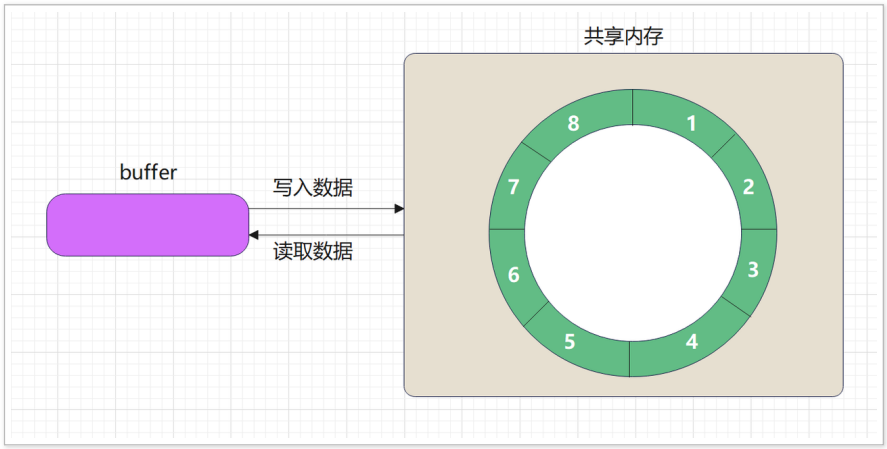
# 2.3 环形队列设计 (三)- 环形队列数据读写实现_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 2.3 环形队列设计 (三)– 环形队列数据读写实现涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。




- 环形队列的写数据的实现逻辑如下:
  - 判断队列是否已满，这里使用 信号量来判断
  - sem_p(fifo->p_head->semid,SEM_FULL_ID);
- 
  - 获取互斥锁，因为队列使用的空间为共享内存的空间
  - sem_p(fifo->p_head->semid,SEM_MUTEX_ID);
  - 计算当前的写位置（字节数）
  - 
  - pos = fifo->p_head->wpos * fifo->p_head->blksz;
- 
- 
  - 将数据入队列, 这里使用 memcpy 函数
  - memcpy(fifo->p_payload + pos ,buf,fifo->p_head->blksz);
  - 释放互斥锁
  - sem_v(fifo->p_head->semid,SEM_MUTEX_ID);
  - 释放判空信号量
  - sem_v(fifo->p_head->semid,SEM_EMPTY_ID);
- 完整代码如下:

```
void shmfifo_put(shm_fifo_t *fifo,const void *buf)
{
        int pos = 0;

        sem_p(fifo->p_head->semid,SEM_FULL_ID);
        sem_p(fifo->p_head->semid,SEM_MUTEX_ID);
```

```
        pos = fifo->p_head->wpos * fifo->p_head->blksz;

        memcpy(fifo->p_payload + pos,buf,fifo->p_head->blksz);
        fifo->p_head->wpos = (fifo->p_head->wpos + 1) % (fifo->p_head->blocks);
        sem_v(fifo->p_head->semid,SEM_MUTEX_ID);
        sem_v(fifo->p_head->semid,SEM_EMPTY_ID);
}
```

- 环形队列的读数据的实现逻辑如下:
    - 判断队列是否为空，这里使用 信号量来判断
    - `sem_p(fifo->p_head->semid,SEM_EMPTY_ID);`
- 
- 
    - 获取互斥锁，因为队列使用的空间为共享内存的空间
    - `sem_p(fifo->p_head->semid,SEM_MUTEX_ID);`
    - 计算当前的读位置（字节数）
    - 
    - `pos = fifo->p_head->wpos * fifo->p_head->blksz;`
- 
- 
    - 将数据出队列, 这里使用 memcpy 函数

      `memcpy(buf,fifo->p_payload + pos,fifo->p_head->blksz);`
- 通过信号量判断环形队列是否为空
    - `sem_p(fifo->p_head->semid,SEM_EMPTY_ID);`
    - 释放互斥锁
    - `sem_v(fifo->p_head->semid,SEM_MUTEX_ID);`
    - 释放判满信号量
    - `sem_v(fifo->p_head->semid,SEM_FULL_ID);`

```
void shmfifo_get(shm_fifo_t *fifo, void *buf)
{
        int pos = 0;

        sem_p(fifo->p_head->semid,SEM_EMPTY_ID);
        sem_p(fifo->p_head->semid,SEM_MUTEX_ID);

        pos = fifo->p_head->rpos * fifo->p_head->blksz;

        memcpy(buf,fifo->p_payload + pos,fifo->p_head->blksz);
        fifo->p_head->rpos = (fifo->p_head->rpos + 1) % (fifo->p_head->blocks);
        sem_v(fifo->p_head->semid,SEM_MUTEX_ID);
        sem_v(fifo->p_head->semid,SEM_FULL_ID);
}
```

```
#include <stdio.h>
#include <string.h>

#include "shmfifo.h"

typedef struct person{
    int age;
    char name[32];

}person_t;


int main(void)
{
    person_t person;

    shm_fifo_t *fifo = shmfifo_init(3,sizeof(person_t));
```

```
    for(;;){
        shmfifo_get(fifo,&person);
        printf("name = %s,age = %d\n",person.name,person.age);
    }
    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

#include "shmfifo.h"

typedef struct person{
    int age;
    char name[32];

}person_t;



int main(void)
{
    int i;
    pid_t cpid;
    shm_fifo_t *fifo = shmfifo_init(3,sizeof(person_t));
    person_t  person;

    cpid = fork();
    if (cpid == -1){
        perror("[ERROR]: fork()");
        exit(EXIT_FAILURE);
    }else if (cpid == 0){
        for (i = 0;i < 10;i++){
            strcpy(person.name,"lisi");
            person.age = 20;
            shmfifo_put(fifo,&person);
            sleep(1);
        }

        exit(EXIT_SUCCESS);
    }else if (cpid > 0){
        cpid = fork();
        if (cpid == -1){
            perror("[ERROR]: fork()");
            exit(EXIT_FAILURE);

        }else if (cpid == 0){
            for (i = 0;i < 10;i++){
                strcpy(person.name,"zhangsan");
                person.age = 30;
                shmfifo_put(fifo,&person);
                sleep(2);
            }
            exit(EXIT_SUCCESS);
        }else if(cpid > 0){
            wait(NULL);
            wait(NULL);
        }
    }

    return 0;
}
```

全文完