שאלה 1

א. כאשר תהליך שרץ ב-$user\ mode$ מבצע קוד $int\ 0x80$, כיצד מוצאים את תחילת מחסנית הגרעין של אותו תהליך? מתי ובאיזה פונקציה המצביע למחסנית מתעדכן באותו מקום.

- על מנת למצוא את תחילת מחסנית הגרעין של תהליך המבצע פסיקת תוכנה $int\ 0x80$, משתמשים בשדה $esp0$ שבמבנה TSS של המעבד עליו רץ התהליך.

ב. ידוע שקיים מצביע לתחילת מחסנית הגרעין של התהליך הנוכחי שאחד השדות ב-$thread$, מדוע אם כן לא היה ניתן למצוא את תחילת מחסנית הגרעין של התהליך במעבר לקרנל באמצעות הפקודה $current \rightarrow thread.esp0$

- נשים לב כי המקרו $current$ משתמש ברגיסטר esp כדי למצוא את תחילת המחסנית של התהליך ברמת ה-$kernel$, וכן בזמן המעבר מרמת ה-$user$ לרמת ה-$kernel$ השדה esp מצביע למחסנית ברמת ה-$user$ ולכן השימוש במקרו $current$ אינו אפשרי עדיין. במעבר מרמת $user$ לרמת $kernel$, תחילה, נשמרים על המחסנית ב-$kernel$ ערכי הרגיסטרים הדרושים לחזרה להרצת התהליך ברמת ה-$user$, ורק לאחר מכן מתעדכנים הערכים החדשים המאפשרים שימוש במקרו $current$.

ג. במאקרו $switch\_to\_$ מופיעה התווית "1:". מדוע הכרחי לדרוס את $prev \rightarrow thread.eip$

- הכרחי לדרוס את $prev \rightarrow thread.eip$ משום לפני החזרה לרמת ה-$user$ יש (לרוב) לשחזר את הרגיסטרים $esi, edi\ ebp$ של התהליך המזומן למצב בו הם היה לפני מסירת המעבד בפעם האחרונה (פעולה זו הכרחית לכל תהליך שאינו מקבל את המעבד בפעם הראשונה מאז היווצרו). לכן, יש לדרוס את ערכו של $prev \rightarrow thread.eip$ ולהחליפו בערך של תווית "1", כדי לבצע את קטע הקוד לשחזור הרגיסטרים מהמחסנית של ה-$kernel$.

שאלה 2

א. האם ייתכן מצב בו תהליך אינטראקטיבי A יהיה באותה עדיפות דינאמית כמו תהליך חישובי B. אם כן, תארו מצב כזה, אחרת נמקו מדוע.

- מצב בו תהליך A יהיה באותה עדיפות כמו תהליך חישוב B ייתכן. נציג דוגמא למקרה כזה.

יהי A תהליך אשר קיבל $nice$ של 0 וה-$bonus$ שלו חושב להיות 5. על פי הגדרה מתקיים כי $delta(A) = 5 \cdot \frac{nice(A)}{20} + 2 = 2$, לכן העדיפות הדינאמית של התהליך הינה $120 - nice(A) - bonus(A) = \mathbf{115}$ וגם נשים לב שמתקיים כי $115 \leq 120 - 2 = 118$ ולכן התהליך אכן אינטראקטיבי.

יהי תהליך B אשר קיבל $nice$ של -10 וה-$bonus$ שלו חושב להיות -5. על פי הגדרה מתקיים כי $delta(B) = 5 \cdot \frac{nice(B)}{20} + 2 = -\frac{1}{2}$, לכן העדיפות הדינאמית של התהליך הינה $120 - nice(B) - bonus(B) = \mathbf{115}$ וגם נשים לב שמתקיים כי $115 > 120 - 10 + \frac{1}{2} = 110.5$ ולכן התהליך אכן חישובי.

מתקיים כי התהליך A הינו אינטראקטיבי והתהליך B הינו חישובי ושניהם בעלי עדיפות דינאמית של 115, עבור ערכי ה-$bonus$ וערכי ה-$nice$ הנתונים.

ב. **בתרגול מצוינות 4 סיבות בגינן עשוי תהליך להגיע לפונקציה $schedule$. בחנו את הקוד של sys_sched_yield ו-$interruptible\_sleep\_on$. בכל אחת מהפונקציות ישנה קריאה ישירה לפונקציה $schedule$. אם היינו משנים קריאה זו ל-$set\_tsk\_need\_resched(current)$, כיצד היה משפיע שינוי זה אם היה מתבצע רק ב-sys_sched_yield. כיצד היה משפיע שינוי זה אם היה מתבצע רק ב-$interruptible\_sleep\_on$. נמקו.**

- עבור שינוי הקריאה הישירה ל-$schedule$ בקריאה $set\_tsk\_need\_resched(current)$, בפונקציה sys_sched_yield, התהליך יעבר בצורה תקינה למקומו החדש ב-$runqueue$. מאחר והפונקציה אינה משנה את מצב התהליך ולא מכניסה אותו לרשימת המתנה כלשהי, המערכת תמשיך להתנהל בצורה תקינה. השינוי היחיד הוא שהתהליך ימשיך לרוץ פרק זמן נוסף כלשהו לאחר מסירת המעבד ועד קריאה לפונקציה $schedule$.

- עבור שינוי הקריאה הישירה ל-$schedule$ בקריאה $set\_tsk\_need\_resched(current)$, בפונקציה $interruptible\_sleep\_on$, התהליך לא היה מוצא מהקשר משום שהמקרו $SLEEP\_ON\_TAIL$ היה מתבצע לפני קריאה ל $schedule$ וכתוצאה מכך התהליך היה מוצא מתור ההמתנה מיד לאחר הכנסתו (כלומר, לא מוכנס בפועל). לכן, לאחר ביצוע $deactivate\_task$ ב$schedule$ אין אפשרות להחזיר את התהליך לריצה. לכן מערכת ההפעלה אינה תמשיך לעבוד בצורה תקינה.

ג. **נתון קטע הקוד הבא הלקוח מהפונקציה $schedule$:**

```
perv = current;
rq = this_rq();
…
spin_lock_irq(&rq->lock);
switch(prev->state) {
    case TASK_INTERRUPTIBLE:
            if (signal_pending(prev)) {
                    prev->state = TASK_RUNNING;
                    break;
            }
    default:
            deactivate_task(prev,rq);
    case TASK_RUNNING:
    ;
    }
```

מדוע ה- $case$ הראשון הכרחי?

- ה- $case$ הראשון הכרחי משום שייתכן מצב בו התהליך אשר יצא להמתנה במצב $TASK\_INTERRUPTIBLE$ קיבל בפועל $signal$ שטרם טופל ולמעשה יכול לחזור לרוץ, לכן יש להחזיר אותו למצב $TASK\_RUNNING$ ואין צורך להוציאו מה-$runqueue$.

ד. לצורך שאלה זו נתבונן בחישוב $sleep\_avg$ של תהליך בגרעין

1. נניח שקיים תהליך אינטראקטיבי A בעדיפות סטטית 100 ובעל $sleep\_avg = MAX\_SLEEP\_AVG$. בנקודת זמן $t = 0$ התהליך החל לבצע משימה חישובית אורכה. מהו הזמן המקסימלי (במילישניות) שהתהליך ירוץ לפני שיעבור ל-$expired$, בהנחה שהוא התהליך היחיד ב-$runqueue$.

- נשים לב כי מאחר התהליך הוא היחיד ב-$runqueue$ הוא ימשיך לרוץ בלי הפסקה כל עוד הוא יחשב בעני המערכת כתהליך אינטראקטיבי, מאחר ותהליך אינטראקטיבי אינו מועבר ל-$expired$ אלא מוחזר לסוף הרשימה ב-$active$ (במקרה זה התהליך יבחר מיד לרוץ שוב). כפי שראינו במסגרת הקורס, כל עוד ה-$bonus$ של התהליך יהיה גדול מה-$delta$ שלו, התהליך יחשב אינטראקטיבי.

עבור התהליך A מתקיים כי $delta(A) = -3$ תמיד וכן ה-$time\_slice$ אותו הוא מקבל הינו 300 מילישניות (אלו תלויים בעדיפות הסטטית בלבד). התהליך יהפוך להיות חישובי לאחר שה-$bonus$ שלו יהפוך להיות קטן יותר מ- -3, כלומר, כאשר ה $sleep\_avg$ שלו יהיה קטן מ- 0.4. מאחר ובכל קוונטום ה $sleep\_avg$ קטן ב 0.3, יעברו שישה קוונטומים עד אשר התהליך יהפוך לחישובי ויעבור ל-$expired$.

לכן הזמן המקסימלי שהתהליך ירוץ לפני שיעבור ל-$expired$ הוא $300 * 6 = \mathbf{1800}\ ms$.

2. כיצד תשובתכם לסעיף הקודם הייתה משתנה אם נתון שקיים תהליך B ב-$expired$, והחל מהרגע $t = 0$ שתואר קודם, נותרו 1000 מילישניות עד כלוי ה $expired\_timestamp$? (כלומר כד שהמאקרו $EXPIRED\_STARVING$ מתחיל להחזיר $true$).

- הבדיקה להרעבה מתבצעת עם סיום אפוק (במקרה זה עם סיום הקוונטום של תהליך A), ולכן רק לאחר ארבעה קוונטומים יתגלה כי ישנה הרעבה במערכת והתהליך A, למרות היותו אינטרקטיבי יעבור ל-$expired$.

לכן, בהינתן המידע הנוסף, הזמן המקסימלי שהתהליך ירוץ לפני שיעבור ל-$expired$ הוא $300 * 4 = \mathbf{1200}\ ms$.

ה. מהו פרק הזמן המקסימלי שיכול לעבור מהרגע שהודלק הדגל $need\_resched$ בתהליך שרץ ועד שהוא מגלה את הצורך בהחלפת הקשר (במערכת מרובת מעבדים)?

- הדגל $need\_resched$ נבדק בכל פעם שעתיד להתבצע מעבר מרמת ה-$kernel$ לרמת ה-$user$. נשים לב כי בכל פסיקת שעון ישנו מעבר לרמת ה$kernel$ (משום שהטיפול בפסיקה מתבצע ברמה זו) ולאחריו ישנה חזרה לרמת ה-$user$ במהלכה נבדק הדגל $need\_resched$ ובמידת הצורך תתבצע החלפת הקשר בזמן זה.

לכן, מרגע שהודלק הדגל $need\_resched$ ועד גילוי הצורך בהחלפת הקשר יעבור לכל היותר זמן של פסיקת שעון אחת.

ו. האם אפשר לתת דוגמא בה תהליך יכול להתחיל לרוץ קצת אחרי פסיקת שעון ולעזוב את ה-CPU קצת לפני פסיקת השעון הבאה (פחות מ-$tick$)? אם הדבר בלתי אפשרי הסבירו מדוע, אחרת תנו דוגמא מפורטת.

- נציג את הדוגמא הבאה:

במערכת רצים שני תהליכים, A ו-B אשר שניהם באותה עדיפות. התהליך B מתחיל לרוץ ועם תחילת ה-$time\_slice$ שלו מופעלת הפקודה $sched\_yield$. כפי שראינו במסגרת הקורס, הפונקציה sys_sched_yield קוראת בצורה ישירה לפונקציה $schedule$, ולכן, במצב זה מתבצעת החלפת הקשר לתהליך A שמתחיל את ריצתו קצת אחרי פסיקת שעון, כנדרש.

הפקודה הראשונה שמבוצעת בתהליך A הינה $sched\_yield$ במהלכה נקראת הפונקציה sys_sched_yield ונעשית קריאה מפורשת ל $schedule$ ושוב נעשית החלפת הקשר חזרה לתהליך B עוד לפני שמתקבלת פסיקת השעון הבאה, כנדרש.

לכן, התהליך A מתחיל את ריצתו קצת אחרי פסיקת שעון, ועוזב את ה-CPU קצת לפני פסיקת השעון הבאה.

שאלה 3

בהרצאות הוכח כי במערכת בעלת מעבד יחיד אלגוריתם הזימון SJF נותן את התוצאות האופטימליות עבור מדד זמן ההמתנה הממוצע.

בסעיפים הבאים נבחן מערכות מרובות מעבדים ומדדים שונים. בכל מקרה בו עליכם להפריך טענה שרטטו את הדוגמה הנגדית שלעם בטבלאות זמן.

א. נגדיר אלגוריתם $LJBF$ ($Longest\ Job\ Backfilled\ First$) שעובד בדיוק כמו $SJBF$ רק שבכל החלטה למלא מקום ריק בתהליך ה-$backfilling$ בוחרים תמיד את המשימה הארוכה ביותר שיכולה לאכלס את המקום הריק.

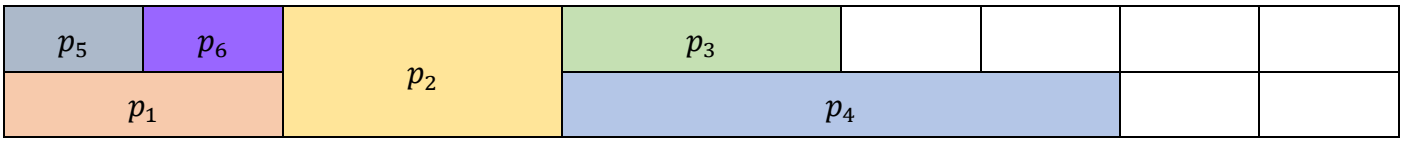
1. האם $LJBF$ תמיד טוב יותר מ-$SJBF$ לפי מדד זמן ההמתנה הממוצע, או שקיימים מקרים בהם $SJBF$ נותן תוצאות טובות יותר? אם כן הוכיחו, אחרת הראו דוגמא נגדית.

- הטענה אינה נכונה, נראה דוגמא נגדית:

מצב לפני $backfilling$ תחת ההנחה כי סדר הגעת המשימות הוא p_1, p_2 ולאחר מכן הגיעו p_3, p_4, p_5, p_6 באותו זמן.

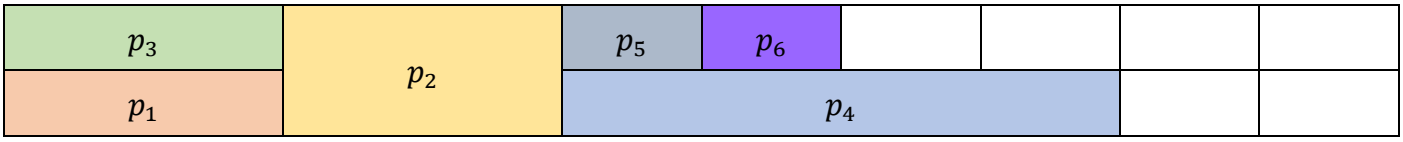
<table>
<tr><td></td><td></td><td rowspan="2">p_2</td><td>p_3</td><td>p_5</td><td>p_6</td><td></td><td></td></tr>
<tr><td colspan="2">p_1</td><td colspan="3">p_4</td><td></td><td></td></tr>
</table>

עבור $SJBF$:



מתקיים כי זמן ההמתנה הממוצע הינו $\frac{0+0+1+2+4+4}{6} = 1.8333$

עבור $LIBF$:



מתקיים כי זמן ההמתנה הממוצע הינו $\frac{0+0+2+4+4+5}{6} = 2.5$

לכן, מתקיים כי עבור מקרה זה $SJBF$ הוא בעל ביצועים טובים יותר מבחינת זמן המתנה ממצוע.

2. האם $SJBF$ תמיד טוב יותר מ-$LJBF$ לפי מדד זמן התגובה הממוצע, או שקיימים מקרים בהם $LJBF$ נותן תוצאות טובות יותר? אם כן הוכיחו, אחרת הראו דוגמא נגדית.

- הטענה אינה נכונה, נראה דוגמא נגדית:

מצב לפני $backfilling$ תחת ההנחה כי $p_4, p_5,$ הגיעו באותו זמן.

			2	2	4	4	5	5	5							
1	1	1	2	2	3	3	3	3	6	6	6	6				

עבור SJBF:

4	4		2	2	5	5	5	6	6	6	6					
1	1	1	2	2	3	3	3	3								

נקבל : זמן התגובה הממוצע : $\frac{2+3+5+8+9+12}{6}$=6.5

עבור LJBF:

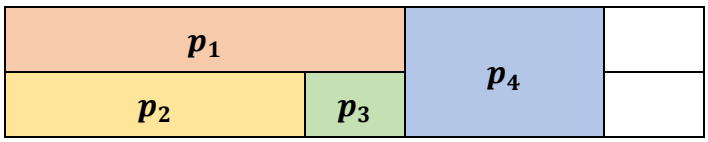
5	5	5	2	2	4	4	6	6	6	6						
1	1	1	2	2	3	3	3	3								

נקבל : זמן התגובה הממוצע :$\frac{3+3+5+7+9+11}{6}$=6.333

ב. עבור מדד הניצולת ($Utilization$), תנו דוגמא לקבוצה של תהליכים עבורם $EASY$ נותן תוצאה טובה יותר מ-SJF, או שהוכיחו שהדבר אינו אפשרי.

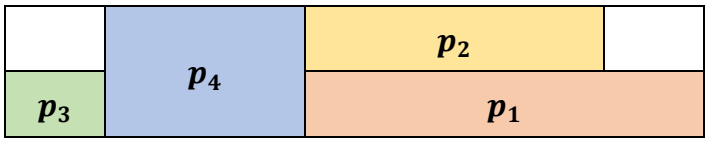
- נראה את הדוגמא הבאה:

<u>לפי $EASY$</u> :



מתקיים כי הנצילות היא 100%.

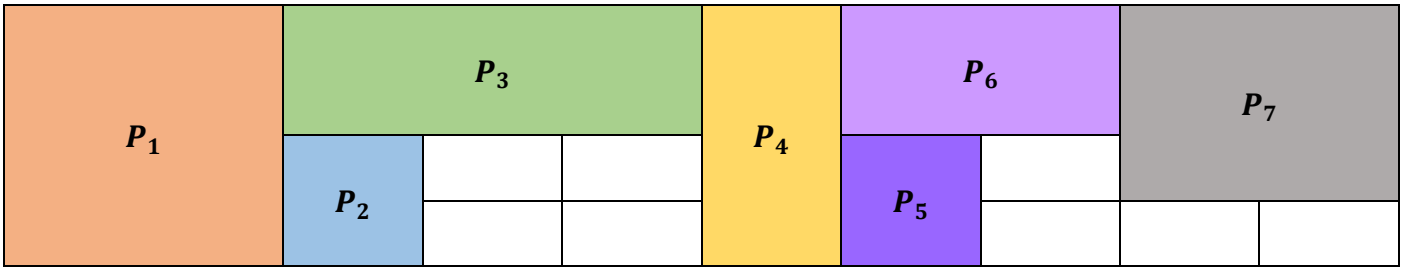
<u>לפי SJF</u> :



מתקיים כי הנצילות היא $\frac{12}{14} = 85.71\%$.

לכן, מתקיים כי עבור מקרה זה $EASY$ הוא בעל ביצועים טובים יותר מבחינת ניצולת.

ג. ענו על השאלות הבאות ביחס למבחן המדדים השונים על התזמון הנתון:



1. מהו זמן ההמתנה הממוצע?

- $\frac{0+2+2+5+6+6+8}{7} = 4.1428$

2. מהו זמן התגובה הממוצע?

- $\frac{2+3+5+6+7+8+10}{7} = 5.8571$

3. מהו זמן ההאטה ($slowdown$) הממוצע?

- $\frac{\frac{2}{2}+\frac{3}{1}+\frac{5}{3}+\frac{6}{1}+\frac{7}{1}+\frac{8}{2}+\frac{10}{2}}{7} = 27.666667$

4. מהי הניצולת?

- $\frac{4+10-8}{4\cdot 10} = \frac{32}{40} = 80\%$

5. מהי התפוקה $(throughput)$?

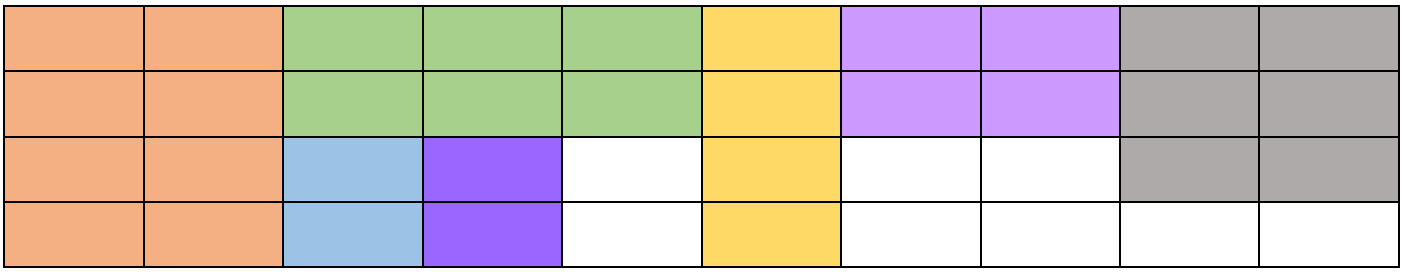
- $\frac{7\ jobs}{10\ minutes} = 0.7\frac{jobs}{minute}$

6. מהו ה-$makespan$?

- $10\ sec$

7. אלו מהמדדים הנ"ל היו מספקים תוצאות טובות יותר אם היינו מפעילים על התזמון הנתון את $SJBF$?

- עבור $SJBF$ תמונת המערכת הינה:

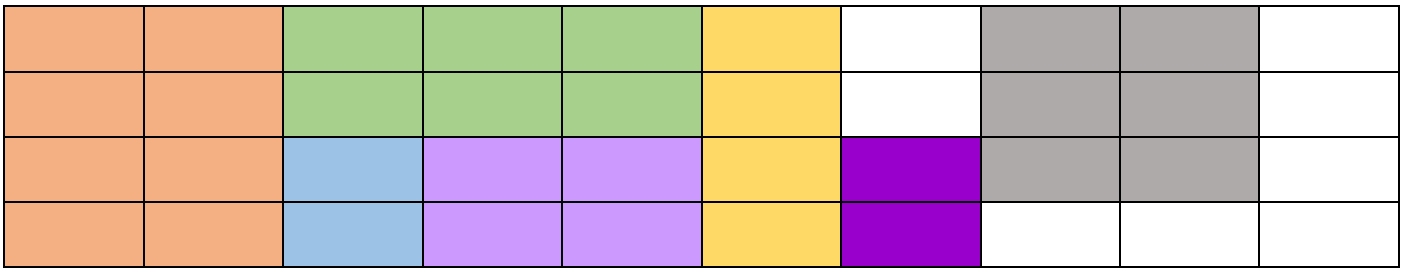


ישנו שיפור במדדים הבאים :

- זמן המתנה ממוצע: $\frac{0+2+2+3+5+6+7}{7} = 3.5714$
- זמן תגובה ממוצע: $\frac{2+3+5+5+6+7+9}{7} = 5.2857$
- זמן האטה ממוצע: $\frac{\frac{2}{2}+\frac{3}{1}+\frac{5}{2}+\frac{5}{3}+\frac{6}{1}+\frac{7}{1}+\frac{9}{2}}{7} = 25.666667$
- ניצולת : $\frac{4\cdot 9-4}{4\cdot 9} = 88.8\%$
- תפוקה : $\frac{7\ jobs}{9\ minutes} = 0.777\frac{jobs}{minute}$
- $Makespan$: הוא 9.

8. אלו מהמדדים הנ"ל היו מספקים תוצאות טובות יותר אם היינו מפעילים על התזמון הנתון את $LJBF$?

- עבור $LJBF$ תמונת המערכת הינה:



ישנו שיפור במדדים הבאים :

- זמן המתנה ממוצע: $\frac{0+2+2+3+5+6+8}{7} = 3.7143$

- זמן תגובה ממוצע: $\frac{2+3+4+5+6+8+10}{7} = 5.4185$

- זמן האטה ממוצע: $\frac{\frac{2}{2}+\frac{3}{1}+\frac{4}{1}+\frac{5}{3}+\frac{6}{1}+\frac{8}{2}+\frac{10}{2}}{7} = 24.666667$