操作系统实验笔记

题目一：mysys.c

题目要求：

- mysys的功能与系统函数system相同，要求用进程管理相关系统调用自己实现一遍
- **使用fork/exec/wait系统调用实现mysys**
- 不能通过调用系统函数system实现mysys
- 测试程序

解决思路：

把主函数中的字符串传递给函数mysys，在mysys中对字符串按照空格进行分割成一个词向量，调用fork函数开启一个子进程，在子进程中把词向量传递给函数execvp，子进程执行该条命令。

代码：

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>
void mysys(char* instc)
{
    char **sinstc ;
    char* i,*word;
    int j,wl;
    for (i=instc,wl=0,j=0;*i!='\0';i++)
    {
        if (*i == ' ') {
            wl++;
            word = (char*)malloc(wl * sizeof(char));
            int k,m;
            for (k = 0,m=wl-1; k < wl; k++,m--) {
                if (k != wl - 1) *(word + k) = *(i - m);
                else *(word + k) = '\0';
            }
            sinstc[j++] = word;
            wl = 0;
        }
        else wl++;
    }
    wl++;
    word = (char*)malloc(wl * sizeof(char));
    int k, m;
    for (k = 0, m = wl - 1; k < wl; k++, m--) {
        if (k != wl - 1) *(word + k) = *(i - m);
        else *(word + k) = '\0';
    }
    sinstc[j++] = word;
```

```
    pid_t pid;
    pid = fork();
    if(pid==0){
        sinstc[j]=NULL;
        execvp(sinstc[0],sinstc);
    }
    else
    wait(NULL);
}
int main()
{
    printf("--------------------------------------\n");
    mysys("echo HELLO WORD");
    printf("--------------------------------------\n");
    mysys("ls /");
    printf("--------------------------------------\n");
    return 0;
}
```

运行结果：

```
--------------------------------------
HELLO WORD
--------------------------------------
bin    dev   initrd.img      lost+found  opt   run   sys  var
boot   etc   initrd.img.old  media       proc  sbin  tmp  vmlinuz
cdrom  home  lib             mnt         root  srv   usr  vmlinuz.old
--------------------------------------
```

题目二：sh3.c

题目要求：

- 该程序读取用户输入的命令，调用函数mysys执行用户的命令
- 实现管道
- 只要求连接两个命令，不要求连接多个命令
- 不要求同时处理管道和重定向

解决思路：

- 调用parse_command函数，在其中使用strtok函数，对输入指令按照" "进行分割。
- 对于cd指令：判断是否为cd指令，读取改变地址，使用chdir函数改变程序运行文件路径。
- 对于pwd指令：判断是否为pwd指令，使用getcwd函数，获取程序当前运行文件路径，打印出来。
- 对于exit：直接使用 exit(0) 退出程序，每次子进程运行完后要退出，不然就会造成退出不了的结果。
- 对于重定向：在指令中读取输入和输出文件路径，使用dup2函数，把文件描述符重新定向至input.txt和output.txt，删除原有的文件描述符。

- 对于管道：使用parse_commands函数把输入的指令分割成多条指令，把每条指令存在结构体中，结构体由单条指令向量，输入文件描述符，输出文件描述符组成；使用exec_pipe函数，把最后一条指令当做一条指令，前面所有指令当做一条指令递归执行，递归调用前使用dup2函数重定向标准输出到pipe，执行当前指令时重定向标准输入到pipe，调用exec_command函数来执行当前指令（exec_command函数中开一个子进程，使用execvp函数来执行指令）。

代码：

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<sys/wait.h>
#include<fcntl.h>
#include<sys/stat.h>

#define MAX_ARGC 16
#define MAX_COMMANDS 100

typedef struct command {
    int argc;
    char* argv[MAX_ARGC];
    char* input;
    char* output;
}com;

int command_count;
com commands[MAX_COMMANDS];
char input[100];
char output[100];
int keybord , monitor;

int ifin(char* line, char sign)
{
    char* a;
    a = line;
    while(*a!='\0'){
        if(*a == sign) return 1;
        a++;
    }
    return 0;
}
void mystrcpy(char* b, char* a)
{
    char* x, * y;
    x = a;
    y = b;
    while (*x != '\0') {
        *y = *x;
        x++;
        y++;
    }
    *y = '\0';
}
void parse_command(char* line)
```

```
{
    char* word;
    int i = 0;
    command_count = 1;
    com instruction;
    instruction.argc = 0;
    instruction.input = NULL;
    instruction.output = NULL;
    word = strtok(line, " ");
    while (word != NULL) {
        if(word[0] == '<'){
            char *re = word+1;
            instruction.input = (char*)malloc(sizeof(word)-1);
            mystrcpy(instruction.input,re);
            word = strtok(NULL, " ");
        }
        else if(word[0] == '>'){
            char *re = word+1;
            instruction.output = (char*)malloc(sizeof(word)-1);
            mystrcpy(instruction.output,re);
            word = strtok(NULL, " ");
        }
        else{
            instruction.argc++;
            instruction.argv[i] = word;
            i++;
            word = strtok(NULL, " ");
        }
    }
    instruction.argv[i] = NULL;
    commands[0] = instruction;
}
parse_commands(char* line)
{
    char* words,*word;
    char wordsc[100];
    char* wordsIndex[100];
    int i;
    command_count = 0;
    words = strtok(line, "|");
    i = 0;
    while (words != NULL) {
        command_count++;
        wordsIndex[i] = words;
        i++;
        words = strtok(NULL, "|");
    }
    for (i = 0; i < command_count; i++) {
        word = strtok(wordsIndex[i], " ");
        com instruction;
        instruction.input = NULL;
        instruction.output = NULL;
        int j = 0;
        instruction.argc = 0;
        while (word != NULL) {
            if(word[0] == '<'){
```

```
            char *re = word+1;
            mystrcpy(instruction.input,re);
            instruction.input = input;
            word = strtok(NULL, " ");
        }
        else if(word[0] == '>'){
            char *re = word+1;
            mystrcpy(output,re);
            instruction.output = output;
            word = strtok(NULL, " ");
        }
        else{
            instruction.argc++;
            instruction.argv[j] = word;
            j++;
            word = strtok(NULL, " ");
        }
        }
        instruction.argv[j] = NULL;
        commands[i] = instruction;
    }
}
void teststring()
{
    printf("\u6709%d\u6761\u6307\u4EE4\uFF1A\n", command_count);
    int i = 0, j;
    for (; i < command_count; i++) {
        printf("\u7B2C%d\u6761\uFF1A", i + 1);
        printf("argc=%d ,", commands[i].argc);
        for (j = 0; j < commands[i].argc; j++) printf("%s$",
commands[i].argv[j]);
        printf("\n");
        if (commands[i].input != NULL) printf("input: %s\n", commands[i].input);
        if (commands[i].output != NULL) printf("output: %s\n",
commands[i].output);
    }
}
void execone(com instruction)
{
    if(!strcmp(instruction.argv[0],"cd")){
        int ifsc;
        ifsc = chdir(instruction.argv[1]);
        if(ifsc) printf("Error : adr not exsit\n");
    }
    else if(!strcmp(instruction.argv[0],"pwd")){
        char cpwd[100];
        getcwd(cpwd,sizeof(cpwd));
        printf("%s\n",cpwd);
    }
    else if(!strcmp(instruction.argv[0],"exit")){
        exit(0);
    }
    else{
        if(instruction.input != NULL){
            int fdin;
            fdin = open(instruction.input,O_RDONLY);
```

```
            dup2(fdin,0);
            close(fdin);
        }
        if(instruction.output != NULL){
            int fdout;
            fdout = open(instruction.output,O_CREAT|O_RDWR,0666);
            dup2(fdout,1);
            close(fdout);
        }
        execvp(instruction.argv[0],instruction.argv);
        exit(0);
    }
}
void execsimple(com instruction)
{
    if(!strcmp(instruction.argv[0],"cd")){
        int ifsc;
        ifsc = chdir(instruction.argv[1]);
        if(ifsc) printf("Error : adr not exsit\n");
    }
    else if(!strcmp(instruction.argv[0],"pwd")){
        char cpwd[100];
        getcwd(cpwd,sizeof(cpwd));
        printf("%s\n",cpwd);
    }
    else if(!strcmp(instruction.argv[0],"exit")){
        exit(0);
    }
    else{
        pid_t pid;
        pid = fork();
        if(pid==0){
        if(instruction.input != NULL){
            int fdin;
            fdin = open(instruction.input,O_RDONLY);
            dup2(fdin,0);
            close(fdin);
        }
        if(instruction.output != NULL){
            int fdout;
            fdout = open(instruction.output,O_CREAT|O_RDWR,0666);
            dup2(fdout,1);
            close(fdout);
        }
        execvp(instruction.argv[0],instruction.argv);
        exit(0);
        }
        else wait(NULL);
    }
}
void exec_pipe(int commands_count)
{
    int fd_arry[2];
    pipe(fd_arry);
    pid_t pid;
    pid = fork();
```

```
    if(pid == 0){
        if(commands_count == command_count){
            dup2(fd_arry[0],0);
            close(fd_arry[0]);
            close(fd_arry[1]);
            dup2(monitor,1);
            close(monitor);
            execone(commands[commands_count-1]);
            exit(0);
        }
        else{
            dup2(fd_arry[0],0);
            close(fd_arry[0]);
            close(fd_arry[1]);
            execone(commands[commands_count-1]);
            exit(0);
        }
    }
    if(commands_count > 1) {
        dup2(fd_arry[1],1);
        close(fd_arry[0]);
        close(fd_arry[1]);
        exec_pipe(commands_count-1);
        wait(NULL);
    }
    else{
        wait(NULL);
        dup2(monitor,1);
        dup2(keybord,0);
        close(monitor);
        close(keybord);
    }
}

int main()
{
    char cwd[100];
    getcwd(cwd,sizeof(cwd));
    while(1)
    {
        keybord = dup(0);
        monitor = dup(1);
        char cpwd[100];
        getcwd(cpwd,sizeof(cpwd));
        if (strcmp(cpwd,cwd)){
            char *p , *q;
            p = q = cpwd;
            while(*q!='\0'){
                if(*q == '/') p = q+1;
                q++;
            }
            printf("%s >",p);
        }
        else printf("~ >");
        char line[100];
        gets(line);
```

```
        if(ifin(line,'|')){
            parse_commands(line);
            exec_pipe(command_count);
        }
        else{
            parse_command(line);
            execsimple(commands[0]);
        }
    }
}
```

运行结果：

```
user@instant-contiki:~$ ./sh3
~ >cd /bin
bin >pwd
/bin
bin >cd /home
home >cd /Home
Error : adr not exsit
home >exit
user@instant-contiki:~$ ./sh3
~ >echo 123 >log
~ >cat log
123
~ >cat log | wc -l
1
~ >exit
user@instant-contiki:~$
```

题目：pi1.c

题目要求：

- 使用2个线程根据莱布尼兹级数计算PI
- 莱布尼兹级数公式: 1 - 1/3 + 1/5 - 1/7 + 1/9 - ... = PI/4
- 主线程创建1个辅助线程
- 主线程计算级数的前半部分
- 辅助线程计算级数的后半部分
- 主线程等待辅助线程运行結束后,将前半部分和后半部分相加

解决思路：

将级数设置成 200 项相加，主进程先调用子线程把前100项相加（子线程结果存在全局变量worker中），再把前 100 项相加存在全局变量master中，主函数等待子线程运行结束，把worker和master两个全局变量相加得到结果pi/4，再把结果乘四输出。

代码：

```
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>

#define NUMBER 200

double PI;
double worker_output;
double master_output;

void *worker(void* arg)
{
    int i;
    double j;
    worker_output = 0;
    for(i = 1;i<=NUMBER;i++){
        j = i;
        if(i%2 == 0) worker_output -= 1/(2*j-1);
        else worker_output += 1/(2*j-1);
    }
}

void master()
{
    int i;
    double j;
    master_output = 0;
    for(i = NUMBER+1; i<= 2*NUMBER; i++){
        j=i;
            if(i%2==0)
                    master_output-=1/(2*j-1);
            else
                    master_output+=1/(2*j-1);
    }
}

int main()
{
    pthread_t worker_tid;
    pthread_create(&worker_tid,NULL,worker,NULL);
    master();
    pthread_join(worker_tid,NULL);
    PI = (worker_output+master_output)*4;
    printf("%f\n",PI);
    return 0;
}
```

运行结果：

```
user@instant-contiki:~$ ./pi1
3.139093
```

题目：pi2.c

题目要求：

- 使用N个线程根据莱布尼兹级数计算PI
- 与上一题类似，但本题更加通用化，能适应N个核心
- 主线程创建N个辅助线程
- 每个辅助线程计算一部分任务，并将结果返回
- 主线程等待N个辅助线程运行结束，将所有辅助线程的结果累加
- 本题要求 1: 使用线程参数，消除程序中的代码重复
- 本题要求 2: 不能使用全局变量存储线程返回值

解决思路：

与上一题类似，把级数(200项)分成N(10)个部分，每个部分给出起始的位置，从起始的位置开始计算到Start+NUM/N项，每次计算结果加到全局变量PI/4中，每次加的时候要加锁，主函数调用十次开启子线程的函数并且等待所有子线程运行结束后输出结果乘四。

代码：

```
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>
#include<malloc.h>

#define NUMBER 200
#define N 10

double qPI;
pthread_mutex_t mutex;

void *compute(void* arg)
{
    int *rarg = arg,num;
    int i;
    double j;
    double sum = 0;
    num = *rarg;
    for(i=num+1; i<=num+(NUMBER/N); i++){
        if(i%2==0) {
            j = i;
            pthread_mutex_lock(&mutex);
            qPI -=  1/(2*j-1);
            sum -=  1/(2*j-1);
```

```
            pthread_mutex_unlock(&mutex);
        }
        else{
            j = i;
            pthread_mutex_lock(&mutex);
            qPI +=  1/(2*j-1);
            sum += 1/(2*j-1);
            pthread_mutex_unlock(&mutex);
        }
    }
    return NULL;
}

int main()
{
    pthread_t workers[N];
    int sub = NUMBER/N,i;
    for(i = 0; i<N; i++) {
        int *arg;
        int subs;
        arg = NULL;
        subs = i*sub;
        arg = (int*)malloc(sizeof(int));
        *arg = subs;
        pthread_create(&workers[i], NULL, compute,arg);
    }
    for(i = 0; i<N; i++)
    pthread_join(workers[i], NULL);
    pthread_mutex_destroy(&mutex);
    printf("PI = %f\n",qPI*4);
    return 0;
}
```

运行结果：

```
user@instant-contiki:~$ ./pi2
PI = 3.136593
```

题目：pc1.c

题目要求：

- 使用条件变量解决生产者、计算者、消费者问题
- 系统中有3个线程：生产者、计算者、消费者
- 系统中有2个容量为4的缓冲区：buffer1、buffer2
- 生产者生产'a'、'b'、'c'、'd'、'e'、'f'、'g'、'h'八个字符，放入到buffer1
- 计算者从buffer1取出字符，将小写字符转换为大写字符，放入到buffer2
- 消费者从buffer2取出字符，将其打印到屏幕上

解决思路：

- 生产者：如果array1满，等待条件变量：array1空，放入字母，唤醒等待array1满的线程。
- 计算者：如果array1空，等待条件变量：array1满，拿走一个字母；如果array2满，等待array2空，把字母变成大写后放入array2，唤醒等待array2满的线程；唤醒等待array1空的线程。
- 消费者：如果array2空，等待条件变量：array2满，拿走一个字母，唤醒等待array2空的线程。

代码：

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

#define CAPACITY 4
#define ITEM_COUNT 8

char array1[CAPACITY];
char array2[CAPACITY];
int in1, in2;
int out1, out2;

int buffer_is_empty(char* array)
{
    if(array == array1) return in1 == out1;
    else return in2 == out2;
}
int buffer_is_full(char* array)
{
    if(array == array1)
        return (in1 + 1) % CAPACITY == out1;
    else
        return (in2 + 1) % CAPACITY == out2;
}
char getitem(char* array)
{
    char item;

    if(array == array2){
        item = array2[out2];
        out2 = (out2 + 1) % CAPACITY;
    }
    else{
        item = array1[out1];
        out1 = (out1 + 1) % CAPACITY;
    }
    return item;
}
void putitem(char item,char* array)
{
    if(array == array1){
        array1[in1] = item;
        in1 = (in1 + 1) % CAPACITY;
    }
    else{
```

```
        array2[in2] = item;
        in2 = (in2 + 1) % CAPACITY;
    }
}

pthread_mutex_t mutex1;
pthread_mutex_t mutex2;
pthread_cond_t waita1_empty_buffer;
pthread_cond_t waita1_full_buffer;
pthread_cond_t waita2_empty_buffer;
pthread_cond_t waita2_full_buffer;

void* productor(void* arg)
{
    int i;
    char item;
    for(i = 0; i < ITEM_COUNT; i++){
        pthread_mutex_lock(&mutex1);
        while(buffer_is_full(array1))
            pthread_cond_wait(&waita1_empty_buffer, &mutex1);
        item = 'a' + i;
        putitem(item,array1);
        printf("productor put item in a1: %c\n",item);
        pthread_cond_signal(&waita1_full_buffer);
        pthread_mutex_unlock(&mutex1);
    }
    return NULL;
}
void* computer(void* arg)
{
    int i;
    char item;
    for(i = 0; i < ITEM_COUNT; i++){
        pthread_mutex_lock(&mutex1);
        while(buffer_is_empty(array1))
            pthread_cond_wait(&waita1_full_buffer,&mutex1);
        item = getitem(array1);
        printf("    computer get item in a1: %c\n",item);

        pthread_mutex_lock(&mutex2);
        while(buffer_is_full(array2))
            pthread_cond_wait(&waita2_empty_buffer,&mutex2);
        item = item - 32;
        putitem(item,array2);
        printf("        computer put item in array2: %c \n",item);
        pthread_cond_signal(&waita2_full_buffer);
        pthread_mutex_unlock(&mutex2);

        pthread_cond_signal(&waita1_empty_buffer);
        pthread_mutex_unlock(&mutex1);
    }
    return NULL;
}
void* consumer(void* arg)
{
    int i;
```

```
    char item;
    for(i = 0; i < ITEM_COUNT; i++){
        pthread_mutex_lock(&mutex2);
        while(buffer_is_empty(array2))
            pthread_cond_wait(&waita2_full_buffer,&mutex2);
        item = getitem(array2);
        printf("            consumer get item in array2: %c\n",item);
        pthread_cond_signal(&waita2_empty_buffer);
        pthread_mutex_unlock(&mutex2);
    }
    return NULL;
}

int main()
{
    pthread_t computer_tid;
    pthread_t consumer_tid;
    pthread_mutex_init(&mutex1, NULL);
    pthread_mutex_init(&mutex2, NULL);
    pthread_cond_init(&waita1_empty_buffer, NULL);
    pthread_cond_init(&waita1_full_buffer, NULL);
    pthread_cond_init(&waita2_empty_buffer, NULL);
    pthread_cond_init(&waita2_full_buffer, NULL);
    pthread_create(&consumer_tid, NULL, consumer, NULL);
    pthread_create(&computer_tid,NULL,computer,NULL);
    productor(NULL);
    pthread_join(computer_tid, NULL);
    pthread_join(consumer_tid, NULL);
    return 0;
}
```

运行结果：

```
user@instant-contiki:~$ ./pc1
productor put item in a1: a
productor put item in a1: b
productor put item in a1: c
    computer get item in a1: a
        computer put item in array2: A
    computer get item in a1: b
        computer put item in array2: B
    computer get item in a1: c
        computer put item in array2: C
productor put item in a1: d
productor put item in a1: e
productor put item in a1: f
    computer get item in a1: d
            consumer get item in array2: A
            consumer get item in array2: B
            consumer get item in array2: C
        computer put item in array2: D
    computer get item in a1: e
        computer put item in array2: E
    computer get item in a1: f
        computer put item in array2: F
```

```
productor put item in a1: g
productor put item in a1: h
    computer get item in a1: g
            consumer get item in array2: D
            consumer get item in array2: E
            consumer get item in array2: F
        computer put item in array2: G
    computer get item in a1: h
        computer put item in array2: H
            consumer get item in array2: G
            consumer get item in array2: H
```

题目：pc2.c

题目要求：

- 功能和pc1.c相同，使用信号量解决

解决思路：

解决思路与上一题类似，不过把条件变量换成了信号量，设置六个信号量，每个数组三个：mutex，full，empty；mutex初始为1，full初始为0，empty初始为数组长度，wait把信号量-1并且等待，signal把信号量+1并且唤醒一个线程。

代码：

```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

#define CAPACITY 4
#define ITEM_COUNT 8

char array1[CAPACITY];
char array2[CAPACITY];
int in1, in2;
int out1, out2;

int buffer_is_empty(char* array)
{
    if(array == array1) return in1 == out1;
    else return in2 == out2;
}
int buffer_is_full(char* array)
{
    if(array == array1)
        return (in1 + 1) % CAPACITY == out1;
    else
        return (in2 + 1) % CAPACITY == out2;
}
char getitem(char* array)
{
```

```
    char item;

    if(array == array2){
        item = array2[out2];
        out2 = (out2 + 1) % CAPACITY;
    }
    else{
        item = array1[out1];
        out1 = (out1 + 1) % CAPACITY;
    }
    return item;
}
void putitem(char item,char* array)
{
    if(array == array1){
        array1[in1] = item;
        in1 = (in1 + 1) % CAPACITY;
    }
    else{
        array2[in2] = item;
        in2 = (in2 + 1) % CAPACITY;
    }
}

typedef struct {
    int value;
    pthread_mutex_t mutex;
    pthread_cond_t cond;
}sema_t;

void sema_init(sema_t *sema, int value)
{
    sema->value = value;
    pthread_mutex_init(&sema->mutex, NULL);
    pthread_cond_init(&sema->cond, NULL);
}
void sema_signal(sema_t *sema)
{
    pthread_mutex_lock(&sema->mutex);
    ++sema->value;
    pthread_cond_signal(&sema->cond);
    pthread_mutex_unlock(&sema->mutex);
}
void sema_wait(sema_t *sema)
{
    pthread_mutex_lock(&sema->mutex);
    while (sema->value <= 0)
        pthread_cond_wait(&sema->cond, &sema->mutex);
    sema->value--;
    pthread_mutex_unlock(&sema->mutex);
}
sema_t a1mutex_sema;
sema_t a1empty_buffer_sema;
sema_t a1full_buffer_sema;
sema_t a2mutex_sema;
sema_t a2empty_buffer_sema;
```

```
sema_t a2full_buffer_sema;

void* productor(void* arg)
{
    int i;
    char item;
    for(i = 0; i < ITEM_COUNT; i++){
        sema_wait(&a1empty_buffer_sema);
        sema_wait(&a1mutex_sema);
        item = 'a' + i;
        putitem(item,array1);
        printf("productor put item in a1: %c\n",item);
        sema_signal(&a1mutex_sema);
        sema_signal(&a1full_buffer_sema);
    }
    return NULL;
}
void* computer(void* arg)
{
    int i;
    char item;
    for(i = 0; i < ITEM_COUNT; i++){
        sema_wait(&a1full_buffer_sema);
        sema_wait(&a1mutex_sema);
        item = getitem(array1);
        printf("    computer get item in a1: %c\n",item);

        sema_wait(&a2empty_buffer_sema);
        sema_wait(&a2mutex_sema);
        item = item - 32;
        putitem(item,array2);
        printf("        computer put item in array2: %c \n",item);
    sema_signal(&a2mutex_sema);
        sema_signal(&a2full_buffer_sema);

    sema_signal(&a1mutex_sema);
        sema_signal(&a1empty_buffer_sema);
    }
    return NULL;
}
void* consumer(void* arg)
{
    int i;
    char item;
    for(i = 0; i < ITEM_COUNT; i++){
        sema_wait(&a2full_buffer_sema);
        sema_wait(&a2mutex_sema);
        item = getitem(array2);
        printf("            consumer get item in array2: %c\n",item);
    sema_signal(&a2mutex_sema);
        sema_signal(&a2empty_buffer_sema);
    }
    return NULL;
}

int main()
```

```
{
    pthread_t computer_tid;
    pthread_t consumer_tid;
    sema_init(&a1mutex_sema, 1);
    sema_init(&a2mutex_sema, 1);
    sema_init(&a1empty_buffer_sema, CAPACITY - 1);
    sema_init(&a2empty_buffer_sema, CAPACITY - 1);
    sema_init(&a1full_buffer_sema, 0);
    sema_init(&a2full_buffer_sema, 0);

    pthread_create(&consumer_tid, NULL, consumer, NULL);
    pthread_create(&computer_tid, NULL, computer, NULL);
    productor(NULL);
    pthread_join(computer_tid, NULL);
    pthread_join(consumer_tid, NULL);
    return 0;
}
```

运行结果：

```
user@instant-contiki:~$ ./pc2
productor put item in a1: a
productor put item in a1: b
productor put item in a1: c
    computer get item in a1: a
        computer put item in array2: A
    computer get item in a1: b
        computer put item in array2: B
    computer get item in a1: c
        computer put item in array2: C
productor put item in a1: d
productor put item in a1: e
productor put item in a1: f
    computer get item in a1: d
            consumer get item in array2: A
            consumer get item in array2: B
            consumer get item in array2: C
        computer put item in array2: D
    computer get item in a1: e
        computer put item in array2: E
    computer get item in a1: f
        computer put item in array2: F
productor put item in a1: g
productor put item in a1: h
    computer get item in a1: g
            consumer get item in array2: D
            consumer get item in array2: E
            consumer get item in array2: F
        computer put item in array2: G
    computer get item in a1: h
        computer put item in array2: H
            consumer get item in array2: G
            consumer get item in array2: H
```

总结

这几周的实验让我对操作系统中的多线程、多进程、信号量、管道、重定向的使用有了更深刻的认识；学习了linux系统的一些基本指令和操作方法；提高了C语言的编程水平，让自己认识到在没有好用的IDE下编写程序是多么的困难，以后还要继续努力。