

מערכות הפעלה -- פתרון מבחן ההודעות

שאלה 1

רוצים לשפר ב-Xinu את מערכת שליחת ההודעות. כל הודעה תוכל להיות בגודל בלתי מוגבל. בשליחת ההודעה תנתן כתובת ההודעה ואורך בבתיים של ההודעה. בקבלת ההודעה תינתן כתובת להודעה והאורך המקסימלי לקבלת ההודעה (כל שארית החורגת מאורך זה תזרק). מנגנון משלוח ההודעות ידאג להעברת ההודעות כך שהשולח לא יידרש לשמור אותם אצלו, ויוכל להשתמש במשתנים מחדש.

יש לממש את קריאות המערכת הבאות: blk_receive, blk_send, blk_sendf. קריאת המערכת blk_receive תחזיר את אורך ההודעה שנקלטה. מערכת ההודעות החדשה מחליפה את הקודמת. ניתן להשתמש בשדות המיועדים להודעות בטבלת התהליכים proctab בלבד אך אין להשתמש בשדות נוספים (שלא שימשו במקור להודעות).

פתרון:

- השדה pmsg ישמש כעת כמצביע להודעה והשדה phasmmsg יסמן את גודל ההודעה
- phasmmsg = 0 יסמן כי לא ממתינה לתהליך הודעה

```
SYSCALL blk_receive(char *addr, int max_length)
```

```
{
    struct pentry *pptr;
    char *msg;
    int ps;
    int len;
    int i;
    disable(ps);
    if (addr == NULL || max_length < 0) {
        restore(ps);
        return(SYSERR);
    }
    pptr = &proctab[currpid];
    if (pptr->phasmmsg == 0) {
        pptr->pstate = PRRECV;
        resched();
    }
    if (max_length > pptr->phasmmsg)
        len = pptr->phasmmsg;
    else len = max_length;
    msg = (char *) pptr->pmsg;
    for (i=0; i<len; i++)
        addr[i] = msg[i];
    freemem(msg, pptr->phasmmsg);
    pptr->phasmmsg = 0;
    restore(ps);
    return len;
}
```

```

SYSCALL blk_sendf( int pid, char *addr, int length)
{
    struct pentry pptr;
    int i, ps;
    char *msg;
    disable(ps);
    if (isbadpid(pid) || ((pptr=&proctab[pid])->pstate == PRFREE) ||
        (msg = getmem(len) == NULL)) {
        restore(ps);
        return SYSERR;
    }
    if (pptr->phasmmsg != 0) freemem(pptr->pmsg, pptr->phasmmsg);
    for (i=0; i<length; i++) msg[i] = addr[i];
    pptr->pmsg = msg;
    pptr->phasmmsg = length;
    if (pptr->pstate == PRRECV)
    {
        ready(pid);
        resched();
    }
    restore(ps);
    return(OK);
}

```

```

SYSCALL blk_sendf( int pid, char *addr, int length)
{
    struct pentry pptr;
    int i, ps;
    char *msg;
    disable(ps);
    if (isbadpid(pid) || ((pptr=&proctab[pid])->pstate == PRFREE) ||
        pptr->phasmmsg != 0 || (msg = getmem(len) == NULL)) {
        restore(ps);
        return SYSERR;
    }
    for (i=0; i<length; i++) msg[i] = addr[i];
    pptr->pmsg = msg;
    pptr->phasmmsg = length;
    if (pptr->pstate == PRRECV)
    {
        ready(pid);
        resched();
    }
    restore(ps);
    return(OK);
}

```

שאלה 2

יש לשנות את מנגנון העדיפויות של XINU כך שתהליכים בעלי עדיפות גבוהה לא יחסמו תהליכים בעלי עדיפות נמוכה יותר. במקום זאת כל תהליך יקבל זמן ריצה יחסי הפרופורציונלי לעדיפותו. יש לתת תאור מילולי מדויק ומנומק על השינויים שיש לבצע ומיקומם במערכת. יש לשים לב באיזה קריאות מערכת או פונקציות פנימיות יש התייחסות לשינויים אלה.

פתרון:

- התור ready ינוהל בשיטת FIFO ולא בשיטת תור עדיפויות
- השינויים בפונקציה `resched()`
 - ✓ במקום הבדיקה
`if (sys_pcxget() == Q || lastkey(rdytail) < optr->pprio)`
נבצע בדיקה
`if (sys_pcxget() == 0 || isempty(rdyhead) || preempt > 0)`
 - ✓ במקום `nptr = &proctab[(currpri = getlast(rdytail))];` נבצע
`nptr = &proctab[(currpri = getfirst(rdytail))];`
 - ✓ במקום השורה `preempt = nptr->pprio;` נבצע `preempt = QUANTUM;`
 - ✓ במקום `insert(curripid,rdyhead,optr->pprio);` נבצע
`enqueue(curripid,rdytail);`
- בכל מקום בו מתבצעת הכנסה לתור ready, למשל בפונקציה `ready()` יש להשתמש ב-
`enqueue`
במקום `insert`
- יש להתאים טיפול בתהליכי המערכת למדיניות תזמון החדשה

שאלה 3

בחר בכל סעיף את התשובה הכי נכונה מהאפשרויות:

- א. בעיית השברור הפנימי:
1. נפתרת מיידית על ידי שימוש בהקצאת best fit
 2. נפתרת מיידית על ידי שימוש בהקצאת worst fit
 3. לא קיימת ב-Xinu
 4. אף אחת מהתשובות 1, 2, 3 אינה נכונה

ב. זיכרון וירטואלי:

1. פותר את בעיית השברור הפנימי ואת בעיית השברור החיצוני
2. פותר את בעיית השברור הפנימי אך לא את בעיית השברור החיצוני
3. פותר את בעיית השברור החיצוני אך לא את בעיית השברור הפנימי
4. לא פותר אך אחת מהבעיות הנ"ל

ג. אתחול התורים של הסמפורים:

1. נעשה ב-screate
2. לא נעשה בכלל
3. נעשה באתחול Xinu
4. אף אחת מהתשובות 1,2,3 אינה נכונה

ד. לאחר ביצוע

```
{ int ps;  
  disable(ps);  
  disable(ps);  
  ...  
}
```

1. די לבצע restore(ps) על מנת שתוכלנה להתבצע פסיקות
2. דרושות שתי קריאות restore(ps) בכדי שפסיקות תוכלנה להתבצע
3. אין שום דרך לאפשר יותר פסיקות
4. אף אחת מהתשובות אינה נכונה

הסבר: בפעולת disable השניה דורסים את המשתנה ps השומר מצב פסיקות התחלתי ע"י מצב פסיקות חדש שהוא כבוי כתוצאה מפעולת disable הראשונה. לכן לא משנה כמה פעמים נבצע restore(ps) הפסיקות ישאר כבוי. ניתן להדליק דגל פסיקות ע"י enable().

ה. תהליך A מבצע sleep(1) ומייד אחר כך תהליך B מבצע sleep(1).
ניצד יראה ראש התור של sleep (ראש התור משמאל)?

1. A[1], B[0]
2. A[1], B[1]
3. B[1], A[0]
4. B[1], A[1]

ו. כאשר מספר תהליכים שולחים הודעות לאותו תהליך עם sendf(pid, msg) רק ההודעה האחרונה תקרא. האם יכול תהליך המקבל לבדוק כמה הודעות נדרסו?

1. כן, ברמת מערכת הפעלה בלבד
2. כן, ברמת משתמש בלבד
3. כן, גם ברמת מערכת ההפעלה וברמת משתמש
4. לא

הסבר: ניתן לבדוק שדה phasmsg של התהליך ב-proctab אך גישה לטבלה הזו מותרת רק ברמת מערכת הפעלה.

ז. שתי השורות האחרונות בפונקציה intcom הן:

```
add sp, 2
iret
```

מה יקרה אם נחליף אותם ב-:

```
add sp, 6
iret
```

1. לא יהיה שום שינוי
2. ביציאה מ- intcom תתבצע שיגרת הפסיקה המקורית
3. ביציאה מ- intcom תתבצע קפיצה למקום לא מוגדר והמחשב ייתקע
4. ביציאה מ- intcom תתבצע שוב שיגרת פסיקה של Xinu

- ח. מתי מתבצעת ב-resched() קריאה ctxsw(a,a) (אותו ארגומנט לשני הפרמטרים)?
1. לאחר פסיקה וכאשר תהליך האפס הוא היחיד במערכת לעולם לא
 2. תמיד לאחר הפסיקה
 3. כאשר התהליך הנוכחי מבקש זיכרון דינאמי והמערכת אינה יכולה להקצות
 4. כאשר התהליך הנוכחי מבקש זיכרון דינאמי והמערכת אינה יכולה להקצות

ט. מתבצע קטע הקוד הבא:

```
#include<conf.h>
#include<kernel.h>
int sema;

int xmain()
{
    int prA(), prB();
    sema = screate(0);
    resume(create(prA, INITSTK, INITPRIO-2, "prA",0));
    resume(create(prB, INITSTK, INITPRIO-1, "prB",0));
}

int prA()
{
    signal(sema);
    wait(sema);
    do_something();
    printf("a");
    signal(sema);
}
```

```

int prB()
{
    wait(sema);
    do_something_else();
    printf("b");
    signal(sema);
}

```

לגבי `do_something_else()` ו-`do_something()` ידוע ששתי הפונקציות אינן קוראות לשום קריאת מערכת. מהו הפלט המסתבר ביותר ?

- .1 ab
- .2 ab או ba
- .3 ba
- .4 a
- .5 b
- .6 b או a

י. מה היה קורה אם היינו משנים את ערכו של QUANTUM מ-1 ל-1- ?

- 1. הדבר לא היה משפיע על פעולת המערכת
- 2. המערכת היתה עובדת יותר מהר
- 3. תהליכים היו מתחלפים לפני שהיו מספיקים לרוץ
- 4. אף תשובה אינה נכונה