

## טבלת התהליכיים

טבלת התהליכיים, ה-proctab, הינה מערך של רשומות המכיל אינפורמציה של כל תהליך שנחוץ להיות נגיש למערכת הפעלה בכל רגע נתון מספר שיקולים. האינפורמציה היא מצב התהליך, העדיפות שלו, מספר הסמפור שהוא ממתין לו (במידה זהה המצב), מבנה הנתונים של מנגנון ההודעות, מצביעים למחסנית הפרטיה של התהליך ועוד שדות עוזר: שם התהליך, מספר הפרמטרים וקוד התחלתי.

ההכרזה ב-c-initial.h המהווה הקצהה של שטח ל-proctab הוא:  
struct pentry proctab[NPROC].

## מצבי התהליכיים

בכל רגע נתון כל תהליך נמצא באחד מסידרה מוגדרת של מצבים.  
רשימת המצבים ב-XINU הסטנדרטי הם:

**PRFREE**

התהליכי איןו בשימוש, חופשי להקצאה בעתיד.

**PRCURR**

התהליכי הוא זה שמקבל שירות CPU.

**PRREADY**

התהליכי מעוניין בשירותי CPU, אך הוא משרת תהליכי אחר וההתהליכי הזה מתחכה לתורו.

**PRSUSP**

התהליכי מושהה לזמן בלתי מוגבל.

**PRSLEEP**

התהליכי רודם לזמן קבוע.

**PRWAIT**

התהיליך ממתין למספר.

PRRCEV

התהיליך ממתין להודעה.

### מנגנון החלפת התהיליכים

כל הנושא של ייצרת תהיליכים, הפעלתם, והרצתם כביבול במקביל אפשרי אך ורק הודות למנגנון המאפשר עצירת ריצתו של התהיליך המורוץ כרגע ע"י ה-CPU והחלפתו בתהיליך אחר. זה מהלך לא פשוט בכלל, כרוך במספר מהויבויות, אבל אפשר לדבר על שורה תחתונה, או שתיים כאילו:

העובדת שה-CPU משרת **טהיליך מסוים** נובע בעיקר מהעובדת שאוגרי ה-CPU (IP, AX, BX, ..., SP), **מכילים את הערכיהם "שלו"** – הערכיהם שהקוד שלו מעוניין בהם. זה קובע את הפקודות הבאות לבצע, המחסנית הנוכחית, **משמעות ערכי האוגרים הכלליים וכו'**.

בפועל החלפת תהיליכים היא, בין השאר אבל בעיקר, מהלך של **שימור אוגרים של התהיליך היוצא**, ו**שיחזור אוגרים של התהיליך הנכנס**. באנגלית **ההעברה שליטה** נקראית **context switch**, בתרגום **חופשי" החלפת ההקשר**" (של ה-CPU) וזה בעצם מה שהחלפת ערכי האוגרים עשויה. המימוש ב-UNIX ו-UNIX מבוסס על המרכזיות של מחסנית המערכת, זו שמוצבעת על ידי האוגרים BP, SS, SP:  **לכל תהיליך יש שטח זכרון המשמש מחסנית פרטית משלו**, **והחלפת המחסנית הפעילה** (תוכן האוגרים BP, SP, SS) מזו של התהיליך היוצא **לוזה של הגבנס** עושה את עיקר המשימה.

השאלה העיקרית היא מהי הסיבה שהחלפת המחסניות די בכך למש את העברת השליטה מטהיליך אחד לשני. הסיבה שדי בכך היא שיחד עם מנגנון הקריאה לפונקציה של C, פקודת המכונה CALL ומנגנון הפסיקות, קביעת המחסנית קובעת בסופו של דבר את הפקודות הבאות לביצוע (האוגרים IP ו-CS), את המשתנים האוטומטיים הפעילים (דרך BP) ואת משתני אוגר SI ו-DI. על האוגרים הכלליים האחרים (AX, BX, CX, DX, C) מילא אינו מסתמך עליהם. קביעת אוגר הדגלים נעשה ע"י קוד מיוחד של UNIX וגם הוא נשמר במחסנית. אשר לאוגרי הסגמנטים, הם אינם משתנים בקורס במודול של C Turbo שאנו עובדים אליו, הם עשויים לשנות רק בפסיקות, וגם שם הם נשמרים ומשתוחרים דרך המחסנית.

בנוסף להחלפת ערבי האוגרים (בעיקר החלפת המחסנית הפעילה) למנגנון החלפת התהיליכים שתி מחיוביות נוספת: מימוש מדיניות החלפת התהיליכים, ועדכון מבנה הנתונים הגלובלי של UNIX.

בכדי להמשיך אנחנו צריכים להכיר סידרה של מושגים ורקע נוסף.

### מדיניות החלפת התהיליכים של UNIX

UNIX דואג שכל רגע התהיליך בעל העדיפות הגבוהה ביותר במערכת שמעוני ב-CPU הוא זה שמקבל אותו. מידת ויש יותר מתהיליך אחד בעל העדיפות הגבוהה ביותר, התהיליכים הללו מקבלים שירות CPU, לפי מסכת זמן, לפי סדר של ותק בהמתנה.

ב-UNIX כפי שאנו מקבלים אותו, אם יש תהיליך יחיד בעל עדיפות גבוהה ביותר במערכת, הוא יקבל שירות CPU כל עוד הוא מעוניין בהם והוא יוחלף רק אם יותר על ה-CPU ויעבור למצב אחר. רק אם יש יותר מתהיליך אחד בעדיפות הגבוהה יתבצע החלפת תהיליך הנוכחי גם אם הוא מעוניין המשיך.

מבחינת resched משמעות הדבר שאם לתהיליך הנוכחי יש עדיפות גבוהה ממש מכל תהיליך מתינו בתור ה-READY הוא ממשיך לרגע, גם אם המכסה שלו פגה. במקרה של שוויון, (בגרסה שלנו) התהיליך ממשיך לרגע אם המכסה שלו לא פגה.

מצב נוסף שבו לא יתבצע החלפת תהיליכים היא אם המערכת לא יכולה להרשות זאת - יש דגל מערכת מיוחד לסמן מצב כזה. הסיבות למצב כזה הם די טכניות, לא נכנס לזה כאן.

### מימוש מנגנון החלפת התהיליכים

קבוצת התהיליכים שמעוניינים בשירותי ה-CPU הם התהיליך הנוכחי היחיד שבפועל מקבל אותו וקבוצת התהיליכים במצב PRREADY המסורדים בתור עדיפויות (המש FIFO בין תהיליכים שווי עדיפות).

מערכת UNIX כתובה בצורה כזו שכל מקרה שבו יש צורך בהחלפת תהיליכים, מתחבצע קרייה לרוטינה C **resched** בשם **READY**. במידה ויש החלפת תהיליכים, העברת השליטה בפועל נעשית בתוך resched ע"י קרייה לרוטינה אסמלטי פנימית בשם `swctxsw`, התהיליך הנכנס נשלף מתור ה-`READY` ע"י הרוטינה `getlast`. במידה והטהיליך היוצא נעשית `READY`

(אנחנו נראה מתי זה קורה) הוא מוכנס לתוך ה-`ready` ע"י הרוטינה `insert`. כל יתר המחויבויות של מגנון החלפת התהיליכים (יתר שימוש המדיניות ו שינוי מבנה הנתונים הגלובלי במקורה של החלפה) נעשה רקוד של `resched`.

היתרון בכך שיש רוטינה אחת לכל המקרים של החלפת התהיליכים הוא שימוש מגנון החלפת התהיליכים מרוכז במקום אחד, ובכך להכניס שינויים כלשהו צריך להתיחס רק אליו. החסרון הוא ש-`resched` נקרא במצבים שונים של המערכת והקובד צריך להביא את כל המקרים בחשבון.

כאמור, קריאה ל-`resched` היא רק בדיקה, האם יש צורך לבצע החלפת תהיליכים. למעשה, הקריאה ל-`resched` נופלת לתוכה שתי קטגוריות עיקריות:

1. קריאות ל-`resched` ביוזמת התהיליך.
2. קריאות ל-`resched` ביוזמת המערכת.

קריאות ל-`resched` ביוזמת התהיליך הם למעשה חלק מויתור של התהיליך על ה-CPU. זה קורה במקרים שתהיליך מגלה שהוא צריך להמתין לספוף, ממתיין להודעה, משזה את עצמו, מרדדים את עצמו לזמן קצר או פשוט חדל להתקים. במצבים הללו התהיליך הנוכחי דואג לשנות את מצבו לפני הקריאה ל-`resched`. זה גם הדרך ש-`resched` חש שמדובר בסוג הראשון של קריאה ל-`resched`. בסוג זהה של קריאות ל-`resched`, החלפת התהיליכים מתחכמת בכל מקרה, גורל התהיליך היוצא הוא **להישאר במצב שבחר בעצמו**.

קריאות ל-`resched` ביוזמת המערכת הם מצבים שבהם המערכת יוזמת קריאה ל-`resched` משיקולים שלה, למטרות שתהיליך הנוכחי מעוניין להמשיך לרוץ. הדבר קורה או משום שמכסת הזמן של התהיליך נגמר, או שחל שינוי בסטטוס של התהיליך שעד עכשו לא היה מעוניין ב-CPU, אבל עכשו כן מעוניין בו. הדבר יכול לנבוע מכך שתהיליך יצא מההיה, נעשה `signal` לאחד הסופרים, שתהיליך ממתיין להודעה קיבל אחת כזו, או שلتהיליך רדום נגמר זמן ההרדמה.

הדרך ש-`resched` חש שמדובר בסוג כזה של קריאה ל-`resched` הוא שתהיליך הנוכחי מגיע ל-`PRCURR` במצב `PRREADY`. **כאן לא בהכרך** שיתבצע החלפת תהיליכים, בהתאם למединיות החלפת התהיליכים. במידה ויש בכלל זאת החלפת תהיליכים, התהיליך היוצא **נעשה PRREADY**.

| במקרה של החלפה, גורל התהיליך הנוכחי        | האם מחבצתה החלפה?   | כיצד resched מבחין בין 2 סוגי הקריאה?   | הסיבה לקריאה   | סוג הקריאה ל-resched |
|--|---|---|--|----------------------|
| נשאר במצב שבחר לעצמו                       | כן, בכל מקרה  | שדה ה-pstate של התהיליך הנוכחי ≠ PRCURR | ויתור על זכאות על המעבד של התהיליך הנוכחי              | ביוזמת התהיליך       |
| resched אחראי להעביר את התהיליך ל- PRREADY | יכול להיות שלא, resched עשוי להחליט שלא לבצע החלפה, לפי מדיניות החלפת התהיליכים | שדה ה-pstate של התהיליך הנוכחי = PRCURR | שינוי במצב המערכת, הדורש הערכה מחדש על הזכאות על המעבד | ביוזמת המערכת        |

## הרוטינה ctxsw

בכדי להבין את הקוד של ctxsw צריך לקרוא אותו מתוך ההנחה, שכל תהליך שאינו רץ ברגע נמצא באמצע קריאה ל-ctxsw. ההנחה זו היא בפירוש נcona לכל תהליך שרצה פעם, ומדוברה לתהליכיים חדשים שטרם רצנו.

תפקידו העיקרי של ctxsw הוא לשמור את ערך אוגר ה-SP של התהליך היוצא בשדה ה-pregs של הרשומה שלו (התהליך) ב-protab ולשלוף את ערך ה-pregs של התהליך הנכנס לתוכן האוגר SP. שימושות הדבר הפעלת המחסנית של התהליך הנכנס על פני המחסנית של התהליך היוצא.

מלבד האמור לעיל, ctxsw משמר את SI ו-DI ומתחילה את אוגר הדגלים עבור תהליכיים חדשים.

## החלפת ערכי אוגרים בהחלפת הקשר – סיכום

כפי שנאמר קודם, העברת השליטה בין ה手続きים, כולל החלפת הקשר, הוא תהליך של שימור של ערכי האוגרים של התהליך היוצא והחלפתם בערכי התהליך הנכנס. זה מהלך הנפרש על פני כמה רוטינות שבמרכוו החלפת הצבעה של המחסנית ה-CPU.

ב-UINU קבוצת האוגרים הרלוונטיות הינה AX, BX, CX, DX, BP, SP, SI, DI, IP, CS, SS, DS, FLAGS.

כאן נפרט איך נשמר/מוחלף כל אחד מהאגרים הללו:

### האגרים

AX, BX, CX, DX, FLAGS

איןם צריכים להישמר בהסתעפות רגליות של טורבו C ולכן נשמרים רק בהקשר של פסיקות על ידי מה שקרה רוטינת הטיפול בפסיקות של UINU (להלן "סדרון הפסיקות") שאותו נראה בהמשך.

### האגרים

SI, DI, BP

כאשר מדובר בקריאות מערכת, האוגרים הללו נשמרים ומשתוחרים ע"י מוסכמת קרייה של פונקציות C (גם על ידי `ctxsw` כחלק מזה). במקרה של פסיקה הם נשמרים סופית ע"י סדרון הפסיקות.

#### האוגרים

##### CS, SS, DS, ES

איןם משתנים במהלך הריצה של UNIX ולכך הם משמרים ומשתוחרים רק ע"י סדרון הפסיקות.

#### האוגר

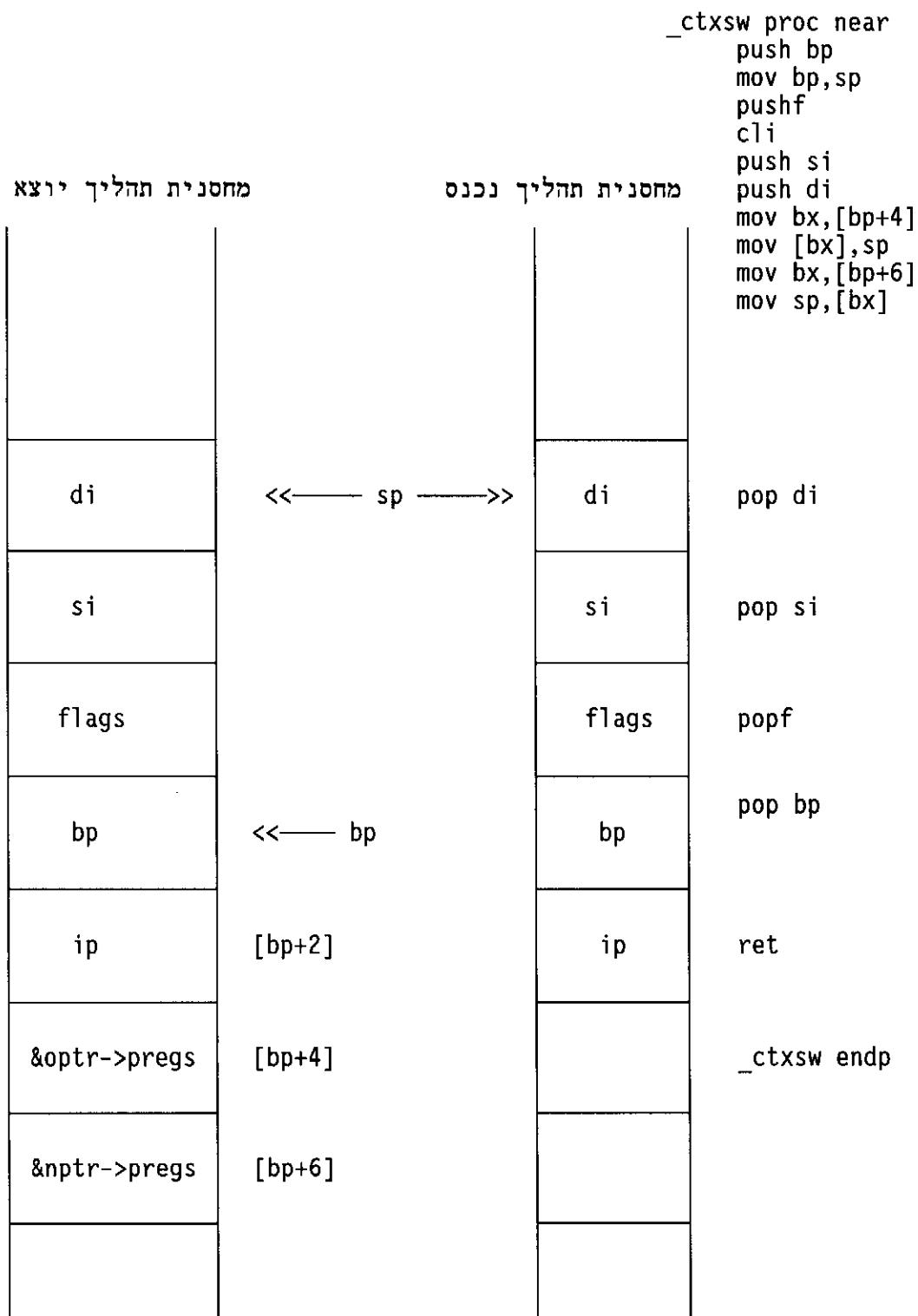
##### SP

נשמר / משתהזר דרך `proctab` על ידי `ctxsw` ובהמשך או, במקרה של קריית מערכת, ע"י מוסכמת הקריאה של טורבו C או, במקרה של פסיקה, ע"י סדרון הפסיקות.

#### האוגר

##### IP

ನשמר ומשתהזר בעקבות השינוי ב-SP, שוב במקרה של קריית מערכת, ע"י מוסכמת הקריאה של טורבו C או, במקרה של פסיקה, ע"י סדרון הפסיקות.



```

/* proc.h - isbadpid */
/* 8088 version */

/* process table declarations and defined constants */

#ifndef NPROC          /* set the number of processes */
#define NPROC           30   /* allowed if not already done */
#endif

/* process state constants */

#define PRCURR          '\01'  /* process is currently running */
#define PRFREE           '\02'  /* process slot is free */
#define PRREADY          '\03'  /* process is on ready queue */
#define PRRECV            '\04'  /* process waiting for message */
#define PRSLEEP           '\05'  /* process is sleeping */
#define PRSUSP            '\06'  /* process is suspended */
#define PRWAIT            '\07'  /* process is on semaphore queue */

/* miscellaneous process definitions */

#define PNMLEN           9    /* length of process "name" */
#define NULLPROC          0    /* id of the null process; it
                                /* is always eligible to run */

#define isbadpid(x)      (x<=0 || x>=NPROC)

/* process table entry */

struct pentry {
    char    pstate;          /* process state: PRCURR, etc. */
    int     pprio;           /* process priority */
    int     psem;             /* semaphore if process waiting */
    int     pmmsg;            /* message sent to this process */
    int     phasmgs;          /* nonzero iff pmmsg is valid */
    char    *pregs;           /* saved environment */
    char    *pbase;            /* base of run time stack */
    word    plen;              /* stack length in bytes */
    char    pname[PNMLEN+1];  /* process name */
    int     pargs;             /* initial number of arguments */
    int     (*paddr)();        /* initial code address */
};

extern struct pentry proctab[];
extern int numproc;          /* currently active processes */
extern int nextproc;         /* search point for free slot */
extern int currid;           /* currently executing process */

```

```

/* resched.c - resched */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>

/*
 * resched -- reschedule processor to highest priority ready process
 *
 * Notes:      Upon entry, currpid gives current process id.
 *              Proctab[currpid].pstate gives correct NEXT state for
 *              current process if other than PRCURR.
 */
int     resched()
{
    register struct pentry  *optr; /* pointer to old process entry */
    register struct pentry  *nptr; /* pointer to new process entry */

    optr = &proctab[currpid];
    if ( optr->pstate == PRCURR )
    {
        /* no switch needed if current prio. higher than next */
        /* or if rescheduling is disabled ( pcxflag == 0 ) */
        if ( sys_pcxget() == 0 || lastkey(rdytail) < optr->pprio
            || (lastkey(rdytail) == optr->pprio) && (preempt > 0) )
            return;
        /* force context switch */
        optr->pstate = PRREADY;
        insert(currpid,rdyhead,optr->pprio);
    } /* if */
    else if ( sys_pcxget() == 0 )
    {
        kprintf("pid=%d state=%d name=%s",
               currpid,optr->pstate,optr->pname);
        panic("Reschedule impossible in this state");
    } /* else if */

    /* remove highest priority process at end of ready list */

    nptr = &proctab[ (currpid = getlast(rdytail)) ];
    nptr->pstate = PRCURR;           /* mark it currently running */
    preempt = QUANTUM;              /* reset preemption counter */
    ctxsw(&optr->pregs,&nptr->pregs);

    /* The OLD process returns here when resumed. */
    return;
}

```

123

```

; ctxsw.asm - _ctxsw

        include dos.asm

        dseg
; null data segment
        endds

        pseg

        public _ctxsw

;-----
; _ctxsw -- context switch
;-----
; void ctxsw(opp,npp)
; char *opp, *npp;
;-----
; Stack contents upon entry to ctxsw:
;     SP+4 => address of new context stack save area
;     SP+2 => address of old context stack save area
;     SP    => return address
; The addresses of the old and new context stack save areas are
; relative to the DS segment register, which must be set properly
; to access the save/restore locations.
;
; The saved state consists of the current BP, SI and DI registers,
; and the FLAGS register
;-----

_ctxsw proc    near
        push   bp
        mov    bp,sp           ; frame pointer
        pushf
        cli               ; disable interrupts just to be sure
        push   si
        push   di           ; preserve registers
        mov    bx,[bp+4]      ; old stack save address
        mov    [bx],sp
        mov    bx,[bp+6]      ; new stack save address
        mov    sp,[bx]
        pop    di
        pop    si
        popf              ; restore interrupt state
        pop    bp
        ret
_ctxsw endp

        endps

        end

```