הטכניון – מכון טכנולוגי לישראל

מעבדה במערכות הפעלה 046210

תרגיל בית מס' 3

תאריך הגשה: 4 באוקטובר 2022 עד השעה 23:55

הקדמה

ברוכים הבאים לתרגיל מס' 3 במעבדה במערכות הפעלה. במסגרת תרגיל זה נבנה מנהל התקן פשוט שתומך בפעולות קריאה, כתיבה ובמנגנון ioctl (ראו הסבר על מנגון זה בהמשך). מנהל ההתקן יטען כמודול לקרנל (LKM) והתקשורת איתו תהיה בעזרת קובץ מסוג Char Device.

תיאור כללי

מטרתכם היא לייצר התקן שמדמה רשת מחשבים. ההתקן ינהל מספר קבצים (מזוהים לפי המספר המינורי שלהם), שכל אחד מייצג מחשב אחד ברשת. תהליכים יוכלו לשלוח חבילות מידע (הודעות עד גודל של MTU=1500 בתים) אחד לשני באמצעות פעולות כתיבה בקובץ אחד (השולח) ופעולת קריאה בקובץ אחר (המקבל) . כל חבילה מגיעה לתהליך אחד בדיוק בצד המקבל. כדי לזהות לאיזה מהתהליכים החבילה מיועדת, כל תהליך שרוצה לקבל הודעות יבקש מספר שנקרא פורט (port) בקובץ המקבל, והתהליך השולח יצטרך לתת את מספר הפורט בנוסף למספר הקובץ בעת שליחת ההודעה.

לדוגמה, נניח וקיימים במערכת שני קבצים עם מספרי מינור 0 ו-1. כדי לשלוח חבילה מתהליך א' לתהליך ב' נבצע את הפעולות הבאות:

- תהליך א' יפתח את קובץ 0 ותהליך ב' יפתח את קובץ 1
- תהליך ב' יבקש להאזין לפורט במספר כלשהו לבחירתו, שהוסכם מראש עם תהליך א', למשל פורט 123.
- תהליך א' ישלח חבילה (יבצע פעולת כתיבה לקובץ 0) לקובץ מספר 1 ופורט 123
- תהליך ב' יקרא את החבילה (יבצע פעולת קריאה על קובץ 1). אם תהליך ג' שגם פתח את קובץ 1 ומאזין לפורט 234 יבצע קריאה לפני תהליך ב', הוא לא יקבל אף חבילה כי ההודעה לא יועדה לפורט שלו.

כל קובץ בניהול ההתקן ישמור עד QUEUE_SIZE=128 הודעות סה"כ, כאשר הודעה נמחקת מתור ההודעות כאשר התהליך שאליו היא יועדה קרא אותה. ברגע שתהליך סוגר את הקובץ, הפורט שהוא האזין לו משתחרר ותהליכים אחרים יכולים להאזין לו. לשם פשטות, יש לשחרר את תור ההודעות רק כאשר כל התהליכים סגרו את הקובץ. בנוסף, בכל מופע פתוח של הקובץ כל

תהליך יוכל או לשלוח או לקבל חבילות. כדי לבצע מספר פעולות (שליחה וקבלה, קבלה בשני פורטים וכו'), התהליך יצטרך לפתוח את הקובץ כמה פעמים.

פירוט על המימוש

שם ההתקן שתיצרו יהיה devnet, ועליו לממש את הפעולות הבאות מתוך ממשק הקובץ הרגיל של יוניקס. בסוגריים מופיע שם הפונקציה המתאימה ב-file_operations:

- LISTEN (פעולת ioctl): הפעולה מקבלת מספר פורט כפרמטר ומתחילה להאזין לחבילות. המשמעות היא שמרגע ביצוע הפעולה התהליך הנוכחי יוכל לקבל חבילות שיועדו לפורט שניתן ע"י ביצוע read על הקובץ הפתוח הנוכחי. הפורט יהיה הבית הראשון (LSB) בפרמטר arg ל-ioctl, ז"א מספר בין 0 ל-255.
  - ערך החזרה: 0 במקרה של הצלחה, מינוס קוד השגיאה במקרה של שגיאה.
  - שגיאות:
    - EINVAL (Invalid argument): התהליך ביצע כבר LISTEN או CONNECT על הקובץ הפתוח הנוכחי
    - EADDRINUSE (Address already in use): יש כבר תהליך שמאזין לפורט הנתון באותו הקובץ
- CONNECT (פעולת ioctl): הפעולה מקבלת מספר קובץ ומספר פורט ומייעדת את כל החבילות העתידיות אליו. המשמעות היא שמרגע ביצוע הפעולה התהליך הנוכחי יוכל לשלוח חבילות לקובץ והפורט הנתונים ע"י ביצוע write על הקובץ הפתוח הנוכחי. מספר הקובץ יהיה הבית התחתון (LSB) בפרמטר arg ל-ioctl, ומספר הפורט הבית אחריו. ז"א שניהם מספרים בין 0-255.
  - ערך החזרה: 0 במקרה של הצלחה, מינוס קוד השגיאה במקרה של שגיאה.
  - שגיאות:
    - EINVAL (Invalid argument): התהליך ביצע כבר LISTEN או CONNECT על הקובץ הפתוח הנוכחי.

- ECONNREFUSED (Connection reufsed): אין תהליך שמאזין (ביצע LISTEN) לפורט הנתון בקובץ הנתון.

- כתיבה (write): בכתיבה יש להתייחס לתוכן החוצץ שהעובר (פרמטר buf) בתור תוכן החבילה. יש לשים את החבילה בסוף תור ההודעות של הקובץ שהוגדר לפני כן בפונקציה CONNECT. אם גודל החוצץ (פרמטר count) גדול מ-MTU, יש לקחת את MTU הבתים הראשונים בלבד.
  - ערך החזרה: מספר הבתים שנכתבו (גודל החוצץ עד MTU) במקרה של הצלחה. מינוס קוד השגיאה במקרה של שגיאה.
  - שגיאות:
    - ECONNRESET (Connection reset by peer): אין תהליך בקובץ היעד שמאזין לפורט המבוקש (למשל כי התהליך סגר את הקובץ).
    - EAGAIN (Try again): תור ההודעות בקובץ היעד מלא.
    - EINVAL (Invalid argument): התהליך לא ביצע CONNECT בקובץ הפתוח הנוכחי.
    - EFAULT (Bad address): שגיאה בקריאה מחוצץ המשתמש או החוצץ הוא NULL.

- קריאה (read) – כל קריאה תעתיק לחוצץ שהעביר המשתמש (פרמטר buf) את החבילה הבאה בתור שמיועדת לפורט שאליו הקובץ הפתוח הנוכחי מאזין, לפי סדר הגעת החבילות. לאחר ההעתקה החבילה תשוחרר מהתור.

  החבילה תיכתב בצורת המבנה הבא:

```
struct packet {
   unsigned char source_file;
   pid_t source_pid;
   int payload_size;
   unsigned char payload[MTU];
}
```

  כאשר `source_file` מכיל את המינור של הקובץ שממנו הגיעה החבילה, `source_pid` מכיל את ה-PID של התהליך ששלח את החבילה, `payload_size` מכיל את אורך התוכן ו-`payload` מכיל את תוכן ההודעה בבתים 0 עד `payload_size-1`. שאר הבתים של `payload` יכולים להיות זבל.
  - ערך החזרה: מספר הבתים שנכתבו לחוצץ ((sizeof(struct packet) במקרה של כתיבה מוצלחת של הודעה. 0 במקרה שאין הודעות שממתינות בתור. מינוס קוד השגיאה במקרה של שגיאה.
  - שגיאות:

- ENOSPC (No space left on device): יש הודעה בתור אבל count < sizeof(struct packet)
- EINVAL (Invalid argument): התהליך לא ביצע LISTEN בקובץ הפתוח הנוכחי.
- EFAULT (Bad address): שגיאה בהעתקה לחוצץ המשתמש או החוצץ הוא NULL.

הערות נוספות:

- עבור כל הפונקציות, אם יש לכם שגיאה בהקצאת זיכרון דינמי יש להחזיר ENOMEM.
- בקבצי העזר לתרגיל ישנם שלדי מימוש לחלק מהפונקציות שעליכם לממש, אך לא בהכרח לכולן. היעזרו בחומרי העזר המצוינים למטה לפי הצורך.
- ההגדרה של struct packet, הקבוע MTU וכן מספרי הפעולות של CONNECT ו-LISTEN עבור מנגון ה-ioctl מוגדרים עבורכם בקובץ שלד המצורף לתרגיל. ראו מידע כללי על ioctl בנספח בסוף ודוגמאות מימוש בלינקים למטה.
- מספר major של ההתקן צריך להינתן באופן דינמי.

מידע שימושי

- אפשר לחלץ את הבית הראשון והשני ממספר ע"י פעולות מודולו (%) וחילוק.
- הסבר על מנהלי התקן מסוג Char Driver + דוגמאות ניתן למצוא בקישור הבא וכן בקישור הבא.
- אפשר למצוא הסבר על איך להוסיף תמיכה ב- IOCTL בקישור הבא.

בדיקה של מנהל ההתקן

כדי לקמפל את מנהל ההתקן יש להשתמש בפקודה הבאה (לחלופין ניתן להשתמש בקובץ Makefile המצורף):

```
gcc –c –I/usr/src/linux-2.4.18-14/include –Wall devnet.c
```

לשם בדיקת מנהל ההתקן יש להתקין אותו ולבדוק האם הוא מבצע נכון את כל הפעולות הנדרשות ממנו. טעינת מנהל ההתקן נעשית בעזרת הפקודה:

```
insmod ./devnet.o
```

כדי ליצור קובץ התקן המשויך למנהל ההתקן ניתן להיעזר בפקודה mknod. לדוגמא, הפקודה הבאה תיצור קובץ התקן בשם devnet המזוהה על ידי מספר מינורי 0:

```
mknod /dev/devnet c major 0
```

במקום major יש לרשום את מספר ה-major שנבחר להתקן שלכם. ניתן למצוא את מספר ה-major שנבחר להתקן בקובץ

/proc/devices

הסרת ההתקן נעשית בעזרת הפקודות:

```
rm -f /dev/devnet
rmmod devnet
```

מומלץ לכתוב סקריפטים שיבצעו את הפעולות הנ"ל בצורה אוטומטית על מנת לחסוך זמן וטעויות אנוש.

בדיקת תפקוד מנהל ההתקן תתבצע על ידי פתיחת קובץ ההתקן, ביצוע הפעולות המפורטות לעיל ובדיקת התוצאות שלהן. ניתן לבצע זאת בעזרת תכנית בשפת C. כמו כן ניתן גם לעשות זאת בשפת סקריפט כגון Python שמאפשרת פיתוח מהיר ונוח של תכניות. תיעוד שפת Python נמצא בקישור הבא, הסבר על גישה לקבצים נמצא בקישור הבא ועל ביצוע פניות IOCTL ניתן לקרוא בקישור הבא. בין קבצי התרגיל מצורף קובץ בדיקה לדוגמא: test.py.

אין צורך להגיש את התוכנית בה השתמשתם לבדיקת מנהל ההתקן שלכם.

הוראות הגשה לתרגיל:

- הגשה אלקטרונית דרך אתר הקורס.
- הגשה בזוגות בלבד. אפשר להשתמש בפורום באתר הקורס למציאת שותפים.
- יש להגיש דרך חשבון של אחד השותפים בלבד.(אין להגיש פעמיים - מכל חשבון).

ההגשה בקובץ **zip** בשם id1_id2.zip , כאשר id1, id2 הם מספרי ת.ז. של השותפים. הקובץ יכיל:

- קבצי התוכנית: devnet.h, devnet.c

- קובץ טקסט submitters.txt עם הנתונים של המגישים לפי המתכונת הבאה:

first_name1 last_name1 id1
first_name2 last_name2 id2

יש להקפיד שקובץ ה zip לא יכיל קבצים נוספים, לרבות תיקיות. דהיינו, על ההגשה להיות בצורה הבאה:

```
zipfile -+
         |
         +- submitters.txt
         +- devnet.c
         +- devnet.h
         |
```

דגשים לגבי הציון

- הקפידו על מילוי הדרישות לשמות הקבצים והממשקים. כל טעות הגשה תוריד **5** נקודות מהציון על התרגיל.

- יש להקפיד על סדר ותיעוד הקוד.

- הגשה באיחור תגרור הורדת ניקוד.

נספח: הסבר על מנגנון ה-*ioctl*

בהרבה התקנים (כמו ההתקן שלנו בתרגיל הזה), המנגנון המועדף והפשוט ביותר להעביר מידע בין המשתמש להתקן הוא דרך קובץ שאותו קוראים וכותבים. במנגון הזה משתמשים כדי לעשות את הפעולות הפשוטות ו\או הפעולות שדורשות להעביר מידע בקצב מהיר. הבעיה היא שמנגנון זה מוגבל לשתי פעולות (קריאה וכתיבה). במקרה וההתקן רוצה לחשוף כלפי המשתמש עוד פונקציות, הוא צריך דרך כלשהי להרחיב את הממשק הקיים. לצורך כך המציאו את מנגנון ioctl – קריאת מערכת כללית שמריצים על קובץ התקן ומעבירים לה מספר פונקציה ופרמטר. מנגנון זה מאפשר להתקן להגדיר פונקציות נוספות, להצמיד להם מספר ולהודיע למשתמש (למשל בתיעוד של ההתקן) מה המשמעות של כל מספר פונקציה ומה הפרמטר שהוא צריך להעביר.

המשתמש לאחר מכן ישתמש בקריאת המערכת ioctl על הקובץ של ההתקן ביחד עם מספר הפונקציה כדי לבצע את הפעולות הנוספות.

דוגמה פשוטה לשימוש ב-ioctl הוא כרטיס קול. ההתקן של כרטיס הקול יצור קובץ התקן, אשר קריאה ממנו תבצע הקלטה (דרך שקע המיקרופון של הכרטיס) וכתיבה אליו תשמיע צליל (דרך שקע הרמקולים). אבל אם המשתמש רוצה לשנות הגדרות של הכרטיס, כמו למשל קצב הדגימה של המיקרופון, הוא יפתח את קובץ ההתקן ויבצע ioctl. במספר הפונקציה הוא יכתוב את מה שכתוב בתיעוד של מנהל ההתקן, ובפרמטר הוא יכול להעביר את קצב הדגימה כמספר חיובי. עבור פונקציה אחרת (של אותו ההתקן או התקן אחר), הפרמטר יכול להיות מספר שלילי או אפילו מצביע. מבחינת ioctl עצמה אין משמעות לפרמטר – המימוש במנהל ההתקן יתייחס לפרמטר לפי הטיפוס המתאים למספר הפונקציה שהמשתמש ביקש להריץ.

שימו לב שמספר הפונקציה הוא שרירותי לחלוטין ופרטי למנהל ההתקן שאתם כותבים. למרות זאת, הוגדרו קונבנציות איך ליצור את המספרים האלה ויש פונקציות עזר לשם כך. זאת המשמעות של שורות ה-define בקובץ השלד – להגדיר את המספרים של פונקציות ה-ioctl בתרגיל לפי הקונבנציות הנהוגות בקרנל.