

פרק 3

ניהול זיכרון

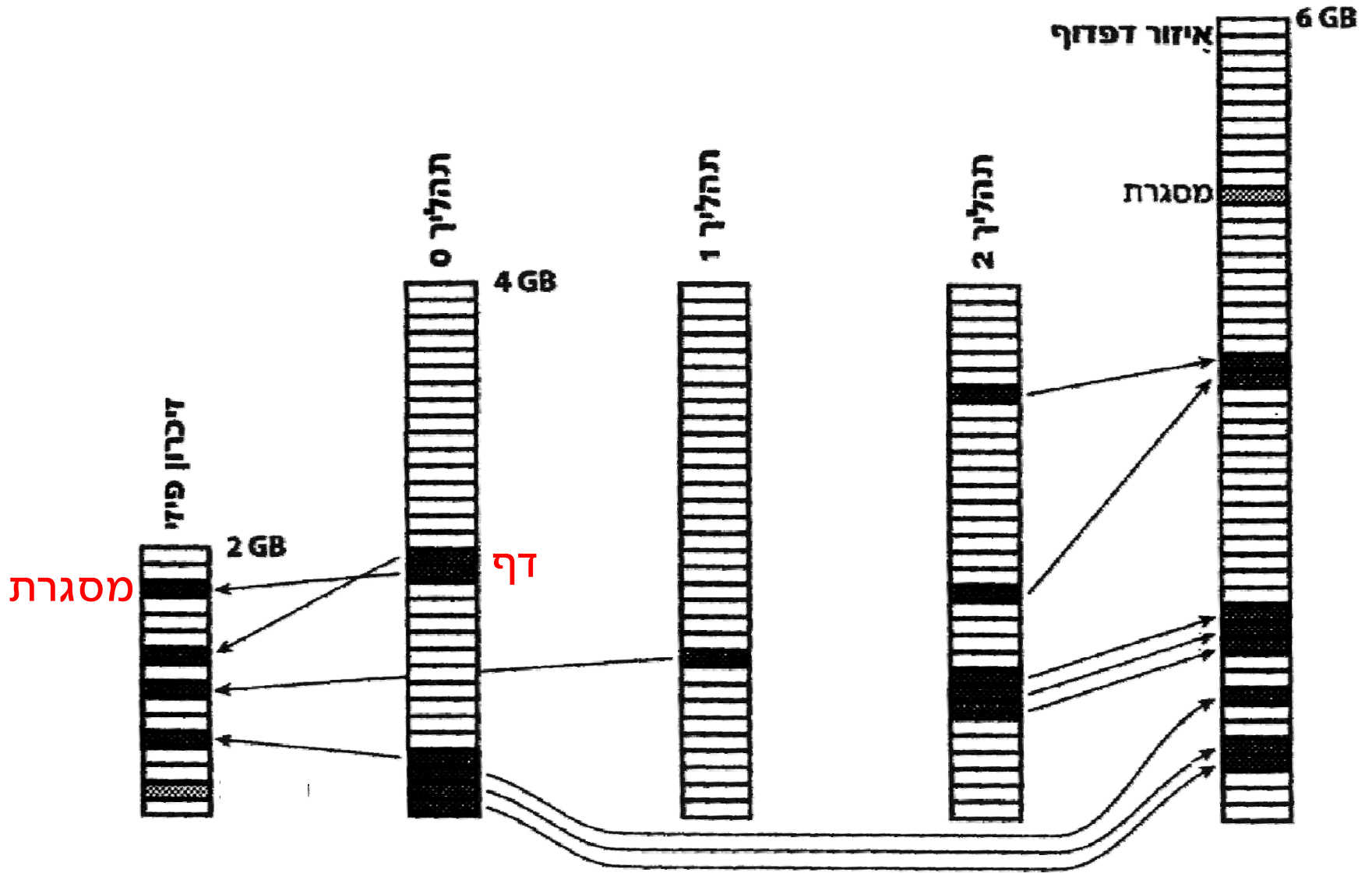
מטרות ניהול הזיכרון

- ❖ יכולת להריץ מספר תוכניות בו-זמנית תוך כדי הגנה על הזיכרון של כל תוכנית מפני האחרות
- ❖ יכולת להריץ בו-זמנית תוכניות שסך כל הזיכרון שהוקצה להן גדול מהזיכרון המותקן במחשב
- ❖ יכולת להשתמש בדיסקים כהרחבה זולה אך איטית של הזיכרון; המעבד אינו יכול להתייחס ישירות למידע בדיסק
- ❖ יכולת להזיז את מבני הנתונים של תוכנית במהלך ריצתה בלי שהתוכנית מודעת להזזות הללו; הזזות כאלו מאפשרות לנצל "חורים" קטנים בזיכרון, או לאחד חורים לזיכרון פנוי רצוף, או להעביר מבנה נתונים מהזיכרון לדיסק ומשם למקום אחר בזיכרון

הפרדה בין כתובות בתוכנית וכתובות על הפס

- ❖ תוכנית מציגה למעבד כתובות של תאי זיכרון שיש לקרוא או לכתוב מהם נתונים
- ❖ הכתובות מאוחסנות ברגיסטרים או בזיכרון
- ❖ מצביע התוכנית מציג למעבד כתובות שיש לקרוא מהן פקודות
- ❖ המעבד אינו מציג את הכתובות הללו על ערוץ הכתובות בפס, אלא מתרגם אותם לכתובות אחרות תוך שימוש במבנה נתונים שמערכת ההפעלה מקימה ומתחזקת
- ❖ כתובות וירטואליות בתוכנית, כתובות פיזיות על הפס

זיכרון וירטואלי



תרגום כתובות וירטואליות לפיזיות

- ❖ מערכת ההפעלה מקימה טבלת דפים, מבנה נתונים שממפה את הדפים של תהליך למסגרות פיזיות
- ❖ טבלת הדפים שמורה בזיכרון ומערכת ההפעלה מודיעה למעבד היכן היא בעזרת אוגר מיוחד
- ❖ המעבד מסוגל לחפש מיפוי של דף למסגרת בטבלה
- ❖ על מנת לחסוך את הצורך בחיפוש בכל גישה לזיכרון, המעבד שומר בהתקן חומרה מיוחד (חלק מהמעבד) בשם TLB מיפויים שנעשה בהם שימוש לאחרונה
- ❖ במעבדים מסוימים מערכת ההפעלה אחראית לביצוע חיפוש בטבלה והכנסת מיפויים ל-TLB; לא נפוץ

תרגום כתובות (המשך)

- ❖ בהינתן כתובת וירטואלית, המעבד מפרק אותה למספר דף (סיביות בכירות) ולהיסט בתוך הדף (סיביות זוטרות)
- ❖ לדוגמה, אם גודל דף 8192 בתים, אזי 13 סיביות מציינות את ההיסט והשאר את הדף
- ❖ המעבד ממפה את הדף למסגרת פיזית ומשרשר את ההיסט למספר המסגרת על מנת ליצור כתובת פיזית שניתן להציג על הפס. נתבונן בכתובת הוירטואלית

0000 0000 0000 0000 0110 0000 0000 1001

- ❖ הכתובת הוירטואלית מתייחסת לבית ה-9 בדף מספר 11; אם הדף הזה ממופה למסגרת מספר 255, הכתובת שתוצג על הפס היא

0000 0000 0000 1111 1110 0000 0000 1001

טבלאות דפים שטוחות

- ❖ מערך בגודל מספר הדפים במרחב זיכרון וירטואלי
- ❖ בכל איבר במערך מצוין מספר המסגרת ועוד מספר סיביות לתיאור מצבים שונים, כמו דף שאין מוקצה כלל לתהליך ([invalid]) או שהמסגרת שמכילה אותו אינה בזיכרון ([not]present)
- ❖ חיפוש פשוט, טבלה גדולה
- ❖ למשל, מרחב של 4 GB (מצביעים של 32 סיביות) עם דפים בגודל 8 KB ו-4 דורש טבלה עם 512 K איברים. אם גודל כל איבר 4 בתים הטבלה של כל תהליך צורכת 2 MB

טבלאות דפים היררכיות

- ❖ חיפוש מורכב יותר מאשר בטבלה שטוחה (ממומש בחומר!)
- ❖ יותר גישות לזיכרון בזמן חיפוש (אחת לכל רמה בעץ)
- ❖ חיסכון עצום בזיכרון עבור מרחבי זיכרון שרק חלק קטן מהם מוקצה

טבלאות דפים הפוכות

- ❖ טבלה אחת לכל התהליכים
- ❖ שומרת רק מיפויים של דפים בזיכרון; מיפויים לאזור הדפדוף שמורים במבנה נתונים אחר
- ❖ ממומשת כטבלת גיבוב (hash) שמפתח החיפוש שלה הוא מספר הדף (ואולי מספר התהליך)
- ❖ כניסה צריכה לציין גם מספר מסגרת וגם מספר תהליך
- ❖ אלגוריתם חיפוש מורכב יחסית
- ❖ גודל הטבלה תלוי רק בגודל הזיכרון הפיזי ואינו תלוי בגודל מרחבי הזיכרון הוירטואליים או במספר התהליכים

תחזוקת טבלאות הדפים וה-TLB

❖ מערכת ההפעלה מחליטה היכן לשכן דפים

❖ אי לכך, מערכת ההפעלה היא שמתחזקת את טבלאות הדפים

❖ בדרך כלל המעבד מחפש בטבלה ומכניס מיפויים ל-TLB

❖ שינוי כניסה בטבלת דפים דורש מחיקת המיפוי מה-TLB אם הוא

נמצא שם, בעזרת פקודת מכונה מיוחדת

❖ מערכת ההפעלה צריכה להודיע למעבד מה הכתובת של טבלת

הדפים של התהליך שרץ (באוגר מיוחד)

הגנה על זיכרון

- ❖ כל הפקודות שעוסקות ב-TLB ובכתובת של טבלת הדפים מותרות רק במצב מיוחס
- ❖ אם טבלת הדפים הפוכה, מערכת ההפעלה מודיעה למעבד מה המזהה של התהליך שרץ כרגע על מנת להשתמש רק במיפויים שלו
- ❖ אי לכך, המעבד משתמש רק במיפויים שמערכת ההפעלה יצרה עבור התהליך שרץ כרגע; תהליך לא יכול להתייחס לכתובות פיזיות כלל, ולא יכול להתייחס למסגרות שאינן ממופות לדפים שלו

הגנה על זיכרון: סיביות גישה

❖ כל כניסה בטבלת הדפים כוללת מספר סיביות שמתארות הרשאות גישה

▪ האם מותרת גישה לדף במצב משתמש או רק במצב מיוחס?

▪ האם מותרות קריאה, כתיבה, ושימוש בתוכן כפקודות מכונה?

❖ היכולת לאסור סוגי גישה מסוימים, יחד עם העובדה שהדפים הראשונים לעולם אינם ממופים, עוזרת לגלות שגיאות בתוכניות (התייחסות דרך מצביע null, שכתוב קוד, ...)

❖ היכולת לאסור גישה במצב משתמש:

▪ מאפשרת למערכת ההפעלה למפות את הקוד ומבני הנתונים שלה לכל התהליכים בלי לאפשר לתוכנית שהתהליך מריץ לגשת לזיכרון הזה

▪ מערכת ההפעלה ממופה בדרך כלל לכל התהליכים באותו מקום (למשל ב-GB העליון של כל מרחב זיכרון וירטואלי

▪ ממזער תקורה בעת קריאות מערכת ופסיקות

חריגי דף (page)

(faults/segmentation exception)

- ❖ המעבד מזהה גישות לדפים לא ממופים, לדפים שממופים למסגרת בדיסק, ולדפים שהרשאותיהם אינן מתירות את סוג הגישה או אינן מתירות גישה במצב משתמש
- ❖ המעבד מייצר חריג (exception) בשם חריג דף שגורם להפעלת שגרה של מערכת ההפעלה, בדומה לטיפול בפסיקות
- ❖ השגרה של מערכת ההפעלה מזהה את הסיבה לחריג ומגיבה
- ❖ אם צריך לקרוא מסגרת מהדיסק, מערכת ההפעלה משעה את התהליך, מביאה את המסגרת, מעדכנת את טבלת הדפים, וחוזרת לתהליך; הפקודה שגרמה לחריג תרוץ שוב
- ❖ רוב המקרים האחרים גורמים להעפת התהליך או הפעלת שגרה מיוחדת של התהליך (שגרה לטיפול באיתות, signal)

סיבות לחריגי דף והטיפול בהם

- ❖ דף שכלל אינו מוקצה לתהליך
- ❖ דף מוקצה בתהליך אך אינו ממופה למסגרת פיזית
- ❖ הרשאות לא מספיקות לסוג הגישה הנדרש
- ❖ במצב מיוחס: דף של התהליך שאינו מוקצה, מוקצה אך אינו ממופה, או מוקצה וממופה אך ללא הרשאות לסוג הגישה (בדרך כלל החריג הוא תוצאה של גישה של מערכת ההפעלה לארגומנט של קריאת מערכת שכתובתו הועברה לקריאה)
- ❖ במצב מיוחס בגלל דף של מערכת ההפעלה: שגיאה בתהליך (ארגומנט לא נכון לקריאת מערכת) או תקלה במערכת ההפעלה

התייחסות לכתובות פיזיות

- ❖ מערכת ההפעלה צריכה להתייחס לכתובות פיזיות ולהקצות מערכים בזיכרון פיזי רצוף בשביל טבלאות דפים וזיכרון שגם בקרים ניגשים אליו, כמו חוצצים שבקר DMA מעביר מהם/אליהם נתונים
- ❖ דרך פשוטה להתייחס לכתובות פיזיות היא למפות את כל הזיכרון הפיזי או חלקו לזיכרון הוירטואלי של כל התהליכים
- ❖ דוגמה: מיפוי GB 2 של זיכרון פיזי לחצי העליון של מרחב כתובות של 32 סיביות: כתובת פיזית X ממופה לכתובת וירטואלית $X+2^{31}$
- ❖ רשומות מיפוי לדפים גדולים (למשל 4 MB) מייעלות את זה

פינוי מסגרות

- ❖ מערכת ההפעלה צריכה מאגר של מסגרות פנויות שניתן להקצות במהירות עבור תהליכים או עבור מערכת ההפעלה עצמה.
- ❖ מסגרות נקיות (שיש עותק עדכני שלהן בדיסק) ניתן לפנות במהירות
- ❖ מסגרות מלוכלכות, שאין עותק עדכני שלהן בזיכרון משום ששנו מאז שנקראו או שמעולם לא נכתבו לדיסק, יש צורך לשמור בדיסק לפני פינויין
- ❖ כניסות בטבלת הדפים וה-TLB כוללות סיבית ניקיון/לכלוך

מנגנונים לפינוי מסגרות

- ❖ המנגנון משתמש בשלושה מאגרים (רשימות) של מסגרות
 - מסגרות מלוכלכות מועמדות לניקוי
 - מסגרות נקיות מועמדות לפינוי
 - מסגרות פנויות וריקות
- ❖ תהליך מיוחד מתעורר מדי פעם, מוצא מסגרות ראויות לפינוי, מוחק אותן מטבלת הדפים המתאימה, ומעביר לרשימה המתאימה
- ❖ תהליך אחר שגם הוא מתעורר מדי פעם כותב לדיסק מסגרות מלוכלכות מועמדות לפינוי ומעביר אותן לרשימת הנקיות
- ❖ תהליך שלישי (פחות חיוני) מאפס מדי פעם מסגרות נקיות ומעביר אותן למאגר הפנויות

הצלת מסגרות מועמדות לפינוי

- ❖ בזמן טיפול בחריג דף מערכת ההפעלה בודקת האם הדף (שאינו ממופה) שמור במסגרת באחד משני ממאגרי המועמדות לפינוי
- ❖ במצב כזה מערכת ההפעלה מוציאה את המסגרת ממאגר המועמדות לפינוי ומשחזרת את המיפוי בטבלת הדפים, ללא גישה לדיסק
- ❖ זו הסיבה שכדאי להחזיק מאגר גדול של מועמדות נקיות אבל לא לפנות אותן

קביעת קצב הפינוי

- ❖ התהליכים המיוחדים ומפנים כמות מסגרות שנקבעת על פי מצב המערכת
- ❖ כאשר יש מעט מסגרות נקיות/פנויות, התהליכים יפנו מספר גדול כל אימת שהם מתעוררים
- ❖ כאשר יש הרבה מסגרות נקיות, התהליכים יפנו מעט או שלא יתעוררו כלל
- ❖ הפרמטרים שקובעים את קצב הפינוי תלויים בגודל הזיכרון הפיזי והם ניתנים לכיוון

מדיניות לבחירת קורבנות לפינוי

- ❖ מיטבי: מספר חריגי הדף ממוזער אם בכל חריג דף מפנים את המסגרת שהשימוש הבא בה רחוק ביותר
- ❖ לא מעשי כי מערכת ההפעלה אינה יודעת לאיזה מסגרות ייגשו בעתיד וגם משום שרצוי להחזיק מאגר של פנויות
- ❖ מיטבי במודל מתמטי שאינו מתיר הצצה לעתיד: least recently used (LRU), מפנים את המסגרת שהשימוש האחרון בה היה רחוק ביותר בעבר
- ❖ מדיניות LRU פועלת היטב בפועל משום שההיסטוריה חוזרת
- ❖ קשה למימוש כי דרושות חותמות זמן שימוש ב-TLB ובטבלת הדפים וצריך למצוא את הישנה ביותר בכל טבלאות הדפים

קירובים ל-LRU

- ❖ מעבדים אינם תומכים בחותמות זמן שימוש מדויקות למסגרות
- ❖ מעבדים כן מממשים חותמת שימוש בת סיבית אחת שמודלקת אוטומטית בכל שימוש; צריך לאפס אותה מפורשות
- ❖ כיבוי של הסיבית הזו מדי פעם בכל מיפוי ובחירה של מועמדות שהסיבית שלהן עדיין כבויה מבטיחה שלא נעשה בהן שימוש לאחרונה
- ❖ על ידי שמירת הסיבית הזו באוגר הזזה מדי פעם ניתן לקבל קירוב מדויק יותר ל-LRU
- ❖ ניתן לסמלץ סיבית כזו במעבד שאינו תומך בה בעזרת מנגנון חריגי הדף

מדיניות פיננסי דו-שלבית

- ❖ במערכות עם טבלת דפים לכל תהליך (היררכית למשל), קשה לסרוק את כל המסגרות על מנת לחפש כאלה שלא היו בשימוש לאחרונה
- ❖ מערכות הפעלה בוחרות מסגרות לפיננסי בשני שלבים: תהליך שמסגרות שלו יפוננו, ואז מסגרות ספציפיות
- ❖ מדיניות דו-שלבית כזו גם מאפשרת לממש עקרונות של הגינות בין תהליכים, למשל לתגמל תהליכים שמתמשים במעט מסגרות

בחירת תהליך בפינוי מסגרות

- ❖ בלינוקס: מדיניות פשוטה, התהליך שגרם למספר חריגי הדף הקטן ביותר בזמן האחרון
- ❖ גרסאות ישנות יותר של לינוקס: התהליך עם מספר המסגרות הגדול ביותר
- ❖ חלונות: מדיניות working sets:

 - מערכת ההפעלה משערכת את מספר המסגרות שכל תהליך זקוק להן, כתלות בקצב חריגי הדף שלו ותוך כיבוד חסמים עליון ותחתון
 - התהליך שייבחר לפינוי מסגרות הוא התהליך שחורג ממספר המסגרות שהוא "זכאי" להן במידה הגדולה ביותר
 - שערך חוזר מדי פעם

אזורי דפדוף

❖ מסגרות שאין להן מקום בזיכרון הראשי מועברות לאזור דפדוף בדיסק

❖ אזור הדפדוף יכול להיות קובץ רגיל (טיפוסי בחלונות), מחיצה (טיפוסי בלינוקס), או אפילו מספר קבצים, מחיצות ודיסקים שלמים (אפשרי בלינוקס)

❖ בלינוקס ניתן להגדיר עדיפות לכל אזור דפדוף, ולהוסיף ולהסיר אזורים בזמן שהמערכת פועלת

```
$ /sbin/swapon -s
```

Filename	Type	Size	Used	Priority
/dev/sda1	partition	17832112	65536	-1
/tmp/swapfile	file	1048568	0	-2

מיפוי קבצים לזיכרון

- ❖ בכל מערכות ההפעלה ניתן להקצות בלוק של זיכרון וירטואלי שממופה לקובץ מסוים
- ❖ מאפשר לקרוא קובץ פשוט על ידי מיפוי וגישה לזיכרון הממופה
- ❖ במיפוי משותף כתיבה לזיכרון הממופה משנה את תוכן הקובץ (מאפשר להעביר מידע בין תהליכים)
- ❖ במיפוי פרטי כתיבה לזיכרון יוצרת עותק פרטי של המסגרת, ואם יהיה צריך לפנות אותה היא תפונה לאזור דפדוף, לא חזרה לקובץ הממופה
- ❖ מיפוי קבצים משמש לטעינה של קוד מכונה של תוכניות וספריות שגרות לזיכרון (DLL, shared objects)
- ❖ מיפוי משותף של תוכניות וספריות חוסך מסגרות פיזיות

דוגמה למיפוי זיכרון בלינוקס

```
$ cat /proc/27394/maps
```

```
00400000-00404000      r-xp 00000000 fd:01 12931 /usr/bin/xeyes
02719000-0275b000      rw-p 00000000 00:00 0      [heap]
3b6f400000-3b6f415000   r-xp 00000000 fd:01 6767  /lib64/libgcc.so.1
3b6fc00000-3b6fd39000   r-xp 00000000 fd:01 52431 /usr/lib64/libX11.so
3e5d200000-3e5d38f000   r-xp 00000000 fd:01 5914  /lib64/libc-2.14.so
3e5d58f000-3e5d593000   r--p 0018f000 fd:01 5914  /lib64/libc-2.14.so
3e5d593000-3e5d594000   rw-p 00193000 fd:01 5914  /lib64/libc-2.14.so
3e5d600000-3e5d683000   r-xp 00000000 fd:01 5921  /lib64/libm-2.14.so
7fff1c7d0000-7fff1c7f1000 rw-p 00000000 00:00 0      [stack]
7fff1c7ff000-7fff1c800000 r-xp 00000000 00:00 0      [vdso]
ffffffffffff600000-ffffffffffff601000 r-xp 00000000 00:00 0      [vsyscall]
```

address

perm offset dev inode path