Homework 5

Wu Jiayao 517370910257

1. EX.1

1.

A deadlock won't happen. For the two processes, each takes at most one resource. In such situation, when either needs for an extra resource, it can take the third resource, which is free.

2.

$n \leq 5$. Each process takes at most one tape drive, leaving one tape drive free if one of the process needs it.

3.

The sum of proportion that each event takes should be less than 1

$$\frac{35}{50}+\frac{20}{100}+\frac{10}{200}+\frac{x}{250}<1 \tag{1}$$

$$x<12.5 \tag{2}$$

4.

The process can be activated more than once. It can be applied in implementation of a scheduler, where each process has a different priority.

5.

- Analyzing the source code: If the program is processing data while inputting and outputting them, it is likely to be I/O bound. Otherwise, it is likely to be CPU bound.
- At runtime: use **top** or **iostat**

2. EX.2

1.

$$\begin{bmatrix} 7 & 4 & 3 \\ 1 & 2 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 4 & 3 & 1 \end{bmatrix} \tag{3}$$

2.

It is safe. Run the process in the order 2,4,3,1,5.

3.

Yes. Take the example in No.2

Process	Allocated	Request	Maxium	Available
				332
P_2	200	122	322	210
				532
P_4	211	011	222	521
				743
P_3	302	600	902	143
				1045
P_1	010	743	753	302
				1055
P_5	002	431	433	624
				1057

3.

Implemented in /src

One fail case works as

```
Input the number of process:
5
Input the number of resource types:
4
======================================
Available:     9   12   8   13
======================================
Proc     ALLOCATED        MAXIUM
======================================
0        10 0 13 11    |   21 19 20 21
--------------------------------------
1        16 21 11 3    |   28 29 27 18
```

```
--------------------------------------
2        5 8 6 7    |   25 25 27 14
--------------------------------------
3        0 4 3 3    |   16 21 11 26
--------------------------------------
4        5 0 4 8    |   25 27 5 11
--------------------------------------
Schdule fail.
```

One success case works as

```
Input the number of process:
5
Input the number of resource types:
4
======================================
Available:     17    7    13    17
======================================
Proc     ALLOCATED        MAXIUM
======================================
0        5 15 4 5    |    14 17 13 14
--------------------------------------
1        8 12 8 2    |    8 14 15 3
--------------------------------------
2        6 7 11 13    |    7 12 15 18
--------------------------------------
3        5 4 4 2    |   9 11 13 10
--------------------------------------
4        4 0 5 6    |   9 0 16 12
--------------------------------------
Schedule success.
```

4.

Here is part from /usr/src/kernel/main.c

```
/* See if this process is immediately schedulable.
   * In that case, set its privileges now and allow it to run.
   * Only kernel tasks and the root system process get to run immediately.
   * All the other system processes are inhibited from running by the
   * RTS_NO_PRIV flag. They can only be scheduled once the root system
   * process has set their privileges.
   */
  proc_nr = proc_nr(rp);
  schedulable_proc = (iskerneln(proc_nr) || isrootsysn(proc_nr) ||
    proc_nr == VM_PROC_NR);
  if(schedulable_proc) {
      /* Assign privilege structure. Force a static privilege id. */
            (void) get_priv(rp, static_priv_id(proc_nr));
            /* Priviliges for kernel tasks. */
      if(proc_nr == VM_PROC_NR) {
                  priv(rp)->s_flags = VM_F;
                  priv(rp)->s_trap_mask = SRV_T;
    ipc_to_m = SRV_M;
    kcalls = SRV_KC;
                  priv(rp)->s_sig_mgr = SELF;
                  rp->p_priority = SRV_Q;
                  rp->p_quantum_size_ms = SRV_QT;
      }
      else if(iskerneln(proc_nr)) {
                  /* Privilege flags. */
                  priv(rp)->s_flags = (proc_nr == IDLE ? IDL_F : TSK_F);
                  /* Allowed traps. */
                  priv(rp)->s_trap_mask = (proc_nr == CLOCK
                      || proc_nr == SYSTEM  ? CSK_T : TSK_T);
                  ipc_to_m = TSK_M;                  /* allowed targets */
                  kcalls = TSK_KC;                   /* allowed kernel calls */
            }
            /* Priviliges for the root system process. */
            else {
        assert(isrootsysn(proc_nr));
                  priv(rp)->s_flags= RSYS_F;        /* privilege flags */
                  priv(rp)->s_trap_mask= SRV_T;     /* allowed traps */
                  ipc_to_m = SRV_M;                 /* allowed targets */
                  kcalls = SRV_KC;                  /* allowed kernel calls */
                  priv(rp)->s_sig_mgr = SRV_SM;     /* signal manager */
                  rp->p_priority = SRV_Q;           /* priority queue */
                  rp->p_quantum_size_ms = SRV_QT;   /* quantum size */
            }
```

When startup, kernel will check whether each process is schedulable. If it is, the kernel will set the process's privileges and allow it to run. Kernel tasks and the root system process get to run immediately, thus have a higher priority. Other processed can only be scheduled once the root system process has set their privileges.

5.

1.

```
void read_lock()
{
  down(count_lock);
  if(counter++==0)
  {
    down(db_lock);
  }
  up(count_lock);
}

void read_unlock()
{
  down(count_lock);
  if(counter--==1)
  {
    up(db_lock);
  }
  up(count_lock);
}
```

2.

The resource will be kept busy. No writer can access the resource because readers are coming continuously.

3.

two of the functions need modification.

```
void read_lock()
{
  down(read_lock);
  down(count_lock);
  if(counter++==0)
  {
    down(db_lock);
  }
  up(count_lock);
  up(read_lock);
}

void write_lock()
{
  down(read_lock);
  down(db_lock);
  up(read_lock);
}
```

4.

No, writers have higher priority than readers.