☐☐☐☐☐☐☐☐☐

מספר התלמיד הנבחן

**רשום את כל תשע הספרות**

**הדבק כאן את מדבקת הנבחן**

N100533067 מס' סידורי

I.D ת"ז:

## האוניברסיטה הפתוחה

**ו' באלול תשע"ד**

1

# מס' שאלון - 492

בספטמבר 2014

מס' מועד 94

**סמסטר 2014ב**

**20594 / 4**

## שאלון בחינת גמר

# 20594 - מערכות הפעלה

**משך בחינה: 3 שעות**

בשאלון זה 9 עמודים

**מבנה הבחינה:**

קראו בעיון לפני שתתחילו בפתרון הבחינה!

א. המבחן מורכב משלושה חלקים.

ב. בחלקים א ו - ב מופיעות שאלות פתוחות. ענו תשובות מלאות, בכתב קריא ובקיצור נמרץ. אין חובה להשתמש בכל השורות המוקצות לצורך התשובות, אך אין לחרוג מהמקום המוקצה.

ג. בחלק ג ( שאלות אמריקאיות ) עליכם לבחור בכל פעם בתשובה יחידה מבין התשובות המוצעות ולהקיף בעיגול את אות התשובה שבחרתם.

**חומר עזר:**

**כל חומר עזר אסור בשימוש, פרט למחשבון.**

**בהצלחה !!!**

**החזירו**
**למשגיח את השאלון**
**וכל עזר אחר שקיבלתם בתוך מחברת התשובות**

## חלק א (55 נקודות)

ענו על **שלוש** השאלות הבאות.

### שאלה 1 (35 נקודות)

(10 נק') א. עליכם לממש רשימה מקושרת חד-כיוונית ציקלית (מעגלית). הרשימה תהיה בעלת איבר סרק (dummy node) יחיד, שמוצבע מראש הרשימה. בצורה זאת אין צורך לבדוק את מקרה הקצה של הוצאת איבר אחרון מהרשימה או הוספת איבר ראשון – ברשימה תמיד יש איבר אחד.

השלימו את הקוד הבא להכנסת איבר לראש הרשימה. עליכם להשלים גם את מבני הנתונים המגדירים את הרשימה ואיבר הרשימה. ניתן להשתמש במנעול אחד בלבד. קוד לא יעיל (מבחינת מספר המשתנים או הביצועים) יקבל ניקוד חלקי בלבד.

```
typedef struct Node_t {
        void* data;
        Node *next;
        ________________
        ________________
        ________________
        ________________
} Node;

    typedef struct List_t {
        Node head;  // dummy empty node. Always first
        Lock lock;
        ________________
        ________________

} List;

void  insert(List list, void* data){
        Node* node = (Node*)malloc(sizeof(Node));   // assuming malloc allways suceeds
        node->data=data;
        lock(lock);
        node->next = head->next;          ✓
        head->next = node;
        unlock(lock);
}
```

(15 נק׳) ב. נסתכל על מימוש של פונקציית remove מרשימה מקושרת דו-כיוונית ציקלית (מעגלית). הרשימה נועדה לשפר את הגישה המקבילית של מספר תהליכונים לרשימה, ולכן משתמשת במנעולים ברמה של איברי הרשימה.

```
typedef struct Node_t {
        void* data;
        Node *next, *prev;
        Lock lock;
} Node;

typedef struct List_t {
        Node head, tail; // dummy empty nodes
} List;

InitList(){
        head->prev=head->next=tail;
        tail->prev=tail->next= head;
}

void  remove(List* list, Node* node)
{
```

should be a lock here ? ->

```
        Node* prev;
        if (node==NULL || node->prev==NULL || node->next==NULL)
return;

        lock(node->prev->lock);
        lock(node->lock);
        lock(node->next->lock);
        node->prev->next = node ->next;
        node->next ->prev = node ->prev;
        prev=node->prev;
        node->next=NULL;
        node->prev=NULL;
        unlock(prev ->next->lock);
        unlock(node->lock);
        unlock(prev->lock);
}
```

במימוש זה קיימת בעיה. מצאו את הבעיה והסבירו מדוע היא יכולה להיגרם ובאילו תנאים.

The node being removed needs to be locked at the begining of the function before checking for NULL. Currently there is a race condition if two processes try to remove the same node. If they both pass the NULL check the firsts to aquire a lock will remove the node, then the second will continue and also try to remove the node and cause a NULL pointer to be ~~dafre~~ dereferenced, leading to a crash.

✓

(10 נק')   ג.   כעת נחליף את פונקציית ה- remove מסעיף ב בפונקציה הבאה :

```
void  remove(List* list, Node* node)
{
        Node* prev;
        if (node==NULL) return;
n       lock(node->lock);
        if (node==NULL || node->prev==NULL || node->next==NULL){
                unlock(node->lock);
                return;
        }
p       lock(node->prev->lock);
next    lock(node->next->lock);
        node->prev->next = node ->next;
        node->next ->prev = node ->prev;
        prev=node->prev;
        node->next=NULL;
        node->prev=NULL;
next    unlock(prev ->next->lock);
n       unlock(node->lock);
p       unlock(prev->lock);
}
```

these need to be switched

locks should always be realosed in reverse order from which they are aquired.

X (-8)

2 processes + Context Switch ~> may cause to deadlock

possibility

גיתכנות

הוכיחו או הפריכו את ייתכנות ה deadlock בעקבות השימוש בפונקציה זו.

Multiple calls to this function will never cause a deadlock because the first to aquire node->lock will complete removal of the node and all following calls will exit/return at the NULL check.

**שאלה 2 (20 נקודות)**

במערכת הפעלה כלשהי קיים מנגנון לניהול זיכרון המקצה לכל **דף** מזהה מספרי i. מדיניות הדפדוף הממומשת על-ידי המנגנון לניהול הזיכרון היא המדיניות הבאה:

אם נרצה להביא לזיכרון דף מספר i, והזיכרון מלא, נפנה מהזיכרון דף j כך ש |j-i| הינו מקסימאלי. אם יש שני דפים כאלה, נבחר אחד באקראי.

בשאלה זו נניח שגודל הזיכרון הפיזי הוא 3 מסגרות (אלא אם כן מצוין אחרת).

(5 נק׳) א. עבור סדרת הגישות הבאה, רשמו את תוכן הזיכרון. כמה Page Faults מתבצעים במהלך הריצה? (המספרים מציינים את מזהי הדפים). הניחו שהזיכרון היה ריק בתחילת הריצה.

1 2 3 4 1 4 3 2 1 4 (משמאל לימין)

LRU

123
423
413
41
41
~~44~~
4
1
1

\ \ \ \ \ \ \ \

| t | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| frame 1 | 1 | 1 | 1 | 4 | 1 | 4 | 4 | 4 | 1 | 4 |
| frame 2 |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| frame 3 |  |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |

סה״כ: page faults 8

(5 נק׳) ב. תנו דוגמה לסדרת גישות שבה אלגוריתם LRU טוב יותר ממדיניותה של המערכת.

1 2 3 4 1 4 1 4 1 4 LRU Gives 5 page faults

the other gives 10

✓

(5 נק׳) ג. הוכיחו או הפריכו את הטענה הבאה: אלגוריתם LRU תמיד טוב יותר ממדיניותה של המערכת.

לא נכון, בדוגמה הנתונה שניהם צריכים 8 page faults.

✓

(5 נק׳) ד. השוו בין כמות המסגרות שיש לבדוק כדי למצוא דף לפינוי מהזיכרון, לפי האלגוריתם LRU ולפי מדיניותה של המערכת.

ב LRU מייד יודע איזה מסגרת להחליף (זו בסוף הרשימה)

במדיניות של מערכת צריך לבדוק כל המסגרות על כל page fault.

✓

you could keep the min and max pge #'s currently in memory to improve th
given algorithm, but it would still require more checks in the case min < i

## חלק ב (25 נקודות)

ענו על **חמש** השאלות הבאות. משקל כל שאלה 5 נקודות.

### שאלה 3

מהו hard-link? מה ההבדל בינו לבין soft-link?

hard link contains pointer to inode and increment the ref count
soft link contains the path to the file

### שאלה 4

מהו condition variable וכיצד משתמשים בו במבנה פיקוח (monitor)?

A condition variable allows another process to enter the monitor if the current ~~[illegible]~~ process needs to block and wait for an event. Once the other process has provided the event, it can use the condition variable to signal the blocked process to continue.

### שאלה 5

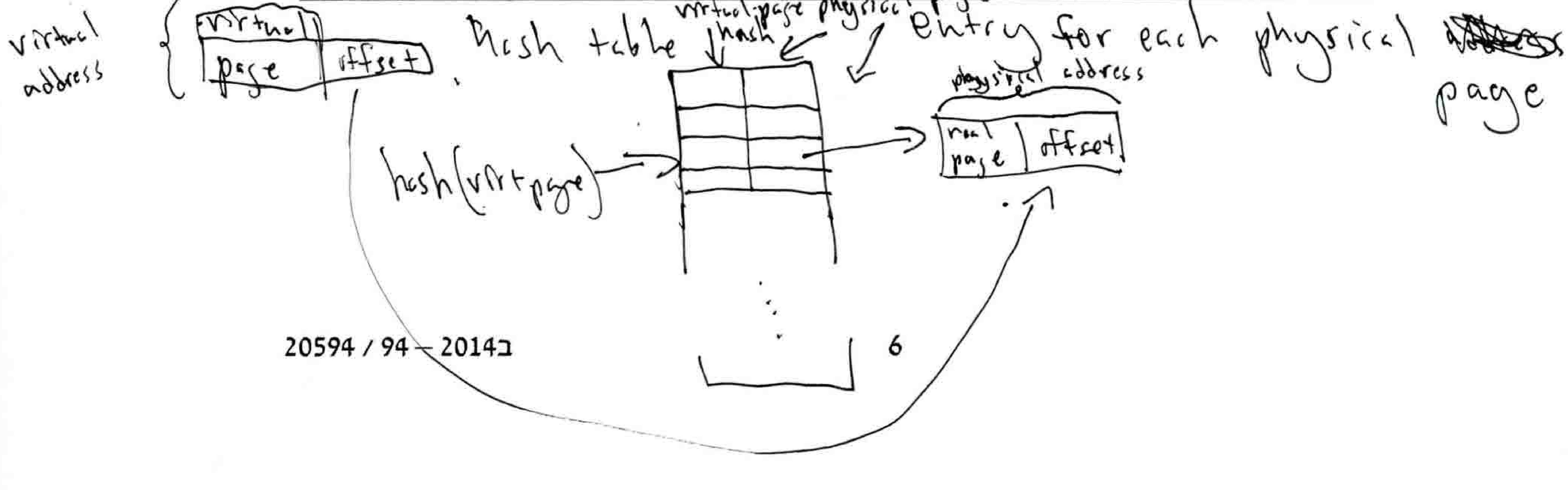
מהי TRAP instruction?

It causes control to switch to the kernal. It is used to execute system calls, for example.

### שאלה 6

מהו inverted page table? ציירו כיצד מתבצע תרגום כתובת לוגית לכתובת פיזית באמצעות inverted page table.

An inverted page table uses a hash function to map virtual address to physical address, and is used when the virtual address space is much ~~much~~ than the physical address space, such as in 64 bit.



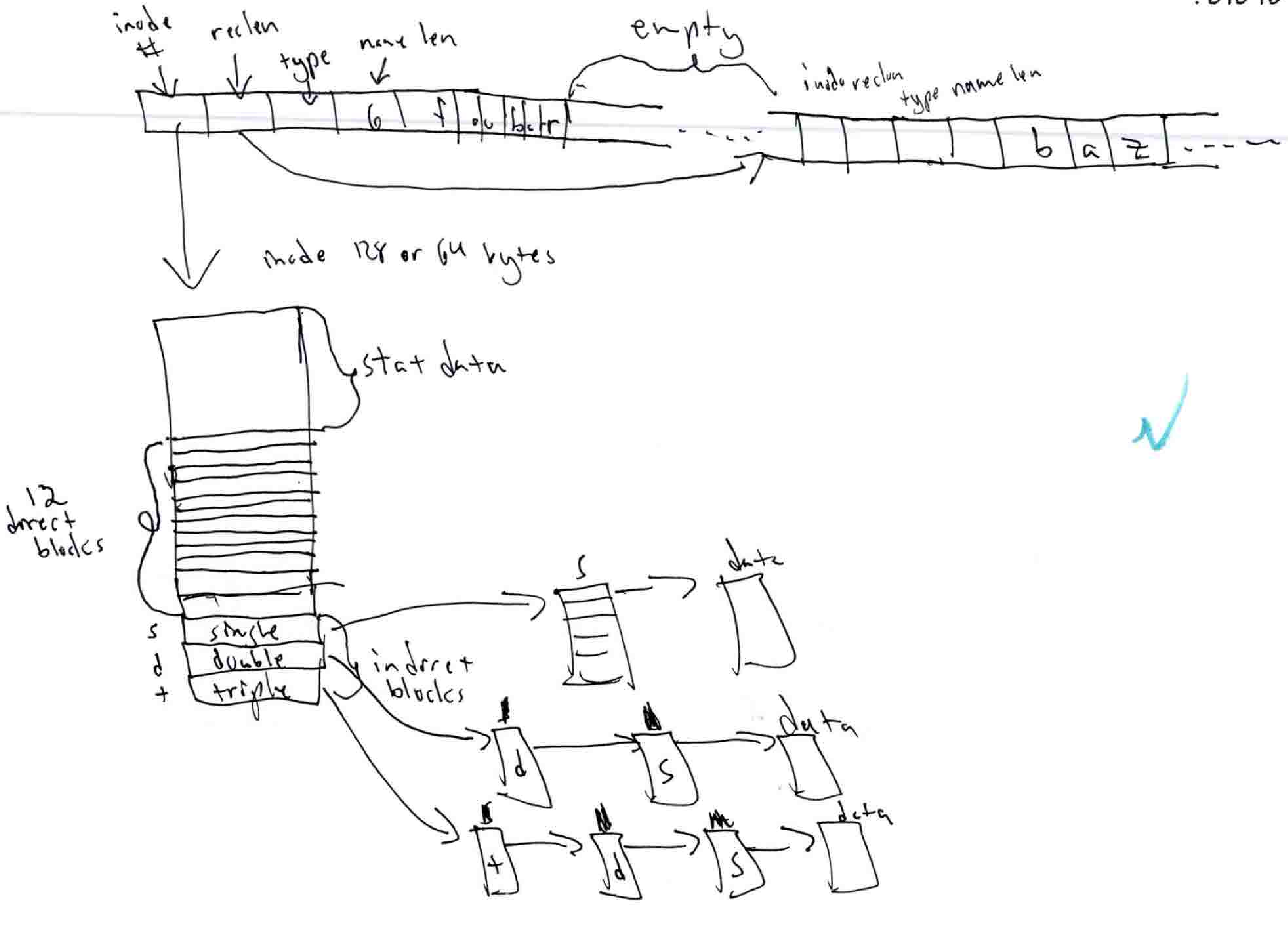
## שאלה 7

כיצד נראה מבנה של קובץ ספרייה במערכת קבצים ext2? שרטטו והסבירו.

4 שדות קבועות ושדה לשם בגודל משתנה. שדה

reclen: can be used to include empty space after the record
type: file type (dir, file, special)

שרטוט:



~~entries in the directory file~~

## חלק ג (20 נקודות)

ענו על **ארבע** שאלות רב-ברירה (אמריקאיות). משקל כל שאלה 5 נקודות.

בכל שאלה יש לבחור את התשובה הנכונה ולהקיף בעיגול את אות התשובה שבחרתם.

### שאלה 8

במערכת עם ניהול זיכרון באמצעות הדפדוף (paging) נתון כי :

- כתובת מדומה (virtual address) היא בת 64 bits
- גודל הדף הוא 64 Kbytes
- גודל המילה הוא 4 bytes

מהי כמות הדפים המקסימלית בזיכרון המדומה?

$64kB = 2^{16}$ bytes / page
assuming only words are addressable
$2^{14}$ words / page

א. 2^40
[✓ ב. 2^50]
ג. 2^54
ד. 2^60

✓

### שאלה 9

מהו מספר הכניסות (entries) בטבלת הדפים המהופכת (inverted page table)?

א. כמספר הדפים בזיכרון המדומה
[✓ ב. כמספר המסגרות (frames) בזיכרון הפיזי]
ג. כמספר הדפים המקסימלי שיכול לדרוש תהליך
ד. כמספר התהליכים המקסימלי שיכולים להתבצע בו-זמנית במערכת

✓

**המשך הבחינה בעמוד הבא**

**שאלה 10**

כדי לבחור את הדף המתאים לפינוי מהזיכרון הראשי הוצע האלגוריתם הבא:

לכל מסגרת (frame) בזיכרון מוצמד מונה. כאשר דף חדש מגיע למסגרת, ערכו של מונה זה נקבע להיות 0 ובכל פנייה לדף, המונה מועלה ביחידה אחת. כאשר יש צורך לפנות דף מהזיכרון, בוחרים תמיד את הדף שהמונה המוצמד למסגרת שלו מכיל את הערך הנמוך ביותר. האם אלגוריתם זה הוא טוב לניהול הדפדוף בזיכרון?

א. כן. המונה משקף את מידת נחיצותו של הדף. ככל שערכו נמוך, כך מספר הפניות לדף היה נמוך יותר ולכן הוא מתאים לפינוי.

ב. כן. מונה גבוה מצביע על דף הנמצא בשימוש מתמיד.

ג. לא. אלגוריתם זה אינו מאפשר כלל כניסה של דפים חדשים כי המונה שלהם תמיד 0 ולכן יפונו מיד.

[✓ ד.] לא. מצב המונה עלול לשקף מציאות היסטורית שאיננה קיימת עוד.

**שאלה 11**

אחד ההבדלים בין פונקציות ספרייה בשפת C (כגון ()printf) לבין קריאות מערכת (system calls) ב- Unix/Linux הוא:

א. רק למנהל המערכת (super user) מותר להשתמש בקריאות מערכת, ואילו בפונקציות ספרייה יכול להשתמש כל אחד.

ב. באמצעות פונקציות הספרייה בשפת C לא ניתן לפתוח קבצים לפעולות קריאה או כתיבה.

ג. כל קריאות המערכת הן חלק מפונקציות הספרייה של השפה.

ד. כל פונקציות הספרייה משתמשות בקריאות מערכת .

ה. התשובות ג׳ וד׳ הן הנכונות.

[✓ ו.] כל התשובות הקודמות אינן נכונות.

**בהצלחה!**