

גרעין מערכת הפעלה

גרעין מערכת הפעלה היא רכיב תוכנה (מבנה נתונים + פקודות) המשמש את אותו רובד השולט בחומרה של המחשב בצוරה צו, שהוא מגדירה את המכונה הווירטואלית המסתיר את החומרה מהמשתמש והאפליקציה. בדרך כלל הגרעין הוא קוד בינארי הנטען לזכרון ומקבל שליטה עם עליית מערכת ונשאר כאן, לצורך קצת להתחאמז בשביב להבוחן בו.

באופן ציורי נהוג לחלק את הגרעין לשידרה של קטגוריות למשל:

- ניהול התהליכים.
- ניהול הפסיכות.
- ניהול הזכרון.
- ניהול הדיסקים.
- ניהול הקלט / פלט.

הרכיב הראשון והעיקרי שנראה ב-UNIX יהיה ניהול התהליכים.

מנהל התהליכים

תפקידיו העיקריים של ניהול התהליכים הם ליציג את התהליכים במערכת, לתמוך ביצירה ושחרור של המשאב שקרויה תהליך, לחמון בהפעלה ועיצוב של תהליכיים, למש את מנגנון החלפת התהליכים, ורכיב אופציונלי של תמייה בשירותי תאום בין תהליכיים.

כמו כל רכיב תוכנה, ניהול התהליכים הוא שילוב של מבנה נתונים ופעולות עליהם.

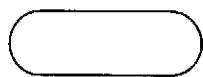
החלק הראשון שנראה מנגנון תורי התהליכים.

תורי התהליכים

ב-UNIX ישנים מצלבים שבהם תהליכיים ממתחנים יחסית לתהליכיים אחרים. לפיכך יש צורך במבנה נתונים לימוש תורים שבו מיוצגים התהליכים. תהליך שהיצוג שלו נמצא בתור נמצא באיזה תור בכל מקרה נמצא נמצא במהלך המבחן. סוג המבחן הם:

תרשים מיקומו של הגרעין במערכת הפעלה

תהליכי ישומים



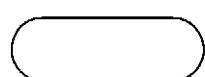
תהליכי מערכת



קריאות מערכת



קריאות מערכת



- המתנה להתקפות ה-CPU.
- המתנה לסםפור.
- הרדמה לזמן קצר.

לצורך תמייה בהמתנות המערכת צריכה לתמוך בשלושה סוגי תורים:

- לצורך המתנה ל-CPU - תור עדפיות.
- לצורך המתנה לסםפור – תור FIFO.
- לצורך הרדמה – תור הפרשיים.

מימוש

ניתן למש את כל התורים במבנה נתוני אחד. זה לא לחוטין הכרחי אבל בMOVED זה או אחר זה נוח שיש מבנה נתוני אחד לכולם.

כל התורים ימושו בראשימה מקושרת זו כיוונית.
יהיה לנו מערך אחד, []q, אשר מבנה הצמתים שלו מוגדר ב-h.q והקצתה שלו, כמו כמעט כל המשתנים הגלובליים, ב-c.initial שימש את הצמתים של הרשימות הקשורות. כל צומת מכיל שדות ליצוג העוקב, קודם, ופתח. האינדקס של כל רשומה במערך ישמש לזיהוי הצומת. המפתח ישמש למילון הרשומות בתור העדיפויות, וגם יאכسن את גודל הרדמה בתור הפרשיים של תור הרדומים.

גודל המערך []q הוא מספר התהליכיים במערכת (NPROC) + פעמים מספר הסטטוסים (NSEM) + NPROC : 4. הנקודות הראשוניות מייצגים את התהליכיים, והיתר הם 2 צמתים לכל תור אפשרי: צומת התחלת (HEAD) וצומת סיום (TAIL). הם תמיד יהיו צמתים עוקבים במערך. בסה"כ מספר התורים 2 + NSEM: תור לכל סםפור וכן שני התורים: תור ה-ready, התהליכיים המתיינים לתורם לקבל שירות CPU, ותור הרדומים. בתור לא ריק יהיו רשומות של נציגי תהליכיים בראשימה המקושרת בין ה-HEAD ל-TAIL. בתור ריק ה-HEAD וה-TAIL יצבעו אחד לשני. ערכי המפתחות של צומתי ה-HEAD תמיד 32768-, של צומתי ה-TAIL תמיד +32767, הערכים הקיצוניים של מספר שלם 16 ביט, ואלה אינם משתנים.

השימוש במבנה זה אפשרי משום שלعالם לא יהיה צורך ליצג תחילה בשני תורים – כל תחילה נמצא רק בסטטוס אחד בכל רגע.

השימוש בסוגי התורים יהיה ע"י הרוטינות הבאות:

טור עדיפיות:

הכנסה ממונת לפי מפתח ע"י הפונקציה `insert`,
הוצאה ע"י הפונקציה `getlast` - הוצאת האחרון (על יד ה-TAIL) מהטור
והחזרת המזהה שלו.
בתור עדיפיות התהליכים מסודרים בסדר עולה מה-HEAD ל-TAIL.

טור FIFO:

הכנסה לסופ הטור (על יד ה-TAIL) ע"י הפונקציה `enqueue`
הוצאת הראשון (ליד ה-HEAD) ע"י הפונקציה `getfirst`

טור הרדומים

לא נדון בתור זהה בשלב זה.

QCL PROCESSES AND

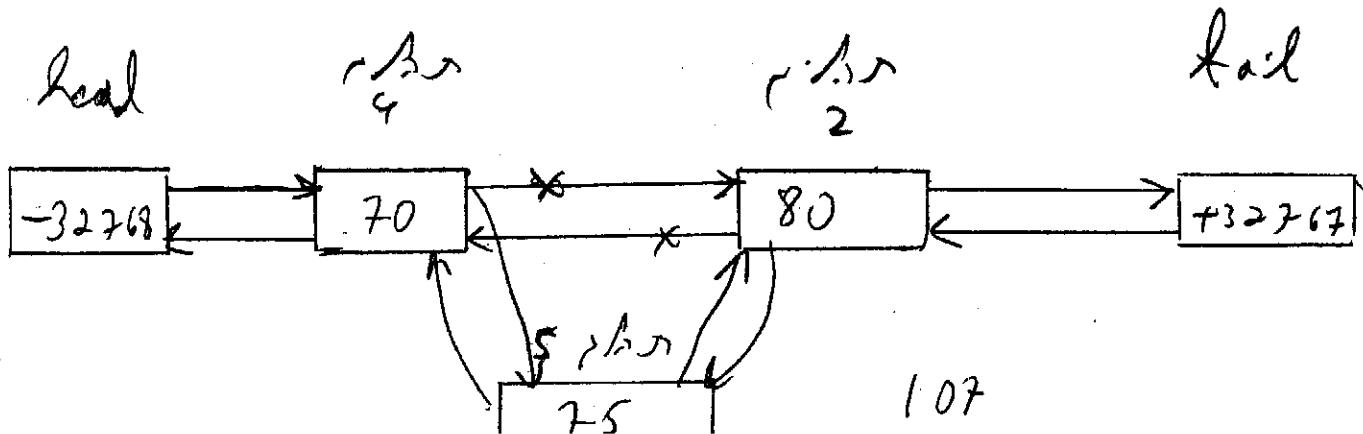
Qkey

Next

Qprev

0				NPROC 2(NESTED +2)
1				
2				
3				
4				
5				2(NESTED +2)
6				
7				2(NESTED +2)
8				
9				2(NESTED +2)
10				
11				2(NESTED +2)
12				
13				2(NESTED +2)
14				
15				2(NESTED +2)
16				
17				2(NESTED +2)
18				
19				2(NESTED +2)
20				
21				2(NESTED +2)
22				
23				2(NESTED +2)
24				
25				2(NESTED +2)
26				
27				2(NESTED +2)
28				
29				2(NESTED +2)
30				
31				2(NESTED +2)
32				
33				2(NESTED +2)
34				
35				2(NESTED +2)
36				
37				2(NESTED +2)
38				
39				2(NESTED +2)
40				
41				2(NESTED +2)
42				
43				2(NESTED +2)
44				
45				2(NESTED +2)
46				
47				2(NESTED +2)
48				
49				2(NESTED +2)
50				
51				2(NESTED +2)
52				
53				2(NESTED +2)
54				
55				2(NESTED +2)
56				
57				2(NESTED +2)
58				
59				2(NESTED +2)
60				
61				2(NESTED +2)
62				
63				2(NESTED +2)
64				
65				2(NESTED +2)
66				
67				2(NESTED +2)
68				
69				2(NESTED +2)
70				
71				2(NESTED +2)
72				
73				2(NESTED +2)
74				
75				2(NESTED +2)
76				
77				2(NESTED +2)
78				
79				2(NESTED +2)
80				
81				2(NESTED +2)
82				
83				2(NESTED +2)
84				
85				2(NESTED +2)
86				
87				2(NESTED +2)
88				
89				2(NESTED +2)
90				
91				2(NESTED +2)
92				
93				2(NESTED +2)
94				
95				2(NESTED +2)
96				
97				2(NESTED +2)
98				
99				2(NESTED +2)
100				

<i>key</i>	<i>next</i>	<i>prev</i>	
0			10013
1			11101
2	80	tail	X 15
3			
4	70	2 5	head
5	75	2	4
6			
head	-32768	4	-
tail	+32767	-	2

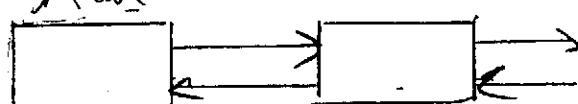


प्राइम एडेट मूलता वित्त

दोस्त

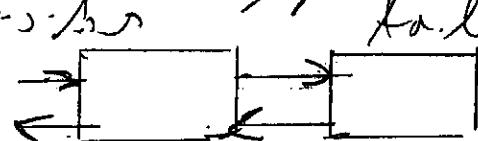
insert

head



get first

push



दोस्त

10238

108

endane

FIFO

108

```

/* q.h - firstid, firstkey, isempty, lastkey, nonempty */

/* q structure declarations, constants, and inline procedures */

#define NQENT           NPROC + NSEM + NSEM + 4 /* for ready & sleep */

struct qent {           /* one for each process plus two for */
    /* each list */
    int   qkey;          /* key on which the queue is ordered */
    int   qnext;          /* pointer to next process or tail */
    int   qprev;          /* pointer to previous process or head */
};

extern struct qent q[];
extern int      nextqueue;

/* inline list manipulation procedures */

#define isempty(list)   (q[(list)].qnext >= NPROC)
#define nonempty(list)  (q[(list)].qnext < NPROC)
#define firstkey(list)  (q[q[(list)].qnext].qkey)
#define lastkey(tail)   (q[q[(tail)].qprev].qkey)
#define firstid(list)   (q[(list)].qnext)

#define EMPTY     -1           /* equivalent of null pointer */

```

```
/* insert.c - insert */

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

/*
 * insert -- insert a process into a q list in key order
 */
int insert(proc, head, key)
int proc;                      /* process to insert          */
int head;                      /* q index of head of list   */
int key;                       /* key to use for this process */
{
    int      next;              /* runs through list         */
    int      prev;

    next = q[head].qnext;
    while (q[next].qkey < key)      /* tail has MAXINT as key   */
        next = q[next].qnext;
    q[proc].qnext = next;
    q[proc].qprev = prev = q[next].qprev;
    q[proc].qkey  = key;
    q[prev].qnext = proc;
    q[next].qprev = proc;
    return(OK);
}
```

```
/* getitem.c - getfirst, getLast */

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

/*
 *  getfirst -- remove and return the first process on a list
 */
int      getfirst(head)
        int      head;                      /* q index of head of list */
{
        int      proc;                     /* first process on the list */

        if ((proc=q[head].qnext) < NPROC)
            return( dequeue(proc) );
        else
            return(EMPTY);
}

/*
 *  getLast -- remove and return the last process from a list
 */
int      getLast(tail)
        int      tail;                      /* q index of tail of list */
{
        int      proc;                     /* last process on the list */

        if ((proc=q[tail].qprev) < NPROC)
            return( dequeue(proc) );
        else
            return(EMPTY);
}
```

```

/* queue.c - dequeue, enqueue */

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

/*
 * enqueue -- insert an item at the tail of a list
 */
int enqueue(item, tail)
int item;                                /* item to enqueue on a list */
int tail;                                 /* index in q of list tail */
{
    struct qent    *tptr;                  /* points to tail entry */
    struct qent    *mptr;                  /* points to item entry */

    tptr = &q[tail];
    mptr = &q[item];
    mptr->qnext = tail;
    mptr->qprev = tptr->qprev;
    q[tptr->qprev].qnext = item;
    tptr->qprev = item;
    return(item);
}

/*
 * dequeue -- remove an item from a list and return it
 */
int dequeue(item)
int item;
{
    struct qent    *mptr;                /* pointer to q entry for item */

    mptr = &q[item];
    q[mptr->qprev].qnext = mptr->qnext;
    q[mptr->qnext].qprev = mptr->qprev;
    return(item);
}

```

```
/* newqueue.c - newqueue */

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

/*
 * newqueue -- initialize a new list in the q structure
 */
int newqueue()
{
    struct qent *hptr;           /* address of new list head */
    struct qent *tptr;           /* address of new list tail */
    int hindex, tindex;          /* head and tail indexes */

    hptr = &q[ hindex=nextqueue++ ]; /* nextqueue is global variable */
    tptr = &q[ tindex=nextqueue++ ]; /* giving next used q pos. */
    hptr->qnext = tindex;
    hptr->qprev = EMPTY;
    hptr->qkey = MININT;
    tptr->qnext = EMPTY;
    tptr->qprev = hindex;
    tptr->qkey = MAXINT;
    return(hindex);
}
```