Prvi kolokvijum iz Operativnih sistema 1
Odsek za računarsku tehniku i informatiku
April 2023.

1. (10 poena)

a)(5)

```
const size_t BIOS_PADDR = 0x0;
const size_t BIOS_SIZE = 0x2000;
const size_t BIOS_VADDR = 0xFFE000;

const size_t MMIO_PADDR = 0x2000;
const size_t MMIO_SIZE = 0x80000;
const size_t MMIO_VADDR = 0xF7E000;

const size_t SRAM_PADDR = 0x82000;
const size_t SRAM_SIZE = 0x1000;
const size_t SRAM_VADDR = 0xF7D000;

const size_t KTXT_PADDR = 0x83000;
const size_t KTXT_SIZE = 0x10000;
const size_t KTXT_VADDR = 0xF6D000;

const size_t KDATA_PADDR = 0x93000;
const size_t KDATA_SIZE = 0x6D000;
const size_t KDATA_VADDR = 0xF00000;
```

b)(5)

```
static const size_t PG_SIZE = 0x1000;
static const size_t BIOS_SZ_PG = BIOS_SIZE/PG_SIZE+(BIOS_SIZE%PG_SIZE!=0);
static const size_t BIOS_START_PG = BIOS_VADDR/PG_SIZE;
static const size_t BIOS_START_FR = BIOS_PADDR/PG_SIZE;

static const size_t MMIO_SZ_PG = MMIO_SIZE/PG_SIZE+(MMIO_SIZE%PG_SIZE!=0);
static const size_t MMIO_START_PG = MMIO_VADDR/PG_SIZE;
static const size_t MMIO_START_FR = MMIO_PADDR/PG_SIZE;

static const size_t SRAM_SZ_PG = SRAM_SIZE/PG_SIZE+(SRAM_SIZE%PG_SIZE!=0);
static const size_t SRAM_START_PG = SRAM_VADDR/PG_SIZE;
static const size_t SRAM_START_FR = SRAM_PADDR/PG_SIZE;

static const size_t KTXT_SZ_PG = KTXT_SIZE/PG_SIZE+(KTXT_SIZE%PG_SIZE!=0);
static const size_t KTXT_START_PG = KTXT_VADDR/PG_SIZE;
static const size_t KTXT_START_FR = KTXT_PADDR/PG_SIZE;

static const size_t KDATA_SZ_PG =KDATA_SIZE/PG_SIZE+(KTXT_SIZE%PG_SIZE!=0);
static const size_t KDATA_START_PG = KDATA_VADDR/PG_SIZE;
static const size_t KDATA_START_FR = KDATA_PADDR/PG_SIZE;
```

```
void mapKernelVMem (PMT* pmt) {
  for (size_t i=0; i<BIOS_SZ_PG; i++)
    initPMTEntry(pmt,BIOS_START_PG+i,BIOS_START_FR+i,PROT_EXEC|PROT_READ);
  for (size_t i=0; i<BIOS_SZ_PG; i++)
    initPMTEntry(pmt,MMIO_START_PG+i,MMIO_START_FR+i,PROT_READ|PROT_WRITE);
  for (size_t i=0; i<BIOS_SZ_PG; i++)
    initPMTEntry(pmt,SRAM_START_PG+i,SRAM_START_FR+i,PROT_READ|PROT_WRITE);
  for (size_t i=0; i<BIOS_SZ_PG; i++)
    initPMTEntry(pmt,KTXT_START_PG+i,SRAM_START_FR+i,PROT_EXEC);
  for (size_t i=0; i<BIOS_SZ_PG; i++)
    initPMT(pmt,KDATA_START_PG+i,KDATA_START_FR+i,PROT_READ|PROT_WRITE);
}
```

2. (10 poena)

```
int handleProtectionFault (Process* proc, size_t page, int op) {
  SegDsc* sd = getSegDesc(proc,page);
  if (!sd) return ERR_ILLEGAL_ADDRESS;

  int lprot = getSegProt(sd);
  if ((lprot & op) == 0) return ERR_ILLEGAL_OP;
  if ((lprot & PROT_USR) == 0) return ERR_ILLEGAL_USR_OP;

  PMT* pmt = getPMT(proc);
  PgDsc* pd = getPageDesc(pmt,page);
  int prot = getPageProt(pd);

  if ((lprot & PROT_WRITE) && !(prot & PROT_WRITE)) return copyOnWrite(pd);

  return ERR_UNKNOWN; // Should never occur
}
```

3. (10 poena)

```
int getDLL () {
  static const char* dllName = ”mydll.dll”;
  static int dll = 0;
  if (dll==0) dll = mapDLL(dllName);
  if (dll<=0) handleError(”DLL cannot be mapped: %s.\n”,dllName);
  return dll;
}

int f1 (int* arg0, int arg1) {
  static int (*pf) (int*, int) = 0;
  if (pf==0) {
    int dll = getDLL();
    pf = mapDLLSymbol(dll,”f1@int@int*@int”);
    if (pf==0)
      handleError(”DLL symbol cannot be mapped: %s.\n”,”int f1(int*,int)”);
  }
  return pf(arg0,arg1);
}

double f2 (X* arg0) {
  static double (*pf) (X*) = 0;
  if (pf==0) {
    int dll = getDLL();
    pf = mapDLLSymbol(dll,”f2@double@X*”);
    if (pf==0)
      handleError(”DLL symbol cannot be mapped: %s.\n”,”double f2(X*)”);
  }
  return pf(arg0);
```