מערכות הפעלה
234123

תרגיל יבש: 1

הוגש ע״י:

רון ביידר	302894784	Roni2706@gmail.com
שם	מספר סטודנט	אימייל
דביר פרי	300411659	sdperry@campus.technion.ac.il
שם	מספר סטודנט	אימייל

<u>שאלה 1</u>

כפי שראינו תתבצע בערך שרשרת הקריאות הבאה:

$$\boldsymbol{open \rightarrow int\ 0x80 \rightarrow system\_call\ \rightarrow sys\_open \rightarrow \cdots \rightarrow iret \rightarrow open}$$

א. מהי הפקודה האחרונה המבוצעת ב $\boldsymbol{user\ mode}$ לפני המעבר ל $\boldsymbol{kernel}$?

- הפקודה האחרונה המבוצעת ב $user\ mode$ היא פסיקת התוכנה $int\ 0x80$. אחריה מטופלת הפסיקה ברמת ה- $kernel$.

ב. לאיזה מבין הפונקציות הנ"ל מפתחי ה-$\boldsymbol{kernel}$ היו צריכים לדאוג לנכונות ה-$\boldsymbol{C\ calling\ conventions}$ בעצמם? איזה חלק בשרשרת משמש אותם לכך?

- מפתחי ה-$kernel$ היו צריכים לדאוג שבזמן המעבר לפונקציה $system\_call$ יתקיימו $C\ calling\ conventions$ מאחר וסוג הפסיקה המבוקשת נמצא ברגיסטר. לשם כך משמשת אותם המעטפת $open$ אשר מקבלת את אופי פסיקת התוכנה המבוקש מהמשתמש , מסדרת את הפרמטרים לפי מוסכמות הקריאה ובעזרת הפקודה $int\ 0x80$ מבקשת פסיקת תוכנה ממערכת ההפעלה.

ג. האם לא היה פשוט יותר לתת לקומפיילר לעשות זאת? מדוע בעצם צריך לעשות זאת באופן ידני? (לומר, הסבירו מדוע יש צורך לגרום לחריגה, ומדוע יש צורך ברכיב התוכנה על מנת לסדר את המחסנית).

- פעולות רבות, כמו פעולת $open$ מצריכות גישה לחומרה חיצונית (במקרה זה לדיסק) ומערכת הפעלה אינה מאפשרת למשתמש לבצע פעולות אלו מסיבות רבות (כדי לא לפגוע בניהול המשאבים, צורך בגישה לחומרה וכו'). הפתרון הוא לאפשר למשתמש לבצע פסיקת תוכנה, שעם קבלתה קוטעת את ריצת התוכנית לטובת ביצוע הפעולה על ידי מערכת ההפעלה. רכיב התוכנה ($wrapper$) הכרחי משום שפונקציית ה $kernel$, $system\_call$ הינה ספציפית למערכת ההפעלה ומערכות הפעלה שונות יכולות לממש אותה, כמו גם את העברת הפרמטרים, בצורות שונות. הקומפיילר אינו ספציפי למערכת ההפעלה ולכן יש צורך לסדר את הפרמטרים בצורה בה תוכל להתבצע הפסיקה עם המעבר לרמת ה- $kernel$.

שאלה 2

א. הסטודנט גל התבקש לכתוב $system\ call$ שמקבל מצביע למערך ומבצע איתו פעולה מסויימת. להלן מימוש (חלקי) ב-$kernel$ של ה-$system\ call$.

```
int gal_system_call (cons tint *arr, int length) {
        int x;
        if (length <=0)
                return 0;
        else {
                /*do some stuff*/
                return gal_system_call (arr+1, length-1);
        }
}
```

גל שם לב שקריאת המערכת שלו עובדת עבור מערכים קצרים, אבל החל ממערכים באורך מסויים $x = liength$ ה-$kernel$ קורס לעיתים קרובות בקריאה $system\ call$ שלו. מהו המספר הקרוב ביותר לx הנ"ל?

- המספר הקרוב ביותר לx הינו $0.25K$.

על פי הנלמד, לכל תהליך ישנו בלוק זיכרון בגודל $8KB$ ב-$kernel$ אשר משמשת אותו. רוב קטע הבלוק משמש לניהול מחסנית אשר מתחילה מהכתובת הגבוהה ביותר בו וגדלה לכיוון הכתובות הנמוכות. בחלקו ההתחלתי של הבלוק נמצא מתאר התהליך ולכן המחסנית אינה יכולה לגדול מעבר ל 7200 בתים, כדי לא לדרוס אותו.

נשים לב כי לכל פונקציה ב$kernel$ מוקצית מחסנית בבלוק הזיכרון (כלומר משתמשת במחסנית של התהליך) אשר נראית כך:

פרמטרים לפונקצייה
כתובת חזרה
ebp
esi
edi
ebx
משתנים מקומיים

ה-$system\ call$ שכתב גל מבצעת קריאות רקורסיביות ועבור כל קריאה משורשרת על מחסנית של התהליך מחסנית חדשה עבור הקריאה הנוכחית של הפונקציה. בארכיטקטורת המעבד עליה עבד כל מילה הינה באורך 4 בתים ומאחר וכל ריצת פונקציה מייצרת שימוש בכ- $8x4 = 32$ בתים נוספים, עומק רקורסיה של $0.25K$ קריאות יגרום לחריגת המחסנית אל תוך מתאר התהליך ודריסתו. כלומר, כאשר $x = 0.25k$ יגרמו שינויים במתאר התהליך ובעקבות זאת טעויות וקריסות של ה-$kernel$.

ב. אילו משיטות התזמון עלולות לגרום לאפקט השיירה ($convoy\ effect$) ?

- כל מדיניות זימון ללא הפקעה עלולה לגרום לאפקט השיירה. תשובה E.

אפקט השיירה הינו מצב בו המעבד (או המעבדים) מבצע משימה גדולה וממושכת, כאשר משימות פשוטות וקצרות ממתינות לסיום משימה זו כדי להתבצע. כל מדיניות זימון ללא הפקע ($non-preemtiv$) עלולה ליצור מצב בו כל כוח העיבוד נמצא בידי משימה גדולה במשך זמן רב וכך לגרום לאפקט השיירה. בפרט שיטת התזמון $FCFS$ אשר מבצעת משימות לפי סדר קבלתן ללא הבחנה בגודלן או במשך הריצה שלהן.

שאלה 3

כזכור ב $super\ computers$ עם תזמון $EASY$ על המשתמש לספק הערכה לזמן חישוב ה job שלו. חוקרי מערכות הפעלה גילו כי אם המשתמש משקר ומספק הערכה כפולה לזמן זה אז מתקבלים ביצועים טובים יותר במשמעות של $wait\ time$.

נניח כי המשתמש העריך את הזמן במדויק ולאחר קביעת התזמון, נראת המערכת כך:

p2	p2	p2	p2			p3	p3	p3	p3	p3	p5	p5	p5	p6	p6
p1	p1	p1	p1	p1	p1	p3	p3	p3	p3	p3	p4	p4	p4	p4	

כאשר סדר הגעת המשימות הוא $p1, p2, p3, p4, p5, p6$.

א. חשבו את זמן ההמתנה הממוצע של המערכת (לאחר ביצוע $backfilling$ ל $p6$).

- זמן ההמתנה הממוצע מוגדר להיות $\frac{1}{N}\sum_{i=1}^{N} w_i$ כאשר, N הוא מספר התהליכים ו w_i הוא זמן ההמתנה של התהליך ה i.

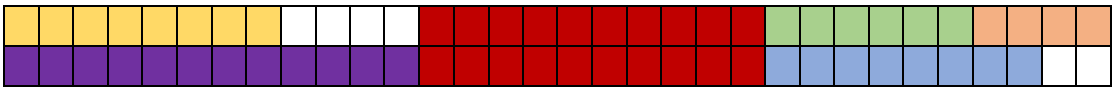
נשים לב כי תחילה יתבצע $backfilling$ ל $p6$ ומערכת תיראה כך:

p2	p2	p2	p2	p6	p6	p3	p3	p3	p3	p3	p5	p5	p5		
p1	p1	p1	p1	p1	p1	p3	p3	p3	p3	p3	p4	p4	p4	p4	

כאשר זמני ההמתנה הם $w_1 = 0, w_2 = 0, w_3 = 6, w_4 = 11, w_5 = 11, w_6 = 4$. לכן זמן ההמתנה הממוצע יהיה $\frac{(4+6+11+11)}{6} = 5.333$.

ב. נניח עתה כי המשתמש משקר ומכפיל ב-2 את הזמן המוערך לביצוע כל job. ציירו מחדש את מצב המערכת החדש לפני תחילת הריצה.

- לפני תחילת הריצה תיראה המערכת כך :



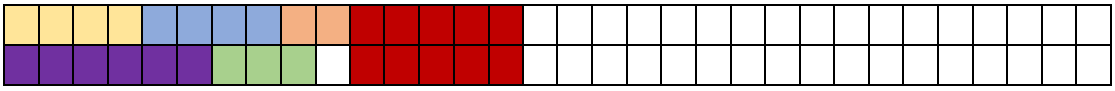
ג. כעט בצעו סימולציה של מהלך החישוב, סמנו מתי מסתיים כל job בפועל.

1. האם ניתן לבצע $backfilling$?

- ניתן לבצע $backfilling$ מספר פעמים במהלך ריצת התהליכים.

2. ציירו את מהלך הריצה בפועל שהתרחש במערכת וחשבו את זמן ההמתנה הממוצע החדש. האם התקבל שיפור בביצועים?

- תיאור זמן הריצה בפועל:



זמני ההמתנה החדשים הם $w_1 = 0, w_2 = 0, w_3 = 10, w_4 = 4, w_5 = 6, w_6 = 8$. לכן זמן ההמתנה הממוצע גדל ל : $\frac{(4+6+8+10)}{6} = 4.6667$.
כפי שניתן לראות, אכן התקבל שיפור בביצועים.

3. הסבירו את התופעה שהחוקרים ראו, הסבירו האם מתקיים שינוי מסוים במדיניות הזימון, אילו תהליכים קודמו והאם ה $fairness$ נשמר.

- מקורה של התופעה שהחוקרים ראו היא בעובדה כי עם סיומו של job בזמן הממשי בו הוא רץ נעשה $backfilling$ במידת האפשר, אלא שמאחר וזמן הריצה הממשי מסתיים במחצית מהזמן המשוערך, נותרים חורים גדולים יותר והמצב מאפשר למשימות קצרות (אשר הזמן המשוערך שלהן גדל באופן לא משמעותי) להתקדם על חשבון משימות ארוכות (אשר הזמן המשוערך שלהן גדל באופן משמעותי). כלומר, הכפלת זמן הריצה המשוערך של כל המשימות מאפשר יותר חופש בזמן ביצוע $backfilling$, ופוטנציאלית ניתן לשפר את זמן ההמתנה הממוצע. במידה והכפלת זמני הריצה גרמה לצמצום זמן ההמתנה הממוצע ניתן להסיק כי משימה קטנה שלא הייתה מקודמת במצב רגיל, התקדמה על חשבון משימה גדולה יותר, (שבפועל נדחפה אחורה) ולכן $fairness$ המערכת אינו נשמר.