缓冲文件。每个缓冲文件有独立的信号量集和 IPC 键值。在创建信号量集时，通过 `ipc_key++` 来生成新的键值，确保每个缓冲文件都有唯一的键值。同时，使用 `fork()` 函数创建多个进程来模拟多个生产者和消费者。