Ayush Patwari

Operating Systems [CS503]

Lab3 Answers and Discussion

Note: I have added "ayush edit" tags in the comments where changes have been made. To check the files edited please do a grep with "ayush"

1. Real time RMS Scheduling:

The following modifications have been made to extend RMS scheduling along with TS scheduling:

1. process.h: added a field for prtype which defines it to be RT_PROC for rt processes and TS_PROC for other processes (both TS and non-TS)
2. rt.h : added a field rmsslk of floating point type
3. queue.h, kernel.h: updated to accommodate a new readylist for RT_processes
4. create.c:
    a. the process prtype is set to TS_PROC be default
    b. newpid() is modified to remove the static associated with nextpid since it will conflict with that in rt_create
5. rt_create.c: (mimics create.c with following changes)
    a. admission control: performs admission when a new rt process is about to be created and returns SYSERR otherwise
    b. priority is set using the following method (read resched modifications to see how it is used)

```
/* Assumption: all RT_PROC will be enqueued at the same level by the scheduler (TS_LEVELS)
 * hence only need to care about within queue priority
 * Computes rt_priority as follows
 * prio = 65535 / tr_period
 * prio = 1 if tr_period > 65535
 */

local pri16 get_rt_priority(int rt_period) {
        int maxperiod = 65535;
        if(rt_period <= 0) return maxperiod;
        if(rt_period >= 65535) return 1;

        else return (maxperiod / rt_period);

}
```

    c. prtype is set to RT_PROC
    d. newpid() is similar to create.c
6. ready.c:
    a. added declaration of rt_readylist

b. added rt_insert() on top of ts_insert()

```
/* rt_insert
 * this method checks if it is a TS_PROC then use ts_insert as before
 * (I have assigned TS_PROC to both TS and higher system processes)
 * else insert into rt_readylist
 */

void rt_insert(pid32 pid) {

        if(isbadpid(pid)) return;

        if(proctab[pid].prtype == TS_PROC) return ts_insert(pid);

        //kprintf("\nRT Insert [%s] pid [%d]", proctab[pid].prname, pid);

        // else insert into rt_readylist
        insert(pid, rt_readylist, proctab[pid].prprio);
}
```

7. resched.c (Scheduling implementation): Policy adopted is as follows:
    a. Simplication to handle rt processes
        i. They are kept in a separate rt_readylist. During scheduling we will assume that rt processes are at a level TS_LEVELS when being compared to other processes
        ii. The priority within this is an integer value inversely proportional to its rt_period (See implementation above) which allows us to round-robin between rt processes
        iii. Highest priority ready process is checked in following order : system processes > RT process > TS process
    b. The simplification allows us to avoid floating point comparisons and implement the RT scheduling with minimal changes to the kernel

8. myperiodicapp.c:

```c
#include <xinu.h>
#include <lab3.h>
#define LOOP1 20
void myperiodicrtapp(int rt_period, int rt_comp) {
        /* ayush edit */
        int i;
        /* vairable names are indicative of their functionalities */
        uint32 period_release, period_start, time_last, time_this, cpu_received, slp_next;
        /* assumed that resume() puts the release time into the unused prsem
         * update initial period_release to stored value of prsem */
        period_release = proctab[getpid()].prsem;
        /* arrays to note the important timestamps */
        int release[LOOP1], start[LOOP1], cpu_met[LOOP1];
        slp_next = -1;
        for(i = 0; i < LOOP1; i++) {
                // only sleep if there is time left till the next release
                if(slp_next > 0) {
                        sleepms(slp_next);
                } else {
                        // if entered here means process missed a deadline
                        // use debug print below to check if that happens
                        //kprintf("PID[%d] missed deadline", getpid());
                }
                // start new period execution time
                period_start = myglobalclock;
                // initialize cpu_received to 0 for current period
                cpu_received = 0;
                // initialize time_start
                time_last = myglobalclock;
                release[i] = period_release;
                start[i] = period_start;

                while(1) {
                        time_this = myglobalclock;
                        uint32 diff = time_this - time_last;
                        // update cpu_received if this was still current process
                        if(diff < 3) {
                                cpu_received += diff;
                        }
                        // if computation completed move sleep till next period
                        if(cpu_received >= rt_comp) break;

                        // update the last time
                        time_last = time_this;
                }
                cpu_met[i] = myglobalclock;
                // determine sleep time before next period
                slp_next = rt_period - (myglobalclock - period_start);
                //kprintf("\nPID[%d] sleeptime [%d]", getpid(), slp_next);
                // update period_release
                period_release += rt_period;
        }
        /* output the timestamps */
        pid32 pid = getpid();
        for(i = 0; i < LOOP1; i++)
                kprintf("\nPID: %d, period_release: %d period_start: %d, cpu_met: %d", pid, release[i], start[i], cpu_met[i]);
        kprintf("\n\n");
}
```

9. rmsslk: The CPU utilization allowed for 1, 2, and 3 processes is 100%, 83% and 78% respectively according to the formula. To account for scheduling overhead and prevent starvation of TS processes I have kept the slack factor to be 20% (0.2) which caps the max utilization to 80% for single rt process.
10. Results: I have used resched_cntl to defer the scheduling and release all the processes at the same time: e.g.

```c
resched_cntl(DEFER_START);
resume( rt_create(50, 10, myperiodicrtapp, 2048, "RTProc1", 2, 50, 10));
resume( rt_create(200, 20, myperiodicrtapp, 2048, "RTProc2", 2, 200, 20));
```

a. 2 RT processes: (50, 10), (200, 20) for 20 iterations. We can see that no process misses the deadline (= next period_release) and they complete in inverse order of their periods.

```
PID: 3, period_release: 2 period_start: 3, cpu_met: 13
PID: 3, period_release: 52 period_start: 53, cpu_met: 63
PID: 3, period_release: 102 period_start: 103, cpu_met: 113
PID: 3, period_release: 152 period_start: 153, cpu_met: 163
PID: 3, period_release: 202 period_start: 203, cpu_met: 213
PID: 3, period_release: 252 period_start: 253, cpu_met: 263
PID: 3, period_release: 302 period_start: 303, cpu_met: 313
PID: 3, period_release: 352 period_start: 353, cpu_met: 363
PID: 3, period_release: 402 period_start: 403, cpu_met: 413
PID: 3, period_release: 452 period_start: 453, cpu_met: 463
PID: 3, period_release: 502 period_start: 503, cpu_met: 513
PID: 3, period_release: 552 period_start: 553, cpu_met: 563
PID: 3, period_release: 602 period_start: 603, cpu_met: 613
PID: 3, period_release: 652 period_start: 653, cpu_met: 663
PID: 3, period_release: 702 period_start: 703, cpu_met: 713
PID: 3, period_release: 752 period_start: 753, cpu_met: 763
PID: 3, period_release: 802 period_start: 803, cpu_met: 813
PID: 3, period_release: 852 period_start: 853, cpu_met: 863
PID: 3, period_release: 902 period_start: 903, cpu_met: 913
PID: 3, period_release: 952 period_start: 953, cpu_met: 963


PID: 4, period_release: 2 period_start: 13, cpu_met: 33
PID: 4, period_release: 202 period_start: 213, cpu_met: 233
PID: 4, period_release: 402 period_start: 413, cpu_met: 433
PID: 4, period_release: 602 period_start: 613, cpu_met: 633
PID: 4, period_release: 802 period_start: 813, cpu_met: 833
PID: 4, period_release: 1002 period_start: 1013, cpu_met: 1033
PID: 4, period_release: 1202 period_start: 1213, cpu_met: 1233
PID: 4, period_release: 1402 period_start: 1413, cpu_met: 1433
PID: 4, period_release: 1602 period_start: 1613, cpu_met: 1633
PID: 4, period_release: 1802 period_start: 1813, cpu_met: 1833
PID: 4, period_release: 2002 period_start: 2013, cpu_met: 2033
PID: 4, period_release: 2202 period_start: 2213, cpu_met: 2233
PID: 4, period_release: 2402 period_start: 2413, cpu_met: 2433
PID: 4, period_release: 2602 period_start: 2613, cpu_met: 2633
PID: 4, period_release: 2802 period_start: 2813, cpu_met: 2833
PID: 4, period_release: 3002 period_start: 3013, cpu_met: 3033
PID: 4, period_release: 3202 period_start: 3213, cpu_met: 3233
PID: 4, period_release: 3402 period_start: 3413, cpu_met: 3433
PID: 4, period_release: 3602 period_start: 3613, cpu_met: 3633
PID: 4, period_release: 3802 period_start: 3813, cpu_met: 3833
```

b. 3 RT processes (100, 20) , (50, 10) , (150, 10). We see that 2nd process is completed fist, then 1st process and lastly 2rd process. No process missed the deadline

```
===== Lab3 =====

PID: 4, period_release: 2 period_start: 3, cpu_met: 13
PID: 4, period_release: 52 period_start: 53, cpu_met: 63
PID: 4, period_release: 102 period_start: 103, cpu_met: 113
PID: 4, period_release: 152 period_start: 153, cpu_met: 163
PID: 4, period_release: 202 period_start: 203, cpu_met: 213
PID: 4, period_release: 252 period_start: 253, cpu_met: 263
PID: 4, period_release: 302 period_start: 303, cpu_met: 313
PID: 4, period_release: 352 period_start: 353, cpu_met: 363
PID: 4, period_release: 402 period_start: 403, cpu_met: 413
PID: 4, period_release: 452 period_start: 453, cpu_met: 463
PID: 4, period_release: 502 period_start: 503, cpu_met: 513
PID: 4, period_release: 552 period_start: 553, cpu_met: 563
PID: 4, period_release: 602 period_start: 603, cpu_met: 613
PID: 4, period_release: 652 period_start: 653, cpu_met: 663
PID: 4, period_release: 702 period_start: 703, cpu_met: 713
PID: 4, period_release: 752 period_start: 753, cpu_met: 763
PID: 4, period_release: 802 period_start: 803, cpu_met: 813
PID: 4, period_release: 852 period_start: 853, cpu_met: 863
PID: 4, period_release: 902 period_start: 903, cpu_met: 913
PID: 4, period_release: 952 period_start: 953, cpu_met: 963


PID: 3, period_release: 2 period_start: 13, cpu_met: 33
PID: 3, period_release: 102 period_start: 113, cpu_met: 133
PID: 3, period_release: 202 period_start: 213, cpu_met: 233
PID: 3, period_release: 302 period_start: 313, cpu_met: 333
PID: 3, period_release: 402 period_start: 413, cpu_met: 433
PID: 3, period_release: 502 period_start: 513, cpu_met: 533
PID: 3, period_release: 602 period_start: 613, cpu_met: 633
PID: 3, period_release: 702 period_start: 713, cpu_met: 733
PID: 3, period_release: 802 period_start: 813, cpu_met: 833
PID: 3, period_release: 902 period_start: 913, cpu_met: 933
PID: 3, period_release: 1002 period_start: 1013, cpu_met: 1033
PID: 3, period_release: 1102 period_start: 1113, cpu_met: 1133
PID: 3, period_release: 1202 period_start: 1213, cpu_met: 1233
PID: 3, period_release: 1302 period_start: 1313, cpu_met: 1333
PID: 3, period_release: 1402 period_start: 1413, cpu_met: 1433
PID: 3, period_release: 1502 period_start: 1513, cpu_met: 1533
PID: 3, period_release: 1602 period_start: 1613, cpu_met: 1633
PID: 3, period_release: 1702 period_start: 1713, cpu_met: 1733
PID: 3, period_release: 1802 period_start: 1813, cpu_met: 1833
PID: 3, period_release: 1902 period_start: 1913, cpu_met: 1933


PID: 5, period_release: 2 period_start: 33, cpu_met: 43
PID: 5, period_release: 152 period_start: 183, cpu_met: 193
PID: 5, period_release: 302 period_start: 333, cpu_met: 343
PID: 5, period_release: 452 period_start: 483, cpu_met: 493
PID: 5, period_release: 602 period_start: 633, cpu_met: 643
PID: 5, period_release: 752 period_start: 783, cpu_met: 793
PID: 5, period_release: 902 period_start: 933, cpu_met: 943
PID: 5, period_release: 1052 period_start: 1083, cpu_met: 1093
PID: 5, period_release: 1202 period_start: 1233, cpu_met: 1243
PID: 5, period_release: 1352 period_start: 1383, cpu_met: 1393
PID: 5, period_release: 1502 period_start: 1533, cpu_met: 1543
PID: 5, period_release: 1652 period_start: 1683, cpu_met: 1693
PID: 5, period_release: 1802 period_start: 1833, cpu_met: 1843
PID: 5, period_release: 1952 period_start: 1983, cpu_met: 1993
PID: 5, period_release: 2102 period_start: 2133, cpu_met: 2143
PID: 5, period_release: 2252 period_start: 2283, cpu_met: 2293
PID: 5, period_release: 2402 period_start: 2433, cpu_met: 2443
PID: 5, period_release: 2552 period_start: 2583, cpu_met: 2593
PID: 5, period_release: 2702 period_start: 2733, cpu_met: 2743
PID: 5, period_release: 2852 period_start: 2883, cpu_met: 2893
```

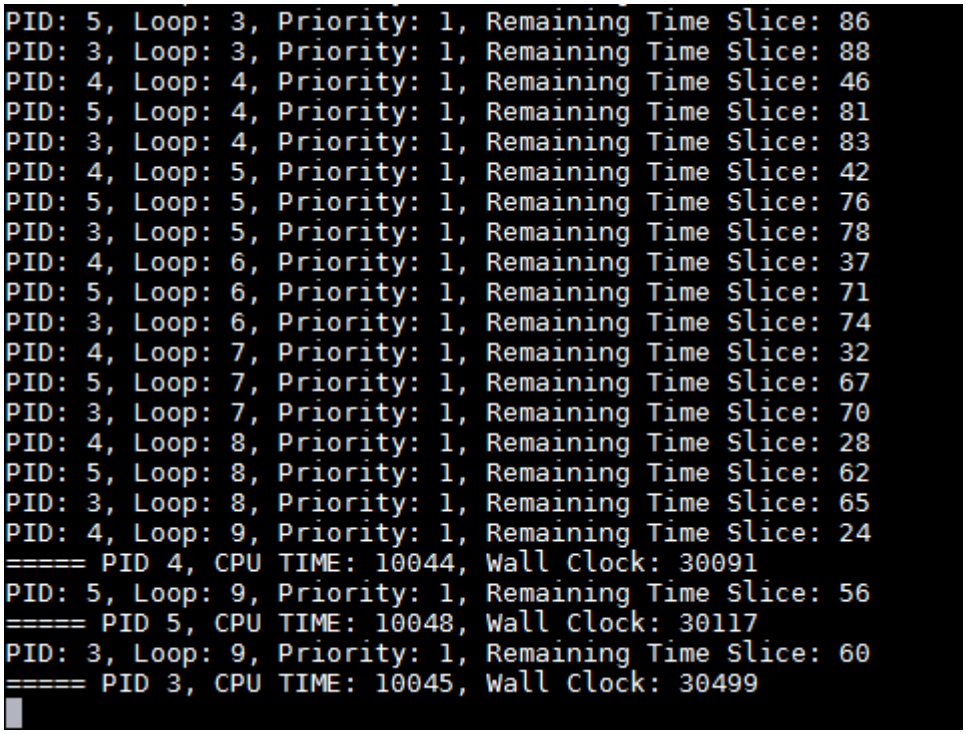
c. 5RT processes: (200, 20) , (500, 10) , (300, 10), (100, 8), (50, 5) . We can see the order of completion PIDs as 7 -> 6 -> 4 -> 3 -> 5 which is in reverse order of their time periods.

```
===== Lab3 =====

PID: 7, period_release: 2 period_start: 3, cpu_met: 11
PID: 7, period_release: 22 period_start: 23, cpu_met: 31
PID: 7, period_release: 42 period_start: 43, cpu_met: 51
PID: 7, period_release: 62 period_start: 63, cpu_met: 71
PID: 7, period_release: 82 period_start: 83, cpu_met: 91
PID: 7, period_release: 102 period_start: 103, cpu_met: 111
PID: 7, period_release: 122 period_start: 123, cpu_met: 131
PID: 7, period_release: 142 period_start: 166, cpu_met: 174
PID: 7, period_release: 162 period_start: 186, cpu_met: 194
PID: 7, period_release: 182 period_start: 206, cpu_met: 214
PID: 7, period_release: 202 period_start: 226, cpu_met: 234
PID: 7, period_release: 222 period_start: 246, cpu_met: 254
PID: 7, period_release: 242 period_start: 266, cpu_met: 274
PID: 7, period_release: 262 period_start: 286, cpu_met: 294
PID: 7, period_release: 282 period_start: 306, cpu_met: 314
PID: 7, period_release: 302 period_start: 326, cpu_met: 334
PID: 7, period_release: 322 period_start: 346, cpu_met: 354
PID: 7, period_release: 342 period_start: 389, cpu_met: 397
PID: 7, period_release: 362 period_start: 409, cpu_met: 417
PID: 7, period_release: 382 period_start: 429, cpu_met: 437


PID: 6, period_release: 2 period_start: 11, cpu_met: 19
PID: 6, period_release: 22 period_start: 31, cpu_met: 39
PID: 6, period_release: 42 period_start: 51, cpu_met: 59
PID: 6, period_release: 62 period_start: 71, cpu_met: 79
PID: 6, period_release: 82 period_start: 91, cpu_met: 99
PID: 6, period_release: 102 period_start: 111, cpu_met: 119
PID: 6, period_release: 122 period_start: 131, cpu_met: 139
PID: 6, period_release: 142 period_start: 174, cpu_met: 182
PID: 6, period_release: 162 period_start: 194, cpu_met: 202
PID: 6, period_release: 182 period_start: 214, cpu_met: 222
PID: 6, period_release: 202 period_start: 234, cpu_met: 242
PID: 6, period_release: 222 period_start: 254, cpu_met: 262
PID: 6, period_release: 242 period_start: 274, cpu_met: 282
PID: 6, period_release: 262 period_start: 294, cpu_met: 302
PID: 6, period_release: 282 period_start: 314, cpu_met: 322
PID: 6, period_release: 302 period_start: 334, cpu_met: 342
PID: 6, period_release: 322 period_start: 354, cpu_met: 362
PID: 6, period_release: 342 period_start: 397, cpu_met: 405
PID: 6, period_release: 362 period_start: 417, cpu_met: 425
PID: 6, period_release: 382 period_start: 457, cpu_met: 465
```

```
PID: 4, period_release: 2 period_start: 223, cpu_met: 525
PID: 4, period_release: 52 period_start: 526, cpu_met: 536
PID: 4, period_release: 102 period_start: 576, cpu_met: 586
PID: 4, period_release: 152 period_start: 626, cpu_met: 636
PID: 4, period_release: 202 period_start: 676, cpu_met: 686
PID: 4, period_release: 252 period_start: 726, cpu_met: 736
PID: 4, period_release: 302 period_start: 776, cpu_met: 786
PID: 4, period_release: 352 period_start: 826, cpu_met: 836
PID: 4, period_release: 402 period_start: 876, cpu_met: 886
PID: 4, period_release: 452 period_start: 927, cpu_met: 937
PID: 4, period_release: 502 period_start: 977, cpu_met: 987
PID: 4, period_release: 552 period_start: 1027, cpu_met: 1037
PID: 4, period_release: 602 period_start: 1077, cpu_met: 1087
PID: 4, period_release: 652 period_start: 1127, cpu_met: 1137
PID: 4, period_release: 702 period_start: 1177, cpu_met: 1187
PID: 4, period_release: 752 period_start: 1227, cpu_met: 1237
PID: 4, period_release: 802 period_start: 1277, cpu_met: 1287
PID: 4, period_release: 852 period_start: 1327, cpu_met: 1337
PID: 4, period_release: 902 period_start: 1377, cpu_met: 1387
PID: 4, period_release: 952 period_start: 1427, cpu_met: 1437


PID: 3, period_release: 2 period_start: 19, cpu_met: 141
PID: 3, period_release: 102 period_start: 142, cpu_met: 162
PID: 3, period_release: 202 period_start: 242, cpu_met: 364
PID: 3, period_release: 302 period_start: 365, cpu_met: 385
PID: 3, period_release: 402 period_start: 485, cpu_met: 505
PID: 3, period_release: 502 period_start: 838, cpu_met: 858
PID: 3, period_release: 602 period_start: 938, cpu_met: 958
PID: 3, period_release: 702 period_start: 1038, cpu_met: 1058
PID: 3, period_release: 802 period_start: 1138, cpu_met: 1158
PID: 3, period_release: 902 period_start: 1238, cpu_met: 1258
PID: 3, period_release: 1002 period_start: 1338, cpu_met: 1358
PID: 3, period_release: 1102 period_start: 1438, cpu_met: 1458
PID: 3, period_release: 1202 period_start: 1538, cpu_met: 1558
PID: 3, period_release: 1302 period_start: 1638, cpu_met: 1658
PID: 3, period_release: 1402 period_start: 1738, cpu_met: 1758
PID: 3, period_release: 1502 period_start: 1838, cpu_met: 1858
PID: 3, period_release: 1602 period_start: 1938, cpu_met: 1958
PID: 3, period_release: 1702 period_start: 2038, cpu_met: 2058
PID: 3, period_release: 1802 period_start: 2138, cpu_met: 2158
PID: 3, period_release: 1902 period_start: 2238, cpu_met: 2258


PID: 5, period_release: 2 period_start: 141, cpu_met: 223
PID: 5, period_release: 152 period_start: 364, cpu_met: 506
PID: 5, period_release: 302 period_start: 514, cpu_met: 524
PID: 5, period_release: 452 period_start: 917, cpu_met: 927
PID: 5, period_release: 602 period_start: 1067, cpu_met: 1077
PID: 5, period_release: 752 period_start: 1217, cpu_met: 1227
PID: 5, period_release: 902 period_start: 1367, cpu_met: 1377
PID: 5, period_release: 1052 period_start: 1517, cpu_met: 1527
PID: 5, period_release: 1202 period_start: 1667, cpu_met: 1677
PID: 5, period_release: 1352 period_start: 1817, cpu_met: 1827
PID: 5, period_release: 1502 period_start: 1967, cpu_met: 1977
PID: 5, period_release: 1652 period_start: 2117, cpu_met: 2127
PID: 5, period_release: 1802 period_start: 2279, cpu_met: 2289
PID: 5, period_release: 1952 period_start: 2429, cpu_met: 2439
PID: 5, period_release: 2102 period_start: 2579, cpu_met: 2589
PID: 5, period_release: 2252 period_start: 2729, cpu_met: 2739
PID: 5, period_release: 2402 period_start: 2879, cpu_met: 2889
PID: 5, period_release: 2552 period_start: 3029, cpu_met: 3039
PID: 5, period_release: 2702 period_start: 3179, cpu_met: 3189
PID: 5, period_release: 2852 period_start: 3329, cpu_met: 3339
```

d. 2 RT process (100, 20), (50, 10), 3 TS process. We see that RT processes are completed first and then TS processes share the CPU fairly.

```
PID: 7, period_release: 362 period_start: 363, cpu_met: 371
PID: 7, period_release: 382 period_start: 383, cpu_met: 391
PID: 7, period_release: 402 period_start: 403, cpu_met: 411
PID: 7, period_release: 422 period_start: 423, cpu_met: 431
PID: 7, period_release: 442 period_start: 443, cpu_met: 451
PID: 7, period_release: 462 period_start: 463, cpu_met: 471
PID: 7, period_release: 482 period_start: 483, cpu_met: 491
PID: 7, period_release: 502 period_start: 503, cpu_met: 511
PID: 7, period_release: 522 period_start: 523, cpu_met: 531
PID: 7, period_release: 542 period_start: 543, cpu_met: 551
PID: 7, period_release: 562 period_start: 563, cpu_met: 571
PID: 7, period_release: 582 period_start: 583, cpu_met: 591
PID: 7, period_release: 602 period_start: 603, cpu_met: 611
PID: 7, period_release: 622 period_start: 623, cpu_met: 631
PID: 7, period_release: 642 period_start: 643, cpu_met: 651
PID: 7, period_release: 662 period_start: 663, cpu_met: 671
PID: 7, period_release: 682 period_start: 683, cpu_met: 691
PID: 7, period_release: 702 period_start: 703, cpu_met: 711
PID: 7, period_release: 722 period_start: 723, cpu_met: 731
PID: 7, period_release: 742 period_start: 743, cpu_met: 751


PID: 6, period_release: 362 period_start: 371, cpu_met: 379
PID: 6, period_release: 382 period_start: 391, cpu_met: 399
PID: 6, period_release: 402 period_start: 411, cpu_met: 419
PID: 6, period_release: 422 period_start: 431, cpu_met: 439
PID: 6, period_release: 442 period_start: 451, cpu_met: 459
PID: 6, period_release: 462 period_start: 471, cpu_met: 479
PID: 6, period_release: 482 period_start: 491, cpu_met: 499
PID: 6, period_release: 502 period_start: 511, cpu_met: 519
PID: 6, period_release: 522 period_start: 531, cpu_met: 539
PID: 6, period_release: 542 period_start: 551, cpu_met: 559
PID: 6, period_release: 562 period_start: 571, cpu_met: 579
PID: 6, period_release: 582 period_start: 591, cpu_met: 599
PID: 6, period_release: 602 period_start: 611, cpu_met: 619
PID: 6, period_release: 622 period_start: 631, cpu_met: 639
PID: 6, period_release: 642 period_start: 651, cpu_met: 659
PID: 6, period_release: 662 period_start: 671, cpu_met: 679
PID: 6, period_release: 682 period_start: 691, cpu_met: 699
PID: 6, period_release: 702 period_start: 711, cpu_met: 719
PID: 6, period_release: 722 period_start: 731, cpu_met: 739
PID: 6, period_release: 742 period_start: 771, cpu_met: 779

PID: 4, Loop: 0, Priority: 1, Remaining Time Slice: 64
PID: 5, Loop: 0, Priority: 1, Remaining Time Slice: 100
PID: 3, Loop: 0, Priority: 1, Remaining Time Slice: 101
PID: 4, Loop: 1, Priority: 1, Remaining Time Slice: 60
PID: 5, Loop: 1, Priority: 1, Remaining Time Slice: 95
PID: 3, Loop: 1, Priority: 1, Remaining Time Slice: 96
PID: 4, Loop: 2, Priority: 1, Remaining Time Slice: 55
PID: 5, Loop: 2, Priority: 1, Remaining Time Slice: 90
PID: 3, Loop: 2, Priority: 1, Remaining Time Slice: 92
PID: 4, Loop: 3, Priority: 1, Remaining Time Slice: 50
```



2. BONUS (EDF scheduling):

To implement the EDF scheduling most of the modifications I have made for the RMS scheduling will work will the following major change-

1. edf_create.c : Admission control EDF has a utilization bound of 100% hence the admission control check will have to be modified to have a value of 1 on the rhs (see below. [Ref Wikipedia])

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1,$$

where the $\{C_i\}$ are the worst-case computation-times of the n processes and the $\{T_i\}$ are their respective inter-arrival periods

We can introduce a slack factor to account for scheduling overhead and prevent starvation of TS processes.

2. Resched.c: Since EDF has dynamic priority allocation (similar to TS processes) the priorities for EDF processes would have to be updated whenever they are context-switched out. The priority would be inversely proportional to remaining time to their deadlines such that the process with earliest deadline is always at the top of the edf_readylist. A initial priority similar to this approach also has be added during edf_create().
3. Assuming that I can apply the same code from the RMS scheduling only other changes would be renaming of the variables and methods to to suit edf scheduling.

2. Deadlock Detection:

1. DataStructures:

a. available[NSEM] : This array maintains the number of resources of each semaphore type
b. request[NPROC][NSEM] : Maintains the current requests for each process for each resource type they are waiting on
c. allocation[NPROC][NSEM] : Maintains the current allocation of each resource type for each process
d. These are defined in rt.h and initialized in initialize.c
e. wait.c, signal.c, semcreate.c, semdelete.c, kill.c are edited to keep the data structures upaeted. See implementation of each for the particular edits. Note: wait is also updated with the notion that both wait() and waitd() could be used by the user.

2. waitd.c:
   a. Return value: waitd returns a SYSERR (non-blocking) if it detects a deadlock else OK. The user can check for return value and decide to kill itself or do something else.
   b. Algorithm: The algorithm is a modified version of Banker's algorithm (ref: http://www.cs.cornell.edu/courses/cs4410/2011su/slides/lecture10.pdf) It is as follows:

```
/* the deadlock detection algorithm is based on resource allocation graph as follows
 * 1. temporaily update the request for the current process on the semaphore requested
 * 2. find processes which are holding at least one resource and mark them not finished
 * 3. do
 * 4.   find a process which can finish its requests with current allocation+available
 * 5.   if found: mark that process as finish and add its resources to available resources
 * 6. while (some process finished in the last iteration)
 * 7. if there is some process which cannot be finish: return SYSERR status
 * 8. else continue with normal blocking wait call procedure
 */
```

   Please check the implementation for further details.
   c. Deadlock inducing methods: I have defined two functions which will be used as processes to create deadlocks or cycles without deadlock.

```
void mytestapp(char c, sid32 sem1, sid32 sem2) {

        waitd(sem1);
        kputc(c);
        sleepms(100);
        waitd(sem2);
        kputc(c);
        signal(sem2);
        signal(sem1);

}

void mytestapp2(char c, sid32 sem1) {

        waitd(sem1);
        kputc(c);
        sleepms(200);
        signal(sem1);
}
```

   d. Results:

i. 2 process deadlock (using wait instead of waitd)

```
resched_cntl(DEFER_START);
resume( create(mytestapp, 2048, 20, "SemProc1", 3, 'C', sem1, sem2));
resume( create(mytestapp, 2048, 20, "SemProc3", 3, 'A', sem2, sem1));
resched_cntl(DEFER_STOP);
```

Output: Both processes output one character and enter deadlock:

```
===== Lab3 =====

Starting Processes
CA
```

Deadlock detection using waitd instead of waitd in mytestapp. Output:

```
Starting Processes
CA
Unsafe to proceed with process 3!
AC
```

Process 3 is detected as the 1st one to be in deadlock and waitd returns as a non-blocking call. Since, I have printed another character in mytestapp A is printed followed by C.

ii. Multiple processes : Cycle without deadlock example (using waitd)

```
void test_deadlock() {
        sid32 sem1, sem2, sem3;

        if((sem1 = semcreate(2)) == SYSERR)
                return;
        if((sem2 = semcreate(1)) == SYSERR)
                return;
        if((sem3 = semcreate(1)) == SYSERR)
                return;

        kprintf("\nStarting Processes\n");

        resched_cntl(DEFER_START);
        resume( create(mytestapp, 2048, 20, "SemProc1", 3, 'C', sem1, sem2));
        resume( create(mytestapp2, 2048, 20, "SemProc2", 3, 'B', sem1));
        resume( create(mytestapp, 2048, 20, "SemProc3", 3, 'A', sem2, sem1));
        resume( create(mytestapp, 2048, 20, "SemProc4", 3, 'D', sem3, sem2));
        resched_cntl(DEFER_STOP);

}
```

Output: No process enters deadlock

```
===== Lab3 =====

Starting Processes
CBADACD
```

iii. Multiple processes: Cycle with deadlock

```
void test_deadlock() {
        sid32 sem1, sem2, sem3;

        if((sem1 = semcreate(1)) == SYSERR)
                return;
        if((sem2 = semcreate(1)) == SYSERR)
                return;
        if((sem3 = semcreate(1)) == SYSERR)
                return;

        kprintf("\nStarting Processes\n");

        resched_cntl(DEFER_START);
        resume( create(mytestapp, 2048, 20, "SemProc1", 3, 'C', sem1, sem2));
        resume( create(mytestapp, 2048, 20, "SemProc3", 3, 'A', sem2, sem3));
        resume( create(mytestapp, 2048, 20, "SemProc4", 3, 'D', sem2, sem1));
        resched_cntl(DEFER_STOP);
```

Output: Deadlock is detected

```
===== Lab3 =====

Starting Processes
CAAD
Unsafe to proceed with process 3!
DC
```

Note: Only the first deadlocked process is shown. All 3 processes are in deadlock here.

e. Overhead:
   i. Space : Have to maintain data structures with total size = NSEM + 2*(NPROC* NSEM)
   ii. Computation: The algorithm takes a worst O(#deadlocks * NPROC * NSEM) in the worst case which can take a lot of time if there are many processes (for e.g. in linux or Windows).

Note: While testing wait() and waitd() can be used concurrently in my implementation. So if you are requesting first semaphore then using wait() can save overhead.