

מעבדה במערכות הפעלה 046210

תרגיל בית מס' 1

תאריך הגשה: 29 באפריל 2021 עד השעה 23:55

הקדמה

ברוכים הבאים לתרגיל מס' 1 במעבדה במערכות הפעלה.

במסגרת תרגיל זה נבנה מנהל התקן פשוט שתומך בפעולות קריאה, כתיבה ובמנגנון ioctl (ראו הסבר על מנגנון זה בהמשך).

פרוט התרגיל שיש להגיש

בתרגיל זה יש ליצור מנהל התקן מסוג Char Device אשר יטען כ-LKM. לפני המשך קריאת פרוט התרגיל יש לקרוא את

ההסבר על מנהלי ההתקנים (תחת סעיף [מידע שימושי](#) שבהמשך).

מטרתכם היא לייצר התקן עבור מערכת צ'אט. קריאה וכתיבת הודעות יתבצעו ע"י קריאה וכתיבה לקובץ בשליטת ההתקן. על ההתקן לתמוך במספר חדרים שפעילים במקביל, כל אחד בקובץ התקן משלו. לכל חדר פעיל, ההתקן ישמור רשימה מקושרת של ההודעות (ראו פירוט מה לשמור לכל הודעה בהמשך). כל תהליך שפותח את הקובץ יוכל לקרוא הודעות בקצב שלו – ההתקן יעקוב אחרי המיקום בתוך רשימת ההודעות של כל תהליך בנפרד. בנוסף כל תהליך יוכל לשאול את ההתקן כמה הודעות נשארו לו לקרוא ולבצע חיפוש להודעה הבאה שנשלחה מהתהליך מסוים.

פירוט פונקציונליות ההתקן:

- פתיחה של קובץ ההתקן - כאן יש לבצע איתחול של החדר – ליצור ולאתחל מבנים שתשתמשו בהם ולהקצות זיכרון אם צריך. אם התקבלה שגיאה בהקצאת הזיכרון יש להחזיר ENOMEM (שימו לב למינוס). עבור כל שגיאה אחרת באתחול יש להחזיר EFAULT. ההתקן יחזיר 0 בעת אתחול נטול שגיאות.
- שימו לב: במקרה של פתיחה של אותו קובץ התקן (מזוהה ע"י אותו מספר מג'ורי ואותו מספר מינורי) מספר פעמים זה מצב שבו כמה תהליכים נמצאים ביחד באותו החדר, ולכן כולם כותבים וקוראים מאותה רשימת הודעות. בנוסף, החל מהפתיחה השנייה של החדר אין צורך לבצע איתחול ראשוני, כי התהליך רק מצטרף לחדר ולא יוצר חדר חדש.
- שחרור המשחק (סגירת קובץ התקן) – יש לשחרר את רשימת ההודעות ומשאבים נוספים שהקצתם למען פעילות תקינה של החדר.

שחרור חדר הצ'אט

שימו לב: רק בסגירת כל קבצי ההתקן הפתוחים יש לשחרר את רשימת ההודעות (ניתן להשתמש בעיקרון

reference counting על מנת לנהל כמה קבצי התקן פתוחים בכל רגע נתון במערכת).

- קריאה – כל קריאה תעתיק לחוצץ שהעביר המשתמש מספר הודעות לפי גודל החוצץ. אתם יכולים להניח כי החוצץ שאתם מקבלים הוא מסוג `message_t[]` (מערך של `message_t`, ראו פירוט למטה) ושפרמטר הגודל הוא כפולה שלמה של `sizeof(message_t)`. אם אין מספיק הודעות, יש למלא את החוצץ חלקית. לבסוף יש להחזיר את מספר הבתים (לא ההודעות) שנכתבו לחוצץ, שיהיה שווה למספר ההודעות שנכתבו כפול `sizeof(message_t)`. למשל, אם קיבלתם חוצץ בגודל `3*sizeof(message_t)` אבל יש רק הודעה אחת בחדר, יש לכתוב רק הודעה אחת לחוצץ ולהחזיר `1*sizeof(message_t)`. בנוסף יש לעקוב אחרי ההיסט (offset) – אם יש שתי הודעות בחדר ומתבצעות שתי בקשות קריאה מאותו התהליך, כל אחת להודעה אחת, הקריאה הראשונה תחזיר את ההודעה הראשונה והקריאה השנייה תחזיר את ההודעה השנייה. עושים זאת בעזרת הפרמטר `f_pos` – בתחילת הקריאה הוא יציין את המיקום הנוכחי (בבתים, ז"א בדאי לחלק ב `sizeof(message_t)`) של התהליך בתוך רשימת הקריאות, ובסוף בקריאה יש לקדם אותו במספר הבתים שקראנו. אם המצביע לחוצץ הוא NULL יש להחזיר EFAULT. אם יש בעיה בלהעתיק נתונים למרחב המשתמש יש להחזיר EBADF. בכל שגיאה אחרת יש להחזיר EFAULT.
- כתיבה – כל כתיבה תוסיף את ההודעה (מחרוזת) לסוף רשימת ההודעות. על ההודעות להופיע ברשימה לפי הסדר שבו ההתקן קיבל אותן. יכול להיות שמספר תהליכים יבצעו כתיבה, אך אפשר להניח כי הן לא נעשות במקביל (אין צורך להשתמש בנעילות או בטכניקות אחרות לתכנות מקבילי). על ההתקן לוודא שאורך ההודעה (ללא תו NULL) לא עולה על `MAX_MESSAGE_LENGTH`. ההתקן יצרף להודעה את ה-PID של התהליך ששלח אותה ואת הזמן שבו ההודעה נשלחה. את הזמן יש לשמור כ `time_t` (מספר שלם המייצג שניות מאז 1 בינואר 1970). לבסוף יש להחזיר את מספר הבתים שנכתבו להתקן (במקרה שלנו זה יהיה תמיד אורך החוצץ שקיבלתם). במקרה של הודעה ארוכה מדי, יש להחזיר ENOSPC. במקרה של חוצץ NULL יש להחזיר EFAULT. במקרה של שגיאה בקריאה מחוצץ המשתמש יש להחזיר EBADF. בכל שגיאה אחרת יש להחזיר EFAULT.
- שינוי היסט (lseek) – תהליך יוכל לזוז בתוך רשימת ההודעות קדימה ואחורה. עליכם לתמוך בכל סוגי ההזזה:
 - SEEK_SET – שינוי המיקום הנוכחי של התהליך ברשימת ההודעות, ביחס להתחלה.
 - SEEK_CUR – שינוי המיקום הנוכחי של התהליך ברשימת ההודעות, ביחס למיקום הנוכחי.

○ SEEK_END – שינוי המיקום הנוכחי של התהליך ברשימת ההודעות, ביחס לסוף הרשימה.

המיקום שאתם תקבלו יהיה בבתים (לכן שוב, כדאי לחלק ב-`sizeof(message_t)` כדי להשיג מיקום ברשימה) והוא עשוי להיות שלילי. אם כתוצאה משינוי ההיסט המיקום חורג מעבר לסוף הרשימה, יש לאפס את המיקום לאחר ההודעה האחרונה (ז"א שקריאה תחזיר 0 עד הכתיבה הבאה). באופן דומה, אם הזזה עם מיקום שלילי חורגת מעבר לתחילת הרשימה, יש לאפס את המיקום ל-0. לבסוף יש להחזיר את המיקום החדש (בבתים) של התהליך. אם קיבלתם סוג שינוי לא מוכר, יש להחזיר -EINVAL.

• COUNT_UNREAD – פעולת `ioctl`. תחזיר את מספר ההודעות שעדיין לא נקראו – מספר ההודעות מהמיקום

הנוכחי של התהליך ברשימה עד הסוף. למשל, אם יש 3 הודעות ברשימה והמיקום שלנו הוא 1 (ז"א, לפני ההודעה

לפי איזה FPOS עובדים?
של התהליך המחפש או התהליך מהARG?

השניה), COUNT_UNREAD תחזיר 2.

• SEARCH – פעולת `ioctl`. מקבלת מספר תהליך ומחזירה את המיקום (בבתים) של ההודעה הבאה (לפי המיקום

הנוכחי של התהליך) שנשלחה ממספר התהליך הזה. למשל – נגיד ויש 3 הודעות שנשלחו מתהליכים A, B, A. אם

תהליך נמצא כרגע במיקום 0 ועושה SEARCH לתהליך A הוא יקבל 0. אם הוא נמצא במיקום 1 ועושה את אותו

מחזיר מיקום אבסולוטי
במערך באריתמטיקה של
פוינטרים, כלומר מרחק
בבתים מתחילת המערך.

הSEARCH הוא יקבל $2 * \text{sizeof}(\text{message_t})$.

שם מנהל ההתקן צריך להיות `chat`. מספר `major` של ההתקן צריך להינתן באופן דינמי. קבצי התרגיל מכילים שלד (`chat.h` ,

`chat.c`) עליו ניתן לבנות את מנהל ההתקן. בנוסף, בקבצי השלד ישנם פונקציות עזר לשליפה של הזמן הנוכחי כ-`time_t`

מהמערכת ולקבל את מספר התהליך הנוכחי. שימו לב שהקבצים שקיבלתם לא בהכרח מכילים שלד לכל הפונקציות שעליכם

לממש. היעזרו בחומרי העזר המצוינים למטה לפי הצורך.

השתמשו בהגדרה הבאה עבור `message_t`:

```
struct message_t {
    pid_t pid;
    time_t timestamp;
    char[MAX_MESSAGE_LENGTH] message;
}
```

שימו לב שמכיוון שיש אורך מקסימלי ידוע ל-`message`, יש להוסיף תו NULL בסוף רק אם אורך ההודעה שאתם כותבים קטן מ-

`MAX_MESSAGE_LENGTH`.

את פעולות ה-COUNT_UNREAD וה-SEARCH יש לממש על ידי מנגנון ה-IOCTL לפי הפרוט הבא (ראו מידע על כללי על

ioctl למטה):

```
#define MY_MAGIC 'r'
#define COUNT_UNREAD_IO(MY_MAGIC, 0)
#define SEARCH_IO(MY_MAGIC, 1)
```

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. במנגנון הזה משתמשים כדי לעשות את הפעולות הפשוטות ו\או הפעולות שדורשות להעביר מידע בקצב מהיר. הבעיה היא שמנגנון זה מוגבל לשתי פעולות (קריאה וכתובה). במקרה וההתקן רוצה לחשוף כלפי המשתמש עוד פונקציות, הוא יכול לעשות זאת בעזרת 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) וכתובה 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.

מידע שימושי

- הסבר על מנהלי התקן מסוג Char Driver + דוגמאות ניתן למצוא ניתן למצוא [בקישור הבא](#) וכן [בקישור הבא](#).
- אפשר למצוא הסבר על איך להוסיף תמיכה ב- IOCTL [בקישור הבא](#).
- ניתן, ורצוי, לדבג את המודול בעזרת פקודת `printk`.
- הגרעין לא מקופל אל מול הספריות הסטנדרטיות (GNU CLib). לכן לא ניתן לעשות שימוש בפונקציות מתוך `'string.h'`, `'stdio.h'`, `'stdlib.h'` וכדומה. עם זאת, לרוב הפונקציות השימושיות יש תחליף ומימוש מתאים בגרעין.
- הגרעין עושה שימוש בזיכרון פיזי. לשם הקצאת זיכרון דינמי בגרעין יש לעשות שימוש בפקודות `kfree` ו- `kmalloc` אשר מקצות זיכרון באופן דינאמי. שימו לב, עליכם לדאוג לשחרר זיכרון שאתם מקצים, דליפות זיכרון יחשבו כשגיאה.
- זיכרון השייך למרחב הגרעין חייב להיות מופרד מטעמי אבטחה מזיכרון המשתמש. לכן, קוד מנהל ההתקן לא יכול לגשת ישירות למידע הנמצא במרחב הזיכרון של המשתמש באופן ישיר! בעקבות זאת עליכם לעשות שימוש בשגרות `copy_to_user`, `copy_from_user`, `strlen_user` על מנת להעביר מידע בין המרחבים. ניתן לקרוא עליהן [בקישור הבא](#) ו- [בקישור הבא](#).
- בקובץ ההתקן ישנו שדה בשם `private_data` בו אתם יכולים להשתמש על מנת לשמור מידע שיישאר לכל אורך חיי ההתקן (כל עוד הוא לא נסגר בעזרת פונקציית `close()` או כאשר התהליך שפתח את קובץ ההתקן נסגר בעצמו). לדוגמא, תוכלו לשמור מבנה אשר מנהל את החוצץ בשדה זה. שימו לב, פתיחת אותו קובץ התקן מספר פעמים יחזיר מבנה חדש עם מצביע `private_data` חדש (אשר מאותחל בהתחלה תמיד ל-NULL). זכרו, בתרגיל זה כל התקן מצלמה מנהל חוצץ אחד בלבד, ולכן יש לדאוג ש- `private_data` יצביע לאותו מבנה נתונים עבור קבצי התקן זהים (בעלי אותו מספר מג'ורי ומספר מינורי).
- המיקום הנוכחי בקובץ פתוח (מסוג `struct file`) נשמר בשדה `f_pos`. ניתן לראות את ההגדרה המלאה של המבנה בקובץ `include/linux/fs.h`.

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 chat.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 ./chat.o
```

כדי ליצור קובץ התקן המשויך למנהל ההתקן ניתן להיעזר בפקודה `mknod`. לדוגמא, הפקודה הבאה תיצור קובץ התקן בשם

`chat` המזוהה על ידי מספר מינורי 0:

```
mknod /dev/chat 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/chat
```

```
rmmod chat
```

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 השתמשותם לבדיקת מנהל ההתקן שלכם.

הוראות הגשה לתרגיל:

- תאריך הגשה: 29 באפריל 2021 עד השעה 23:55.
- 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 יכול:
 - קבצי התוכנית: `chat.h`, `chat.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
+- chat.c
+- chat.h
|
```

דגשים לגבי הציון

- משקל התרגיל הינו 30% מהציון הסופי.
- הקפידו על מילוי הדרישות לשמות הקבצים והממשקים. כל טעות הגשה תוריד 5 נקודות מהציון על התרגיל.
- התרגילים יבדקו אוטומטית. עבודה אשר אינה עוברת את הבדיקה האוטומטית, הציון עבורה יהיה 0.
- יש להקפיד על סדר ותיעוד הקוד.
- על כל יום איחור יופחתו 4 נקודות, עד לתקרה של 24 נקודות.

קובץ `errno.h`
הוספת הפונקציות ל-`file_operations`
הוספת דגל לדיבוגים
קראית `DMESG`
לוודא
`include <asm/current.h>#include#`
`linux/slab.h>#include<`
`linux/limits.h>#include<`
`<stdbool.h>`
לעבוד ב-`ROOT`