Student: Jeremiah Webb
ID: 2545328
Course: CS420
Assignment: Chapter 2 Zybook Exercise Questions

2.1

Indicate whether each series of state transitions is valid or invalid.

(a)

new ➛ ready ➛ suspended ➛ blocked ➛ suspended.

Invalid

(b)

suspended ➛ blocked ➛ suspended ➛ blocked ➛ ready ➛ running.

Valid

(c)

running ➛ blocked ➛ ready ➛ blocked ➛ suspended.

Invalid

(d)

new ➛ ready ➛ running ➛ ready ➛ new.

Invalid

2.3.1

Processes 0-4 are related as follows: 1, 2, 3 are children of 0, and 4 is a child of 2. PCBs are implemented as an array indexed by the process number. Each PCB has the links: parent (p), first child (c), younger sibling (ys), and older sibling (os).

(a)

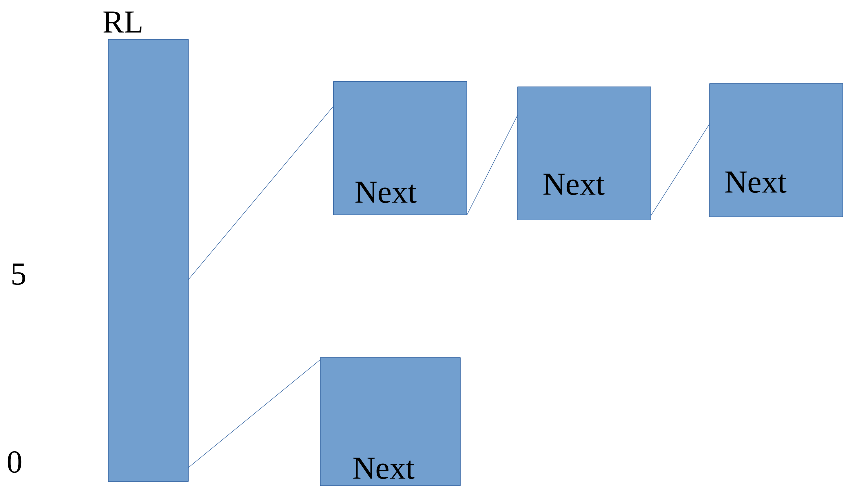
Complete the PCB array to show the values of the 4 links (p, c, ys, os) for all processes, to reflect the parent-child hierarchy.

	0	1	2	3	4	5
p	?	0	0	0	0	
c	1	-	4	-	-	
ys	-	2	3	-	-	
os	-	-	1	2	-	

(b)
Modify the array to reflect the creation of a new child, 5, of process 2.

	0	1	2	3	4	5
p	?	0	0	0	0	2
c	1	-	4	-	-	-
ys	-	2	3	-	5	-
os	-	-	1	2	-	4

<u>2.3.2</u>
(a)
Assume that RL contains 3 processes at level 5 and 1 process at level 0. Draw a diagram showing the RL and the modified PCBs



<u>2.3.3</u>
(a) Derive a formula for computing the value of p, below which the proposed scheme will outperform the linked list implementation.

Linked List insert and removal is constant time.
Array implementation remove takes r time and each insert is r+o*p ms, overhead of extending array.
Half of all operations are inserts and half are removes. Thus the time for one operation is (2r*o*p)/2
Break even point is when s = (2r*o*p)/2.

Thus solving for p = (2s-2r)/o

(b) Compute the value of p when s = 10r and o = 100r.
P = (2*10r-2r)*100r = 18/100 = .18

2.3.4

(a)

How many memory operations (allocate or free) will be performed without PCB reuse?

Each process requires a 1 allocate and 1 free operation. Thus 2*10,000 = 20,000 Operations.

(b)

How many memory operations (allocate or free) will be performed with PCB reuse?

Max of 600 PCBs needed, so 2*600 = 1200 memory.

(c)

How many PCBs will coexist in memory, on average, during the entire run without PCB reuse?

100 PCBs can be assumed to coexist on average, as stated.

(d)

How many PCBs will coexist in memory, on average, during the entire run with PCB reuse?

In the first half, the PCB amount will increase from 0 to 600, so average 300 PCBs. In the second half of freeing, the increase will be from 300 to 600, so over the entire run, (300+600)/2 = 450 PCBs.

(e)

What could be done to reduce the much larger number of PCBs maintained in memory when PCBs are reused?

OS could automatically free PCBs, avoid manual freeing.

2.4.1
(a)
What changes must be made to the scheduler or other functions to make suspend/activate work correctly?

Scheduler must be able to differentiate between states that are ready and suspended ready, both types of processses remain on the RL, but only a ready process must be selected to physically run.

(b)

Why is the scheduler called only in activate but not in suspend?

Activate enables a suspended process to compete for CPU. If the process has a higher priority, a context switch must take place. The suspend function only removes the process from the compeition for the CPU, and thus invoking the process to continue to run.

(c)

A process must be prevented from calling suspend() or activate() on itself. Why?

Calling suspend on itself would not stop itself, no context switch or actual call to the scheduler.

Calling activate on itself would make itself blocked.

(d)

A process must be prevented from calling suspend() on an already suspended process, or calling activate() on a currently active (ready) process. Why?

Suspending an already suspended process can incorrectly change the state of a process to suspended ready. Calling activate on the active process can incorrectly change the state of a process to be blocked.

2.5.1
(a) Show all changes to the data structures after process 8 requests resource 15.
PCB
Process state 7 → Running
Other Resources 7 → 15
next 7 → None

Process state 8 → blocked
other resources – None
Next – None

RCB
Resource 15
state → allocated
waiting_list → 8
next → 25

Resource 25
state → allocated
waiting_list → None
Next → None
(b)
PCBs
Process 7
process_state → running
other_resource → 25
next → 8
Process 8
process_state → ready
other_resources → 15
next →None
RCBs
Resource 15
state → allocated
waiting_list → None
next → None
Resource 25
State → allocated
waiting_list → None
next → None

<u>2.5.2</u>
(a)
1. Process 0 creates child process 1.
2. Process 0 creates a second child process 2. New process goes behind, process 2 on the RL and allows process 0 to continue to run.
3. Timeout function moves process 0 to the end of the queue, now process 2 is running.
4. Process 2 creates a child process 3, continues to run.
5. Process stops, process 2 goes to the end of the queue, process 0 now runs.
6. Process 0 creates a third child process, 4.

(b)
1. RL contains processes 0,2,4. Timeout function moves process 0 to end and process 2 now runs.
2. Resource 1 is free, process 2 acquires the resource and continues to run.
3. Timeout function moves process 2 to the end of the queue, process 4 begins to run.
4. Resource 2 is free, process 4 acquires the resource.
5. Resource 1 is already allocated, thus process 4 is blocked on the resource, process 0 now runs.
6. Next timeout function moves process 0 to the end of the queue, now process 2 runs.
7. Resource 2 is already allocated, thus process 2 is blocked on 2, process 0 now runs.

(c)
1. 0,1,4 processes remain in the system.
2. since process 2 was destroyed, resource 1 is allocated to resource 4. Resource 0 is completely free.

2.5.3

(a)

```
/* psuedo code
request(r, k){
  if (r.state >= k)
    r.state = r.state - k
    insert (r, k) into self.other_resources
  else
    self.state = blocked
    remove self from RL
    insert (self, k) in r.waiting_list
    scheduler()
}

release(r){
  remove (r, k) from self.other_resources
  r.state = r.state + k
  while (r.waiting_list != empty and r.state > 0)
    get next (p, k) from r.waiting_list
    if (r.state >= k)
      r.state = r.state - k
      insert (r, k) into p.other_resources
      p.process_state = ready
      remove (p, k) from r.waiting_list
      insert p in RL
    else break
  scheduler()
}

*/
```

(b)

RCB records the initial number of units in addition to the available number of units. Any request that exceeds that initial number has to be rejected, than be inserted into the waiting list of r.

(c)

The loop in the release function will stop as the request by the process p at the head of the queue cannot be satisfied.

2.6.1
(a)
The server should be implemented as a multi-threaded process, each process incurs the thread creation item for each request but does not block on the disk's access. Thus average CPU time is 25.
(b)
1-n is percentage with disk access

n*15 ms (1-n) * 90 = 25
find n
n = .8667.
So 87% of requests must be satisfied without accessing the server's disk.

2.6.2
Combination 2 & 7 can occur.
Combination 2 can occur because a thread is in the running state but the kernel temporarily takes the CPU away from the process. The thread is unaware of the interruption, and continues to run as soon as the process is again activated.
In combination 7, both process and thread are in the running state, a valid state.

2.6.3
Each thread could have different scheduling info. An application can tailor its own thread scheduling to any of its application needs.