חלק א (55 נקודות)

ענו על **שלוש** השאלות הבאות.

שאלה 1 (24 נקודות)

ממשו סמפור (semaphore) לתיאום בין מספר תהליכונים באותו תהליך ע"י שימוש ב-pipe. אפשר להגביל את המימוש שלכם על ידי כך שלא יתמוך במונה של סמפור אשר ערכו גדול מ- PIPE_BUF.

תזכורת:

Function: int **pipe** (*int filedes* [2])

The `pipe` function creates a pipe and puts the file descriptors for the reading and writing ends of the pipe (respectively) into *filedes* [0] and *filedes* [1].

If successful, `pipe` returns a value of `0`. On failure, `-1` is returned.

Reading or writing pipe data is *atomic* if the size of data written is not greater than `PIPE_BUF`. This means that the data transfer seems to be an instantaneous unit, in that nothing else in the system can observe a state in which it is partially complete. Atomic I/O may not begin right away (it may need to wait for buffer space or for data), but once it does begin it finishes immediately.

Reading or writing a larger amount of data may not be atomic; for example, output data from other processes sharing the descriptor may be interspersed. Also, once PIPE_BUF characters have been written, further writes will block until some characters are read.

(המשך השאלה בעמוד הבא)

```
typedef struct Semaphore_t {
        int ds[2];
} Semaphore;

int sem_init(Semaphore* sem, int count) {    //assume pointer is allocated
        ____________________________________________________
        ____________________________________________________
        ____________________________________________________
}

int sem_wait(Semaphore* sem) {
        ____________________________________________________
        ____________________________________________________
        ____________________________________________________
}

int sem_post(Semaphore* sem) {
        ____________________________________________________
        ____________________________________________________
}
```

הערה: אין צורך לכתוב קוד המטפל בכישלון של קריאות מערכת וגם אפשר להניח שהתוכנית שמשתמשת בסמפורים מתעלמת מ SIGPIPE.

שאלה 2 (16 נקודות)

(8 נק') א. נתון קוד הבא:

```
int memory;

void
handler(int signum)
{
   printf ("%d\n", memory);
   alarm (1);
}

int
main (void)
{
   static int zero = 0, ones = 1;

   /* Initialize the data structures for signal handling. */
   sa.sa_flags = SA_RESTART;
   sigfillset(&sa.sa_mask);
   sa.sa_handler = handler;

   /* Install the signal handler */
   if (sigaction(SIGALRM, &sa, NULL) < 0)
               abort();

   memory = zero;
   alarm (1);
   while (1)
     {
       memory = zero;
       memory = ones;
     }
}
```

הפונקציה alarm(int sec) שולחת את הסיגנל SIGALARM לאחר sec שניות. מה יהיה פלט התכנית? אם יש יותר מאפשרות אחת, ציינו את כל האפשרויות.

(המשך השאלה בעמוד הבא)

(8 נק') ב. נשנה את התכנית בצורה הבאה:

```
struct two_words { int a, b; } memory;

void
handler(int signum)
{
   printf ("%d,%d\n", memory.a, memory.b);
   alarm (1);
}

int
main (void)
{
   static struct two_words zeros = { 0, 0 }, ones = { 1, 1 };

   /* Initialize the data structures for signal handling. */
   sa.sa_flags = SA_RESTART;
   sigfillset(&sa.sa_mask);
   sa.sa_handler = handler;

   /* Install the signal handler */
   if (sigaction(SIGALRM, &sa, NULL) < 0)
               abort();

   memory = zeros;
   alarm (1);
   while (1)
     {
       memory = zeros;
       memory = ones;
     }
}
```

מה יהיה פלט התכנית כעת? אם קיימת יותר מאפשרות אחת, ציינו את כל האפשרויות.

שאלה 3 (15 נקודות)

עקב השימוש במנגנון ה-non write through buffer cache הכתיבות לדיסק נדחות לזמן מאוחר יותר (זמן אקראי בין 30 שניות לדקה). יתרה מזו, הגרעין יכול לשנות את סדר הכתיבות לצורך שיפור הביצועים (אך לא מחליפים סדר של פעולות על אותו block). הציעו מבנה נתונים המנהל בלוקים ב-buffer cache.

(5 נק') א. תארו את מבנה הנתונים ופרטו כיצד מתאפשר שלא יופר סדר פעולות על אותו בלוק.

(5 נק') ב. מהם זמני ההכנסה/הוצאה של הבלוקים למבנה נתונים (זמן במונחים של O)?

(5 נק') ג. האם יש הבדל בכמות הגישות לדיסק בין שתי התכניות הבאות:

```
#define FILE "tmpfile"
int main (int argc, char **argv)
{
    int i, fd;
    for (i = 0; i < 100; i++) {
      fd = open (FILE, O_CREAT|O_WRONLY);
      close (fd);          unlink (FILE);   /* delete
FILE */
    }
    return 0;
}
```

```
#define FILE "/tmp/dir1/dir2/dir3/tmpfile"
int main (int argc, char **argv)
{
    int i, fd;
    for (i = 0; i < 100; i++) {
      fd = open (FILE, O_CREAT|O_WRONLY);
      close (fd);          unlink (FILE);   /* delete
FILE */
    }
    return 0;
}
```

ענו על השאלה במקרה של non write through buffer cache ובמקרה של write through buffer cache. נמקו.

המשך הבחינה בעמוד הבא

חלק ב (25 נקודות)

ענו על השאלות הבאות. משקל כל שאלה 5 נקודות.

שאלה 4

מהו TLB (translation lookaside buffer)?

שאלה 5

מהו condition variable?

שאלה 6

מהו האלגוריתם האופטימאלי להחלפת דפים?

שאלה 7

מדוע נועלים דפים בזיכרון בעת העברת נתונים על ידי DMA (Direct Memory Access)?

שאלה 9

מהו Write-Through Cache?

חלק ג (20 נקודות)

שאלות רב-ברירה (אמריקאיות). משקל כל שאלה 5 נקודות.

שאלה 10

במערכת ניהול זיכרון מדפדף נהוגה מדיניות של prepaging – הבאת מספר כלשהו של דפים השייכים לתהליך בכל פעם שהתהליך עובר מהדיסק לזיכרון. בחרו טענה נכונה:

א. הבאת קבוצת דפים שנבחרה בקפידה היא פעולה שיכולה להוריד את כמות פסיקות הדפים במערכת

ב. אין טעם להביא דפים מראש שכן הדבר כרוך בפעולת פינוי בשלב מאוחר יותר

ג. מדיניות זו עומדת בסתירה לעקרון קבוצת העבודה

ד. מדיניות זו ניתנת למימוש רק בשילוב עם האלגוריתם האופטימאלי להחלפת הדפים

שאלה 11

בחרו את הפעולה היקרה ביותר במונחים של מעברי בלוקים של הדיסק (disk block transfers) בהנחה שלא קיימים נתונים רלוונטיים בזיכרון המטמון (buffer cache):

א. פתיחת קובץ באמצעות open

ב. קריאת בלוק אחד באמצעות read

ג. קריאת תו אחד באמצעות getc

ד. התשובות א' וב' הן הנכונות

שאלה 12

איזו פעולה מן הפעולות הבאות אפשר לבצע אך ורק במצב ראשוני (kernel mode) במערכת ההפעלה Linux?

א. חסימת פסיקות החומרה (disabling hardware interrupts)

ב. החלפת תהליכונים (thread switch) כאשר מדובר בספריית תהליכונים ברמת המשתמש

ג. השמת ערך במשתנה גלובאלי

ד. את כל שלוש הפעולות הנ"ל יש לאפשר אך ורק במצב ראשוני

המשך הבחינה בעמוד הבא

שאלה 13

לפניכם פסאודו-קוד של משחק רובוטים שבו 3 רובוטים בצבעים (אדום, כחול, ירוק) מבצעים תזוזות בסדר כלשהו.

```
int sem[3];
sem[0] = 1; sem[1] = 1; sem[2] = 1;
```

R_Robot (){	G_Robot (){	B_Robot (){
while(true) {	while(true) {	while(true) {
down(sem[0]);	down(sem[1]);	down(sem[2]);
<Make Move>	<Make Move>	<Make Move>
up(sem[1]);	up(sem[2]);	up(sem[0]);
}	}	}
}	}	}

```
int main(){
        run_new_thread(R_Robot);
        run_new_thread(G_Robot);
        run_new_thread(B_Robot);
}
```

האם סדר התזוזות של הרובוטים נקבע על ידי השימוש בסמפורים כדלהלן?
אם כן, מהו הסדר?

א. כן. אדום, ירוק, כחול וחוזר חלילה

ב. כן. אדום, כחול,ירוק וחוזר חלילה

ג. כן. אדום, כחול, אדום, כחול, ירוק וחוזר חלילה

ד. לא. הסדר ייקבע על ידי מתזמן (scheduler)

בהצלחה!