Rešenja drugog kolokvijuma iz Operativnih sistema 2, decembar 2021.

1. (10 poena)

- U klasu `Thread` treba dodati atribut `waitingOn` tipa `Mutex*`, inicijalizovan na `0`.
- U klasu `Mutex` treba dodati zaštićenu operaciju `wouldMakeCycle` koja vraća `int`.

```
int Mutex::wait () {
  lock();
  if (val<=0 && wouldMakeCycle())
    { unlock(); return -1; } // Deadlock detected
  if (--val<0) {
    Thread::running->waitingOn = this;
    block();
  } else
    holder = Thread::running;
  unlock();
  return 0;
}

int Mutex::signal () {
  if (holder!=Thread::running) return -1; // Error
  lock();
  if (val<0) {
    val++;
    Thread* t = blocked.get();
    t->waitingOn = 0;
    holder = t;
    Scheduler::put(t);
  } else {
    val = 1;
    holder = 0;
  }
  unlock();
  return 0;
}

int Mutex::wouldMakeCycle () {
  Mutex* s = this;
  Thread* t = Thread::running;
  while (1) {
    t = s->holder;
    if (t==0) return 0;  // Never occurs
    if (t==Thread::running) return 1;
    s = t->waitingOn;
    if (s==0) return 0;
    if (s==this) return 1;
  }
}
```

Obrazloženje ispravnosti detekcije mrtve blokade u operaciji `wouldMakeCycle`:

- Kako se potencijalna mrtva blokada detektuje i sprečava odmah pri zatvaranju petlje u operaciji `wait`, uvek važi da mrtve blokade, odnosno petlje nema.
- Iako na isti `Mutex` može da čeka više niti, samo jedna nit trenutno drži taj `Mutex`, a nit može da čeka samo na jedan `Mutex`.

- Zato je dovoljno samo slediti po jednu putanju oblika: nit – `Mutex` na kom ta nit čeka – nit koja drži taj `Mutex` itd. počev od tekuće niti i semafora na kom ona treba da se suspenduje. Ukoliko se pretragom naiđe na tekuću nit, detektovana je nastajuća petlja.
- Ova pretraga se uvek završava, ili tako što se detektuje nastajuća petlja, ili tako što se naiđe na nit koja ne čeka na `Mutex`.

2. (10 poena)

```
inline ProcessFrames::ProcessFrames (size_t f) : cursor(f) {
  frames[cursor].prev = frames[cursor].next = cursor;
}

inline size_t ProcessFrames::getPage (size_t f) const {
  return frames[f].page;
}

inline void ProcessFrames::setPage (size_t f, size_t p) {
  if (f<NUM_FARMES) frames[f].page = p;
}

inline size_t ProcessFrames::getVictimFrame () const {
  return cursor;
}

inline void ProcessFrames::replaceVictimFrame () {
  cursor = frames[cursor].next;
}

inline void ProcessFrames::addFrame (size_t f) {
  frames[f].prev = frames[cursor].prev;
  frames[f].next = cursor;
  frames[frames[cursor].prev].next = f;
  frames[cursor].prev = f;
}
```

3. (10 poena)

```
unsigned getBucket (size_t slotSize) {
  unsigned minBck = MAX_BUCKET;
  size_t bucketSize = (1 << minBck) * BLOCK_SIZE,
         minFragment = bucketSize % slotSize;

  bucketSize = BLOCK_SIZE;
  for (unsigned bck=0; bck<MAX_BUCKET; bck++) {
    if (bucketSize < slotSize) continue;
    size_t fragment = bucketSize % slotSize;
    if (fragment<minFragment) {
      minFragment = fragment;
      minBck = bck;
    }
    bucketSize *= 2;
  }
  return minBck;
}
```