שאלה 1

כרגיל בבחינה במ"ה, הכי חשוב זה להיות פשוט.

בקרנל שלנו יש כמה וכמה מקומות בהם יש טווח גדול למזהה עצם, ומשתמשים במאגר קטן. לדוגמא, כך זה בווקטור התהליכים, וכך זה במאגר ה־inode's.

אז אפשרות הסבירה למאגר האירועים היא:

```
#define EVT_SIZE 50
struct evt {
    int num;
    int state;
} evt[EVT_SIZE];

#define EVT_SET 2
#define EVT_CLEAR 1
#define EVT_UNUSED 0

struct spinlock evtLock;
```

המימוש עכשיו הוא מיידי בהתאם לתיאור בשאלה. יש כמובן כל מיני שיטות אפשריות, ורובן יותר פשוטות ממוטקס.

כיון שברור שצריך לחפש במאגר, ולהכניס למאגר אם אין, אזי צריך רוטינה שעושה זאת, נממש אותה בסוף.

המימושים של שלושת הפונקציות הם די מידיים. העברה למצב CLEAR:

```
int event_clear(int num) {
    acquire(&evtLock);
    if ((struct evt *e = evt_insert(num)) == NULL) {
        release(&evtLock);
        return (-1);
    }
    int ret = e->state;
    e->state = EVT_CLEAR;
    release(&evtLock);
    return (ret);
}
```

מעבר למצב SET די זהה למעבר למצב CLEAR. השנוי הוא רק בזה שצריך להעיר אם מישהוא מחכה.

```
int event_set(int num) {
    acquire(&evtLock);
    if ((struct evt *e = evt_insert(num)) == NULL) {
        release(&evtLock);
        return (-1);
    }
    int ret = p->state;
    e->state = EVT_SET;
    wakeup(e);
    release(&evtLock);
    return (ret);
}
```

אחרון חביב, ההמתנה. קוד קלאסי של המתנה:

```
int event_wait(int num) {
    acquire(&evtLock);
    if ((struct evt *e = evt_insert(num)) == NULL) {
        release(&evtLock);
        return (-1);
    }
    while (e->state != EVT_SET) {
        if (proc->killed) {
            release(&evtLock);
            return(-1);
        }
        sleep(e, &evtLock);
    }
    e->state = EVT_CLEAR;
    release(&evtLock);
    return(0);
}
```

החוב שנשאר:

```
struct evt *evt_insert(int num) {
    struct evt *e, *f = NULL;
    if (num >= MAX_EVENTS)
```

```
        return (-1);
    for (e = &evt[0]; e < &evt[EVT_SIZE]; e++) {
        if (e->state != EVT_UNUSED && e->num == num)
            return(e);
        if (f == NULL && e->state == EVT_UNUSED)
            f = e;
    }
    if (f != NULL) {
        f->num = num;
        f->state = EVT_CLEAR;
    }
    return (f);
}
```

שיטה אחרת, דורשת קצת יותר עבודה. יש צורך להוסיף שדות למבנה proc:

```
strut proc {
    int usingEvent;
    int numEvent;
}
```

העברה למצב CLEAR. השנוי ביחס למימוש הקודם הוא שננסה להיפטר מהכניסה אם אין כרגע המתנה, מה שידרוש רוטינת חיפוש חדשה, שתמומש בסוף. בנוסף יש אילוץ מנעולים בגלל הקוד של ההמתנה, אז הפונקציה תפוצל לשניים:

```
int event_clear(num) {
    acquire(&evtlock);
    int ret = do_eventclear(num);
    release(&evtlock);
    return(ret);
}

int do_eventclear(int num) {
    if ((struct evt *e = evt_find(num)) == NULL)
        return (EVT_CLEAR);

    int ret = e->state;
```

```
    acquire(&ptable.lock);
    for (struct proc *p = &ptable.proc[0]; p < &ptable.proc[NPROC]; p++) {
        if (p->state != UNUSED && p->usingEvent  &&  p->numEvent == num) {
            release(&ptable.lock);
            e->state = EVT_CLEAR;
            return (ret);
        }
    }
    release(&ptable.lock);
    e->state = EVT_UNUSED;
    return (ret);
}
```

מעבר למצב SET זהה לקוד הקודם.

ההמתנה בגירסה הנוכחית **תציף** את העובדה שיש המתנה. בסיום נקרא ל־do_eventclear כדי לנסות לנקות את הכניסה. הקוד העיקרי הוא כמו הקוד קודם.

```
int event_wait(int num) {
    acquire(&evtLock);
    if ((struct evt *e = evt_insert(num)) == NULL) {
        release(&evtLock);
        return (-1);
    }
    proc->usingEvent = 1;
    proc->numEvent = num;
    while (e->state != EVT_SET) {
        if (proc->killed)
            break;
        sleep(e, &evtLock);
    }
    proc->usingEvent = 0;
    do_eventclear(num);
    release(&evtLock);
    return(0);
}
```

החוב שנשאר:

```
struct evt *evt_find(int num) {
```

```
    struct evt *e;
    if (num >= MAX_EVENTS)
        return (-1);
    for (e = &evt[0]; e < &evt[EVT_SIZE]; e++) {
        if (e->state != EVT_UNUSED && e->num == num)
            return(e);
    }
    return (NULL);
}
```

שאלה 2 זוהי האופציה החביבה עלי.

```
int off[] = {
    (int)&((struct trapframe *)0)->eax,
    (int)&((struct trapframe *)0)->ecx,
    (int)&((struct trapframe *)0)->edx,
    (int)&((struct trapframe *)0)->ebx,
    (int)&((struct trapframe *)0)->esi,
    (int)&((struct trapframe *)0)->edi,
    (int)&((struct trapframe *)0)->ebx,
    (int)&((struct trapframe *)0)->esp,
};

int store(int reg, int val) {
    addr = *(int *)(((char *)proc->tf) + off[reg]);
    if (addr >= proc->sz  ||  addr+4 >= proc->sz)
        return (-1);

    *addr = val;
    return (0);
}
```

הקוד הבא סביר גם כן.

```
int store(int reg, int val) {
    switch (reg) {
    case 0:
        addr = proc->tf->eax;
        break;

    case 1:
        addr = proc->tf->ecx;
        break;

    case 2:
        addr = proc->tf->edx;
        break;
```

```
    case 3:
        addr = proc->tf->ebx;
        break;

    case 4:
        addr = proc->tf->esi;
        break;

    case 5:
        addr = proc->tf->edi;
        break;

    case 6:
        addr = proc->tf->ebx;
        break;

    case 7:
        addr = proc->tf->esp;
        break;
    }
    if (addr >= proc->sz  ||  addr+4 >= proc->sz)
        return (-1);

    *addr = val;
    return (0);
}
```