电子科技大学 计算机科学与工程学院

实验报告

（实验）课程名称： 计算机操作系统

电子科技大学教务处制表

**电 子 科 技 大 学**

**实 验 报 告**

|  |  |
| --- | --- |
| **学生姓名：李逢君** | **学 号：2016060601010** |
| **一、实验室名称：主楼A2-413** | |
| **二、实验项目名称：系统化思维模式下计算机操作系统进程与资源管理设计** | |
| **三、实验原理：**  系统总体架构如图1所示，最右边部分为进程与资源管理器，属于操作系统内核的功能。该管理器具有如下功能：完成进程创建、撤销和进程调度；完成多单元资源的管理；完成资源的申请和释放；完成错误检测和定时器中断功能。    图1 系统总体结构  图1中间绿色部分为驱动程序test shell, 设计与实现test shell，该test shell将调度所设计的进程与资源管理器来完成测试。Test shell的应具有的功能：   * + 从终端或者测试文件读取命令；   + 将用户需求转换成调度内核函数（即调度进程和资源管理器）；   + 在终端或输出文件中显示结果：如当前运行的进程、错误信息等。   图1最左端部分为：通过终端（如键盘输入）或者测试文件来给出相应的用户命令，以及模拟硬件引起的中断 | |
| **四、实验目的：**  设计和实现进程与资源管理，并完成Test shell的编写，以建立系统的进程管理、调度、资源管理和分配的知识体系，从而加深对操作系统进程调度和资源管理功能的宏观理解和微观实现技术的掌握。 | |
| **五、实验内容：**  在实验室提供的软硬件环境中，设计并实现一个基本的进程与资源管理器。该管理器能够完成进程的控制，如进程创建与撤销、进程的状态转换；能够基于优先级调度算法完成进程的调度，模拟时钟中断，在同优先级进程中采用时间片轮转调度算法进行调度；能够完成资源的分配与释放，并完成进程之间的同步。该管理器同时也能完成从用户终端或者指定文件读取用户命令，通过Test shell模块完成对用户命令的解释，将用户命令转化为对进程与资源控制的具体操作，并将执行结果输出到终端或指定文件中。 | |
| **六、实验器材（环境配置）：**  Windows10，Python，JetBrains PyCharm 2018.2.4 x64 | |
| **七、实验步骤及操作：**  **(a) 系统功能需求分析：**  **1. 进程管理设计：**  **1.1 进程状态与操作**  进程状态： ready / running / blocked  进程操作：   * 创建(create)：(none) -> ready * 撤销(destroy)：running/ready/blocked -> (none) * 查看进程(list process)：running list/block list/ready list * 请求资源(Request)：running -> blocked (当资源没有时，进程阻塞) * 释放资源(Release)：blocked -> ready (因申请资源而阻塞的进程被唤醒) * 时钟中断(Time out)：running -> ready * 调度：ready -> running / running ->ready   **1.2 进程控制块结构**   * PID（name） * resources //: resource which is occupied * Status: Type & List // type: ready, block, running   // List: RL(Ready list) or BL(block list)   * Creation tree: Parent/Children * Priority: 0, 1, 2 (Init, User, System)   **1.3 主要函数**   * 创建进程： * 传入参数为进程的ID和优先级，优先级：初始进程0、用户进程1和系统进程2。 * Init进程在启动时创建，可以用来创建第一个系统进程或者用户进程。 * 新创建的进程或者被唤醒的进程被插入到就绪队列（RL）的末尾。 * 优先级高的进程对抢占优先级低的进程，即进行调度。 * 新创建进程的父进程为当前正在运行的进程，同样当前正在运行的进程的子进程为新创建的进程，一个进程只能有一个父进程，但是可以有多个子进程。 * 撤销进程： * 传入参数为进程的ID，对指定的进程进行删除。 * 解除父子进程的关系。 * 将其占用的所有资源释放。 * 尝试唤醒阻塞队列的优先级较高的且队列中靠前的进程。 * 若被删除的进程是正在运行的进程则进行调度操作。 * 查看进程： * 返回运行队列、就绪队列和阻塞队列 * 请求资源： * 传入参数为资源的ID和数量，向指定资源请求指定数量的资源。 * 若被请求资源剩余数量大于指定数量，则该进程成功占有该资源。 * 否则，该进程放入阻塞队列队尾，然后进行调度操作。 * 释放资源 * 传入参数为资源的ID和数量，当前正在运行的进程释放占有的指定资源指定数量。 * 尝试唤醒阻塞队列的优先级较高的且队列中靠前的进程。 * 时钟中断 * 模拟时间片到或外部硬件中断。 * 进行调度操作。 * 调度 * 将当前正在运行的进程放入就绪队列的队尾。 * 取就绪队列中优先级高的且先到达的进程运行。   **2. 资源管理设计：**  **2.1 资源状态与操作**   * 资源状态：设置固定的资源数量，4类资源，R1，R2，R3，R4，每类资源Ri有i个 * 资源操作： * 请求资源 * 释放资源 * 查看资源：(list resource)   **2.2 资源控制块结构**   * RID: 资源的ID * Status: 空闲单元的数量 * Waiting list: list of blocked process * Allocated list: list of allocated process   **2.3 主要函数**   * 请求资源： * 所有的资源申请请求按照FIFO的顺序进行。 * 若满足请求数量，则向指定进程分配指定数量的资源，并将该进程放入资源占用队列。 * 若不满足，则将该进程放入资源等待队列。 * 释放资源 * 将该进程从资源占用队列里移除。 * 如果资源等待队列不为空，并且该队列首部进程的需求资源数小于等于可用资源数则唤醒这个阻塞进程，放入就绪队列。 * 查看资源 * 返回资源的RCB信息以及资源占用队列和资源等待队列。   **3. 系统初始化设计：**   * 初始化Init进程 * 4类资源，R1，R2，R3，R4，每类资源Ri有i个 * 等待用户输入   **(b) 总体框架设计：**    **(c) 详细设计：**   * main模块为主函数，主要实现为等待用户的输入，并且做了一些输入正确错误的判断，并予以错误输入的提示，对于正确的输入，main模块解析这些输入并调用其他模块的函数。能够解析的命令如下，在命令行中输入help也可查看。分别是进程管理模块：列出所有进程的信息，查看某个进程的具体信息，创建一个优先级为[1,2]的进程，根据pid删除进程；资源管理模块：列出所有资源的信息，为正在运行的进程请求若干资源，为正在运行的进程释放若干资源；模拟时间片进行调度；按行读取文本命令作为输入。 * process: * •   show [lp] | [lp <pid>] * •   create [cr <pid> <priority>] * •   delete [de <pid>] * resources: * •   show [lr] * •   request [req <rid> <number>] * •   release [rel <rid> <number>] * dispatch: * •   time out [to] * run test shell: * •   run [filename] * exit * •   exit [exit] * Processor模块为实现一个进程控制类，类中主要维护了运行队列，就绪队列，阻塞队列三个属性变量，初始化为三个空的列表，之后会自动创建一个init进程加入到运行队列中。外部输入命令通过解析之后将直接调用该模块的函数对内部进行操控，其主要函数实现代码与实现思路如下   **1.1 创建进程**   1. **def** create\_process(self, pid, priority): 2. # 判断进程是否存在 3. **if** len(self.\_running\_list) != 0 **and** pid **in** [x.get\_pid() **for** x **in** self.get\_process\_list()]: 4. **print**("create failed, process '" + pid + "' existed!") 5. **return** 6. # 新创建的进程插入到RL队列的末尾 7. new\_pcb = PCB(pid, int(priority)) 8. self.\_ready\_list.append(new\_pcb) 9. # 如果创建的不是init进程 10. **if** len(self.\_running\_list) != 0: 11. new\_pcb.set\_parent(self.\_running\_list[0]) 12. self.\_running\_list[0].set\_children(new\_pcb) 13. self.schedule()   实现思路：对于用户输入的进程pid和priority，首先查看所有队列中是否存在相同pid的进程，若存在则退出；若不存在则将其放入就绪队列中。然后查看运行队列是否有进程，若没有，则将就绪队列的第一个进程取出放入运行队列中；若有，则设置对应的父子节点的关系，最后进行调度。  **1.2 撤销进程**   1. **def** delete\_process(self, resource, pid): 2. # 从所有队列中找到该pid的状态，从该状态的队列中删除 3. processes = [x **for** x **in** self.get\_process\_list() **if** x.get\_pid() == pid] 4. **if** len(processes) == 0: 5. **return** 6. **else**: 7. process = processes[0] 8. # 删除父母节点 9. process\_parent = process.get\_parent() 10. process\_parent.delete\_children(pid) 11. # 级联删除 12. process\_children = process.get\_children() 13. [self.delete\_process(resource=resource, pid=x.get\_pid()) **for** x **in** process\_children] 15. # 将自己拥有的资源进行释放 16. **for** x **in** process.get\_all\_resource(): 17. self.release\_resource(process=process, resource=resource, rid=x['rid'], release\_status=x['status']) 19. # 根据状态查找对应的队列进行删除 20. process\_status = process.get\_status() 21. **if** process\_status == 'running': 22. self.\_running\_list.pop([x.get\_pid() **for** x **in** self.\_running\_list].index(pid)) 23. # 如果删除的进程是当前正在运行的进程则立即进行调度 24. **elif** process\_status == 'blocked': 25. self.\_block\_list.pop([x.get\_pid() **for** x **in** self.\_block\_list].index(pid)) 26. **elif** process\_status == 'ready': 27. self.\_ready\_list.pop([x.get\_pid() **for** x **in** self.\_ready\_list].index(pid)) 28. self.schedule()   实现思路，对于用户输入的进程pid，首先查看所有队列中是否有这个进程，如果没有则返回。如果有则首先删除其与父节点的关系，然后级联删除其创建的子进程，并将被删除的节点所占有的资源进行释放，查找其在哪个队列并将其移除队列，最后进行调度。  **1.3 查看进程**   1. **def** get\_process\_info(self, pid): 2. processes = self.get\_process\_list() 3. info = [x.get\_info() **for** x **in** processes **if** x.get\_pid() == pid] 4. **if** len(info) == 0: 5. **print**("process: " + pid + " not exist!") 6. **return** 7. **print**(info[0]) 9. **def** get\_process\_list(self): 10. **return** self.\_ready\_list + self.\_block\_list + self.\_running\_list   实现思路：用户输入lp则返回所有进程队列的信息，如果lp后面带有参数pid则返回具体某个进程PCB信息 。  **1.4请求资源**   1. **def** request\_resource(self, resource, rid, request\_status, process=None): 2. **if** process **is** None: 3. process = self.\_running\_list[0] 4. # if self.\_running\_list[0].get\_pid() == 'init': 5. #     print("the process init can not request resource!") 6. #     return 7. request\_status = int(request\_status) 8. code = resource.request(process=process, rid=rid, request\_status=request\_status) 9. # 若资源请求成功，修改进程状态 10. **if** code == 0: 11. process\_resource = process.get\_resource(rid=rid) 12. # 若资源不存在则分配 13. **if** len(process\_resource) == 0: 14. process.set\_resources(resource={ 15. "rid": rid, 16. "status": request\_status 17. }) 18. # 若资源存在则进行叠加 19. **else**: 20. process\_resource['status'] += request\_status 21. # 若资源不足，则修改当前进程为阻塞态, 添加到阻塞队列，移除运行队列，进行调度 22. **elif** code == 1: 23. **for** x **in** self.\_running\_list: 24. self.\_block\_list.append(process) 25. x.set\_status("blocked") 26. self.\_running\_list.pop(self.\_running\_list.index(x)) 27. self.schedule()   实现思路：该函数调用了Resource模块的request要求其查看资源是否满足请求资源情况，0表示满足，1表示不满足。如果满足，则对修改进程的资源状态；如果不满足，则将当前你的进程放入阻塞态中，最后进行调度。  **1.5释放资源**   1. **def** release\_resource(self, resource, rid, release\_status, process=None): 2. release\_status = int(release\_status) 3. # 获取当前已分配的资源 4. **if** process **is** None: 5. process = self.\_running\_list[0] 6. status\_allocated = int(process.get\_resource(rid)['status']) 7. pid = process.get\_pid() 8. # if pid == 'init': 9. #     print("the process init can not request resources!") 10. #     return 11. # 如果已分配资源大于等于要求释放资源，则释放资源，并修改进程状态 12. **if** status\_allocated >= release\_status: 13. code = resource.release(process=process, rid=rid, release\_status=release\_status) 14. **if** code == 0: 15. process.set\_resources(resource={ 16. "rid": rid, 17. "status": status\_allocated - release\_status 18. }) 19. # 遍历进程的阻塞队列 20. **for** priority **in** [2, 1]: 21. # 根据优先级 22. block\_list = [x **for** x **in** self.\_block\_list **if** x.get\_priority() == priority] 23. flag = False 24. **for** x **in** block\_list: 25. # 查看资源的等待队列 26. rcb = resource.get\_rcb(rid=rid) 27. waiting\_list = rcb.get\_waiting\_list() 28. # 标志位用于判断是否遇见阻塞队列 29. **for** y **in** waiting\_list: 30. # 查看是否能唤醒并分配资源 31. **if** y["pid"] == x.get\_pid(): 32. # 标志位为真则证明有先到的进程被阻塞，且不满足唤醒条件，故而后续的队列不进行唤醒询问 33. **if** flag: **return** 34. **if** rcb.get\_status() >= y['status']: 35. x.set\_status('ready') 36. self.\_ready\_list.append(x) 37. self.\_block\_list.pop(self.\_block\_list.index(x)) 38. rcb.get\_waiting\_list().pop(rcb.get\_waiting\_list().index(y)) 39. resource.request(process=x, rid=rid, request\_status=y['status']) 40. # 如果被唤醒的进程优先级大于正在进行资源优先级则进行调度 41. **else**: 42. flag = True 43. **else**: 44. **print**("error, the process '" + process.get\_pid() + "' only request", resource.get\_status(), 45. "resource(s), your input has exceeded it") 46. self.schedule()   实现思路：该函数先查看资源是否满足释放状态，比如说有一些用户输入错误的rid或者其释放资源大于其占有资源则认为不满足。如果满足，则调用Resource模块的release函数进行相应的资源管理，同时processor自身需要修改正在运行进程的占有的资源状态，遍历进程的阻塞队列，查看相应的资源等待队列进行对比是否满足资源请求条件，然后根据优先级和到来顺序唤醒被阻塞的进程，如果有优先级较高且先到来的进程不满足资源分配的情况，即使有后到来的进程满足了其要求的资源分配也不允许唤醒，对于同种资源的阻塞必须按照时间到来的先后进行唤醒，最后进行调度。  **1.6 调度**   1. **def** schedule(self): 2. # 选取优先级最高而且到达时间最早的进程 3. system = [x **for** x **in** self.\_ready\_list **if** int(x.get\_priority()) == 2] 4. user = [x **for** x **in** self.\_ready\_list **if** int(x.get\_priority()) == 1] 5. **if** len(system) != 0: 6. tasks = system 7. **elif** len(user) != 0: 8. tasks = user 9. **else**: 10. tasks = [x **for** x **in** self.\_ready\_list **if** int(x.get\_priority()) == 0] 12. # 如果运行队列为空 13. **if** len(self.\_running\_list) == 0: 14. self.\_running\_list.append(tasks[0]) 15. tasks[0].set\_status("running") 16. self.\_ready\_list.pop(self.\_ready\_list.index(tasks[0])) 17. **return** 19. # 如果优先级大于正在运行的进程则进行抢占 20. **if** tasks[0].get\_priority() > self.\_running\_list[0].get\_priority(): 21. # 取出正在运行的队列 22. self.\_running\_list[0].set\_status("ready") 23. self.\_ready\_list.append(self.\_running\_list[0]) 24. self.\_running\_list.pop() 26. # 运行被选中的进程 27. tasks[0].set\_status("running") 28. self.\_ready\_list.pop(self.\_ready\_list.index(tasks[0])) 29. self.\_running\_list.append(tasks[0])   实现思路：在就绪队列中找到优先级最高并且到达时间最早的进程，将其放入运行队列中，并且正在运行的队列放入就绪队列的队尾，当然这个过程需要做一些细节的判断，比如说运行队列中是否存在进程，如果没有则认为删除了正在运行的进程，需要额外的操作；如果运行队列中仅有init进程，那么执行调度的下一个时间片仍然执行init进程。  **1.7 时间片**   1. **def** time\_out(self): 2. self.\_running\_list[0].set\_status("ready") 3. self.\_ready\_list.append(self.\_running\_list[0]) 4. self.\_running\_list.pop() 5. self.schedule()   实现思路：将正在运行的进程置为就绪，然后进行调度。   * Resource模块为实现一个资源控制模块，类中主要维护了进程控制块RCB的队列，保存所有RCB的信息。外部请求或释放资源不会直接调用Resource中的函数，而是通过Processor的request\_resource/release\_rouserce来调用Resource的request/release来完成的，Resource模块起到的另一个作用就是通过检查RCB的状态是否满足条件，若满足则修改对应的RCB的状态，并将结果反馈给Processor让其修改当前进程占用的资源状态。   **2.1请求资源**   * **def** request(self, process, rid, request\_status): * **if** len([x **for** x **in** self.\_resource\_list **if** x.get\_rid() == rid]) != 0: * resource = [x **for** x **in** self.\_resource\_list **if** x.get\_rid() == rid][0] * # 如果剩余资源大于等于请求资源，则请求资源成功 * **if** resource.get\_status() >= request\_status: * # 维护剩余资源状态 * resource.set\_status(status=resource.get\_status() - request\_status) * # 维护已分配资源状态 * allocated\_status = resource.get\_allocated\_status(pid=process.get\_pid()) * resource.set\_allocated\_list(process={ * "pid": process.get\_pid(), * "priority": process.get\_priority(), * "status": allocated\_status + request\_status * }) * **return** 0 * # 否则阻塞 * **else**: * resource.set\_waiting\_list(process={ * "pid": process.get\_pid(), * "priority": process.get\_priority(), * "status": request\_status * }) * **return** 1 * **else**: * **print**("Resource", rid, "not exist!") * **return** -1   实现思路：该函数先根据rid查看是否存在该类资源，否则报错。如果存在则判断该资源剩余资源数量是否大于等于资源请求数量，如果满足，则修改该资源PCB的状态；否则将该进程放入自己的等待队列中。并把资源情况返回给processor，-1代表资源不存在，0代表资源存在且满足分配条件，1代表资源存在但是不满足分配条件，需要等待。  **2.2释放资源**   * **def** release(self, process, rid, release\_status): * **if** len([x **for** x **in** self.\_resource\_list **if** x.get\_rid() == rid]) != 0: * resource = [x **for** x **in** self.\_resource\_list **if** x.get\_rid() == rid][0] * allocated\_status = resource.get\_allocated\_status(pid=process.get\_pid()) * # 如果释放资源大于该进程已分配资源 * **if** release\_status > allocated\_status: * **print**("request failed, the release number exceeds!") * **return** -1 * # 如果该进程已分配资源大于等于释放资源，则释放资源成功 * **else**: * # 维护资源状态 * resource.set\_status(status=resource.get\_status() + release\_status) * # 维护已分配资源的状态 * resource.set\_allocated\_list(process={ * "pid": process.get\_pid(), * "priority": process.get\_priority(), * "status": allocated\_status - release\_status * }) * **return** 0 * **else**: * **print**("Resource", rid, "not exist!") * **return** -1   实现思路：该函数先根据rid查看是否存在该类资源，否则报错。如果存在则判断该资源剩余资源数量是否需小于等于该进程已分配资源，如果不是则认为用户输入错误并报错；如果是则修改该资源RCB对应的状态。  **2.3查看资源**   * **def** get\_rl(self): * **return** [{"rid": x.get\_rid(), * "max": x.get\_max(), * "status": x.get\_status(), * "waiting\_list": x.get\_waiting\_list(), * "\_allocated\_list": x.get\_all\_allocated\_list()}**for** x **in** self.\_resource\_list]   实现思路：查询Resource维护的资源队列并返回所有资源的RCB信息。   * PCB模块为实现一个进程控制块，其内部只有set和get属性的方法实现，这么设计的原因是只有处理器能对进程做出修改，即改变PCB的状态。PCB的数据结构如下： * def \_\_init\_\_(self, pid, priority): * self.\_pid = pid * # 0: init, 1: user, 2: system * self.\_priority = **int**(priority) * # -1: none, 0: ready, 1:running, 2: blocked * self.\_status = 'ready' * self.\_parent = None * self.\_children = [] * self.\_resources = []  # {rid: rid, status:status}   需要说明的是其父亲节点只有一个，但是其子进程可以有很多个，\_resoures是其占有的资源rid及其数量status。   * RCB模块为实现一个资源控制块，其内部同样只有set和get属性的方法实现，只有处理器能通过总的资源控制块来对资源的状态来修改，Resource为保存所有RCB信息的类，RCB的数据结构如下： * def \_\_init\_\_(self, rid, status): * self.\_rid = rid * self.\_max = status * self.\_status = status * self.\_waiting\_list = []  # {pid: pid, status:status} * self.\_allocated\_list = []  # {pid: pid, status:status}   需要说明的是\_waiting\_list和\_allocated\_list分别是资源的分配队列和等待队列，里面包括了进程的pid及其资源数量status。  **测试程序设计：**  **1. 样例测试程序**   1. cr x 1 2. cr p 1 3. cr q 1 4. cr r 1 5. to 6. req R2 1 7. to 8. req R3 3 9. to 10. req R4 3 11. to 12. to 13. req R3 1 14. req R4 2 15. req R2 2 16. to 17. de q 18. to 19. to   期望输出：init x x x x p p q q r r x p q r x x x p x  **2. 测试程序设计**   1. cr A 1 2. cr B 1 3. cr C 1 4. to 5. cr D 1 6. cr E 1 7. cr F 1 8. to 9. to 10. to 11. req R1 1 12. req R2 1 13. to 14. req R2 1 15. to 16. req R3 3 17. req R4 3 18. req R4 3 19. to 20. req R1 1 21. cr G 2 22. req R1 1 23. de B 24. req R3 2 25. cr H 2 26. cr I 2 27. to 28. req R3 3 29. req R3 2 30. to 31. rel R3 1 32. to   示意图如下（数字代表优先级）    期望输出：init A A A B B B B C A D D D E E F F F B C A G D A A H H I H C A A C  设计思路：前22行用于为D、E、F分配资源，然后再使C阻塞，最后删除B，此时C得到了唤醒，用以测试进程的级联删除以及资源释放和唤醒阻塞进程。后面创建优先级较大的进程，为不同的进程请求相同资源不同数量，然后某进程释放资源使资源不满足阻塞队列队头进程要求但满足阻塞队列队头后的进程，用以测试优先级较高的进程进行抢占，以及测试先到的进程被阻塞，且不满足唤醒条件，故而后续的（申请同样资源且满足唤醒条件）队列不得进行唤醒。 | |
| **八、实验数据及结果分析：** | |
| **九、实验结论：**  设计和实现进程与资源管理，并完成Test shell的编写，以建立系统的进程管理、调度、资源管理和分配的知识体系。 | |
| **十、总结及心得体会：**  加深对操作系统进程调度和资源管理功能的宏观理解和微观实现技术的掌握。 | |
| **十一、对本实验过程及方法、手段的改进建议：**  可以给学弟学妹们加入死锁的判断，多进程的调度等，进一步加深对操作系统的理解。 | |
| **报告评分：**  **指导教师签字：** | |