实验2 进程间通信和命令解释器

要求4月28日晚上12:00之前交

实验目的

1. 进程间共享内存实验，初步了解这种进程间通讯
2. 实现简单的shell命令解释器：了解程序运行。

1.进程间共享内存实验

- 完成课本第三章的练习3.10的程序
  - 共享内存的系统调用的使用例见课本

2.实现shell的要求

- 完成课本上第三章的项目：实现shell。除此之外满足下面要求：
  - 实现程序的后台运行

实验指导

1.进程间共享内存

- 共享内存的系统调用(参考Linux共享内存.pdf)
- **创建或打开共享存储区(shmget)：依据用户给出的整数值key，创建新区或打开现有区，返回一个共享存储区ID。**
- **连接共享存储区(shmat)：连接共享存储区到本进程的地址空间，返回共享存储区首地址。父进程已连接的共享存储区可被fork创建的子进程继承。**
- **拆除共享存储区连接(shmdt)：拆除共享存储区与本进程地址空间的连接。**
- **共享存储区控制(shmctl)：对共享存储区进行控制。如：共享存储区的删除需要显式调用shmctl(shmid, IPC_RMID, 0);**
- **用系统调用ftok给出IPC键值key：保证同一个用户的两个进程获得相同的IPC键值key（本页备注中有详细说明）**

共享内存各个函数使用的例

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>
int main()
{
    /* the identifier for the shared memory segment */
    int segment_id;
    /* a pointer to the shared memory segment */
    char* shared_memory;
    /* the size (in bytes) of the shared memory segment */
    const int segment_size = 4096;
    /** allocate  a shared memory segment */
    segment_id = shmget(IPC_PRIVATE, segment_size, S_IRUSR | S_IWUSR);
    /** attach the shared memory segment */
    shared_memory = (char *) shmat(segment_id, NULL, 0);
    printf("shared memory segment %d attached at address %p\n", segment_id, shared_memory);
    /** write a message to the shared memory segment   */
    sprintf(shared_memory, "Hi there!");
    /** now print out the string from shared memory */
    printf("*%s*\n", shared_memory);
    /** now detach the shared memory segment */
    if ( shmdt(shared_memory) == -1) {
            fprintf(stderr, "Unable to detach\n");
    }
    /** now remove the shared memory segment */
    shmctl(segment_id, IPC_RMID, NULL);
    return 0;
}
```

父子进程间共享内存使用的例

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define PERM S_IRUSR|S_IWUSR    //(见注1)
```

父子进程间共享内存使用的例（续1）

```
int main(int argc, char **argv)
{
    key_t shmid;
    char   *p_addr, *c_addr;
    pid_t pid;
    if(argc != 2) {
        fprintf(stderr, "Usage:%s\n\a", argv[0]);
        exit(1);
    }
    if( (shmid = shmget(IPC_PRIVATE, 1024, PERM)) == -1 )  { //(见注1)
        fprintf(stderr, "Create Share Memory Error:%s\n\a",
     strerror(errno));
        exit(1);
    }
```

父子进程间共享内存使用的例（续2）

```
pid = fork();
  if(pid > 0) {
     p_addr = shmat(shmid, 0, 0);
     memset(p_addr, '\0', 1024);
     strncpy(p_addr, argv[1], 1024);
     wait(NULL);
     exit(0);
  }
  else if (pid == 0){
     sleep(1);
     c_addr = shmat(shmid, 0, 0);
     printf("Client get %s\n", c_addr);
     exit(0);
  }
}
```

- (注1)

IPC_PRIVATE　　保证使用唯一ID(或key)

S_IRUSR | S_IWUSR　　使当前用户可以读写这个区域

2.实现shell的提示

- 实验要求：
  - 完成课本上第三章的项目：实现shell。除此之外满足下面要求：
    - 实现程序的后台运行。

main函数实现的建议

```
int main(void)
{
   char inputBuffer[MAX_LINE]; /* 这个缓存用来存放输入的命令*/
   int background;          /* ==1时，表示在后台运行命令，即在命令后加上'&' */
   char *args[MAX_LINE/2+1];/* 命令最多40个参数 */


   while (1){          /* 程序在setup中正常结束*/
      background = 0;
      printf(“ COMMAND->”); //输出提示符，没有换行，仅将字符串送入输出缓存
                            //若要输出输出缓存内容用fflush(stdout);头文件stdio.h
     setup(inputBuffer,args,&background);     /* 获取下一个输入的命令 */

     /* 这一步要做:
      (1) 用fork()产生一个子进程
      (2) 子进程将调用execvp()执行命令,即 execvp(args[0],args);
      (3) 如果 background == 0, 父进程将等待子进程结束, 即if(background==0) wait(0);
          否则，将回到函数setup()中等待新命令输入.
     */
   }
}
```

execvp()的使用

函数原型： execvp(char cmd[], char * args[]);

若键盘输入的命令为：

ls -l -R

则这样调用execvp()来执行命令“ls -l -R”

args[0]=“ls”；

args[1]=“-l”；

args[2]=“-R”； args[3]=NULL;

execvp(args[0],args);

下面的setup()程序就是将输入的串存入args.

实现shell的部分程序

```
#include <stdio.h>
#include <unistd.h>
#define MAX_LINE 80
 /* 每次输入的命令规定不超过80个字符 */
/* * setup() 用于读入下一行输入的命令，并
   将它分成没有空格的命令和参数存于数组
   args[]中，
 * 用NULL作为数组结束的标志
 */
```

setup函数的实现

```
/*
 * setup() 用于读入下一行输入的命令，并将它分成没有空格的命令和参数存于数组args[]中，
 * 用NULL作为数组结束的标志
 */

void setup(char inputBuffer[], char *args[],int *background)
{
   int length, /* 命令的字符数目 */
       i,     /* 循环变量 */
       start,  /* 命令的第一个字符位置 */
       ct;    /* 下一个参数存入args[]的位置 */

   ct = 0;

   /* 读入命令行字符，存入inputBuffer */
   length = read(STDIN_FILENO, inputBuffer, MAX_LINE);

   start = -1;
   if (length == 0) exit(0);          /* 输入ctrl+d，结束shell程序 */
   if (length < 0){
      perror("error reading the command");
      exit(-1);          /* 出错时用错误码-1结束shell */
   }
```

```
/* 检查inputBuffer中的每一个字符 */
   for (i=0;i<length;i++) {
      switch (inputBuffer[i]){
         case ' ':
         case '\t' :            /* 字符为分割参数的空格或制表符(tab)'\t'*/
                  if(start != -1){
            args[ct] = &inputBuffer[start];
                     ct++;
                  }
          inputBuffer[i] = '\0'; /* 设置 C string 的结束符 */
                  start = -1;
                  break;

        case '\n':              /* 命令行结束 */
                  if (start != -1){
            args[ct] = &inputBuffer[start];
                     ct++;
                  }
          inputBuffer[i] = '\0';
          args[ct] = NULL; /* 命令及参数结束 */
                  break;

          default :          /* 其他字符 */
                  if (start == -1)
                     start = i;
          if (inputBuffer[i] == '&'){
                     *background  = 1;       /*置命令在后台运行*/
            inputBuffer[i] = '\0';
                  }
       }
    }
    args[ct] = NULL; /* 命令字符数 > 80 */
}
```