מנהל הזיכרון

מנהל הזיכרון הוא אחד הרכיבים החשובים בגרעין. במערכת הפעלה רצינית, מנהל הזיכרון יממש את המשימות הבאות:

1. מעקב אחרי שטחי הזיכרון הפנויים, הקצאת שטחי זיכרון לבקשת תוכנת מערכת ההפעלה ותוכניות האפליקציה וקבלת שטחים חזרה מתוכנות אלה.
2. מעקב אחרי שטחי זיכרון שהוקצו לתהליכים, ושחרורם כאשר התהליך חדל להתקיים.
3. במערכות שקיימות בהם **מנגנוני הגנה**, ניהול הגנה והשליטה (**מנגנון המיעון הוירטואלי**) על שטחי זיכרון, כך ששום קוד לא יוכל לגשת או לשנות שטחי זיכרון שלא יועדו לו.
4. במערכות שבהם קיים **זיכרון וירטואלי**, מנגנון המאפשר הרחבה מרחב הכתובות ושימוש יעיל בזיכרון האלקטרוני, מימוש המנגנון וניהולו.

ב-xinu, אין מנגנוני הגנה או מנגנונים מתקדמים כלשהם, לכן 3. ו-4. לא קיימים שם, וגם 2. לוקה בחסר. מנהל הזיכרון של xinu מממש את 1. ובאשר ל-2. רק את שחרור מחסנית התהליך כאשר התהליך חדל להתקיים. זה האחרון מתבטא בשמירת הפרטים על המחסנית ב-create ושחרור השטח הזה ב-kill. ככלל, מנהל הזיכרון של xinu הוא קוד פשוט ומינימלי בכדי שישמש בסיס לתרגילים.

הגרסה של xinu שאנחנו עובדים איתה עובדת במודל small שהיא תוכנית אפליקציה שמערבת שני סגמנטים של עד 64k: סגמנט קוד וסגמנט מידע.

סגמנט הקוד יכול, בעיקרו של דבר, לשמש רק כקוד. חריג לכך הוא רק קוד שכתוב בשפת אסמבלי.
באפליקציות turboc במודל small סגמנט המידע מממשת גם את המידע הגלובלי, גם את המחסנית וגם את מאגר השטח הדינמי. בקוד של xinu, בזמן עלית מערכת בקובץ initiali.c מתבצע malloc אחד גדול שבו התוכנית שלוקחת כמעט את כל הזיכרון הפנוי ,למעט שטח קטן למען רוטינות פנימיות של turboc שקוראות ל-int 21h. מכאן ואילך xinu מנהל את המחסניות והשטחים הדינמיים בעצמו.

לפיכך סגמנט המידע ב-xinu מחולק לשלוש:

1. חלק גלובלי / סטטי, מתחילת הסגמנט עד לנקודה המוצבעת ע"י משתנה גלובלי בשם end.
2. השטח הדינמי של xinu, משמש למימוש המחסניות והשטחים המבוקשים באופן דינמי ע"י התהליכים, המנוהל על ידי xinu עצמו, מהנקודה המוצבעת ע"י end עד לנקודה המוצבעת ע"י משתנה גלובלי אחר בשם maxaddr.
3. השטח ש-xinu ויתר עליו, מהנקודה המוצבעת ע"י maxaddr עד סוף הסגמנט.

השטח המוקצה למחסניות והבקשות הדינמיות של תהליכים הוא אותו שטח, חלק 2, והשטח מסוים יכול, במהלך הזמן, לשמש לשני המטרות.

מבנה הנתונים

מבנה הנתונים היחיד שמנהל הזיכרון מתחזק, מלבד הרישום של מחסניות ב-proctab, הוא **רשימת הבלוקים הפנויים memlist**. זוהי רשימה מקושרת של בלוקים של זיכרון הפנויים להקצאה, וזהו פשוט מעקב על איזה שטחים ניתן להקצות לבקשות הקצאה. הרשימה הזו קצת שונה מרשימות מקושרות שאנחנו נוהגים בדרך כלל לממש. מלבד רשומה המשמשת כקישור לתחילת הרשימה הנמצאת בחלק הגלובלי של הסגמנט (חלק 1), כל הצמתים ברשימה הם **הבלוקים הפנויים עצמם**. כלומר הצמתים ברשימה הזו אינם בגודל קבוע, כפי שאנחנו רגילים לחשוב. לכל צומת / בלוק ברשימה יש חלק עליון קבוע בגודל 4 בתים המשמש למימוש הרשימה – פוינטר לעוקב וציון גודל הבלוק – ושאר הבלוק שלא ישמש לכלום עד שהבלוק יוקצה למטרה כלשהיא. הרשימה גם **ממוינת** לפי **כתובת ההתחלה** של הבלוקים, זאת בכדי שכאשר בלוק משתחרר ומצורף חזרה לרשימה, יהיה קל יחסית להבחין אם הבלוק הזה יכול להתמזג עם בלוק קיים ברשימה – זיכרון שקיים ממש לפניו או ממש אחריו- , או אפילו לגרום למיזוג של שתי בלוקים קיימים ברשימה להתמזג יחד איתו לבלוק אחד גדול שיכיל את שלושתם.

בזמן עלית מערכת המצב ההתחלתי של המבנה הוא שיש שטח גדול אחד שפניו להקצאה, והרשימה היא memlist שמצביע עליו.

מדיניות הקצאת זיכרון

מדיניות הקצאה של xinu הוא הפשוטה ביותר: כל בקשה לזיכרון ע"י תהליך כלשהוא נענה במידה והדבר אפשרי – קיים בלוק רציף לפחות אחד בגודל הזה

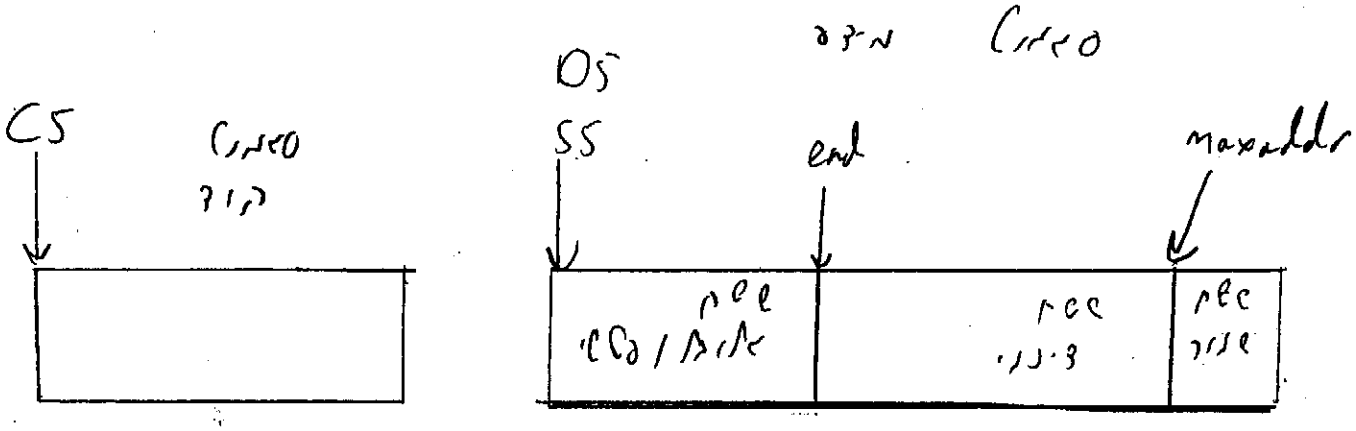
או יותר. במידה וניתן להיענות לבקשה, ההקצאה נעשית מהבלוק הראשון ברשימה שבו הדבר ניתן. המדיניות הזו נקראית first fit. מדיניות יותר מתוחכמת הוא best fit – הקצאה מהבלוק הקטן ביותר שהדבר אפשרי, אם כי מימוש המדיניות של best fit מחייבת סריקת כל הרשימה בכל מקרה.
מנהל הזיכרון אינו מקבל החלטות ניהוליות, הוא לא שואל האם זה בלתי רצוי לתת כמות כזו או אחרת של זיכרון לתהליך או משהוא מהסוג הזה.

הרוטינות העיקריות

הרוטינה getmem - פונקצית מערכת שהיא בקשה להקצאת זיכרון - מקבלת כפרמטר את גודל השטח הרצוי, מממשת את מדיניות הקצאת הזיכרון ומחזירה פוינטר לתחילת השטח שהוקצה במקרה של הצלחה (NULL במידה שהדבר בלתי אפשרי).

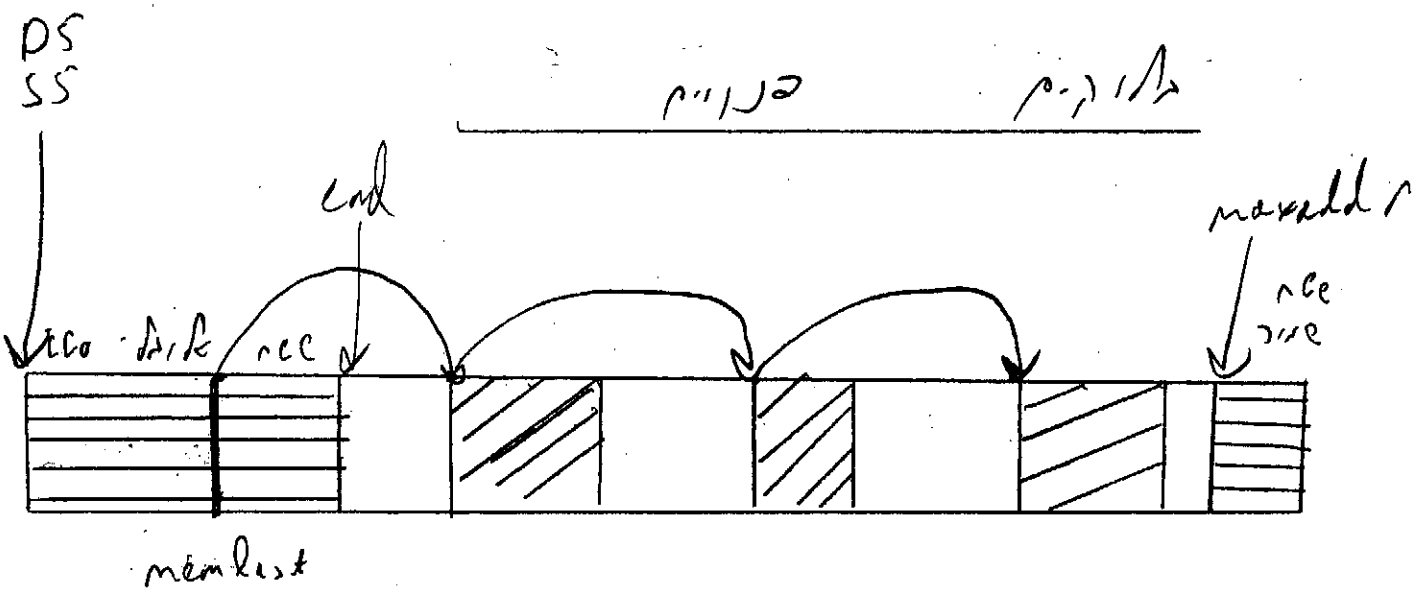
הרוטינה freemem – משמש לשחרור שטחי זיכרון – מקבלת את פרטי הבלוק המשתחרר – גודל ונקודת התחלה – ומחזירה אותה לרשימה תוך ניסיון עד כמה שאפשר למזג את הבלוק החדש עם בלוקים קיימים ברשימה.
עקרונית, השטח המשתחרר **אינו** חייב להיות בדיוק בלוק זיכרון שהוקצה בעבר (כפי שזה קורה ב-malloc ו-free הרגילים) או אפילו להיות הרישא של השטח כזה. יחד עם זאת, אני לא בטוח שהקוד לגמרי נכון למצבים כאילו. על מנת להבין את הקוד, אנחנו נצא מתוך הנחה שהמגבלה הזאת **כן קיימת**.
freemem אינו מממש איזה שהוא רעיון של הרשאות: הוא אינו בודק קשר כלשהוא בין התהליך המשחרר את השטח לזה שביקש אותו. הוא כן בודק אם השטח המשתחרר נמצא כולו בשטח הדינמי של xinu (בין end ל-maxaddr) וכן ששום חלק מהבלוק המשתחרר אינו חופף לשטחים שנמצאים ב-memlist.
בכל מקרה מהסוגים האלו גורם מיידית לחזרה מיידית עם ערך SYSERR.

מיפוי בזיכרון של UNIX



כשיש מקושרת של

בלוקים פנויים



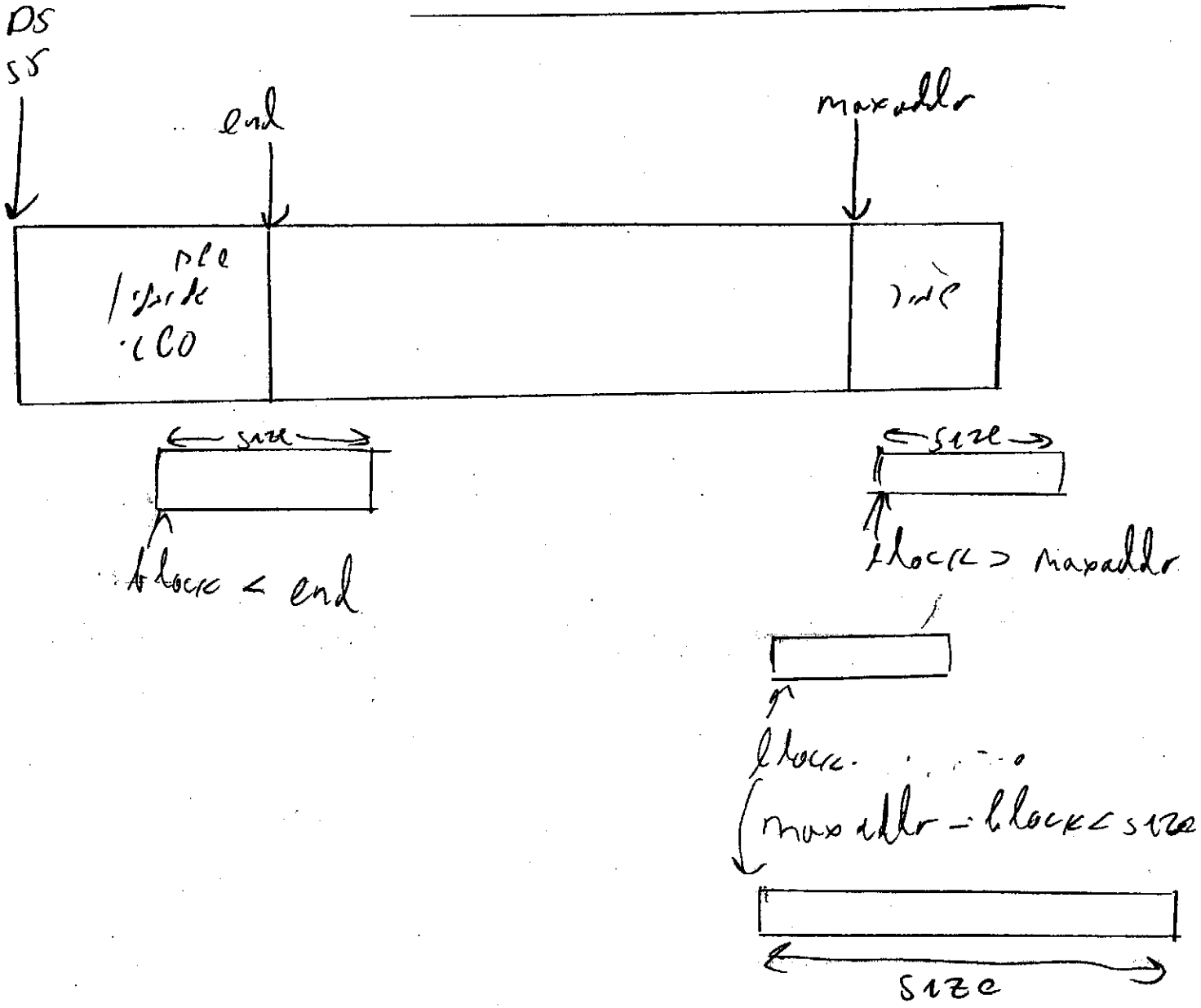
```
/* mem.h - roundew, truncew, getstk, freestk */

/*------------------------------------------------------------------------
 * roundew, truncew - round or truncate address to next even word
 *------------------------------------------------------------------------
 */
#define roundew(x)      ( (3 + (WORD)(x)) & (~3) )
#define truncew(x)      ( ((WORD)(x)) & (~3) )

#define getstk(n)       getmem(n)
#define freestk(b,s)    freemem(b,s)

typedef struct  mblock  {
        struct  mblock  *mnext;
        word    mlen;
} MBLOCK;

#define end     endaddr                 /* avoid C library conflict     */

extern  MBLOCK  memlist;                /* head of free memory list     */
extern  char    *maxaddr;               /* max memory address           */
extern  char    *end;                   /* address beyond loaded memory */

#define MMAX    65024                   /* maximum memory size          */
#define MBLK    512                     /* block size for global alloc  */
#define MMIN    8192                    /* minimum Xinu allocation      */
#define MDOS    1024                    /* save something for MS-DOS    */

extern  char    *getmem();
extern  int     freemem();
```

```
/* getmem.c - getmem */

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

/*------------------------------------------------------------------------
 *  getmem  --  allocate heap storage, returning lowest integer address
 *------------------------------------------------------------------------
 */
char *getmem(nbytes)
word nbytes;
{
        int     ps;
        struct  mblock  *p, *q, *leftover;

        disable(ps);
        if ( nbytes==0 ) {
                restore(ps);
                return( NULL );
        }
        nbytes = roundew(nbytes);
        for ( q=&memlist, p=q->mnext ;
                (char *)p != NULL ;
                q=p, p=p->mnext )
                if ( p->mlen == nbytes) {
                        q->mnext = p->mnext;
                        restore(ps);
                        return( (char *) p );
                } else if ( p->mlen > nbytes ) {
                        leftover = (struct mblock *)( (char *)p + nbytes );
                        q->mnext = leftover;
                        leftover->mnext = p->mnext;
                        leftover->mlen = p->mlen - nbytes;
                        restore(ps);
                        return( (char *) p );
                }
        restore(ps);
        return( NULL );
}
```

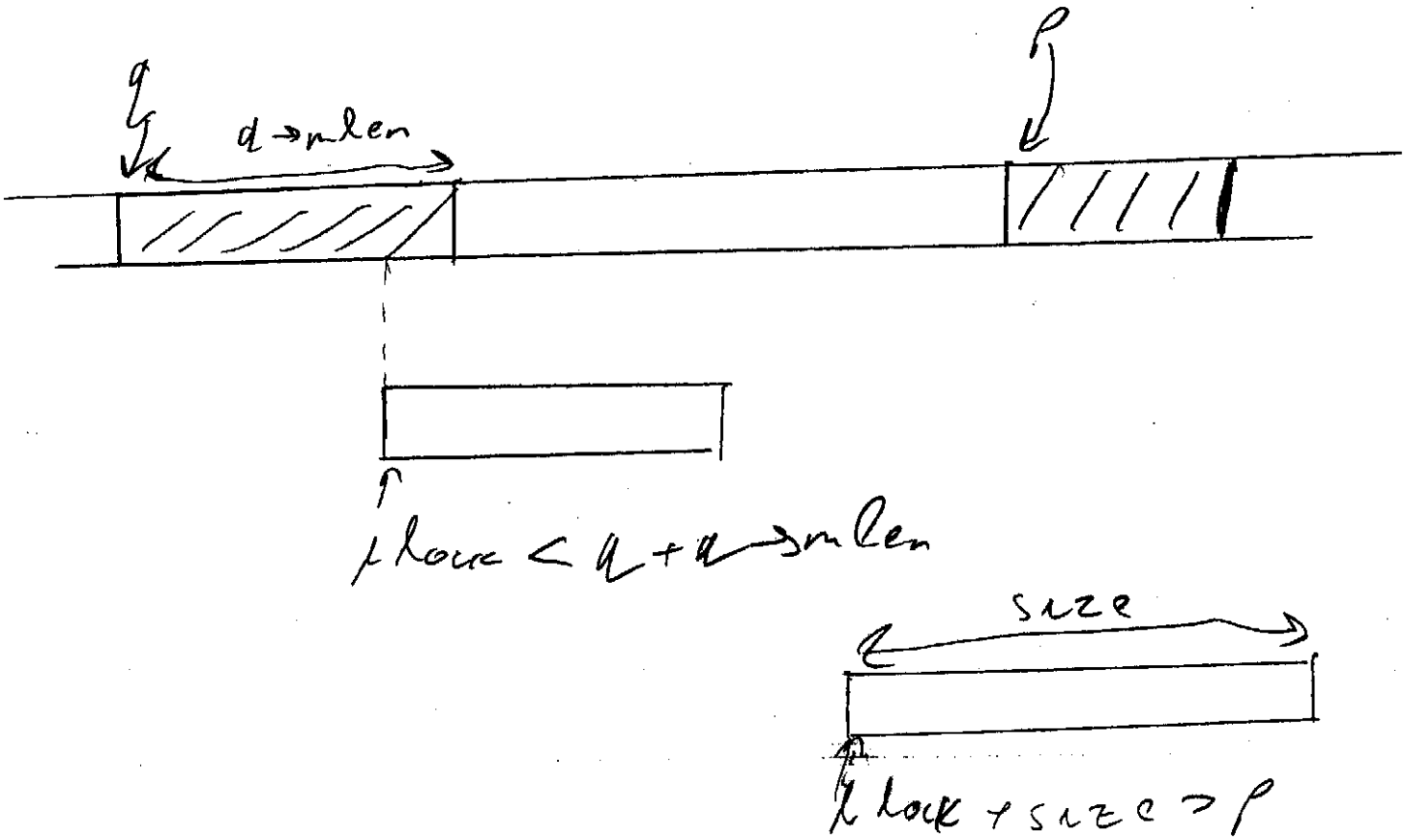
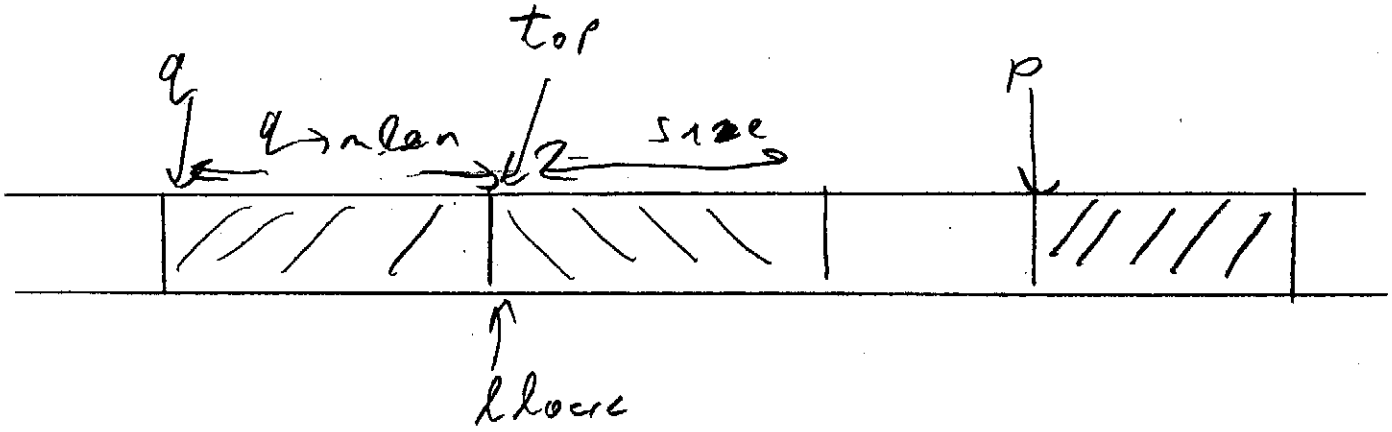
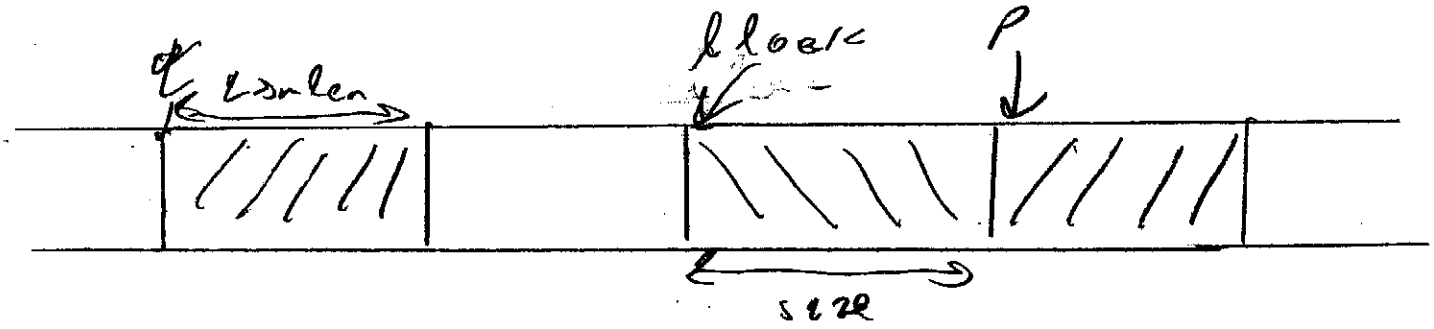
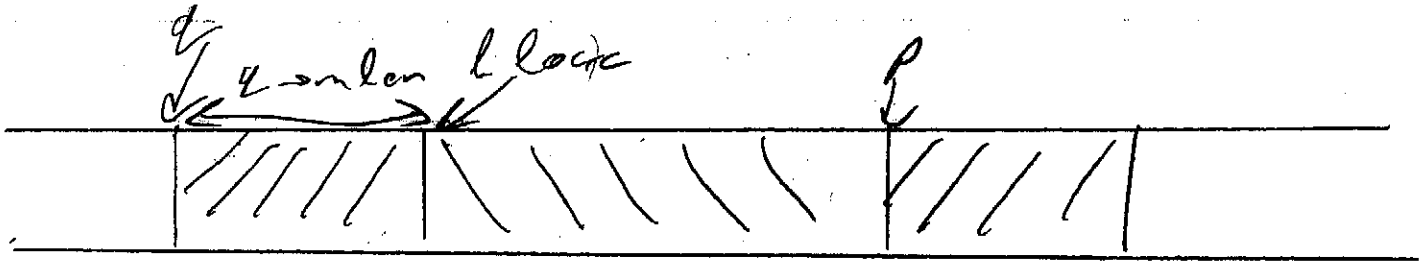
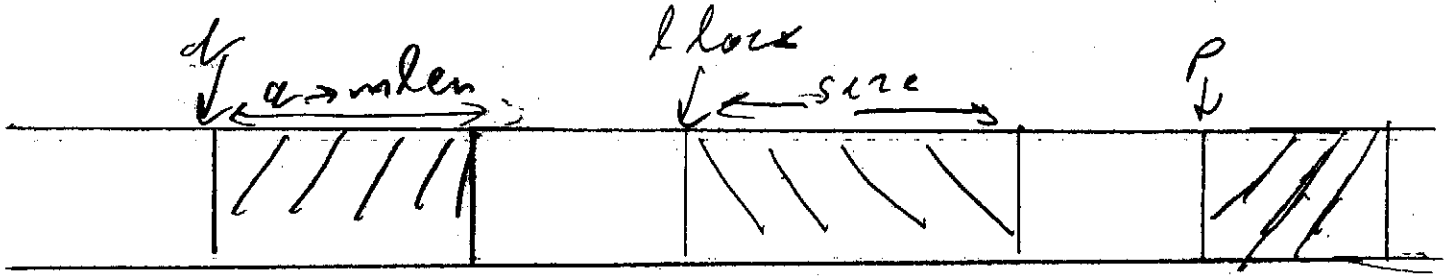
```
/* freemem.c - freemem */

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

/*------------------------------------------------------------------------
 *  freemem  --  free a memory block, returning it to memlist
 *------------------------------------------------------------------------
 */
SYSCALL freemem(block, size)
char *block;
word size;
{
        int     ps;
        struct  mblock  *p, *q;
        char    *top;

        size = roundew(size);
        block = (char *) truncew( (word)block );
        if ( size==0 || block > maxaddr || (maxaddr-block) < size ||
                block < end )
                return(SYSERR);
        disable(ps);
        for( q = &memlist, p=memlist.mnext ;
                (char *)p != NULL && (char *)p < block ;
                q=p, p=p->mnext )
                        ;
        if ( q != &memlist && (top=(char *)q+q->mlen) > block
                 || (char *)p != NULL && (block+size) > (char *)p ) {
                restore(ps);
                return(SYSERR);
        }
        if ( q != &memlist && top == block )
                q->mlen += size;
        else {
                ((struct mblock *)block)->mlen = size;
                ((struct mblock *)block)->mnext = p;
                q->mnext = (struct mblock *)block;
                (char *)q = block;
        }
        if ( ((char *)q + q->mlen) == (char *)p ) {
                q->mlen += p->mlen;
                q->mnext = p->mnext;
        }
        restore(ps);
        return(OK);
}
```

Foreman

I תכניות



II בדיקות בקונות



אפשרויות שלוח block