ICS143B Project1 Process and Resource Manager - Preliminary Document

Anbang Xu(35086995)

1. Introduction

Design/implement a simplified process and resource manager. The basic operations include create/destroy process, request/release resource, time-out interrupt, I/O processing and schedule processes.

2. Data Structure

a. Process Control Block(PCB)

```
+-----+
| PID |
+-----+
| RList |->|R1|->|R2|->... |Rn|->NULL
+-----+
| Status.Type |, | Status.List |->|N1|->|N2|->... |Nn|->NULL
+-----+
| CreationTree.Parent |->|Parent1|,
| CreationTree.Children |->|Child1|->|Child2|->... |Childn|->NULL
+-----+
| Priority |
+-----+
```

The PCB is a structure having Process ID, Resource List - RList, Process Status, Priority, CreationTree including Parent and Children. “RList” is a pointer to the linked list of resources this PCB is using and is used by the process manager to update the resource usage list. Status.Type indicates the status of the process and Status.list is a pointer to the linked list of “Ready List” when the status is ‘ready’ while it’s a pointer to the linked list of “Waiting List” when the status is ‘wait’. “Parent” is a pointer to the parent process of current process. And “Children” is a pointer to the linked list of the children processes of current process. The creation tree need to be updated when some updates occur in this PCB. “Priority” indicates the priority of this PCB, such as 0, 1 or 2.

b. Resource Control Block(RCB)

```
+-----+
| RName |
+-----+
| Available |, | Used |
+-----+
| Waiting List |->|PCB1|->|PCB2|->... |PCBn|->NULL
+-----+
```

The RCB is a structure having Resource Name, a counter for remaining available resources, a counter for used resources and a pointer named “Waiting List”, which points to the linked list of the PCBs which are waiting to use these resources. Two counters will be updated when the PCB succeeds to request the resources. While the failed PCB will be added to the **end** of the “Waiting List” of this RCB.

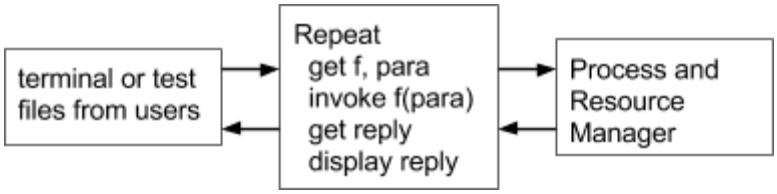
c. Ready List(RL)

```
+-----+
| Priority-2 |->|PCB1|->|PCB2|->... |PCBn|->NULL
+-----+
```

| Priority-1 |->|PCB1|->|PCB2|->... |PCBn|->NULL
+-----+
| Priority-0 |->|PCB1|->|PCB2|->... |PCBn|->NULL
+-----+
The RL is a structure having three linked list. Each one maps to a pointer to the Ready List of PCBs in the particular priority. PCB will be added to the **end** of the corresponding linked list when this PCB succeeds to request all the required resources.

d. Process and Resource Manager(PRManager)
+-----+
| Running_Process |->|PCB|
+-----+
| All Resources |->|RCB1|->|RCB2|->... |RCBn|->NULL
+-----+
| Ready List |->|PCB1|->|PCB2|->... |PCBn|->NULL
+-----+
The PRManager is a structure having the running process, a pointer to the linked list of all resource and a pointer to the “Ready List”. There is always only one process in the running status. “All Resources” is used to find the particular RCB using rid. The running process can be picked up from “Ready List” when rescheduling.

3. System Architecture
a. Overall Organization



The terminal or test files give the input. **Main function** reads the input command line by line and calls the corresponding function in **Process and Resource Manager(PRManager)** with parameters. After operation, **PRManager** gives reply to **main function**, and then **main function** displays reply to users.

b. Important functions in PRManager:
(1) Create Process
- Describe: create a new PCB data structure, initialize PCB using parameters, link PCB to creation tree, insert new process at the end of the corresponding RL
- Call Hierarchy: PRManager.createProcess(pid) -> new PCB(pid) -> PCB.initiate() -> RL.insert(PCB) -> PRManager.scheduler()
- What data structures may be changed? PRManager.runningProcess(when this PCB has highest priority), PRManager.ReadyList(otherwise)

(2) Destroy Process:
- Describe: find the corresponding PCB using pid, recursive kill the children of this particular PCB and itself including free using resources, delete PCB and update all pointers to this PCB
hint: process can be destroyed by any of its ancestors or by itself (exit).

- Call Hierarchy: PRManager.destroyProcess(pid) -> PRManager.findPCB(pid) -> PCB.killTree(pid) -> RCB.release() -> PRManager->RL.remove(pid) -> PRManager.scheduler()
- What data structures may be changed? PRManager.runningProcess, PCBs, RCBs, PRManager.RL

(3) Request Resource:
- Describe: find the corresponding RCB using rid, check its status, (1). if the status is 'free', then change this RCB's status to 'allocated' and insert this RCB to the end of RList of current process; (2). otherwise, then change the status of current process to 'blocked', remove current process from "Ready List" and insert current process to the "Waiting List" of this RCB. In the end, call rescheduler.
hint: all requests are satisfied in strict FIFO order.
- Call Hierarchy: case1: PRManager.requestResource(rid) -> PRManager.findRCB(rid) -> RCB.status = 'allocated' -> PRManager.runningProcess.RList.insert(RCB)
- What data structures may be changed? PRManager, RCB

(4) Release Resource:
- Describe: find the corresponding RCB using rid, remove this RCB from RList of current process. (1). if the "Waiting List" of this RCB is NULL, then change the RCB's status to 'free'; (2) otherwise, remove the first process q of RCB's "Waiting List", change q->status.type to 'ready' and q->status.list pointing to this RCB, remove current process from "Ready List" and insert current process to the "Waiting List" of this RCB. In the end, call rescheduler.
- Call Hierarchy: PRManager.releaseResource(rid) -> PCB.RList.remove(RCB)
- What data structures may be changed? PRManager, PCB.RList

(5) Scheduler:
- Describe: if some conditions are satisfied, change status of p to running and output the name of the running process.
- Call Hierarchy: PRManager.scheduler() -> PCB.changeStatus()
- What data structures may be changed? PRManager.runningProcess

(6) Time Out:
- Describe: use this function to simulate time-sharing. Find the running process q, change q->status.type to 'ready', insert q into "Ready List", and call rescheduler finally.
- Call Hierarchy: PRManager.timeOut() -> RL.insert(pid) -> PRManager.scheduler()
- What data structures may be changed? PRManager.runningProcess, PRManager.RL

4. Test Cases
What are the test cases you're planning to use (or have used) in order to check the correctness of the project. Try to design test cases so that they cover a large part of functionality of the project.

a. Create Process
shell> cr A 1
*Process A is running
shell> cr B 2
*Process B is running
shell> cr C 1
*Process B is running

b. Destroy Process

shell> de A
*Process A is destroyed

c. Request Resource
shell> req R1
*Process B is requesting R1

d. Release Resource
shell> rel R1
*Process B is releasing R1

e. Time Out
shell> to
*Process B is timing out

f. Request IO
shell> rio

g. IO Completion
shell> ioc

5. Pseudo Code

a.

```
Create(initialization parameters){
        create PCB data structure
        initialize PCB using parameters
        link PCB to creation tree
        insert(RL, PCB, priority)
        Scheduler()
}
insert(RL, PCB, priority){
        find the “Ready List” with the particular priority
        insert this PCB to the end of this linked list
}
```

b.

```
Destroy (pid) {
        Find_PCB(pid)
        Kill_Tree(p)
        Scheduler()
}
Kill_Tree(p) {
        for all child processes q Kill_Tree(q){
                Free_Resources()
                delete PCB and update all pointers
        }
}
Find_PCB(pid){
```

```
        Search from current process to its children process until geting the pointer p to PCB with the
        particular pid
}
Free_Resources(){
        for all resources in the RList of current process{
                Release(rid)
        }
}
```

c.

```
Request(rid) {
        r = Get_RCB(rid);
        if (r->Status == 'free') {
                r->Status = 'allocated‘;
                insert(self->RList, r);
        }
  } else {
        self->Status.Type = 'blocked';
        self->Status.List = r;
        remove(RL, self);
        insert(r->Waiting_List, self);
        Scheduler();
  }
}
Get_RCB(rid){
        search from All_Resource until getting the RCB with the particular rid
}
```

d.

```
Release(rid) {
        r = Get_RCB(rid);
        remove(self->RList, r);
        if (r->Waiting_List == NIL} {
                r->Status = 'free';
        } else {
                remove(r->Waiting_List, q);
                q->Status.Type = 'ready';
                q->Status.List = RL;
                insert(q->RList, r); //missing in book
                insert(RL, q);
                Scheduler();
        }
}
```

e.

```
Scheduler() {
        find highest priority process p
        if (self->priority < p->priority ||
```

```
                    self->Status.Type != 'running' ||
                    self == NIL)
                    preempt(p, self)
}
```

f.

```
Time_out() {
        find running process q;
        remove(RL, q);
        q->Status.Type = 'ready';
        insert(RL, q);
        Scheduler();
}
```

g.

```
Request_IO() {
        self->Status.Type = 'blocked';
        self->Status.List = IO;
        remove(RL, self);
        insert(IO->Waiting_List, self);
        Scheduler();
 }
```

h.

```
IO_completion() {
        remove(IO->Waiting_List, p);
        p->Status.Type = 'ready';
        p->Status.List = RL;
        insert(RL, p);
        Scheduler();
 }
```