Lab1 Util

本lab主要内容是通过使用ulib和系统调用完成用户态的一些函数

Sleep:

根据lab要求，通过调用sleep系统调用来实现sleep。sleep函数即用户指定系统睡眠指定的时间周期。这道题没有要求做usage和错误处理，直接实现功能就可以了

首先读取额外参数:

```
// check param num

      if(argc<=1){
            printf("not enough param\n");
            exit(1);
      }

      //check the last char is number
      //char* mystring;
      //strcpy(mystring,argv[1]);
      int num = atoi(argv[1]);
```

然后根据用户指定的参数进行睡眠:

```
for(int i=0;i<num;i++){
            sleep(1);

      }
```

pingpong

这道题主要难点在于理解fork和管道。

> fork：将当前的进程的所有状态拷贝一份，并赋予一个新的进程号。区别父进程和子进程的方法在于fork后的返回值。若为0则是子进程，若为非0（子进程号）就是父进程

> 管道：管道本身是一个具有分离的读口和写口的文件。传输信息的方法为打开一个管道文件，然后fork进程，这样两个进程都拥有指向读口的指针和指向写口的指针。关掉一个读口指针和一个写口指针，就可以实现一个进程向另一个进程输出信息了。

因此，在这道题中使用了两个管道，一个是父进程向子进程传输字符，一个是子进程向父进程传输字符。

传输字符与输出代码如下:

```
if(childpid==0){
            //close unuse pipe
            close(PWCRpipefd[1]);
            close(PRCWpipefd[0]);
            read(PWCRpipefd[0],readbuffer,1);
            printf("%d: received ping\n",getpid());
            write(PRCWpipefd[1],writebuffer,1);
            exit(0);

    }
     else
    {
            close(PWCRpipefd[0]);
            close(PRCWpipefd[1]);
            write(PWCRpipefd[1],writebuffer,1);
            read(PRCWpipefd[0],readbuffer,1);
            printf("%d: received pong\n",getpid());
            exit(0);
    }
```

Primes

这道题要求通过使用fork和管道的方式实现一个素数筛。原理为：每个进程读到的所有数的第一个数是素数，然后向下一个进程传输所有模第一个数不是0的数。

在传输数据时可以通过传输int而不是char来避免额外的atoi修改：

```
 int tmp = -3;
while (read(pipefd[0], &tmp, sizeof(tmp)) > 0)
```

通过在每个进程中用一段数组来记录上个进程传输的数以及要向下一个进程传输的数。同时根据题目要求，父进程需要等待最后一个子进程完成传输后才倒序exit。因此，当某个进程作为子进程时它是不应该exit的（除非它是最后一个进程）；作为父进程则使用wait函数等待前面的子进程全关了才exit。

```
while (passnum>0)
    {
           if (pipe(pipefd))
          {
                printf("pipe create error");
                exit(-1);
          }
           int pid = fork();
           if (pid < 0)
          {
                // fail fork
                printf("fork error\n");
                exit(-1);
          }
           else if (pid > 0)
          {
                close(pipefd[0]);
                for (int i = 0; i < passnum; i++)
```

```
                {
                        write(pipefd[1], recordnum + i, sizeof(i));
                }
                close(pipefd[1]);
                wait((int *)0);
                exit(0);
        }
        else
        {
                close(pipefd[1]);
                int index = -1, curprime = -1;
                int tmp = -3;
                while (read(pipefd[0], &tmp, sizeof(tmp)) > 0)
                {

                        if(index<0){
                                curprime=tmp;
                                index++;
                        }
                        else if(tmp%curprime!=0){
                                recordnum[index++]=tmp;
                        }
                        // log(passnum,recordnum);
                }
                close(pipefd[0]);
                printf("prime %d\n", curprime);
                passnum = index;

        }

    }
```

Find

本质就是一个dfs递归搜索树。在这里，由于目录是树状结构，因此思路就是对于当前目录下的目录与文件进行遍历，若遍历到文件则将路径中的最后一段字符串（即文件名）与目标字符串进行匹配，若遍历到目录则打开目录并递归搜索。

注意，要记得排除.和..两个路径，否则会进入死循环。

```
switch (st.type)
    {
    case T_FILE:
        if (strcmp(getlastname(path), aimname) == 0)
        {
                printf("%s\n", path);
        }
        break;
    case T_DIR:
        if (strlen(path) + 1 + DIRSIZ + 1 > sizeof buf)
        {
                printf("path too long\n");
```

```
                break;
            }
             strcpy(buf, path);
             p = buf + strlen(buf);
             *p++ = '/';
             while (read(fd, &de, sizeof(de)) == sizeof(de))
            {
                if (de.inum == 0 || strcmp(de.name, ".") == 0 || strcmp(de.name, "..")==0)
                    continue;

                memmove(p, de.name, DIRSIZ);
                find_cur_path(buf, aimname);
            }
             break;
    }
```

xarg:

该题难点在于理解exec的使用与如何读取标准输入。

读取标准输入方法：

```
n = read(STDINFD, buffer, MAXLENGTH)
```

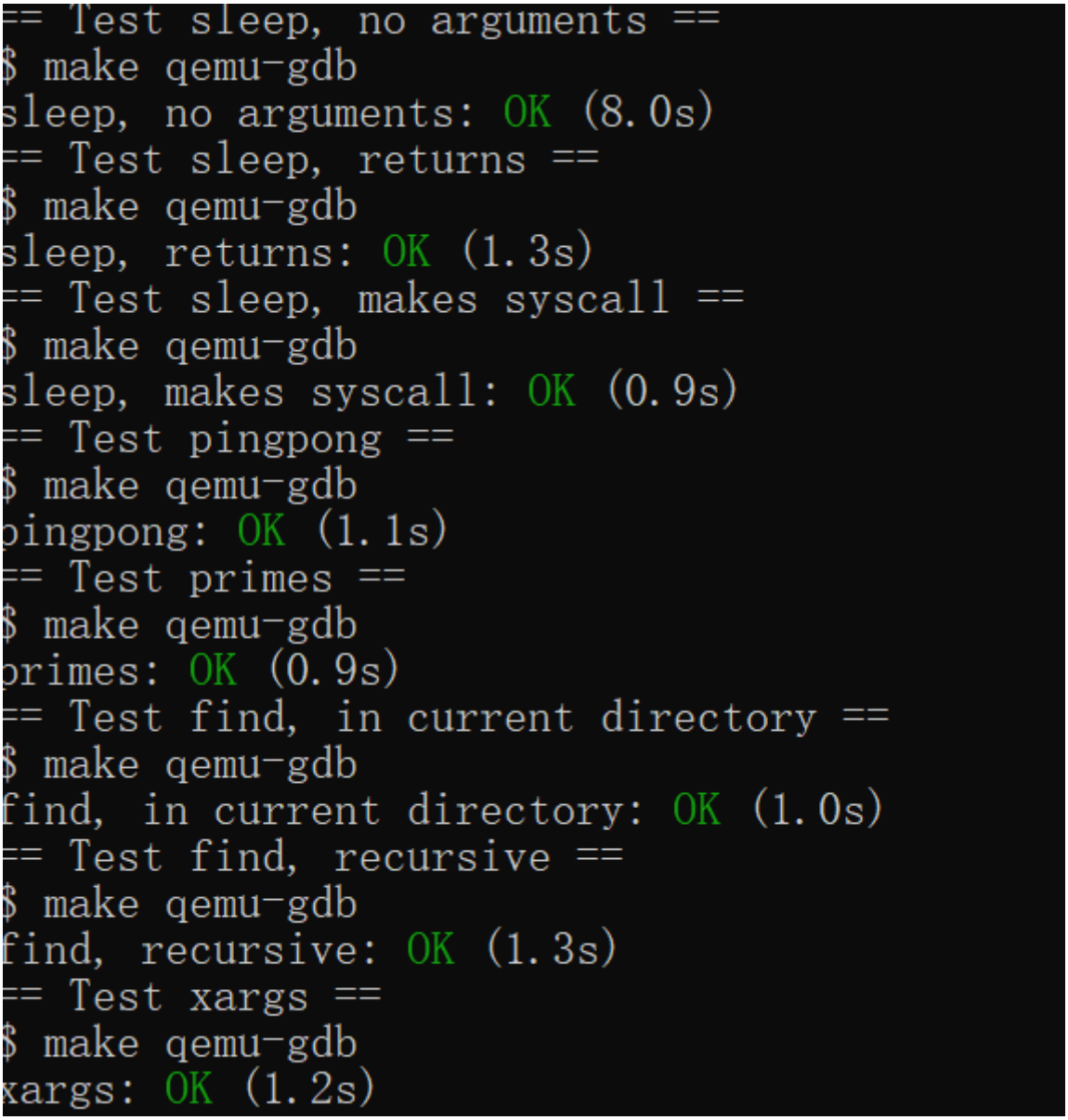
exec系统调用中，第一个参数是使用的系统调用名，第二个参数数组是系统调用的额外参数。注意，在处理前一个标准输入的空格和回车时，要把几个参数拆开。以及由于传递的参数为char**，需要用malloc来插入参数字符串。

通过为父进程fork一个子进程来执行exec来实现xargs

```
int cnt=0;
int nextargs=curargsnum;

// read param from buffer
for(int i=0;i<n;i++){
    if((buffer[i]==0||buffer[i]==' '||buffer[i]=='\n')&&cnt!=0){
        //make it become 0
        buffer[i]=0;

        //malloc a string to add to the param part
        char* tmp=(char*)malloc(cnt*sizeof(char)+1);
        strcpy(tmp,&buffer[i-cnt]);
        cnt=0;

        //add it to nextargc
        nextargc[nextargs++]=tmp;
    }
    else{
        cnt++;
    }
}

// deal with the last param
```

```
char* tmp = (char*) malloc(cnt*sizeof(char)+1);
buffer[n]=0;
strcpy(tmp,&buffer[n-cnt]);
nextargc[nextargs++]=tmp;

exec(argc[1], nextargc);
```

结果：



实验小结

不是很困难，理解lec1中讲的几个系统调用就可以了