# COL 331 Minor 2

Viraj Agashe

TOTAL POINTS

**36.5 / 50**

QUESTION 1

*1* Q1 **10 / 10**

Maintain a hierarchical structure for scheduling entity

✓ **+ 4 pts** *Use a 2-level CFS scheduler*

✓ **+ 5 pts** *Correct explanation of the 2-level scheduler*

+ **2 pts** Incomplete explanation

+ **2 pts** Partially correct explanation

✓ **+ 1 pts** *Saved the history per-user for the next scheduling*

Other than the CFS algorithm for users

+ **2 pts** Round-robin for selecting users

+ **2 pts** Other than RR

+ **4 pts** Correct explanation of the chosen algorithm

+ **2 pts** partially correct

+ **2 pts** Incomplete explanation

+ **1 pts** Mentioned how processes are scheduled after selecting a user

+ **0 pts** Incorrect or not attempted

QUESTION 2

*2* Q2 **0 / 10**

✓ **+ 0 pts** *Not attempted/Wrong*

+ **4 pts** High-level idea (Using MLFQ, CFS)

+ **3 pts** Details (Round-robin, datastruct)

+ **1 pts** Time quanta

+ **2 pts** Managing priorities

QUESTION 3

Q3 10 pts

*3.1* (a) **3 / 3**

✓ **+ 1 pts** *both reference the same list*

✓ **+ 1 pts** *sigpending structure correctly described*

✓ **+ 1 pts** *sigqueue structure correctly described*

+ **0 pts** incorrect/unattempted/unclear

*3.2* (b) **1.5 / 3**

✓ **+ 1.5 pts** *return address set to the instruction immediately following the instruction that was interrupted by the signal*

+ **1.5 pts** return address including other cpu state pushed on stack

+ **0 pts** incorrect/unattempted

*3.3* (c) **3 / 3**

✓ **+ 0.5 pts** *resource grant clause*

✓ **+ 0.5 pts** *inheritance clause*

✓ **+ 2 pts** *sound arguments*

+ **1 pts** unclear/incomplete arguments

+ **0 pts** incorrect/unattempted

*3.4* (d) **1 / 1**

✓ **+ 1 pts** *correct answer*

**+ 0 pts** incorrect/uattempted

QUESTION 4

*4* Q4 **10 / 10**

✓ **- 0 pts** *Correct*

**- 10 pts** Incorrect

**- 2 pts** Semantic errors, (like use or instead of and or incorrectly addressing an array etc)

**- 4 pts** Partial solution/Inefficient/Leads to deadlock/Pseudocode not provided

**- 6 pts** Right track but missing a more detailed explanation/Pseudocode

**- 3 pts** Needs more detailed explanation (How are the semaphore used, which forks to choose etc)

QUESTION 5

Q5 10 pts

*5.1* (a) **5 / 5**

✓ **+ 5 pts** *Correct*

**+ 3.5 pts** Partially correct/ Diagram missing

**+ 0 pts** Incorrect/ Not attempted

*5.2* (b) **3 / 5**

**+ 0 pts** Incorrect/ Not attempted

**+ 5 pts** Correct

✓ **+ 3 pts** *Partially Correct*

Name: VIRAJ AGASHE Entry Number: 2020CS10567

# COL331/COL633
## OPERATING SYSTEMS
## MINOR-2
TIME: 1 HOUR MARKS: 50

INSTRUCTIONS:

1. All the questions are **compulsory**.
2. No doubts will be addressed during the exam. Make your *assumptions*.

## Questions:

1. How can we modify the CFS scheduling policy to *fairly* allocate processing time among all users instead of processes? Assume that you have a single CPU and all the users have the same priority (they have an equal right to the CPU regardless of the processes that they spawn). Each user may spawn multiple processes, where each process will have its individual CFS priority between 100 and 139. Do not consider the real-time or deadline scheduling policies. [10 Marks]

Let us say the users are $u_1, u_2, \ldots u_n$. Instead of ensuring fair time amongst processes, we need to ensure among users. For this, we can allocate CPU time (scheduling slice) on the basis of the cumulative v-runtimes of the processes within a user's processes.

~~Let us say we have a struct user {..} for each user of the system.~~ We will maintain an red-black tree containing the v-runtimes (cumulative) of all processes owned ~~by all the users~~ by each user.

At any time, we select the user with the minimum cumulative v-runtime of all its processes and allot it a time slice. ~~We can calculate the scheduling slice on the basis of the cumulative v-runtime similiar to the common fair scheduler~~ Similiarly to CFS, we allot a fixed time slice to each user since priority of each user is the same (~~the effect of priorities~~ (scheduling - slice).

Name: ________________ Entry Number: ________________

Now within a user's processes we can simply use the CFS algorithm as before — pick an individual task with the currently min vruntime & schedule it.

Use the priorities of the process to appropriately scale its runtime to get the v-runtime, and once it is pre-empted, add this v-runtime to the cumulative v-runtime of the user.

In this way we can have a CFS algorithm for users.

2. Given a mixture of interactive, I/O intensive and long running processes whose execution time is not known a priori, design a scheduling algorithm for a single CPU that optimizes the completion time as well as the responsiveness of the interactive jobs. The algorithm (and associated data structures) should take into account the diversity of jobs and the fact that new jobs and high priority jobs need quick service, whereas low-priority long-running batch jobs can be delayed (read deprioritized).
[10 Marks]

Name: NIRAJ AGARWAL    Entry Number: ______________

3. Answer the following questions about signal handling and real-time scheduling. [3 + 3 + 3 + 1 Marks]
   a. Do *struct sigpending* and *struct sigqueue* reference the same data structure? Explain.

Name: ________________ Entry Number: ______________

struct sigpending is essentially a linked list of sigqueue objects.


The list head of sigpending points to the next sigqueue item in the linked list, while the list head of each sigqueue object point to the next & prev. elements in the sigpending linked list.

b. How is the return address of a signal handler set? What is it set to?

The return address of a signal handler is set to the address of a glibc function (essentially a system call) which restores back the state (context) of original process. It is set by calling the sigreturn instruction

c. Prove that in PCP, once the first resource is acquired, there can be no more priority inversions (provide a very short proof).

In PCP we know that a process ~~can acquire a resource if~~ has ownership of a resource if:

→ It was the one that set the system ceiling
→ It has a prio genuinely higher than the system ceiling.

Now, consider the resource R, under contention. Say that a process $P_1$ is holding it, and $P_2, - - P_K$ are waiting. We claim that $prio(P_1) \geq prio(P_j)\ \forall j$.

If this is not true, then $\exists$ some $P_j$ s.t. $prio(P_j) > prio(P_1)$.

Contradiction

However in the PCP algorithm we set $prio(P_1)$ to the max. of the waiting tasks (soft. increase). Further, when R was acquired, by the conditions above, P must have set the system ceiling, so it was genuinely the highest prio process. ⇒ No priority inversions.

d. For a system with periodic and pre-emptive jobs, what is the utilization bound (maximum value of $U$ till which the system remains schedulable) for EDF? Ans: 1

Name: VIRAJ AGASHE Entry Number: 2020CS10537

4. Write pseudocode (C like syntax) for solving the Dining Philosopher's Problem. Use only semaphores. Use three states for each philosopher: THINKING, HUNGRY, and EATING. [10 Marks]

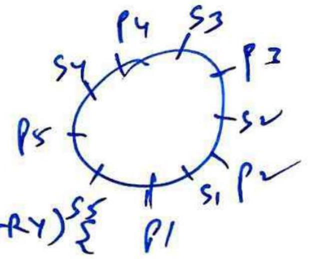
Sol. We can solve the dining philosopher's problem as follows:

semaphore $s_1, s_2, \ldots s_5$;

~~uses~~ philosophers $p_1, p_2 \ldots p_5$;

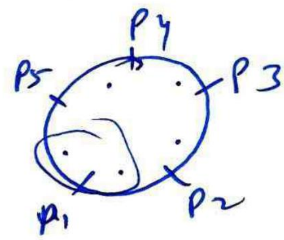
~~if~~ // Let rank denote which philosopher I am.

~~If (rank == p5) {~~
~~wait (s4);~~
~~} else {~~




If (rank == p5 && state == HUNGRY) {
    wait (s4);
    wait (s5);
    state == EATING;
    . . . . .
    state == THINKING;
    POST (s4); POST (s5);
}
else { if state == HUNGRY) {
    wait ($s_{i+1}$);
    wait ($s_{i-1}$);
    state == EATING;
    .
    .
    .
    state == THINKING;
    POST ($s_{i+1}$); POST ($s_{i-1}$);
}

One of the philosophers try left
Others all try right fork

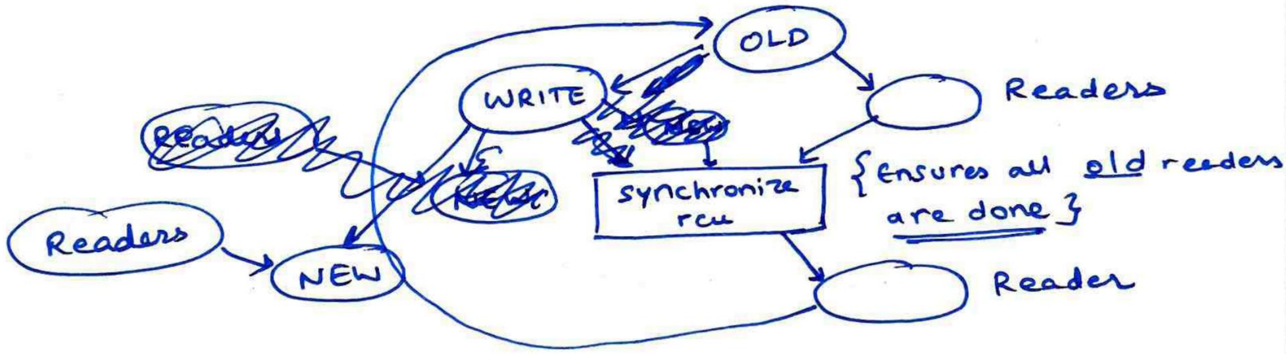


Name: ______________ Entry Number: ______________

5. Answer the following questions regarding RCU:
   a. Prove that no reader can be alive (reading the old value of the pointer) when *synchronize_rcu* returns. Create diagrams with happens-before edges and prove that such a situation is not possible. Show a proof by contradiction. [5+5] marks.

We know that sychronize RCU ensures that all readers have exited their RCU protected regions ⇒ Nobody can be reading that pointer at this time. So we use the following graph —

Name: VIRAJ AGASHE Entry Number: ______________




Consider the above graph of happens-before relationships. Suppose that earlier readers dereferenced the old value of a pointer, which was written sometime. Any reader called after sychronize_rcu has also been called after the old value has been deleted. Reading the old value ⇒ we read it before the write. This however creates a cycle in our happens-before graph, which is not possible ⇒ No reader can see old value after rcu-sychronize has been called.

b. Show the pseudocode for registering and deregistering readers and *synchronize_rcu* with a real time preemptive kernel.

Register.

```
register () {
    ~~CPU_Reading = 1~~
    CPU_Register_Reader ( );
}

deregister {
    CPU_deregister_reader ( ) ;
}
```

This function will set the bit that the CPU is in a RCU protected region in the VEB tree.

This is a call back function which sets the bit in the VEB to 0 again.

Name: ______________ Entry Number: ______________

```
Sychronize_rcu {
bool b = check_bits ( ) ;  ──→ Check if all bits
                                 in VEB are 1
  if (!b) {                      (check root)

      ~~bitmask bm = bits get_VEB_bits ( )~~;

      ~~while (bm) { count = 0 ;~~
      ~~if (bm & 0xfff0) {~~
          ~~send_ipi ( count )~~; // Send IPI to "count"
                                      CPU (index into the
                                           VEB tree)
   ~~array bitarra~~

      count = 0 ;
      while (count <ncpus) {
      if ( VEB-get (count) = = 0 ) {

          send_IPI (count); // send an IPI to
      }   else { count ++; continue;}  "count"-th processor (CPU)
          ~~wait while ( VEB-get (count)==1)~~; calls the dereg function
                                                (callback) .
   } }  while ( VEB-set (count) ==1 ) ); ← Wait till
                                            bit is set to
   }                                         1
```