Rešenja drugog kolokvijuma iz Operativnih sistema 2, januar 2018.

1. (10 poena) a)(7)

```
int resource_allocate (int pid, int rid) {
  if (resources[rid].pid==-1) {
    resources[rid].pid = pid;
    return 1;
  } else {
    processes[pid].rid = rid;
    return 0;
  }
}

void resource_free (int rid) {
  resources[rid].pid = -1;
  for (int pid=0; pid<NProc; pid++)
    if (processes[pid].rid==rid) {
      processes[pid].rid = -1;
      resources[rid].pid = pid;
      return pid;
    }
  return -1;
}
```

b)(3) Problem ovog rešenja je moguće izgladnjivanje: resurs dobija onaj od procesa koji čekaju na dati resurs koji ima najnižu vrednost svog `pid`. Zbog toga neki proces sa visokom vrednošću `pid` može neograničeno da čeka na resurs ako se stalno pojavljuju procesi sa nižim vrednostima `pid` koji traže isti resurs. Rešenje je uvesti neki pravedni (engl. *fair*) protokol dodeljivanja resursa procesima koji čekaju na isti resurs i koji sprečava izgladnjivanje, recimo prost FIFO protokol, uvezivanjem procesa koji čekaju na isti resurs u FIFO listu.

2. (10 poena) a)(6)

```
int Resource::allocate (Process* p) {
  int ret = 0;
  if (this->usedBy==0) {
    this->usedBy = p;
    ret = 1;
  } else
  if (p->timestamp>=this->usedBy->timestamp) {
    ret = -1;
  } else {
    p->next = 0;
    if (this->tail)
      this->tail->next = p;
    else
      this->head = p;
    this->tail = p;
    ret = 0;
  }
  return ret;
}
```

b)(4) Dokaz kontradikcijom: pretpostavimo da je nastala mrtva blokada. Tada važi neophodan uslov „kružno čekanje“ (*circular wait*), tj. postoje procesi p_1, p_2, ..., p_n, takvi da p_1 čeka na resurs koga je zauzeo p_2, itd, p_n čeka na resurs koga je zauzeo p_1. Prema ovom protokolu, p_i može da čeka na resurs koga je zauzeo p_j samo ako je $TS(p_i)<TS(p_j)$. Odatle sledi: $TS(p_1)<TS(p_2)<...<TS(p_n)<TS(p_1)$, tj. $TS(p_1)<TS(p_1)$ – kontradikcija.

3. (10 poena)

```
const uint16 PMTSize = ((uint16)1)<<10;
typedef uint32 PgDesc[2];

uint32 getWorkingSetSize (uint32* pmt) {
  if (pmt==0) return 0; // Exception
  uint32 wssize = 0;
  for (uint16 i=0; i<PMTSize; i++) {
    PgDsc* pmt1 = (PgDsc*)(pmt[i]);
    if (pmt1==0) continue;
    for (uint16 j=0; j<PMTSize; j++) {
      uint32 pgDesc = pmt1[j][1];
      wssize += ((pgDesc&(((uint32)0xf)<<28))!=0);
    }
  }
  return wssize;
```