מספר זהות:

סמסטר א, מועד ב.
תאריך: 3/3/2017
שעה: 0900
משך הבחינה: 3 שעות.
חומר עזר: אסור

בחינה בקורס: **מבוא למערכות הפעלה**

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

מדבקית
ברקוד

הנחיות:

טופס הבחינה כולל 18 עמודים (כולל עמוד זה).

תשובות צריכות לכלול הסבר.

קוד לא קריא לא יבדק!

יש לענות בשטח המוקצה לכך.

בהצלחה!

1. (25 נק') בשאלה זו סביבת העבודה הינה הקרנל xv6. מבנה ה־folder שונה כדי להתאים לשמות קבצים ארוכים. מבנה רשומה חדש נראה בזיכרון כך:

```
struct dirent2 {
        ushort inum;
        uchar  len;
        char   name[255];
};

struct dirent {
        ushort inum;
        char   name[DIRSIZ];
};
```

בדיסק אורך כל רשומה הוא $2+1+\text{len}$. כיתבו את הפונקציה dirlookup2 שתחליף את הפונקציה dirlookup ותטפל במבנה החדש.

```
struct inode *dirlookup(struct inode *dp, char *name, uint *poff) {
        uint off, inum;
        struct dirent de;

        if (dp->type != T_DIR)
                panic("dirlookup not DIR");

        for (off = 0; off < dp->size; off += sizeof(de)) {
                if (readi(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
                        panic("dirlink read");
                if(de.inum == 0)
                        continue;
                if(namecmp(name, de.name) == 0) {
                        if (poff)
                                *poff = off;
                        inum = de.inum;
                        return iget(dp->dev, inum);
                }
        }
        return 0;
}
```

2. (25 נק') סביבת שאלה זו היא הקרנל xv6. כיתבו את הפונקציה int sbrk(uint n) שמטרתה להגדיל את מרחב הכתובות של התהליך הנוכחי ב־n בתים.

כרגיל, בכישלון יוחזר ערך שלילי, ובהצלחה יוחזר אפס. במקרה כישלון יש להשאיר את התהליך בגודלו המקורי.

ניתן להשתמש בשגרות:

kalloc(), kfree(void *p), memset(char *buf, char val, uint len),

walkpgdir(void *pgdir, uint addr, int alloc),

ותו לא.

```
struct proc {
        uint sz; // Size of process memory (bytes)
        pde_t* pgdir; // Page table
        char *kstack; // Bottom of kernel stack for this process
        enum procstate state; // Process state
        volatile int pid; // Process ID
        struct proc *parent; // Parent process
        struct trapframe *tf; // Trap frame for current syscall
        struct context *context; // swtch() here to run process
        void *chan; // If non-zero, sleeping on chan
        int killed; // If non-zero, have been killed
        struct file *ofile[NOFILE]; // Open files
        struct inode *cwd; // Current directory
        char name[16]; // Process name (debugging)
};
```

```
#define PDX(va)         (((uint)(va) >> PDXSHIFT) & 0x3FF)
#define PTX(va)         (((uint)(va) >> PTXSHIFT) & 0x3FF)
#define PTXSHIFT        12
#define PDXSHIFT        22
#define PTE_P           0x001
#define PTE_W           0x002
#define PTE_U           0x004
#define PTE_ADDR(pte)   ((uint)(pte) & ~0xFFF)
static pte_t *walkpgdir(pde_t *pgdir, const void *va, int alloc) {
        pte_t *pgtab;
        pde_t *pde = &pgdir[PDX(va)];

        if (*pde & PTE_P){
                pgtab = (pte_t*)p2v(PTE_ADDR(*pde));
        } else {
                if (!alloc || (pgtab = (pte_t*)kalloc()) == 0)
                        return 0;
                        memset(pgtab, 0, PGSIZE);
                        *pde = v2p(pgtab) | PTE_P | PTE_W | PTE_U;
                }
                return &pgtab[PTX(va)];
        }
```

⋮

3. (35 נק') סביבת שאלה זו היא linux ב־user-mode. ממומשות ונתונות הפונקציות וההגדרות הבאות:

```
#define MAX_FILES 10
#define GOOD_NUM <value>
#define BAD_NUM <value>

int get_random();
int check_number(int);
```

הפונקציה check_number מחזירה אחד מהערכים GOOD_NUM או BAD_NUM.

תיאור תוכניות:

(א) עליכם לכתוב תכנית שנקראת tee. התכנית מתנהגת באופן הבא:

שורת ההפעלה של התכנית מכילה שמות קבצים. מספר שמות הקבצים קטן או שווה ל־ MAX_FILES. כך למשל ניתן להפעיל את התכנית מתוך הטרמינל ע"י השורה: tee f1 f2 f3

התכנית tee מעתיקה את הקלט הסטנדרטי לכל אחד מהקבצים, וכמו כן לפלט הסטנדרטי. הקריאה מהקלט הסטנדרטי מתבצעת על ידי קריאה של בית בודד בכל פעם.

(ב) עליכם לכתוב תכנית שנקראת prog המתנהגת באופן הבא:

שורת ההפעלה של התכנית היא: prog <proc−num> <file1> <file2> .....

כאשר:

- proc−num: מספר הצאצאים (תהליכים) שצריך ליצור.
- file1, file2,...: רשימת קבצים.

התכנית יוצרת proc−num צאצאים.

כל אחד מצאצאים אלו מגריל מספרים ועבור כל מספר בודק אם הוא "טוב" או "רע". אם המספר "רע", הצאצא כותב את ההודעה הבאה (לדוגמא) לשגיאה הסטנדרטית ומסתיים: pid = 875 BAD NUMBER=29858

כאשר 875 הוא ה־pid של הצאצא, ו־29858 הוא המספר "הרע" שהוגרל על ידו.

לעומת זאת, אם המספר שהוגרל הינו "טוב" אזי יש לכתוב אותו לכ"א מהקבצים file1, file2,... לשם כך, עליכם להשתמש בתכנית tee שתוארה בחלק א'.

הערות:

(א) התכנית prog היא "תכנית שקטה" – כלומר איננה כותבת דבר לפלט הסטנדרטי.

(ב) במערכת שבה התכנית רצה, הכתיבה והקריאה לקבצים מתבצעות בכמות של בית בודד בכל פעם.

הנחיות:

- מירב הנקודות בשאלה הזאת לפתרונות פשוטים ואלגנטיים.
- מירב הנקודות לפתרון שבו התכנית prog אינה כותבת בעצמה לקבצים אלא נעזרת בתכנית tee.
- התכנית tee צריכה להתאים לצרכים של התכנית prog מצד אחד, ומאידך לענות לדרישות שמתוארות בחלק א'.
- מירב הנקודות לפתרון שאיננו משתמש בסמפורים. במידת הצורך ניתן להשתמש ב־flock.
- עליכם לאפשר מקביליות רבה ככל שניתן.
- אין להתייחס בתשובה לאפשרויות שקריאות מערכת כגון open, pipe, fork וכו' נכשלות.
- אין להתייחס ל־INCLUDE's.
- תשובה בכתב לא קריא – לא תיבדק!
- לגבי הקריאה ל־exec. התייחסו לחתימה של xv6 שהיא:

```
int exec(char* prog_name, char **argv);
```

הקונבנציה היא שמקום 0 במערך argv מכיל את שם התכנית, והמערך מסתיים ב־NULL-pointer.

(א) כיתבו את התכנית tee.c:

(ב) כיתבו את התכנית prog.c

4. (15 נק') סביבת שאלה זו היא linux ב־user-mode.

```
11 void print(char c)
12 {
13    fprintf(stdout, "%c pid=%d ppid=%d\n", c, getpid(), getppid());
14 }
15
16 int
17 main(int argc, char *argv[])
18 {
19 setbuf(stdout, NULL);
20    if (fork() || fork() )
21       if (fork() &&  fork())
22          print('X');
23    print('Y');
24 }
```

נניח שכשמריצים את התכנית, התהליך הראשון (תהליך ההורה) הוא בעל PID=100 וכל תהליך שנוצר לאחר מכן מקבל מספר PID עוקב.

(א) כמה תהליכים נוצרים (כולל תהליך ההורה)? ציירו את עץ התהליכים שנוצר.

נוצרים בסה"כ _____ תהליכים. עץ התהליכים הוא:

(ב) תארו פלט אפשרי של התכנית:

(ג) למרבה הצער, כשהריצו את התכנית מתוך הטרמינל, ה־prompt של ה־shell התערבב בתוך ההודעות.

i. הסבירו למה זה קורה.

ii. האם זה חייב להיות כך או שאפשר למנוע זאת, ואם אפשר למנוע זאת, כיצד בדיוק לשנות את התכנית.

```
WAIT(2)                    Linux Programmer's Manual                    WAIT(2)

NAME
       wait, waitpid, waitid - wait for process to change state

SYNOPSIS
       #include <sys/types.h>
       #include <sys/wait.h>

       pid_t wait(int *status);

       pid_t waitpid(pid_t pid, int *status, int options);

       int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```

```
DESCRIPTION
       All of these system calls are used to wait for state changes in a child
       of  the  calling  process, and obtain information about the child whose
       state has changed.  A state change is considered to be: the child  ter-
       minated; the child was stopped by a signal; or the child was resumed by
       a signal.  In the case of a terminated child, performing a wait  allows
       the  system  to  release  the resources associated with the child; if a
       wait is not performed, then the terminated child remains in a  "zombie"
       state (see NOTES below).

       If  a  child has already changed state, then these calls return immedi-
       ately.  Otherwise they block until either a child changes  state  or  a
       signal  handler interrupts the call (assuming that system calls are not
       automatically restarted using the SA_RESTART flag of sigaction(2)).  In
       the  remainder  of this page, a child whose state has changed and which
       has not yet been waited upon by one of these  system  calls  is  termed
       waitable.
```

```
     WEXITSTATUS(status)
            returns  the  exit  status  of  the child.  This consists of the
            least significant 8 bits of the status argument that  the  child
            specified  in  a  call to exit(3) or _exit(2) or as the argument
            for a return statement in main().  This  macro  should  only  be
            employed if WIFEXITED returned true.
```

```
PIPE(2)                    Linux Programmer's Manual                    PIPE(2)

NAME
       pipe, pipe2 - create pipe

SYNOPSIS
       #include <unistd.h>

       int pipe(int pipefd[2]);

       #define _GNU_SOURCE             /* See feature_test_macros(7) */
       #include <unistd.h>

       int pipe2(int pipefd[2], int flags);

DESCRIPTION
       pipe()  creates  a pipe, a unidirectional data channel that can be used
       for interprocess communication.  The array pipefd is used to return two
       file  descriptors  referring to the ends of the pipe.  pipefd[0] refers
       to the read end of the pipe.  pipefd[1] refers to the write end of  the
       pipe.   Data  written  to  the write end of the pipe is buffered by the
       kernel until it is read from the read end of  the  pipe.   For  further
       details, see pipe(7).

       If  flags is 0, then pipe2() is the same as pipe().  The following val-
       ues can be bitwise ORed in flags to obtain different behavior:

       O_NONBLOCK  Set the O_NONBLOCK file status flag on  the  two  new  open
                   file  descriptions.   Using  this flag saves extra calls to
                   fcntl(2) to achieve the same result.

       O_CLOEXEC   Set the close-on-exec (FD_CLOEXEC) flag on the two new file
                   descriptors.   See  the  description  of  the  same flag in
                   open(2) for reasons why this may be useful.

RETURN VALUE
       On success, zero is returned.  On error, -1 is returned, and  errno  is
       set appropriately.
```

```
SEM_POST(3)                Linux Programmer's Manual                SEM_POST(3)

NAME
       sem_post - unlock a semaphore

SYNOPSIS
       #include <semaphore.h>

       int sem_post(sem_t *sem);

       Link with -lrt or -pthread.

DESCRIPTION
       sem_post()  increments  (unlocks)  the semaphore pointed to by sem.  If
       the semaphore's value consequently  becomes  greater  than  zero,  then
       another  process  or thread blocked in a sem_wait(3) call will be woken
       up and proceed to lock the semaphore.
```

```
NAME
       sem_wait, sem_timedwait, sem_trywait - lock a semaphore

SYNOPSIS
       #include <semaphore.h>

       int sem_wait(sem_t *sem);

       int sem_trywait(sem_t *sem);

       int sem_timedwait(sem_t *sem, const struct timespec *abs_timeout);

       Link with -lrt or -pthread.

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       sem_timedwait(): _POSIX_C_SOURCE >= 200112L || _XOPEN_SOURCE >= 600

DESCRIPTION
       sem_wait()  decrements (locks) the semaphore pointed to by sem.  If the
       semaphore's value is greater than zero, then  the  decrement  proceeds,
       and  the function returns, immediately.  If the semaphore currently has
       the value zero, then the call blocks until either it  becomes  possible
       to  perform the decrement (i.e., the semaphore value rises above zero),
       or a signal handler interrupts the call.

       sem_trywait() is the same as sem_wait(), except that if  the  decrement
       cannot  be immediately performed, then call returns an error (errno set
       to EAGAIN) instead of blocking.
```

```
NAME
       sem_getvalue - get the value of a semaphore

SYNOPSIS
       #include <semaphore.h>

       int sem_getvalue(sem_t *sem, int *sval);

       Link with -lrt or -pthread.

DESCRIPTION
       sem_getvalue() places the current value of the semaphore pointed to sem
       into the integer pointed to by sval.
```

```
KILL(2)                    Linux Programmer's Manual                    KILL(2)

NAME
       kill - send signal to a process

SYNOPSIS
       #include <sys/types.h>
       #include <signal.h>

       int kill(pid_t pid, int sig);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       kill(): _POSIX_C_SOURCE >= 1 || _XOPEN_SOURCE || _POSIX_SOURCE

DESCRIPTION
       The  kill()  system  call can be used to send any signal to any process
       group or process.

       If pid is positive, then signal sig is sent to the process with the  ID
       specified by pid.
```

to kill a process with pid==100 you need to do:

```
kill(100, Term);     // Term is the signal to terminate the process
```

```
FLOCK(2)                    Linux Programmer's Manual                   FLOCK(2)

NAME
       flock - apply or remove an advisory lock on an open file

SYNOPSIS
       #include <sys/file.h>

       int flock(int fd, int operation);

DESCRIPTION
       Apply or remove an advisory lock on the open file specified by fd.  The
       argument operation is one of the following:

           LOCK_SH  Place a shared lock.  More than one  process  may  hold  a
                    shared lock for a given file at a given time.

           LOCK_EX  Place  an  exclusive  lock.   Only one process may hold an
                    exclusive lock for a given file at a given time.

           LOCK_UN  Remove an existing lock held by this process.

       A call to flock() may block if an incompatible lock is held by  another
       process.   To  make  a  nonblocking request, include LOCK_NB (by ORing)
       with any of the above operations.

       A single file may not simultaneously have  both  shared  and  exclusive
       locks.

       Locks  created by flock() are associated with an open file table entry.
       This means that duplicate file descriptors (created  by,  for  example,
       fork(2)  or  dup(2)) refer to the same lock, and this lock may be modi-
       fied or released using any of these descriptors.  Furthermore, the lock
       is  released  either  by  an explicit LOCK_UN operation on any of these
       duplicate descriptors, or when all such descriptors have been closed.

       If a process uses open(2) (or similar) to obtain more than one descrip-
       tor  for  the same file, these descriptors are treated independently by
       flock().  An attempt to lock the file using one of these file  descrip-
       tors  may  be  denied  by  a  lock that the calling process has already
       placed via another descriptor.
```