北京郵電大學

Beijing University of Posts and Telecommunications

操作系统课设

操作系统课程设计

测试文档

组号：8

时间：2022/05/22

目录

计算机操作系统课程设计测试文档

1 测试概述

1.1 测试范围

我们所完成的 PD 操作系统的测试主要由各模块的单元测试、模块测试和系统整合后的整体测试组成。

1.2 测试人员及对应分工

姓名	模块	分工
张明昱	进程+整体	完成系统的整合和系统测试
董晓雨	进程	完成进程部分的测试
邵逸辰	文件	完成文件部分的测试
张博	UI	完成 UI 部分的测试
朱飞烟	设备	完成设备部分的测试
李佳奕	内存	完成内存部分的测试

1.3 测试目标（各模块的功能、性能、兼容等）

测试的目标是完成各模块的目标功能，并实现模块之间的合作：UI 采用图形化界面为用户提供使用渠道；进程管理处理内存中的进程，实现进程状态的转换、timer 和中断处理；内存管理实现对内存存储空间的管理；文件管理对文件和目录进行处理；设备管理维护 IO 设备的队列和磁盘的管理。

从用户的角度来讲，PD-OS 使用图形化的界面，为用户提供进程的创建与中断（将以运行数个可执行程序的形式实现）、文件的创建，修改和删除、系统内各个资源的监控。

内存模块的目标是：实现主要功能内存分配、内存释放、写入内存、访问内存和页面置换算法。其中内存分配、释放需要实现成功与失败情况，写入内存要求能够节约内存、连续写入，访问内存能够实现两页之内的跨页访问。

进程模块的目标是：实现进程的创建、运行、中止；管理进程就绪队列、阻塞队列并可查看进程队列；实现 CPU 的多种调度算法——包括 FCFS 算法、SJF 算法和多级反馈队列算法，其中多级反馈队列算法中包含 RR 算法；能够完成进程状态转换；能够正确应对中断。

1.4 测试环境

系统本身采用 VS2019 和 QT 插件进行编写，分工测试由组员在自己的电脑上完成，主流环境为 VS2019 和 GTest 测试框架，其中，UI 由于其特殊性，在 QT

Creator 上完成。

2 测试过程评估

2.1 测试总体评估

2.2 测试用例设计

2.2.1 功能模块 UI:

1. 测试对象：输出
功能描述：展示进程参数、展示进程队列，同步演示，文件树，读取文件，内存占用情况，磁盘占用情况，设备状态
测试方法：直接展示输出/调用<QTEST>
测试用例描述：
展示进程参数、展示进程队列、读取文件、内存占用情况、设备状态：展示在 QLabel 的文本框中；
同步演示：动态展示 QLabel；
文件树：通过 QTreeWidget 和 QTreeWidgetItem 展示树形结构；
磁盘占用情况：通过 QChart 根据不同磁盘调度算法画出。
期望结果：正确，便于用户理解且美观的输出。
2. 测试对象：输入
功能描述：中断，创建/修改目录，创建文件，运行文件，删除文件，写入文件，添加设备
测试方法：直接展示输出/调用<QTEST>
测试用例描述
中断：通过信号和槽传给其他模块；
创建/修改目录/创建文件/添加设备：通过 QLineEdit 文本框输入；
运行文件：运行当前目录下可执行文件 QTreeWidgetItem；
删除文件：删除当前目录下文件 QTreeWidgetItem；
写入文件：QTreeWidgetItem 选择可写入文件后通过 QTextEdit 文本框写入内容。
期望结果：能够传给其他模块正确的 QString 和信号。

2.2.2 内存管理

测试对象	测试方法	测试用例	期望
pageAlloc，页面分配函数	给定参数 pid 和申请页面大小，观察返回值和页面分配结果	pid=1,size=188 pid=2,size=200 pid=3,size=100	一二组可成功分配，三组分配失败
writeVirtual，写入内存函数	给定写入字符串，查看是否写入以及写入位置是否正确	pid1,str1 pid1,str1 pid1,str2 pid2,str3	第一组正常写入，第二组能够写在 1 后；第三组超出页面，只能部分写

		str 内容见代码	入；第四组能够正常写入空格、回车符等符号
pageAccess，访问页面内容，同时检查页面置换算法是否正确	指定参数：pid、起始位置、终止位置，查看输出内容	（1，90，104） (1, 1, 10) (2, 0, 10) (2, 100, 110)	全部能够正常读出，包括空格和回车；同时物理内存页面应按（0，1，-1），（0，1，-1）（0，1，2）（3，1，2）顺序输出
pageFree 释放页面	指定参数 pid，释放相应内存，查看返回值是否释放成功	pid=1 pid=2 pid=3	一二组成功释放，三组释放失败

2.2.3 设备管理

1. 添加设备

(1) 功能描述：用户选择添加设备，并输入设备名称与设备的传输速率（没有传输速率则填 0），若设备数量没有到达上限，并且该设备还未在设备列表中，则添加成功。

(2) 测试方法：添加 main 函数，在 main 函数中循环调用添加设备函数进行测试。

```
int main(){
    //test AddDevice
    DeviceSystem devicesys;
    devicesys.initIO();
    while (true) {
        devicesys.createDevice();
        cout<<devicesys.showDevices();
    }
```

(3) 测试用例描述：添加一个新设备 printer；添加一个已有设备 mouse；……；添加第 13 个设备。

(4) 期望结果：添加 printer 成功；添加失败，出现提示信息该设备已存在；……；添加失败，出现提示信息设备已满。

2. 进程请求设备

(1) 功能描述：进程请求使用某个设备，将该请求的具体信息（进程的 pid 和请求使用该设备的时长）存入该设备的请求队列中。

(2) 测试方法：利用 visual studio2019 添加 UnitTest 进行测试。

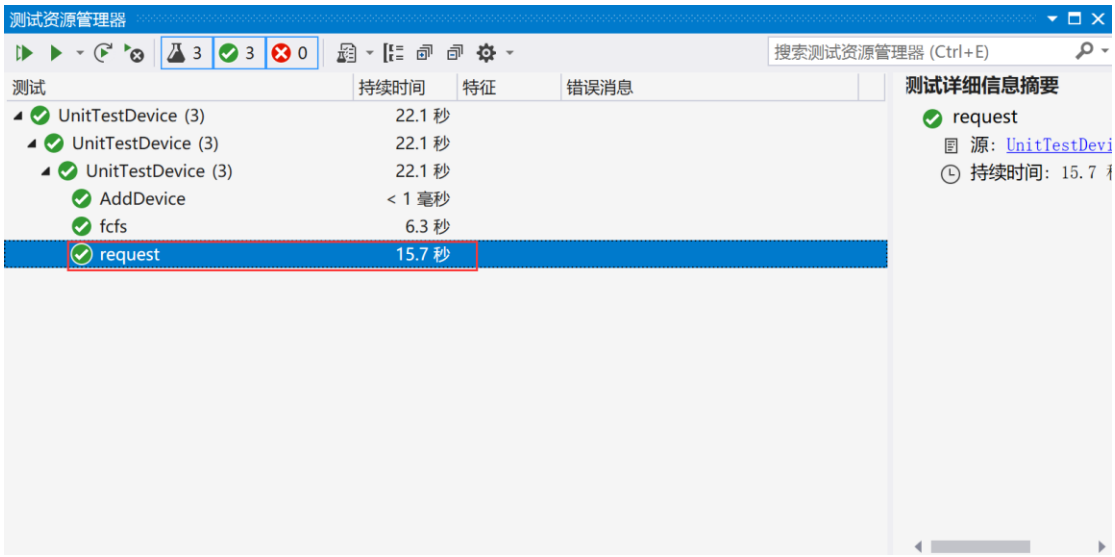
测试程序为：

```
TEST_METHOD(request)
{
    devicesys.initIO();
    for (int i = 0; i < 10; i++) {
        int id = rand();
        int device = rand() % devicesys.DEVICENUM;
        double time = rand();
        devicesys.requestDevice(id, devicesys.devices[device].deviceName, time);
        int retID = devicesys.devices[device].request[devicesys.REQUESTNUM[device]].pid;
        Assert::AreEqual(id, retID);
        double retTime = devicesys.devices[device].request[devicesys.REQUESTNUM[device]].ioTime;
        Assert::AreEqual(time, retTime);
    }
}
```

利用 for 循环，一次测试 10 个用例，提高效率。

(3) 测试用例描述：利用随机数，产生对一个随机设备请求，将随机生成的 pid 和 iotime 存入该设备的请求队列中。再通过 Assert::AreEqual（）判断程序是否正确进行。

(4) 期望结果：



3. FCFS 调度算法

(1) 功能描述： FCFS 调度函数的返回值是该设备的请求队列中的第一个的请求的 pid，来达到先来先服务的目的。

(2) 测试方法：利用 visual studio2019 添加 UnitTest 进行测试。

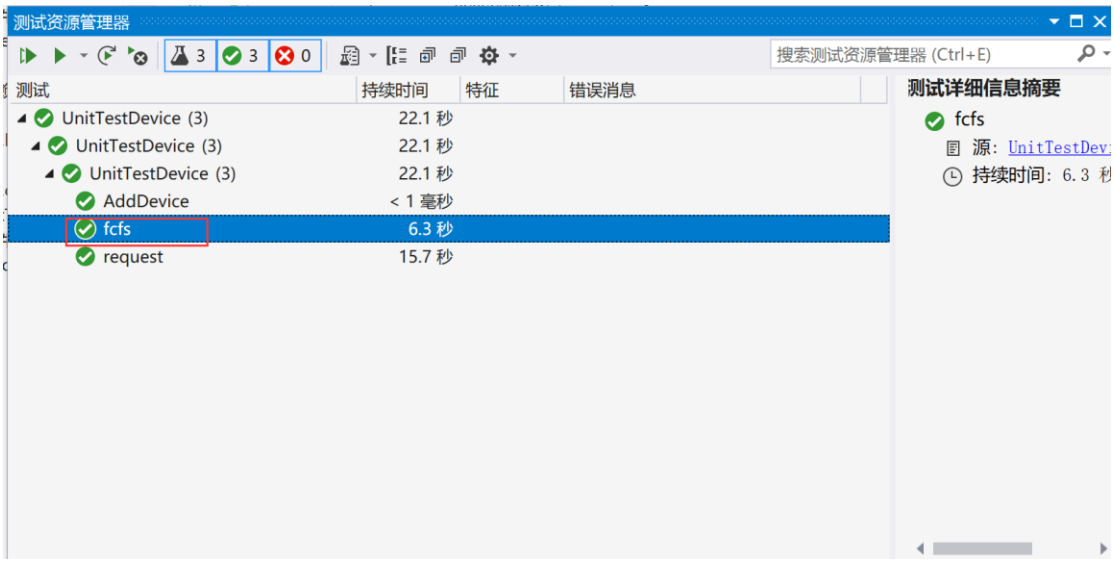
测试程序为：

```
TEST_METHOD(fcfs)
{
    devicesys.initIO();
    for (int i = 0; i < 10; i++) {
        int id = rand();
        int device = rand() % devicesys.DEVICENUM;
        double time = rand();
        devicesys.REQUESTNUM[device] = 1;
        devicesys.devices[device].request[devicesys.REQUESTNUM[device] - 1].pid = id;
        devicesys.devices[device].request[devicesys.REQUESTNUM[device] - 1].ioTime = time;
        int ret = devicesys.FCFS(devicesys.devices[device].deviceName);
        Assert::AreEqual(id, ret);
    }
}
```

利用 for 循环一次测试 10 个用例，提高效率。

(3) 测试用例描述：利用随机数，产生对一个随机设备请求，将随机生成的 pid 和 iotime 存入该设备的请求队列中。然后对该设备运行 FCFS 函数，通过 Assert::AreEqual（）判断 FCFS 函数的返回值是不是请求队列的第一个请求的 pid。

(4) 期望结果：



2.2.4 磁盘管理

1. 磁盘调度算法

(1) 功能描述： 分别实现 FCFS、SSTF、SCAN、LOOK、C-SCAN、C-LOOK 这六种磁盘调度算法。

(2) 测试方法： 添加 initDisk 函数，在函数中给磁盘请求赋值，运行程序观察六中磁盘调度算法是否正确调度。

initDisk 函数为：

```
void Disk::initDisk(){
    disk.trackNum=200;
    srand((unsigned)time(NULL));
    disk.curHead = rand() % 200;
    cout << disk.getHead() << endl;
    disk.outputNum=0;
    disk.seekQueueNum=rand()%20;
    for (int i = 0; i < disk.seekQueueNum; i++) {
        disk.seekQueue[i] = rand() % 200;
    }
```

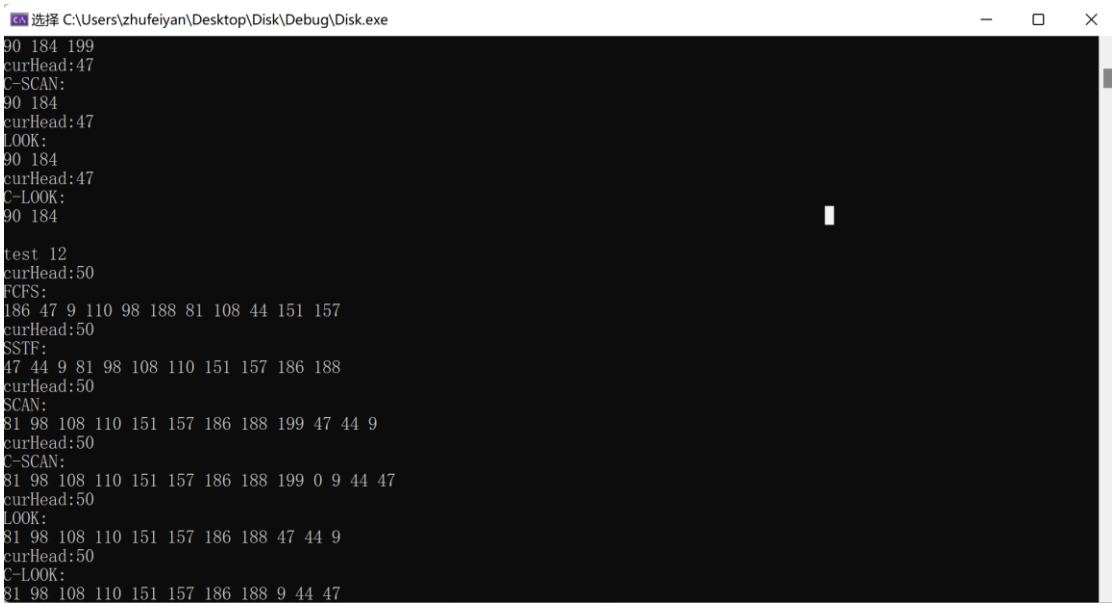
可以在 main 函数中用 for 循环提高测试效率。

```
int main(void){
    for (int i = 0; i < 10; i++) {
        cout << endl << "test " << i << endl;
        disk.FCFS();
        disk.SSTF();
        disk.SCAN();
        disk.C_SCAN();
        disk.LOOK();
        disk.C_LOOK();
    }
    system("pause");
    return 0;
}
```

(3) 测试用例描述：利用随机数，先随机产生当前磁头所在的位置，然后随机产生磁盘请求的个数，最后循环对每一个磁盘请求随机赋值。（总磁道数为 200）

(4) 期望结果：运行程序得到对该磁盘请求队列的经过六种调度算法后的调度顺序。

例如：



2.2.5 进程管理

主要测试函数：

- void runProcess(int pid); //运行进程
- void fork(int pid); //参数：父进程 pi
- void FCFS(QUEUE QueueReady); //first come first served
- void SJF(); //shorest job first
- void MLFQ(); //multilevel feedback queue
- int interruptTimeSlice(double timeslice, PCB* pcb);

测试对象	测试方法	测试用例	期望
fork(int pid) 父进程创建子进程	在同一个进程中设置多个 fork 指令，并同时创建多个类似进程来观察执行结果	pid=1,fork 三次 pid=2,fork 十次 pid=3,fork200 次	打印三个父进程的子进程列表可以得到所有子进程 pid；随机挑选子进程打印父进程 id 查看匹配情况；当父进程结束后查看其所有进程状态全部为 Terminated
FCFS(QUEUE QueueReady) CPU 调度算法——FCFS	调度算法选择 FCFS，添加多个进程，开始调度，观察调度是否正确	依次创建进程 pid1，pid2，pid3，pid4……	进程执行顺序依次为 pid1，pid2，pid3，pid4……
SJF(); CPU 调度算法——SJF 支持抢占与非抢占两种方式，用全局变量 isPreemptive 来判断	调度算法选择 SJF，isPreemptive 分别设置 0 和 1，添加多个进程，开始调度，观察调度	分别在 time=0，1，2，3 时创建进程 pid1，pid2，pid3，pid4，其运行时间分别为	非抢占模式下，进程执行顺序为 pid1，pid3，pid2，pid4；

	是否正确	5，5，1，5	抢占模式下进程执行顺序为 pid1， pid3，pid1， pid2，pid4
MLFQ(); CPU 调度算法——多级反馈队列调度，三个队列优先级分别为 0，1，2；千年两个队列使用 RR 算法，优先级最低的队列使用 FCFS 调度算法	调度算法选择 MLFQ，添加多个不同优先级的进程，开始调度，观察调度是否正确	在初始添加优先级=0 的进程：pid1，pid2； 添加优先级=1 的进程：pid3，pid4； 添加优先级=2 的进程：pid5，pid6；	先运行 pid1，其时间片用尽后运行 pid2，pid2 时间片用尽后依次运行 pid3， pid4，pid1，pid2（都运行至时间片用尽），然后依次运行 pid5，6，3，4，1，2（都运行至进程生命周期结束）
interruptTimeSlice(double timeslice, PCB* pcb) 判断并处理时间片中断	调度算法选择 MLFQ，添加多个进程并赋值较长的运行时间，开始调度，观察当时间片用尽时能否正确执行中断程序	在初始添加优先级=0 的进程：pid1，pid2；	pid1 的时间片用尽后系统自动开始 pid2 的运行
runProcess(int pid) 运行进程	添加多个进程并运行，观察进程的每一条指令执行	将我们规定的所有指令全部执行	指令按照预期正常执行

	情况		

2.3 测试用例执行情况

2.3.1 文件管理

测试对象	测试功能	用例总数	通过数	错误数	未执行数
print_route	多层文件夹与打印路径	>64	64	0	0
new_file	新建并读写（没有删除，测试内存空间不足的情况）	256	256	0	0
old_file	对已经存在的文件进行读写或删除	256	0	0	256
print_file	查找某类文件	64	0	0	64
print_dir	打印文件树	10	10	0	0

2.3.2 UI 界面

测试对象	测试功能	用例总数	通过数	错误数	未执行数
UI	输出	18	18	0	0
UI	输入	14	12	0	2

2.3.3 内存管理

测试对象	测试功能	用例总数	通过数	错误数	未执行数
内存管理	页面分配	3	3	0	0
内存管理	页面写入	4	4	0	0
内存管理	页面访问	4	4	0	0
内存管理	页面释放	3	3	0	0

2.3.4 磁盘管理

测试对象	测试功能	用例总数	通过数	错误数	未执行数
设备管理	添加设备	10	10	0	0
设备管理	进程请求设备	100	100	0	0
设备管理	FCFS 调度算法	100	100	0	0
文件系统	磁盘调度算法	100	100	0	0

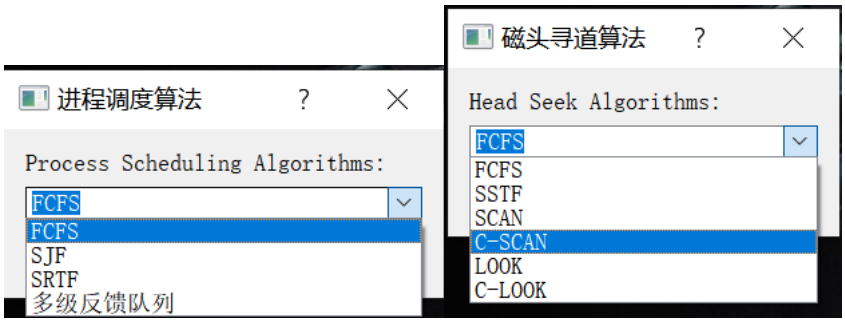
2.4 测试用例执行情况分析

2.4.1 文件管理

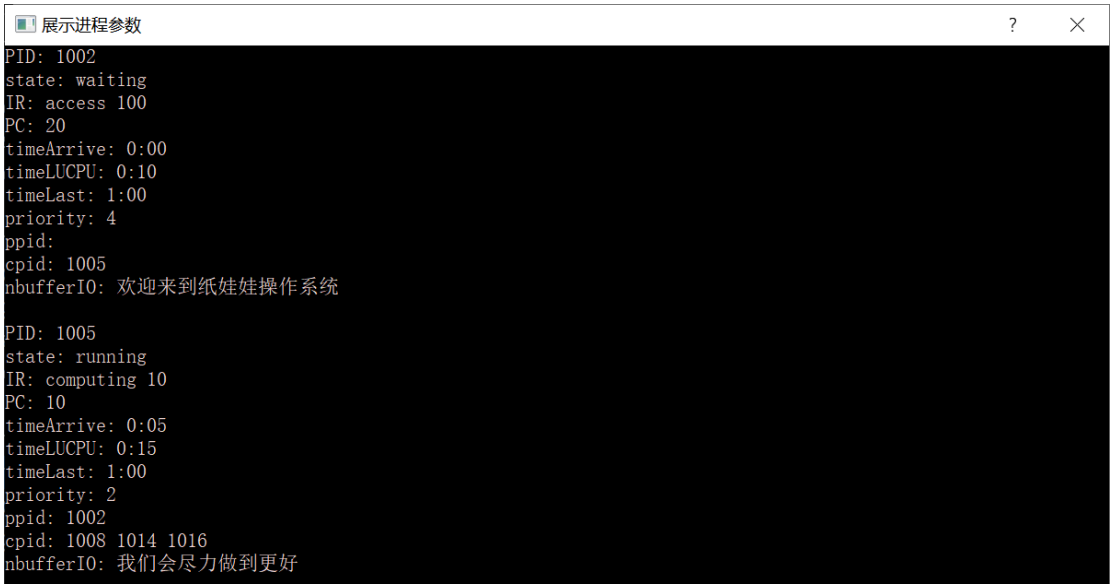
1. print_route
在根目录下，创建 64 个目录 newDir_0_0.f~newDir_63_0.f，每个目录下随机建立 0~7 层目录（命名为 newDir_i_1~7.f），然后打印路径；
记录目录层数，检验路径的“\”数量，若两者不符，则将该条记录输出到“test_record.txt”文件中。
2. new_file
新建文件，并在当前文件中进行读写。写入内容由函数随机生成，长度在 0~5000 之间，即可能需要多个磁盘块来存储。读取时对原函数稍作改动，原函数返回内存起止地址，测试中返回读取的字符串。
对比写入的内容和读取的字符串，若两者不符，则将文件内容、读写信息记录到“test_record.txt”文件中。
3. old_file
随机进入一个目录，随机选取一个文件，进行读写或删除操作
对于读写操作，验证方法同 2；对于删除操作，新建同名文件，若报错“已有同名文件”，则记录错误。
4. print_file
随机生成输入（etf/et/ef/tf/e/t/f），检测输出的 vector，若类型不符合输入，则记录错误。
5. print_dir
打印文件树，由于不太好找参考，只能人工比对。

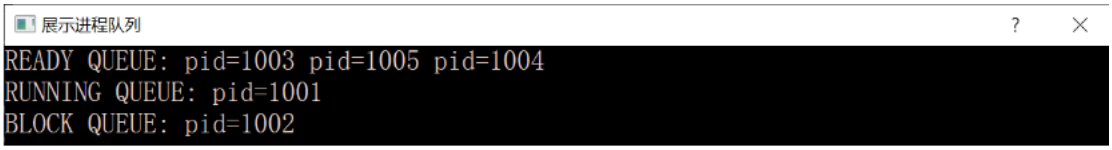
2.4.2 UI

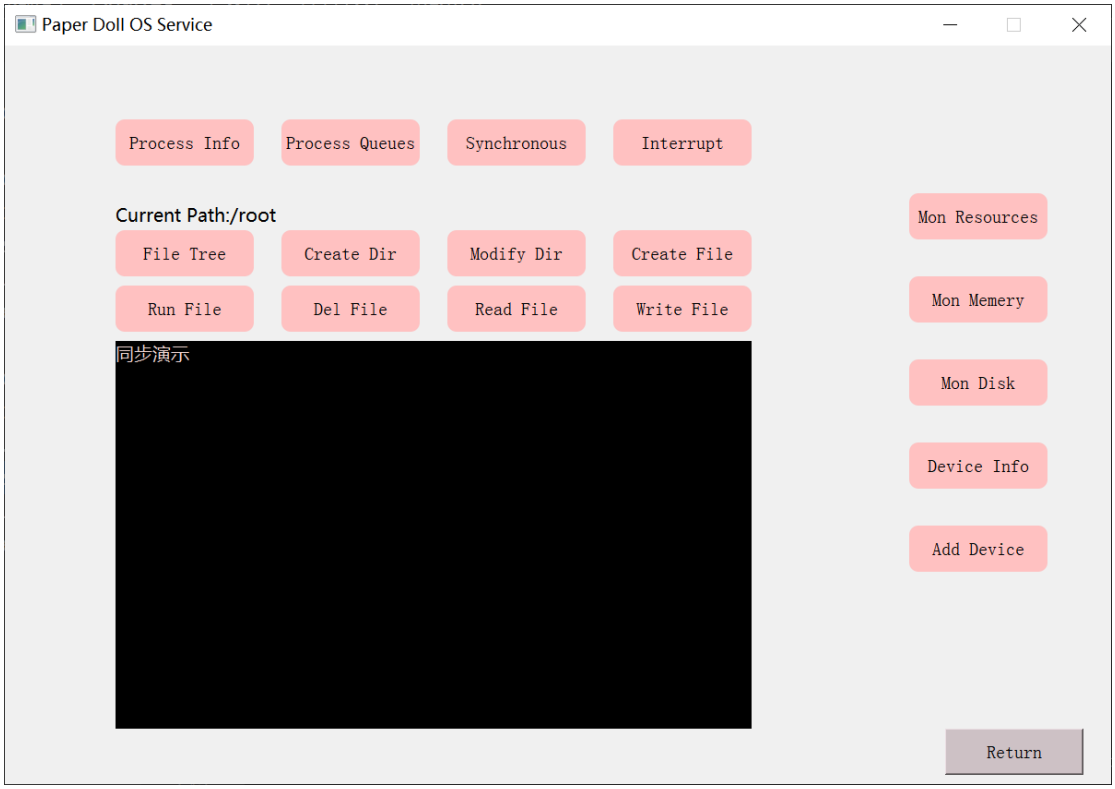
初始设置

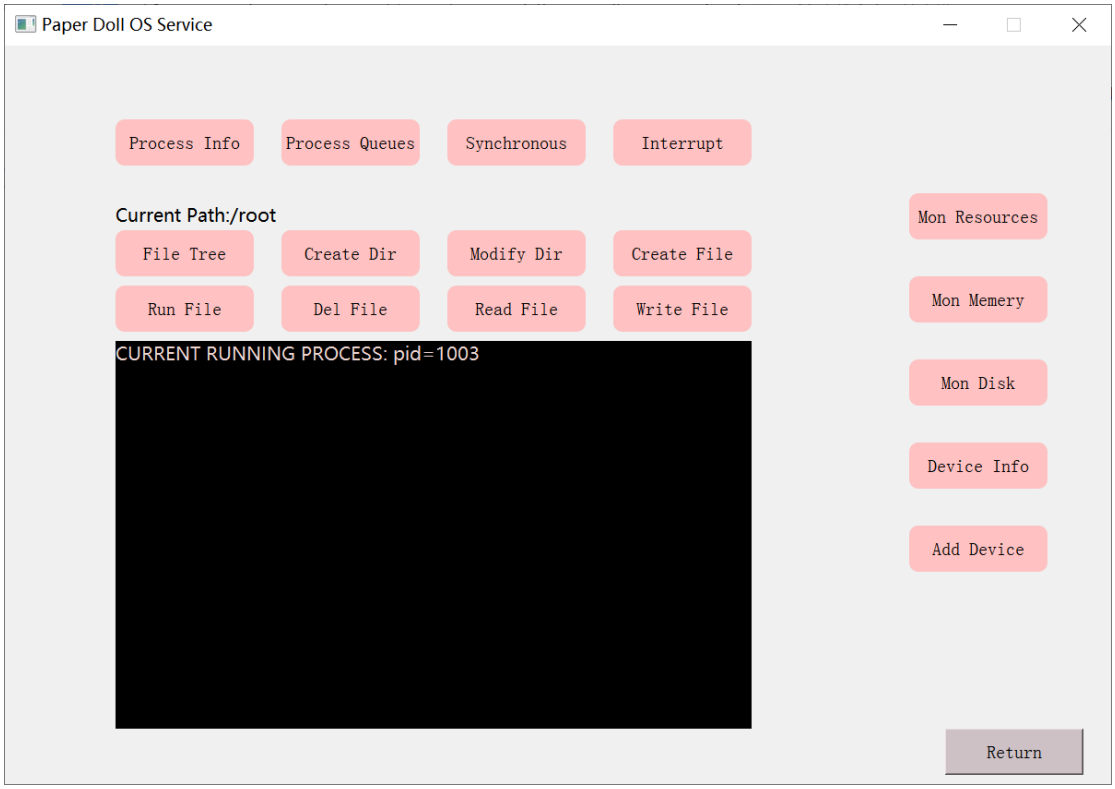

进程调度算法
Process Scheduling Algorithms:
FCFS
FCFS
SJF
SRTF
多级反馈队列
磁头寻道算法
Head Seek Algorithms:
FCFS
FCFS
SSTF
SCAN
C-SCAN
LOOK
C-LOOK

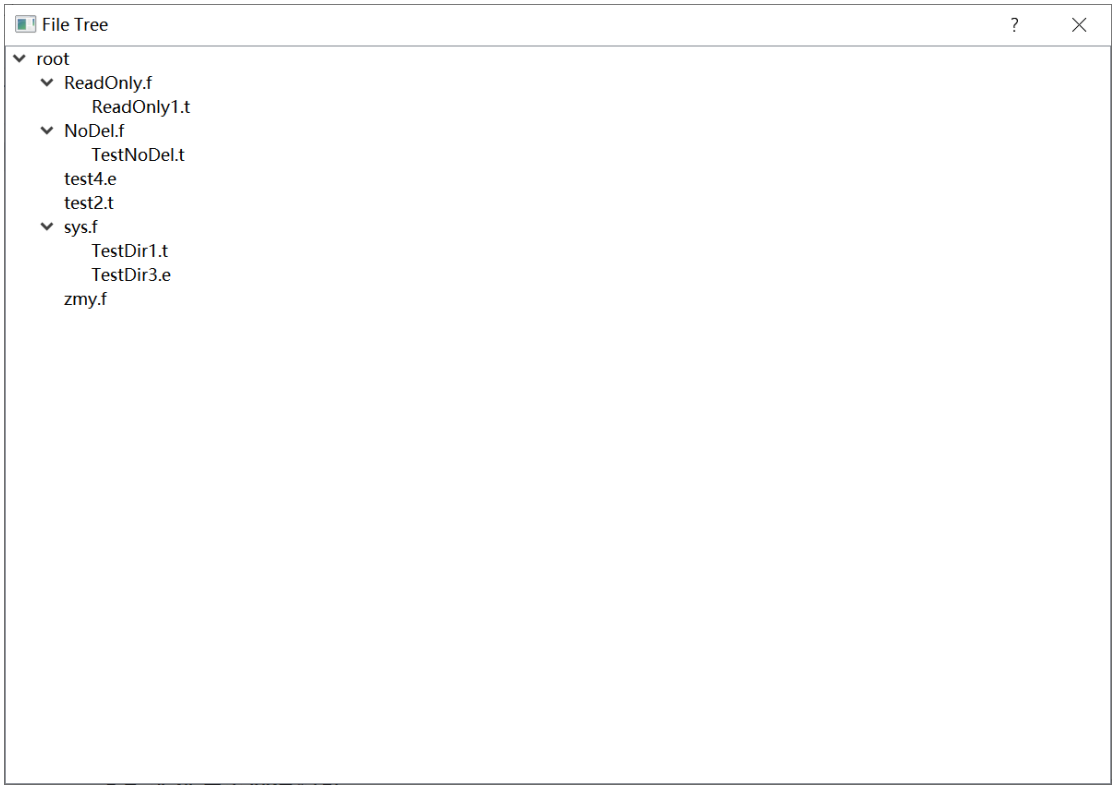
输入部分

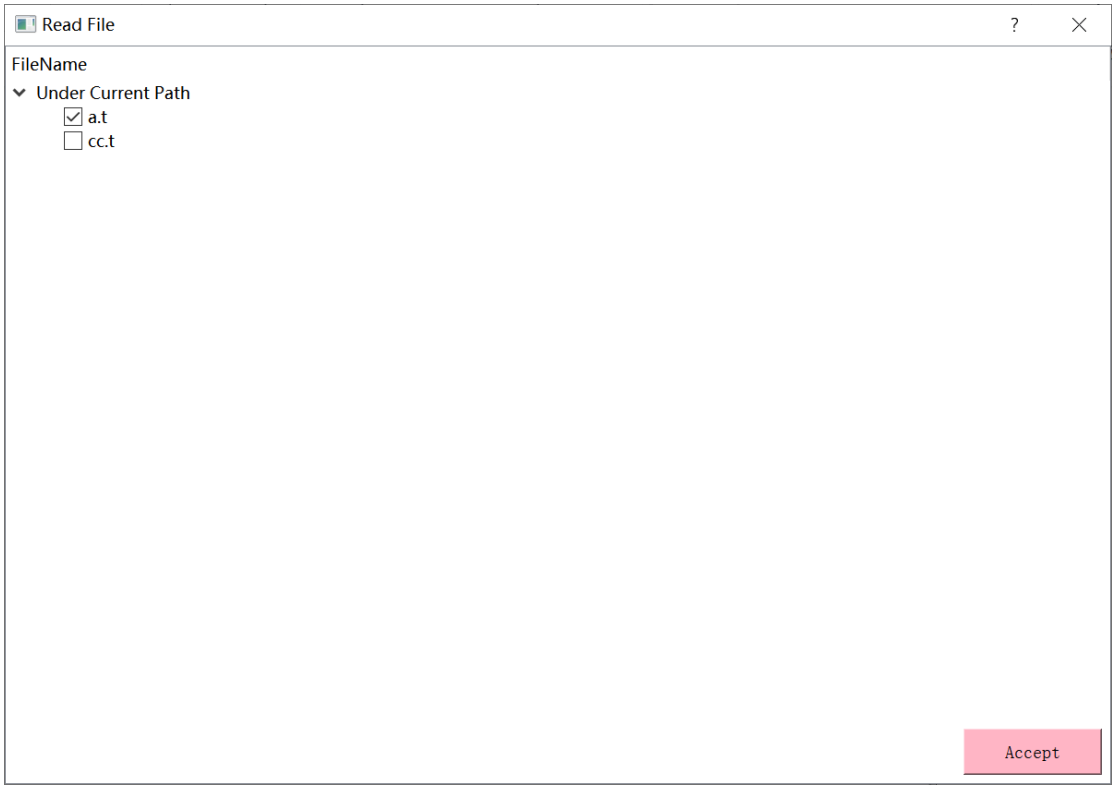

展示进程参数
PID: 1002
state: waiting
IR: access 100
PC: 20
timeArrive: 0:00
timeLUCPU: 0:10
timeLast: 1:00
priority: 4
ppid:
cpid: 1005
nbufferIO: 欢迎来到纸娃娃操作系统
PID: 1005
state: running
IR: computing 10
PC: 10
timeArrive: 0:05
timeLUCPU: 0:15
timeLast: 1:00
priority: 2
ppid: 1002
cpid: 1008 1014 1016
nbufferIO: 我们会尽力做到更好

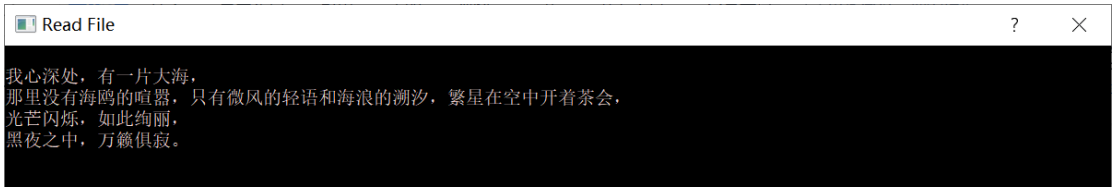

展示进程队列
READY QUEUE: pid=1003 pid=1005 pid=1004
RUNNING QUEUE: pid=1001
BLOCK QUEUE: pid=1002

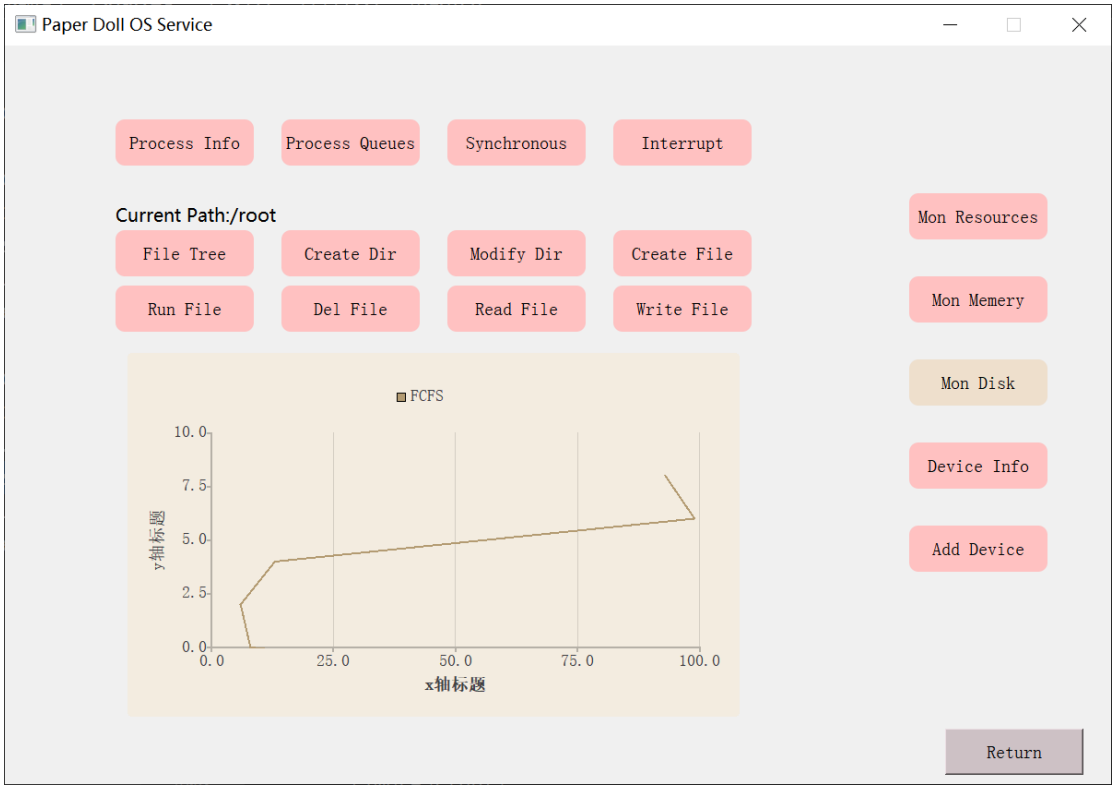

Paper Doll OS Service
Process Info
Process Queues
Synchronous
Interrupt
Current Path:/root
File Tree
Create Dir
Modify Dir
Create File
Run File
Del File
Read File
Write File
同步演示
Mon Resources
Mon Memery
Mon Disk
Device Info
Add Device
Return


Paper Doll OS Service
Process Info
Process Queues
Synchronous
Interrupt
Current Path:/root
File Tree
Create Dir
Modify Dir
Create File
Run File
Del File
Read File
Write File
CURRENT RUNNING PROCESS: pid=1003
Mon Resources
Mon Memery
Mon Disk
Device Info
Add Device
Return


File Tree
root
ReadOnly.f
ReadOnly1.t
NoDel.f
TestNoDel.t
test4.e
test2.t
sys.f
TestDir1.t
TestDir3.e
zmy.f

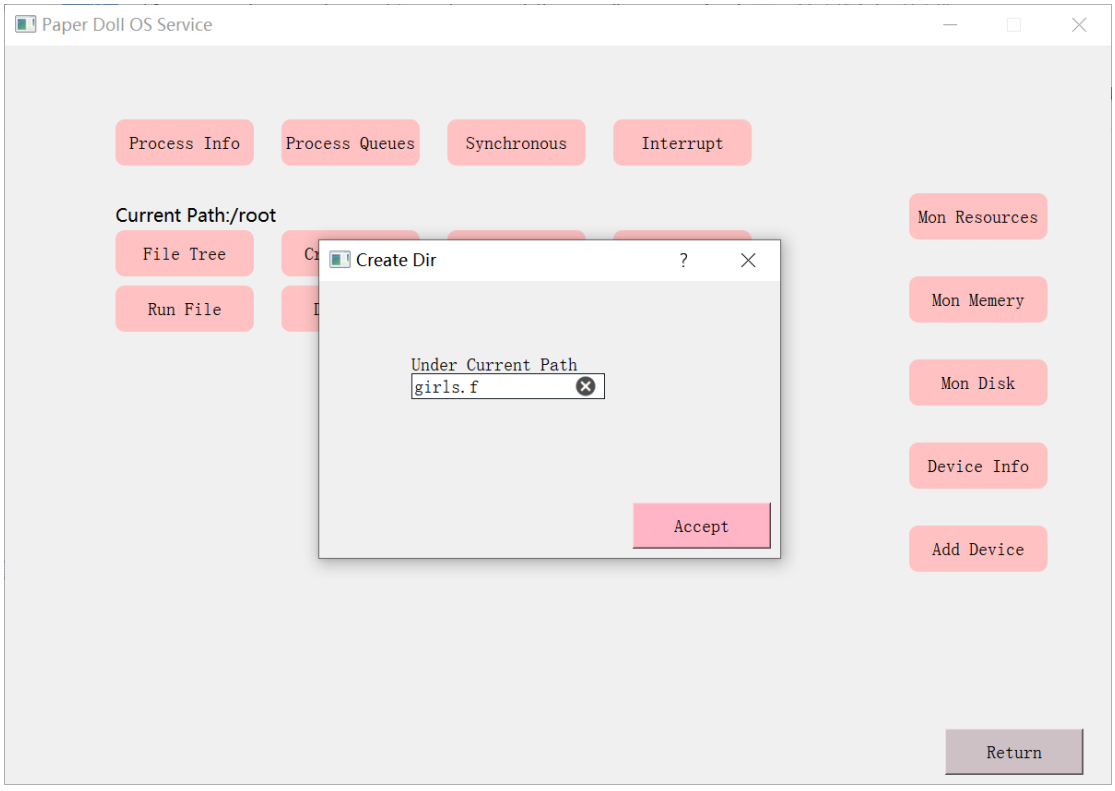

Read File
FileName
Under Current Path
a.t
cc.t
Accept

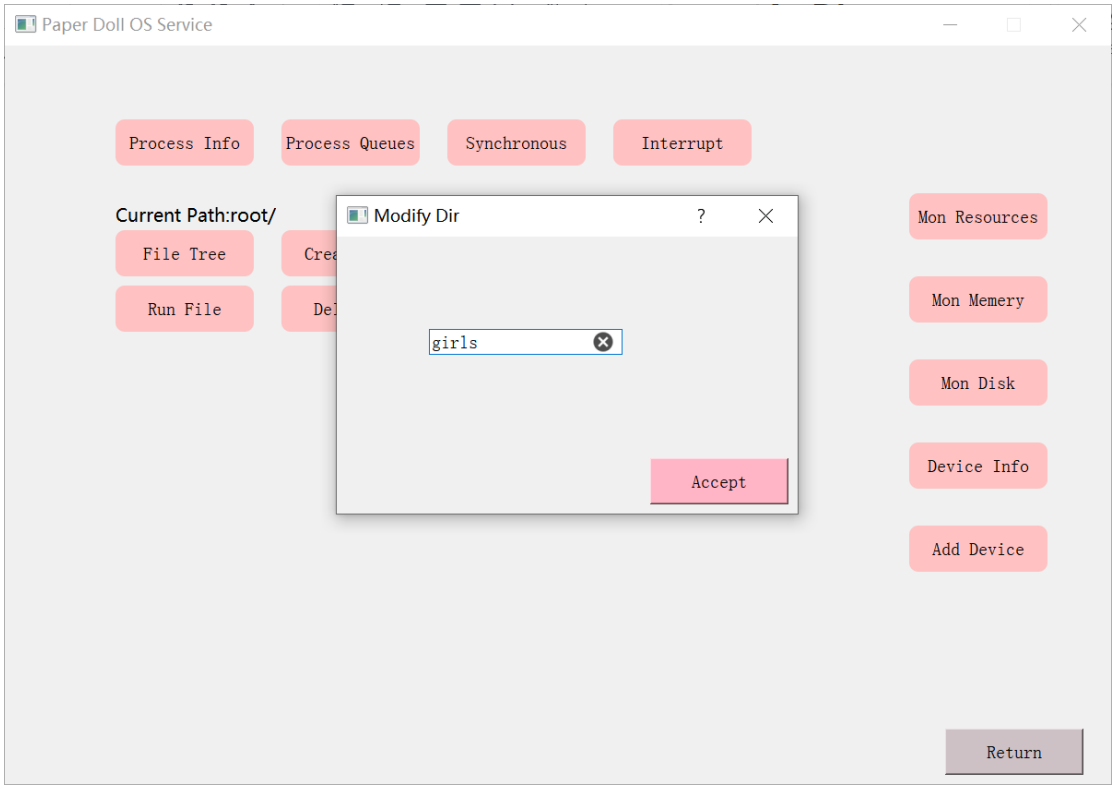

Read File
我心深处，有一片大海，
那里没有海鸥的喧嚣，只有微风的轻语和海浪的溯汐，繁星在空中开着茶会，
光芒闪烁，如此绚丽，
黑夜之中，万籁俱寂。

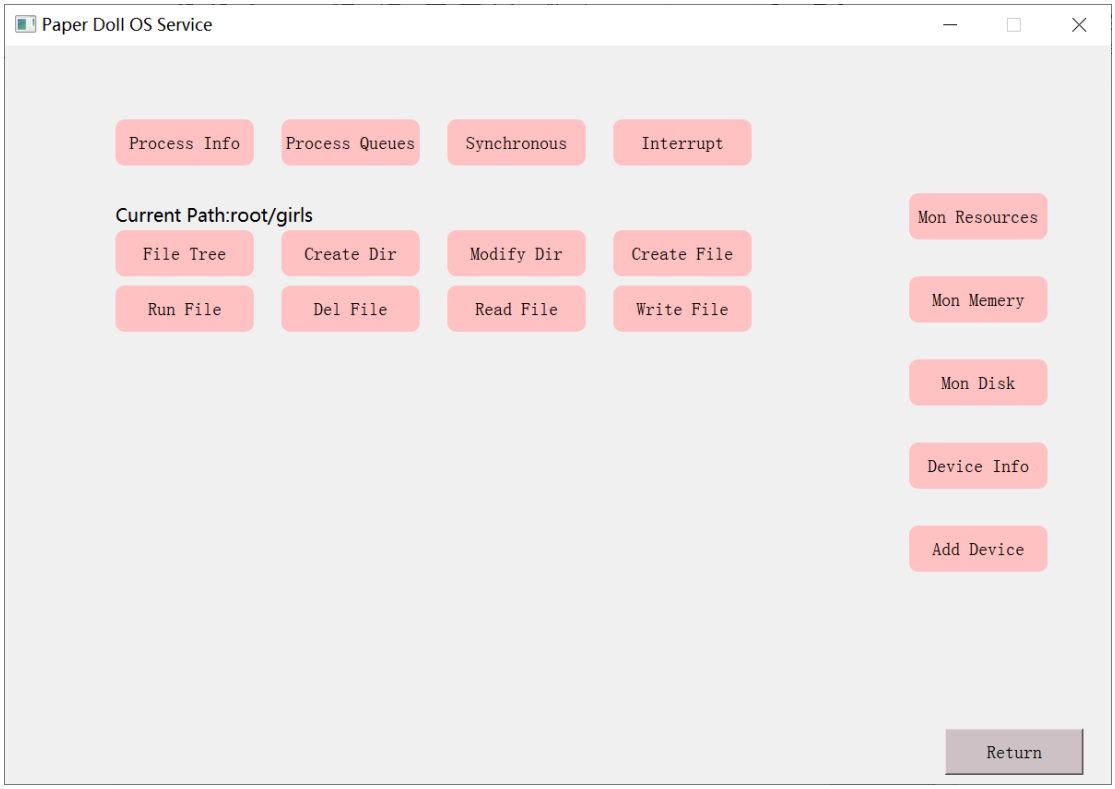

Paper Doll OS Service
Process Info
Process Queues
Synchronous
Interrupt
Current Path:/root
File Tree
Create Dir
Modify Dir
Create File
Run File
Del File
Read File
Write File
Mon Resources
Mon Memery
Mon Disk
Device Info
Add Device
FCFS
10.0
7.5
5.0
2.5
0.0
y轴标题
0.0
25.0
50.0
75.0
100.0
x轴标题
Return

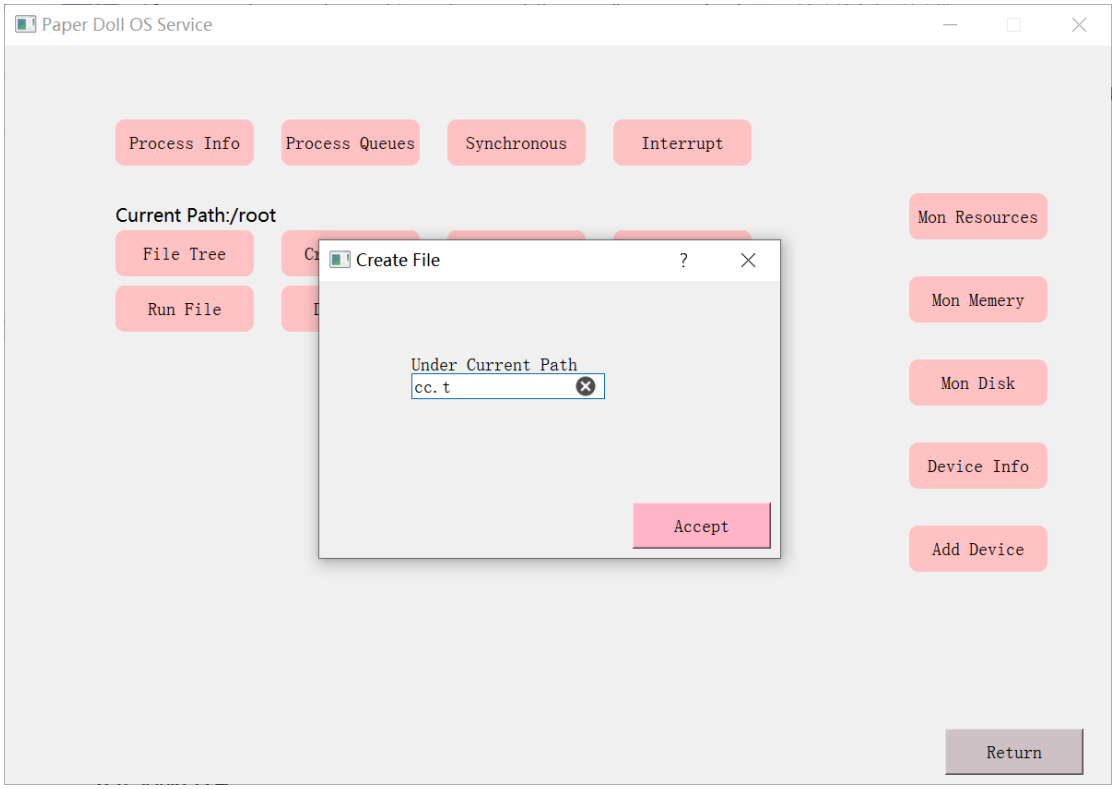

Device Status
键盘：pid=1002
屏幕：
打印机：

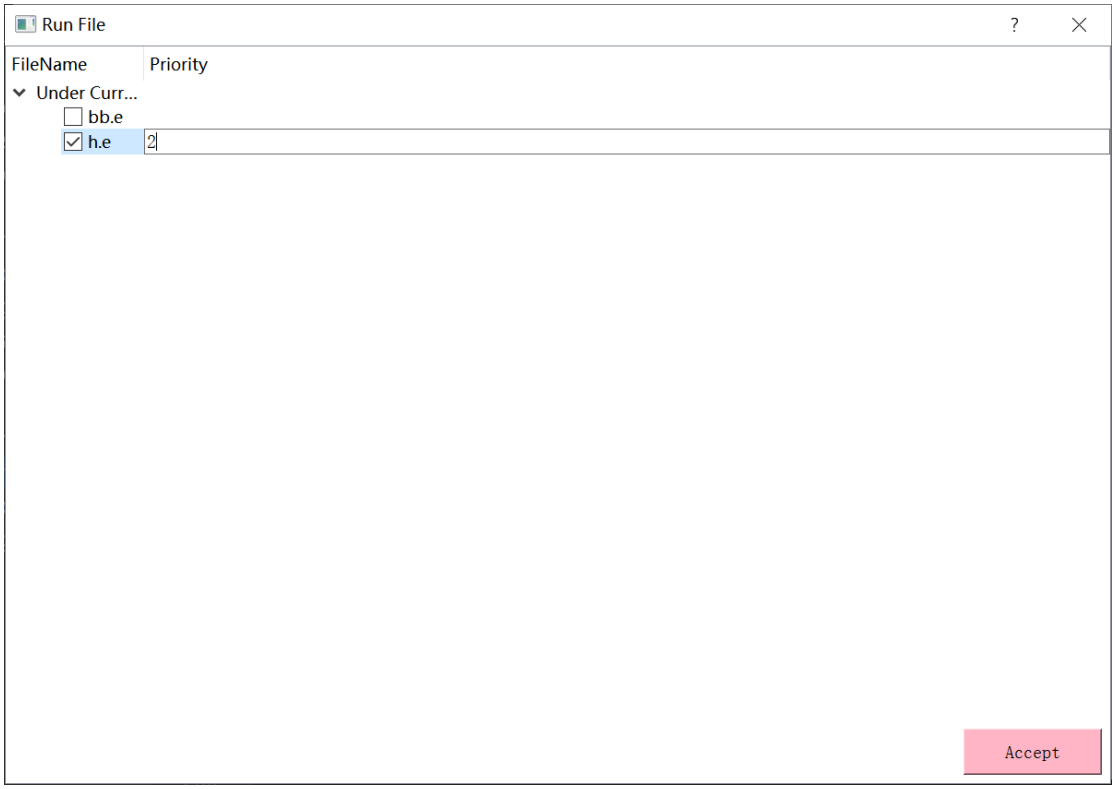
输出部分

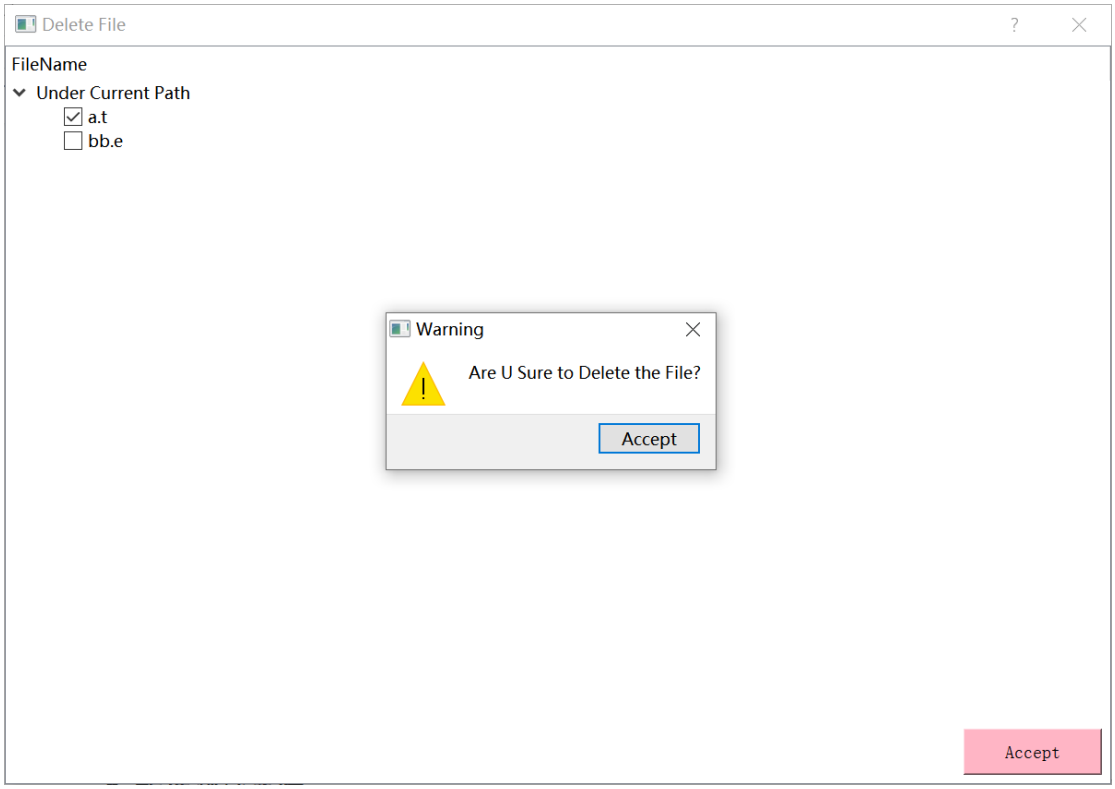

Paper Doll OS Service
Process Info
Process Queues
Synchronous
Interrupt
Current Path:/root
File Tree
Run File
Mon Resources
Mon Memery
Mon Disk
Device Info
Add Device
Create Dir
Under Current Path
girls.f
Accept
Return

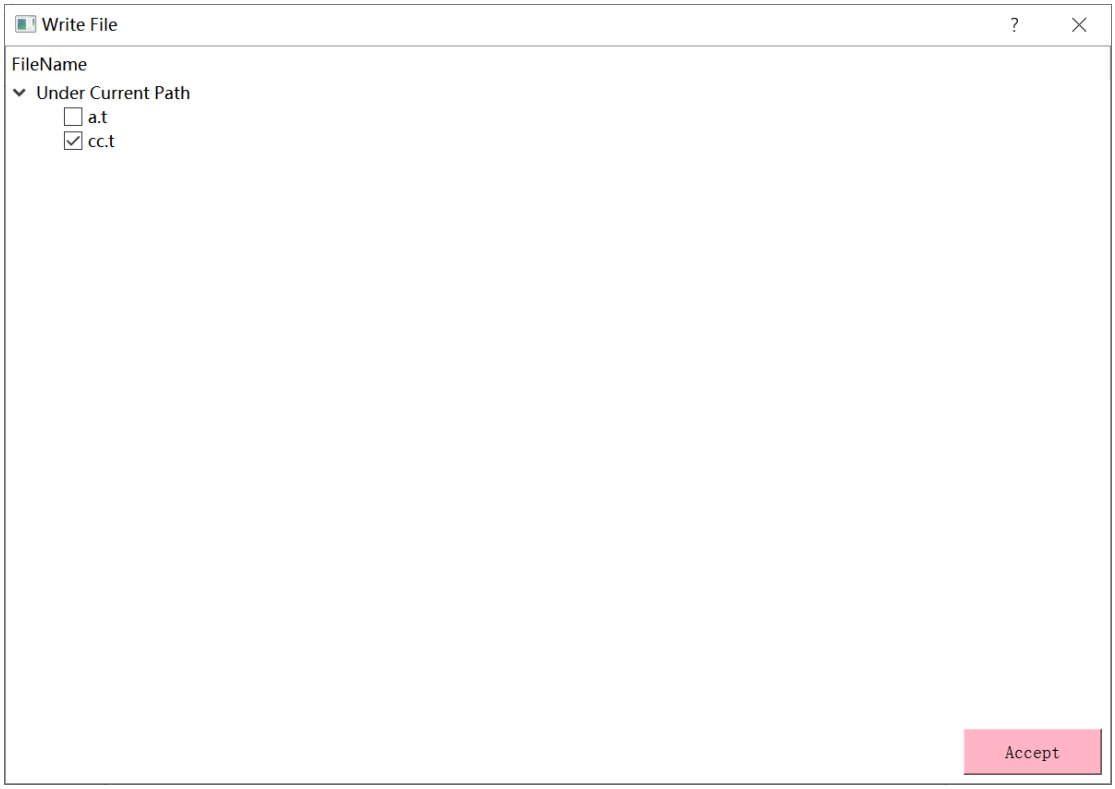

Paper Doll OS Service
Process Info
Process Queues
Synchronous
Interrupt
Current Path:root/
File Tree
Run File
Modify Dir
girls
Accept
Mon Resources
Mon Memery
Mon Disk
Device Info
Add Device
Return

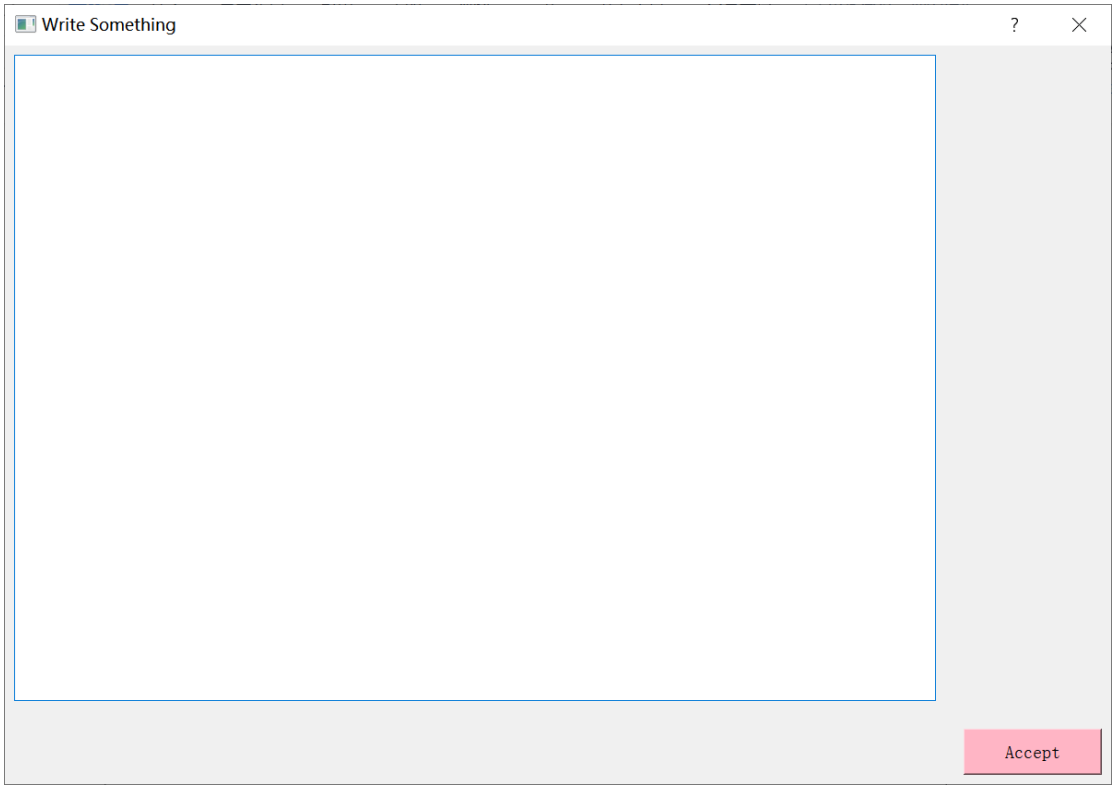

Paper Doll OS Service
Process Info
Process Queues
Synchronous
Interrupt
Current Path:root/girls
File Tree
Create Dir
Modify Dir
Create File
Run File
Del File
Read File
Write File
Mon Resources
Mon Memery
Mon Disk
Device Info
Add Device
Return

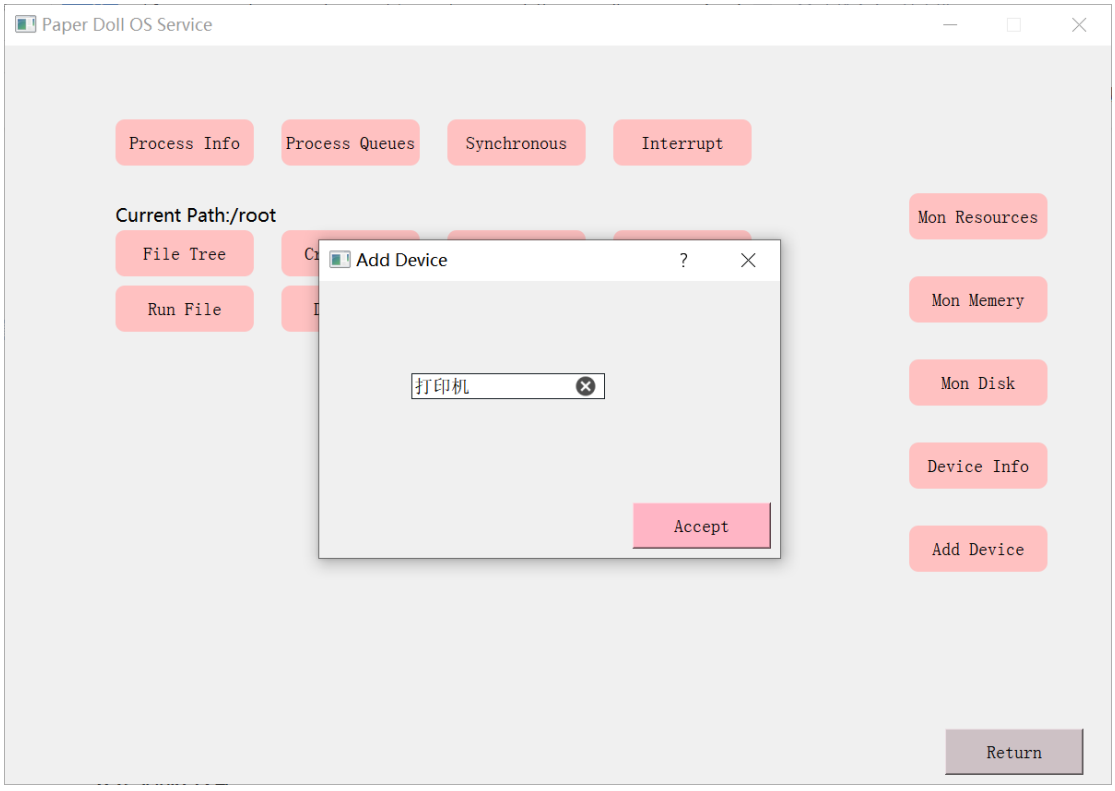

Paper Doll OS Service
Process Info
Process Queues
Synchronous
Interrupt
Current Path:/root
File Tree
Run File
Create File
?
Under Current Path
cc.t
Accept
Mon Resources
Mon Memery
Mon Disk
Device Info
Add Device
Return


Run File
?
FileName
Priority
Under Curr...
bb.e
h.e
2
Accept


Delete File
FileName
Under Current Path
a.t
bb.e
Warning
Are U Sure to Delete the File?
Accept
Accept


Write File
FileName
Under Current Path
a.t
cc.t
Accept





2.4.3 设备管理

设备管理单元的三大功能，添加设备、进程请求设备、FCFS 调度算法，都通过了一定量的测试，并在不同的情况下给出不同的提示信息，得到的运行结果与预期结果相符。

2.5 缺陷统计

2.6 缺陷分析

2.7 改进与未解决问题

2.7.1 文件管理
1.没有限制每个目录下的文件数
2.没有限制文件的最大长度（除非磁盘空间已满）
3.虽然形式上做到了按块读写，但是没有实现进程读取特定块（实际上，每次读写文件必须全部完成）
4.受限于本 OS 的功能，虽然可以实现多线程的读写锁，但并不会真正应用
5. 由于本系统最终为单线程执行，没有将磁盘寻道系统与文件系统很好地融合

3 软件测试总结

3.1 软件测试总结

由于时间原因，我们的测试工作并未能全部完成，系统整合已全部完成，但于各部分配合的完美运行还有一定差距，这大部分是由于我们低估了整合的难度。一方面是各部分之间配合密切，投入实际情况后与各部分的单独测试结果又有不同；另一方面是 QT creator 与 VS2019 的融合，使得前后端的交互也有一定难度。在接下来的时间内，我们一定会抓紧时间，努力完成程序的整合调试和测试。