TECHNION
Israel Institute of Technology

Computer Science Department

Operating Systems – 234123

Homework Exercise 2 – Dry

Name: Nadav Orzech

ID: 311549455

Email: nadav.or@campus.technion.ac.il

Name: Roni Englender

ID: 312168354

Email: roni.en@campus.technion.ac.il

שאלה 1

שאלה זו עוסקת במדיניות זימון התהליכים של לינוקס כפי שנלמדה בתרגולים. לנוחיותכם מצורפים מספר macros המשמשים את זמן התהליכים

```
#define MAX_PRIO 140
#define MIN_TIMESLICE (10 * HZ / 1000)
#define MAX_TIMESLICE (300 * HZ / 1000)
#define TASK_TIMESLICE(p)
MIN_TIMESLICE + (MAX_TIMESLICE – MIN_TIMESLICE) * \
(MAX_PRIO – 1 – (p)>static_prio)/39
#define TASK_INTERACTIVE(p) \
((p)>prio <= (p)>static_prio – DELTA(p))
prio = static_prio – bonus
#define EXPIRED_STARVING(rq) \
((rq)>expired_timestamp && \ ((jiffies – (rq)>expired_timestamp)
>=STARVATION_LIMIT * ((rq)>nr_running + 1))
BONUS(p) = 25% × 40 × ( SleepAvg/MaxSleepAvg – 1/2)
DELTA(p) = 5 × 20 TaskNice(p) + 2
```

א. נניח כי תהליך A מסווג כחישובי על ידי אלגוריתם הזימון של לינוקס ובעל עדיפות סטטית x ותהליך B מסווג כאינטראקטיבי על ידי אלגוריתם הזימון של לינוקס ובעל אותה עדיפות סטטית x .האם יתכן כי העדיפות הדינמית של A טובה יותר משל B?

נתון שלשני התהליכים עדיפות סטטית זהה, ולכן השוני בעדיפות הדינמית שלהם נובע מהבונוס השונה אשר הם מקבלים. כיוון שB מסווג כאינטרקטיבי Aו כחישובי בהכרח הבונוס של תהליך B יותר גדול => עדיפות מספרית יותר נמוכה => עדיפות דינמית טובה יותר.

ב. נניח כי תהליכים A ו B מסווגים כחישוביים על ידי אלגוריתם הזימון של לינוקס ובעלי עדיפות דינמית שווה, אבל עדיפות הסטטית של A טובה יותר (נמוכה יותר מספרית). איזה יתרון מקבל A על B?

נתון שלתהליכים עדיפות דינמית שווה אבל העדיפות הסטטית של A טובה יותר (נמוכה מספרית). אורך הtimeslice שתהליך מקבל הוא לינארי **בעדיפות הסטטית** של אותו תהליך, ולכן A מקבל timeslice יותר גדול מתהליך B.

ג. נניח כי תהליכים A ו B מסווגים כאינטראקטיביים על ידי אלגוריתם הזימון של לינוקס ובעלי עדיפות דינמית שווה. אבל העדיפות הסטטית של A טובה יותר (נמוכה יותר מספרית). איזה יתרון מקבל A על B?

נתון שלתהליכים עדיפות דינמית שווה אך העדיפות הסטטית של A עדיפה, משמע לתהליך A ערך nice קטן יותר ולכן "קל יותר" לתהליך A להיות מסווג כאינטרקטיבי. כלומר התהליך צריך לצבור זמן המתנה ממוצע נמוך יותר על מנת להיחשב כאינטרקטיבי והוא יכול לבצע יותר חישובים לפני שהופך לחישובי.

שימו לב: בסעיף ד, שלושה תת סעיפים (1,2,3)

ד. לפניך קטע מתוך הקוד של פונקציית הגרעין `yield_sched_sys` אשר מממש את הטיפול בתהליכים עם מדיניות זימון OTHER

```
1.list_del(&current¬>run_list);
```

```
2. if(!list_empty(array¬>queue +current¬>prio)){
3.  list_add(&current¬>run_list,array¬>queue[current¬>prio].next);
4.  goto out_unlock;
5. }
6. __clear_bit(current¬>prio, array¬>bitmap);
7. i =sched_find_first_bit(array¬>bitmap); // this would return MAX_PRIO on
fail (in case no set bits found)
8. if(i==MAX_PRIO ||i<=current¬>prio)
9.  i =current¬>prio;
10. else
11.     current¬>prio =i;
12. list_add(&current¬>run_list,array¬>queue[i].next);
13. __set_bit(i,array¬>bitmap);
14. out_unlock:
15. // release locks & call schedule
```

1. בהנחה שקיימים תהליכים נוספים שאינם expired ב runqueue האם יתכן כי ביצוע yield_sched על ידי תהליך עם מדיניות זימון OTHER לא יגרום להחלפת הקשר?

כאשר יש תהליכים נוספים בrunqueue ביצוע yield_sched תמיד יגרום להחלפת הקשר. במידה וקיימים עוד תהליכים בעלי אותו עדיפות כמו התהליך הנ"ל, התהליך יכנס לרשימה שממנה הוצאנו אותו אך למקום השני ולכן תתבצע החלפת הקשר וירוץ התהליך שעבר להיות במקום הראשון. אם אין עוד תהליכים רצים באותה העדיפות כמו התהליך הנ"ל, עדיפות התהליך תשונה לרעה להיות העדיפות של התהליך הבא שקיים אחריו, וגם במקרה הזה הוא יכנס למקום השני ברשימה ותתבצע החלפת הקשר.

2. איזו בעיה הייתה נוצרת אם היינו מחליפים את שורה 8 בשורה:

```
if(i<=current>prio)
```

אם נוריד את התנאי שבודק אם i==MAX_PRIO, נתעלם מהבדיקה שהתהליך הוא היחידי שרץ כעת בrunqueue. במצב הזה כמה שורות מתחת היינו מנסים להכניס את התהליך ל[array->queue[MAX_PRIO, דבר שהיה גורם לגישה לא חוקית לזיכרון.

3. איזו בעיה הייתה נוצרת אם היינו מחליפים את שורה 8 בשורה:

```
if(i==MAX_PRIO)
```

אם נורית את התנאי i<=current->prio, נתעלם מהבדיקה שקיימים תהליכים בעדיפות גבוהה יותר, וכמה שורות מתחת שמבצעים הכנסה ל[array->queue[i היינו מכניסים את התהליך לרשימה של התהליכים בעלי העדיפות הגבוהה ממנו, כלומר מקדמים אותו בעדיפות. מצב זה עלול לאפשר החלפת תהליכי OTHER לתהליכי RT.

שאלה 2

במסגרת הבחינה בקורס, תתבקשו לענות על שאלות הן על החומר הנלמד (בתרגולים ובהרצאות) והן על מערכות שונות ומגוונות אשר לא נלמדו בקורס, דבר הדורש הכללה של החומר ועקרונותיו.

בשאלה זו ננתח מערכת זימון הדומה במהותה ללינוקס, אך במקביל, גם קצת שונה. נתון אלגוריתם SCHED_OTHER במערכת זו:

- עדיפות התהליכים נקבעת על פי השדה **static_prio** אשר טווח ערכיו בין 1 ל 5 וערכו נקבע על ידי המשתמש (1 הוא העדיף ביותר ו 5 הכי פחות עדיף).
- בכל רגע נתון מוגדר **זמן ריצה מינימלי** למשימות בעלות עדיפות i אשר נסמנו q_i או q_i , על פי הנוסחה:

$$q_i = \max\left\{\frac{target\_latency}{N_i}, min\_granularity\right\}$$

כאשר target_latency, min_granulartiy קבועים המוגדרים במערכת, ו- N_i מספר המשימות עם עדיפות i.

- לכל משימה יש שדה vruntime המאותחל ל 0 בעת יצירתה. בכל פסיקת שעון מתעדכן שדה באופן הבא:

$$current \rightarrow vruntime += current \rightarrow static\_prio$$

- בכל פעם שמשימה נבחרת לרוץ נשמר ערכו של השדה vruntime כך:

$$current \rightarrow start\_vruntime = current \rightarrow vruntime$$

- בכל פסיקת שעון נבדקים שני התנאים הבאים:
  - קיימת משימה בעלת vruntime קטן יותר משל המשימה הנוכחית
  - q_{current->static_prio} <= current->vruntime - current->start_vruntime
- אם שני התנאים הללו מתקיימים, מתבצעת החלפת הקשר (בעזרת הדגל need_resched)
- בהחלפת הקשר המשימה הבאה שנבחרת לרוץ היא זאת בעלת ה vruntime המינימלי.

א. בהנחה שהאלגוריתם הנ"ל עושה שימוש בעץ חיפוש מאוזן הממוין לפי vruntime.

1. מהי סיבוכיות הזמן של **בחירת** המשימה הבאה? הסבר.

ניתן להחזיק מצביע לבן השמאלי ביותר בעץ ואז תמיד הבחירה תתבצע ב(o(1

2. מהי סיבוכיות הזמן של **הוספת** משימה חדשה? הסבר.

הוספה בעץ חיפוש מאוזן – (o(logn כאשר n מספר הצמתים בעץ

3. מהי סיבוכיות הזמן של **הסרת** משימה? הסבר.

הסרה בעץ חיפוש מאוזן – (o(logn כאשר n מספר הצמתים בעץ

ב. נניח שבמערכת יש שני תהליכים CPU-Bound בעלי עדיפות זהה ושהשניים הללו הינם התהליכים היחידים במערכת. איזה אחוז מהזמן ירוץ כל אחד מהתהליכים?

כל אחד מהתהליכים ירוץ 50% מהזמן. כיוון שיש להם אותה העדיפות הi_q שלהם זהה ולכן זמן הריצה שלהם שווה. כל מספר פסיקות שעון כאשר התנאי השני (והראשון) מתקיימים תתבצע החלפת הקשר וחוזר חלילה.

ג. נניח שבמערכת שני תהליכים CPU-Bound, אחד בעל עדיפות 1, והשני בעל עדיפות 3. נניח שהשניים הללו הינם התהליכים היחידים במערכת. איזה אחוז מהזמן ירוץ כל אחד מהתהליכים?

תהליך 1 ירוץ 75% מהזמן לעומת תהליך 3 שירוץ 25% מהזמן. מתקיים q_1=q_3 (כי הם שני התהליכים היחידים במערכת), ובגלל שה vruntime של תהליך 1 גדל פי 3 יותר לאט מתהליך 3 הוא ירוץ פי 3 יותר זמן כל פעם שנבחר.

ד. באלגוריתמי הזימון SCHED_OTHER שלמדנו (בתרגול ובהרצאה), המערכת מחשבת לכל תהליך עדיפות דינמית כדי להבדיל בין תהליכים שהם IO-Bound לתהליכים שהם CPU-Bound. באלגוריתם הנ"ל, אין הפרדה כזו. כיצד בכל זאת מתעדפת המערכת תהליכי IO-Bound כראוי?

כיוון שהתהליך אינטרקטיבי יהיה לו vruntime נמוך (כי הוא אינו רץ הרבה זמן אז הvruntime לא מספיק לעלות הרבה) ולכן התהתיך יקבל עדיפות יותר גבוהה להיבחר. בנוסף לרוב כיוון שהוא לא רץ הרבה הוא לא יגיע לרוץ q_i זמן ולכן לא נעצור אותו באמצע.

ה. באיזו בעיה היינו עלולים להיתקל אם לא היה נעשה שימוש בקבוע `min_granulartiy`?

אם לא היה את הקבוע min_granularity, במצב שבו יש הרבה תהליכים בעלי אותה עדיפות סטטית, הq_i שהם מקבלים היה מאוד נמוך (כי הוא מחולק בכמות התהליכים בעלי עדיפות i) והם היו מקבלים זמן ריצה מאוד קטן. מצב זה גם היה גורם לכך שמרבית הזמן מבוזבזת על החלפות הקשר מרובות.

שימו לב: הסעיף הבא מתייחס למערכת כפי שנלמדה בתרגולים. בפרט, אלגוריתם זימון המשימות הוא זה שנלמד בביתה ולא האלגוריתם שהוזכר לעיל.

ו. נניח כי תהליך מסויים מודע לזמן בו מתרחשת פסיקת שעון והוא מסוגל לבחור להריץ קוד כרצונו בדיוק לפני/אחרי שמתקבלת פסיקת שעון. כיצד יכול התהליך לנצל זאת כדי "לרמות" את אלגוריתם הזימון?

התהליך יכול לדאוג לא לרוץ כלל בזמן פסיקת שעון (לצאת לyield או המתנה לפני הפסיקת שעון) ובכך גם לא יגמר לו הtimeslice וגם הוא יקבל בונוס גבוה כי מבחינת המערכת הוא אינו רץ כלל כיוון שהיא אוספת את המידע הזה בפסיקות שעון. (אם התהליך יצא לyield הוא לא יבחר לרוץ הבא בתור ובמידה ויש עוד משימות שנמצאות בתור).