Kolokvijum iz Operativnih sistema 1
avgust 2021.

1. (10 poena)

```
void* malloc (size_t size) {

  // Try to find an existing free block in the list (first fit):
  BlockHeader *blk = freeMemHead, *prev = nullptr;
  for (; blk!=nullptr; prev=blk, blk=blk->next)
    if (blk->size>=size) break;

  // If not found, allocate a new memory segment and add it to the list:
  if (blk == nullptr) {
    size_t sz = max(size,BLOCK_SIZE);
    blk = (BlockHeader*)mmap(nullptr,sz+sizeof(BlockHeader),PROT_RW);
    if (blk == nullptr) return nullptr;  // No free memory
    if (prev) prev->next = blk;
    else freeMemHead = blk;
    blk->next = nullptr;
    blk->size = sz;
  }

  // Allocate the requested block:
  size_t remainingSize = blk->size - size;
  if (remainingSize >= sizeof(BlockHeader) + MIN_BLOCK_SIZE) {
    // A fragment remains
    blk->size = size;
    size_t offset = sizeof(BlockHeader) + size;
    BlockHeader* newBlk = (BlockHeader*)((char*)blk + offset);
    if (prev) prev->next = newBlk;
    else freeMemHead = newBlk;
    newBlk->next = blk->next;
    newBlk->size = remainingSize - sizeof(BlockHeader);
  } else {
    // No remaining fragment, allocate the entire block
    if (prev) prev->next = blk->next;
    else freeMemHead = blk->next;
  }
  blk->next = nullptr;
  return (char*)blk + sizeof(BlockHeader);
}
```

2. (10 poena)

U klasi `Thread` treba uraditi sledeće izmene:

- Dodati privatan član: `Thread* parent`; ukazuje na roditeljsku nit; inicijalizuje se na `Thread::runningThread` u konstruktoru klase `Thread`;
- Dodati privatne članove tipa `bool`, inicijalizovane na `false` u konstruktoru klase `Thread`: `complete` koji ukazuje na to da je ova nit završila i `isParentWaiting` koji ukazuje na to da roditeljska nit čeka na završetak ove niti;
- U funkciji `Thread::wrapper` uraditi sledeće dopune:

```
void Thread::wrapper (Thread* toRun) {
  ...
  unlock();
  toRun->run();
```

```
  lock();
  toRun->complete = true;
  if (toRun->isParentWaiting) {
    toRun->isParentWaiting = false;
    Scheduler::put(toRun->parent);
  }
  ...
}
```

- Dodati javnu fukciju članicu:

```
int Thread::join () {
  if (!this || this->parent!=Thread::runningThread) return -1;
  if (this->complete) return 0;
  lock();
  this->isParentWaiting = true;
  if (setjmp(Thread::runningThread->context)==0) {
    Thread::runningThread = Scheduler::get();
    longjmp(Thread::runningThread->context,1);
  }
  unlock();
  return 0;
}
```

3. (10 poena)

```
interrupt void packetArrived () {
  if (pbTail == pbHead) {  // Buffer full, reject packet
    *ioCtrl = IO_REJECT;
  } else {
    *dmaAddress = pbTail;
    *dmaCount = PACKET_SIZE;
    *dmaCtrl = DMA_START;
  }
}

interrupt void dmaComplete () {
  if (*dmaStatus & DMA_ERROR) {
    *ioControl = IO_REJECT;
  } else {
    *ioControl = IO_COMPLETE;
    if ((pbTail-packetBuffer)/PACKET_SIZE == BUFFER_SIZE-1)
      pbTail = packetBuffer;
    else
      pbTail = pbTail+PACKET_SIZE;
  }
}
```

4. (10 poena)

```
inline void swap (int* x, int* y) { int temp = *x; *x = *y; *y = temp; }

void moveBlk (int blk) {  // Moves the given block to the first free block
  char buffer[BLK_SIZE];
  readBlock(blk,buffer);
  writeBlock(freeBlkHead,buffer);
  swap(&FAT[freeBlkHead],&FAT[blk]);
  for (int i=0; i<FAT_SIZE; i++)
    if (FAT[i]==blk) { FAT[i]=freeBlkHead; break; }
  freeBlkHead = blk;
}

void compactFile (int fcbBlk, int startBlk) {
  for (int b = FAT[fcbBlk]; b!=0; b=FAT[b]) {
```

```
    if (FAT[startBlk]>=0) moveBlk(startBlk);
    moveBlk(b);
    startBlk++;
  }
}
```