מספר זהות:

סמסטר א, מועד ב.
תאריך: 5/3/2018
שעה: 1600
משך הבחינה: 3 שעות.
חומר עזר: אסור

בחינה בקורס: **מבוא למערכות הפעלה**

פתרון קרנל

מרצה: ד"ר כרמי מרימוביץ
מתרגל: מר צבי מלמד

מדבקית
ברקוד

הנחיות:

טופס הבחינה כולל 12 עמודים (כולל עמוד זה).

קוד לא קריא לא יבדק!

בנוסף לנכונות, אלגנטיות תמיד נלקחת בחשבון!

יש לענות בשטח המוקצה לכך.

בהצלחה!

1. (30 נק') עליכם לממש בקרנל xv6 קריאות מערכת לניהול מנעולי קריאה/כתיבה. ב־user-mode הקריאות נראות כך:

```
int lock_read(int lid);
int lock_write(int lid);
int lock_none();
```

תהליך יכול לנעול מנעול אחד לכל היותר ברגע נתון. שיחרור מנעול מתבצע על־ידי קריאה ל־lock_none. כמובן שאם התהליך לא נעל מנעול אז זו שגיאה לשחרר.

מנעול יכול להימצא באחד משלושה מצבים: לא נעול, נעול לקריאה, נעול לכתיבה. מנעול יכול להיות נעול לכתיבה על־ידי תהליך אחד בלבד, ולא יכולה להיות נעילת קריאה עליו באותו זמן. מנעול יכול להיות נעול לקריאה על־ידי מספר לא מוגבל של תהליכים ולא יכולה להיות נעילת כתיבה עליו באותו זמן.

רוטינות הנעילה חוזרות לקורא רק לאחר שהמנעול ננעל, כלומר הן ממתינות עד שיתקיימו התנאים לביצוע הנעילה, ככל שצריך. אין צורך לדאוג מהרעבה.

(אין צורך לממש את הבאת הארגומנטים מ־user-mode.)

מזהה מנעול הוא מספר בן 32־ביטים ואסור באיסור חמור ליצור ווקטור של מנעולים! מותר להוסיף שדות למבנה proc.

```
struct spinlock tickslock;
int sys_sleep(void)
{
  int n;
  uint ticks0;

  if(argint(0, &n) < 0)
    return -1;
  acquire(&tickslock);
  ticks0 = ticks;
  while(ticks - ticks0 < n){
    if(proc->killed){
      release(&tickslock);
      return -1;
    }
    sleep(&ticks, &tickslock);
  }
  release(&tickslock);
  return 0;
}
```

מקום עבור תשובה 1

```
int lock_read(int lid) {
   struct proc *proc = myproc();
   if (proc->lockState != LOCK_NONE)
      return (-1);

   acquire(&ptable.lock);
loop:
   for (p = ptable.proc; p < ptable.proc[NPROC]; p++) {
      if (p->state     == UNUSED)     continue;
      if (p->lockState == LOCK_NONE)  continue;
      if (p->lockID    != lid)        continue;
      if (p->lockState == LOCK_READ)  break;

      if (p->lockState == LOCK_WRITE) {
         if (proc->killed) {
            release(&ptable.lock);
            return(-1);
         }
         sleep(lid, &ptable.lock);
         goto loop;
      }
   }
   proc->lockID = lid;
   proc->lockState = LOCK_READ;
   release(&ptable.lock);
   return (0);
}

int lock_write(int lid) {
   struct proc *proc = myproc();
   if (proc->lockState != LOCK_NONE)
      return (-1);

   acquire(&ptable.lock);
loop:
```

```
    for (p = ptable.proc; p < ptable.proc[NPROC]; p++) {
        if (p->state     == UNUSED) continue;
        if (p->lockState == LOCK_NONE) continue;
        if (p->lockID    != lid) continue;

        if (proc->killed) {
            release(&ptable.lock);
            return(-1);
        }
        sleep(lid, &ptable.lock);
        goto loop;
    }
    proc->lockID = lid;
    proc->lockState = LOCK_WRITE;
    release(&ptable.lock);
    return (0);
}

int lock_none() {
    struct proc *proc = myproc();

    if (proc->lockState == LOCK_NONE)
        return (-1);

    proc->lockState = LOCK_NONE;
    wakeup(proc->lockID);
    return (0);
}
```

2. (20 נק') בשאלה זו סביבת העבודה הינה xv6 במצב קרנל. עליכם לכתוב רוטינה בקרנל שחתימתה
char *readpage(struct inode *ip, int vaddr);

רוטינה זו נקראת בקרנל בזמן ריצה של תהליך שקובץ ה־elf בו הוא משתמש נגיש על־ידי ip. הרוטינה תקצה דף בזיכרון ותקרא אליו מהקובץ את הדף שמכיל את הכתובת הוירטואלית vaddr של התוכנית. הרוטינה תחזיר את הכתובת הקרנלית אליה נקרא הדף. אם הכתובת vaddr לא קיימת הרוטינה תחזיר NULL. אם יש פחות מדף בקובץ, יש למלא באפסים כמובן. גם אם יש בעיית קריאה כלשהי יש להחזיר NULL.

בבקשה לא להתבלבל עם מרחב הכתובות הקיים של התהליך. הוא לא רלוונטי.

```
struct proghdr {
   uint type;
   uint off;
   uint vaddr;
   uint paddr;
   uint filesz;
   uint memsz;
   uint flags;
   uint align;
};
int exec(char *path, char **argv) {
   char *s, *last;
   int i, off;
   uint argc, sz, sp, ustack[3+MAXARG+1];
   struct elfhdr elf;
   struct inode *ip;
   struct proghdr ph;
   pde_t *pgdir, *oldpgdir;
   struct proc *curproc = myproc();

   begin_op();

   if ((ip = namei(path)) == 0){
      end_op();
      cprintf("exec: fail\n");
      return -1;
   }
   ilock(ip);
```

```
    pgdir = 0;

    // Check ELF header
    if (readi(ip, (char*)&elf, 0, sizeof(elf)) != sizeof(elf))
        goto bad;
    if (elf.magic != ELF_MAGIC)
        goto bad;

    if ((pgdir = setupkvm()) == 0)
        goto bad;

// Load program into memory.
    sz = 0;
    for (i=0, off=elf.phoff; i<elf.phnum; i++, off+=sizeof(ph)){
        if (readi(ip, (char*)&ph, off, sizeof(ph)) != sizeof(ph))
            goto bad;
        if (ph.type != ELF_PROG_LOAD)
            continue;
        if (ph.memsz < ph.filesz)
            goto bad;
        if (ph.vaddr + ph.memsz < ph.vaddr)
            goto bad;
        if ((sz = allocuvm(pgdir, sz, ph.vaddr + ph.memsz)) == 0)
            goto bad;
        if (ph.vaddr % PGSIZE != 0)
            goto bad;
        if (loaduvm(pgdir, (char*)ph.vaddr, ip, ph.off, ph.filesz)
            goto bad;
    }
    iunlockput(ip);
    end_op();
    ip = 0;
    ⋮
    ⋮
bad:
    if (pgdir)
        freevm(pgdir);
    if (ip){
```

```
        iunlockput(ip);
        end_op();
    }
    return -1;
}


int loaduvm(pde_t *pgdir, char *addr, struct inode *ip, uint offs
                    uint sz) {
    uint i, pa, n;
    pte_t *pte;

    if ((uint) addr % PGSIZE != 0)
        panic("loaduvm: addr must be page aligned");
    for (i = 0; i < sz; i += PGSIZE) {
        if ((pte = walkpgdir(pgdir, addr+i, 0)) == 0)
            panic("loaduvm: address should exist");
        pa = PTE_ADDR(*pte);
        if (sz - i < PGSIZE)
            n = sz - i;
        else
            n = PGSIZE;
        if (readi(ip, P2V(pa), offset+i, n) != n)
            return -1
    }
    return 0;
}
```

מקום עבור תשובה 2

```
char *readpage(struct inode *ip, int vaddr) {

   char *mem = kalloc();
   if (mem == 0)
      return (0);
   memset(mem, 0, 4096);

   begin_op();
   ilock(ip);

   if (readi(ip, (char*)&elf, 0, sizeof(elf)) != sizeof(elf))
      goto bad;
   if (elf.magic != ELF_MAGIC)
      goto bad;

   for (i=0, off=elf.phoff; i<elf.phnum; i++, off+=sizeof(ph)) {
      if (readi(ip, (char*)&ph, off, sizeof(ph)) != sizeof(ph))
         goto bad;
      if (ph.type != ELF_PROG_LOAD) continue;
      if ((ph.vaddr % PGSIZE) != 0)     goto bad;

      uint vmin = vaddr & ~4095;
      uint vm   = vmin;
      uint vmax = vmin + 4096;

      if (vmax <= ph.vaddr) continue;
      if (ph.vaddr + ph.memsz <= vmin) continue;

      if (vmax > ph.vaddr + ph.memsz)
         vmax = ph.vaddr + ph.memsz

      if (vmin < ph.vaddr)
            vmin = ph.vaddr;

      uint o = vmin - ph.vaddr;
      uint l = vmax - vmin;
```

```
        if (readi(ip, &mem[vmin-vm], ph.off + o, l) != l)
                goto bad;
    }
    iunlockput(ip);
    end_op();
    return (v2p(mem));
bad:
    kfree(mem);
    iunlock(ip);
    end_op();
    return 0;
}
```

3. (50 נק') סביבת שאלה זו היא Linux ב־user-mode. הכלים הבאים נתונים לצורך מימוש תוכנת שרת.

```
struct msg_request {
 .....
};
struct msg_response {
 .....
};
int get_request(struct msg_request*);
int process_request(struct msg_request*, struct msg_response*);
int send_response(struct msg_response*);
```

בקשה מלקוח מתקבלת על־ידי הפונקציה get_request. טיפול בבקשה ובניית תשובה מתבצעים על ידי הפונקציה process_request. שליחת תשובה ללקוח מתבצעת על־ידי הפונקציה send_response.

במערכת השרת צריך להיות תהליך ראשי אשר מקבל בקשות בלולאה אינסופית ומעביר את הבקשות להמשך טיפול על־ידי פייפ.

הפונקציה process_request הינה "החלק הכבד" במערכת. יש לשאוף לכך שבכל רגע נתון מספר התהליכים שמבצעים (או פנויים לבצע) את החלק הכבד הוא כמספר המעבדים במחשב (אך לא יותר), הנתון על־ידי הקבוע הבא: #define NCPU 8

לתהליך הראשי מותר לשלוח מידע לתהליך המבצע עיבוד רק אם ידוע שהתהליך פנוי.

ידוע שקיימת שונות גדולה בזמן שלוקח לעבד בקשה (כלומר העיבוד של בקשות מסוימות יהיה מהיר ואילו של אחרות יהיה איטי).

הנחיות:

- מירב הנקודות יינתנו לפתרונות פשוטים ואלגנטיים (ונכונים).
- אין צורך לבדוק אם קריאות המערכת נכשלות.
- בפייפ יש מספיק מקום למבנים הנ"ל.

יש לתת שני מימושים של המערכת, לפי הסעיפים הבאים.

(א) (20 נק') ממשו מערכת כנ"ל כאשר מעבר מידע בין כל התהליכים יתבצע בעזרת פייפ יחיד.

(ב) (30 נק') ממשו מערכת כנ"ל כאשר פייפ מסויים יכול לשמש לתקשורת בין זוג תהליכים בלבד.