Homework 2 Dry

Due Date: 11/12/2018 23:30

Teaching assistant in charge:

- Andre Kassis

Important: the Q&A for the exercise will take place at a public forum Piazza only. Critical updates about the HW will be published in pinned notes in the piazza forum. These notes are mandatory, and it is your responsibility to be updated. A number of guidelines to use the forum:

- Read previous Q&A carefully before asking the question; repeated questions will probably go without answers
- Be polite, remember that course staff does this as a service for the students
- You're not allowed to post any kind of solution and/or source code in the forum as a hint for other students; In case you feel that you have to discuss such a matter, please come to the reception hour
- When posting questions regarding hw2, put them in the hw2 folder

Only the TA in charge can authorize postponements. In case you need a postponement, please fill out the following form: https://goo.gl/forms/D5nxMxf9Uvgej1SL2

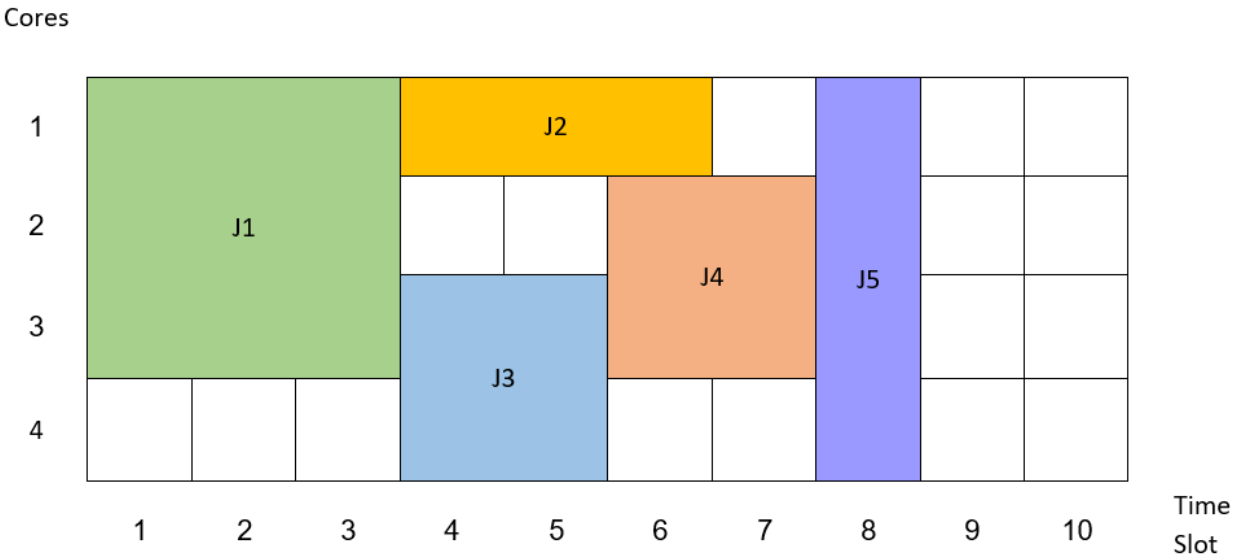
Dry part submission instructions:

1. Please submit the dry part to the electronic submission of the dry part on the course website.
2. The dry part submission must contain a single dry.pdf file containing the following:
    a. The first page should contain the details about the submitters - Name, ID number and email address.
    b. Your answers to the dry part questions.
3. Only typed submissions will be accepted. Scanned handwritten submissions will not be accepted. Only PDF format will be accepted.
4. You do not need to submit anything in the course cell.
5. When you submit, **retain your confirmation code and a copy of the PDF**, in case of technical failure. It is **the only valid proof** of your submission.

יש לנמק כל תשובה, תשובות ללא נימוק לא יתקבלו.

שאלה 1 - זימון תהליכים (50 נק')

שאלה זו עוסקת בנושא batch-scheduling כפי שנלמד בהרצאות. שימו לב שבכל טבלאות התזמון הבאות המספר המייצג time-slot כלשהו מייצג את הזמן שבו החלון הנתון **מסתיים** (כלומר - אם time slot = 3 אזי החלון **מתחיל** ב-t=2 ומסתיים ב- t=3).



1. (18 נק') בהינתן התזמון הנתון למעלה, חשבו את המדדים הבאים (הראו את דרך החישוב). הניחו שכל התהליכים בנתונים למעלה הגיעו באותו הזמן (t=0):
   1. (6 נק') מדד זמן ההמתנה הממוצע (average wait-time):

$$\frac{(0+3+3+5+7)}{5} = \frac{18}{5}$$

   2. (6 נק') מדד זמן התגובה הממוצע (average response-time):

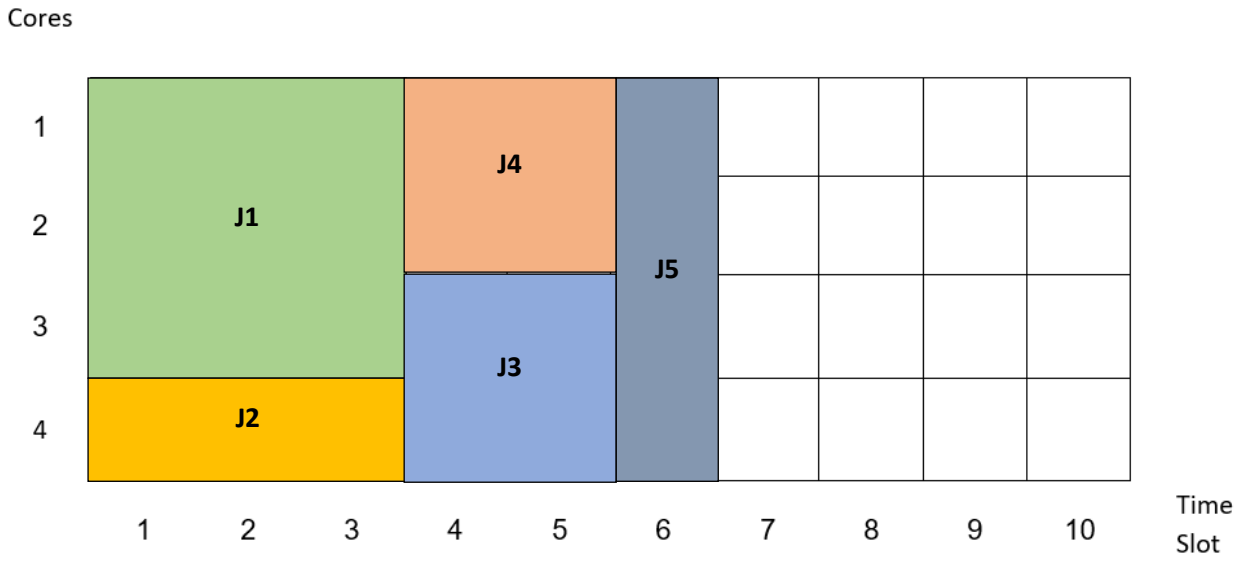
$$\frac{(3+5+6+7+8)}{5} = \frac{29}{5}$$

   3. (6 נק') מדד הניצולת (utilization):

$$\frac{(4*10-16)}{4*10} = \frac{24}{40} = 60\%$$

2. (8 נק') בהנחה שכל התהליכים למעלה מגיעים בזמן 0. איזו מדיניות זימון יכולה לשפר את הניצולת כפי שחושבה בסעיף הקודם? **נמקו** (ראו מצורפת טבלת זימון ריקה על מנת להמחיש את הזימון החדש לפי האלגוריתם, **עליכם למלא את הטבלה בהתאם**)

ע"י שימוש באלגוריתם EASY אשר משלב גם את אלגוריתם FCFS (first come first serve) וגם BACKFILLING נקבל שניתן למלא את הריבועים של המעבדים החל מזמן 0 עד זמן 6 באופן מלא (ראה ציור למטה).
לאחר שימוש באלגוריתם הנ"ל נקבל כי המעבד עובד רק עד 7 ובו הניצולת שלו היא 100%



3. (8 נק') מנו 2 יתרונות ו-2 חסרונות של מדיניות זימון מסוג batch-scheduling על פני מדיניות Round-robin. נמקו.

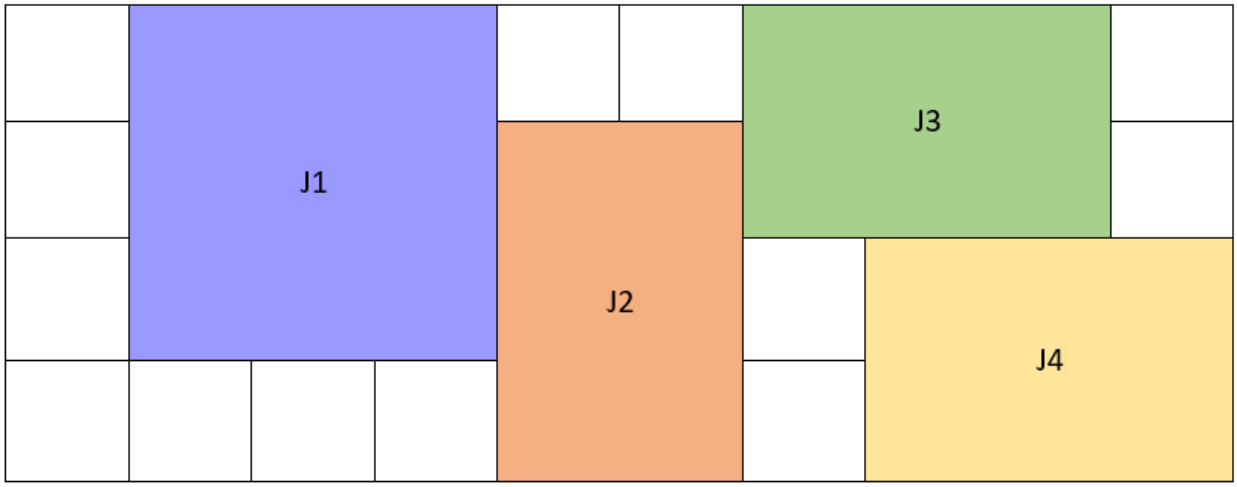
יתרונות:

1. בהנחה שמדובר במערכת בעלת מעבד יחיד וכל התהליכים מגיעים ביחד אזי קיים אלגוריתם מסוג batch-scheduling שזמן ההמתנה הממוצע שלו קטן או שווה לזמן ההמתנה הממוצע של אלגוריתם Round-robin ולפיכך זמן התגובה הממוצע של תהליך במדיניות זו הוא קצר יותר או שווה לזמן התגובה במדיניות Round-robin.
2. Round-robin דורש פסיקות שעון (כל מספר מילי-שניות) ותוכנית יעודית אשר תפקידה לנהל את התהליכים. בנוסף לכך , ב-Round-robin תהליך מופרע באמצע הריצה לעומת זאת ב-batch-scheduling התהליך רץ עד סופו ללא הפרעה

חסרונות:

1. יוצר הרעבה (STARVATION) של תהליכים אחרים כיוון שנותן לתהליך לרוץ עד סיומו
2. ב-batch-scheduling אין התייחסות לעדיפות התהליך ולכן יתכן מצב שיתבצע הרעבה של תהליך בעל עדיפות גבוהה.

נתונה מדיניות זימון חדשה BSAF (Biggest Surface-Area First) לפיה בהינתן 2 תהליכים זה שנבחר לרוץ קודם הוא זה **שהשטח שלו הגדול יותר** (שטח = זמן * מספר-מעבדים). שימו לב שמדיניות זימון זו תומכת ב-backfilling - כלומר שאם התהליך העדיף ביותר לפי BSAF לא יכול להיות מתוזמן בחלון כלשהו, יתוזמן התהליך הטוב ביותר שמתאים לחלון אחריו (זה שמתאים לחלון הפנוי והוא הטוב ביותר לפי BSAF). בהינתן 2 תהליכים בעלי אותו שטח נבחר בזה שיש לו את זמן הריצה הקצר ביותר. בהינתן 2 תהליכים בעלי שטח זהה וזמן ריצה זהה, נבחר בזה עם ה-ID (מספר) הנמוך יותר.
למשל, בהינתן 4 התהליכים הבאים (האיור להמחשה של מימדי התהליכים בלבד):

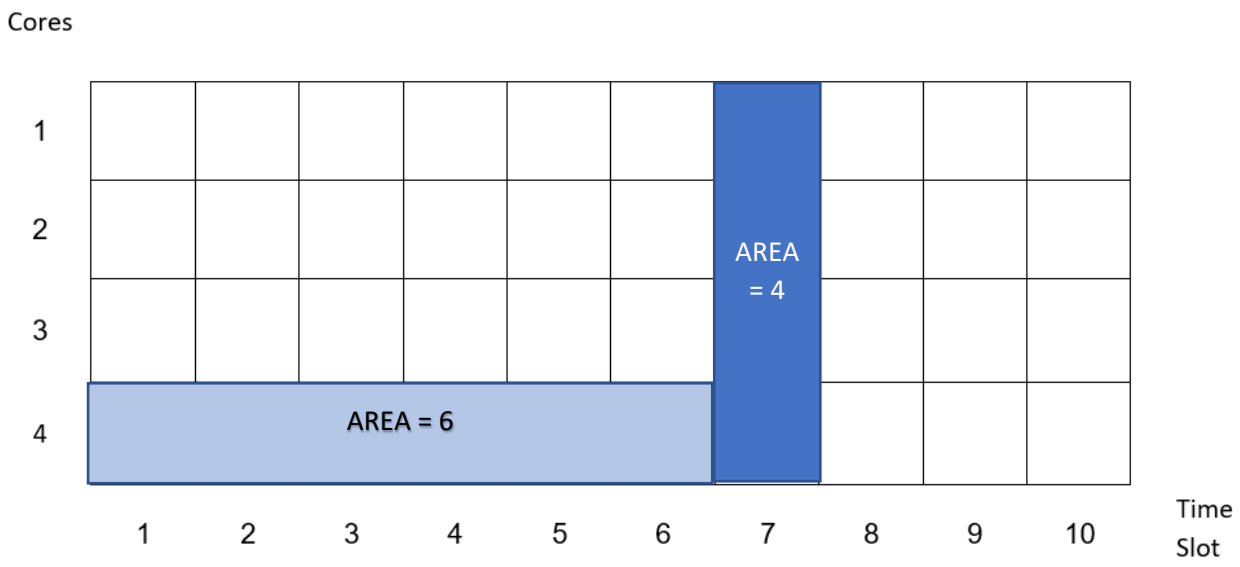


האלגוריתם BSAF יתעדף אותם בסדר הבא:

J1 (3*3=9 surface area)
J2 (3*2=6, shorter runtime than J3 and J4)
J3 (2*3=6, same as J4, but lower ID)
J4

4. (8 נק') האם BSAF סובל מ-convoy effect? נמקו (עליכם להמחיש בעזרת הטבלה הנתונה).

כן , האלגוריתם מתעדף תהליכים עם שטח גדול יותר ולאחר מכן מתעדף אותם לפי זמן ריצה ומספר סידורי. ולכן בהינתן תהליך בעל זמן ריצה ארוך מאוד (בעל שטח מקסימלי מבין כל שאר התהליכים) אשר משתמש במספר מעבדים כך שלא ניתן יהיה לקדם תהליך אחר באמצעות BACKFILLING למעבדים שבהם התהליך הארוך לא משתמש, נקבל את אפקט השיירה.

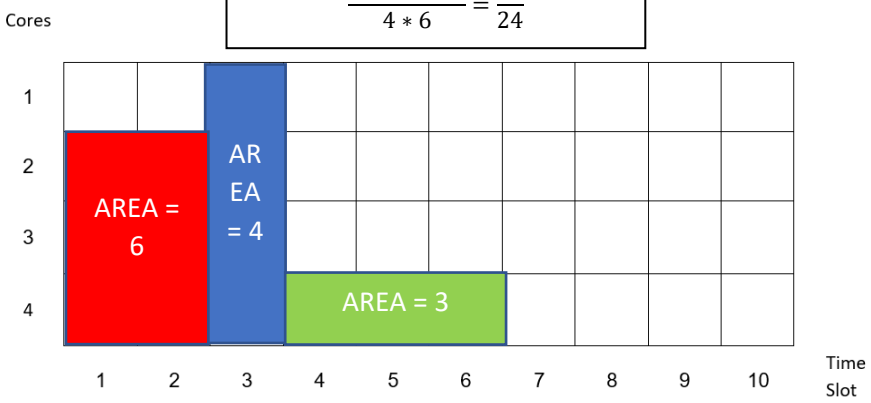


.5 (8 נק') האם אלגוריתם הזימון BSAF הינו אופטימלי מבחינת מדד הניצולת?
הוכיחו או הפריכו (עליכם להמחיש בעזרת הטבלה הנתונה)

לא , דוגמא נגדית:

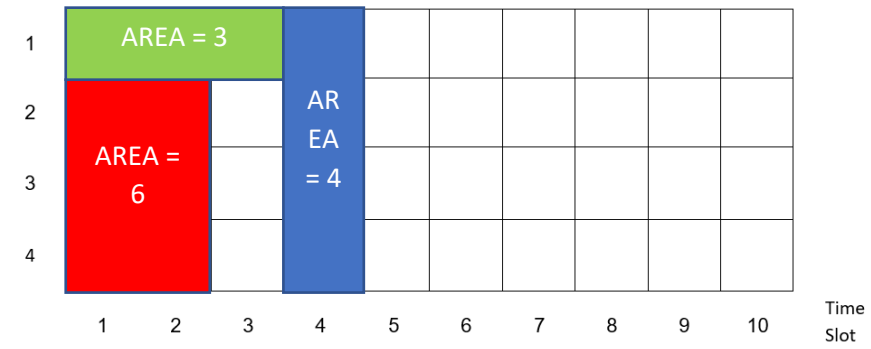
ניצולת BSAF

$$\frac{(6+4+3)}{4*6}=\frac{13}{24}$$



ניצולת אופטימלית

$$\frac{(6+4+3)}{4*4}=\frac{13}{16}$$



שאלה 2 - החלפת הקשר (20 נקודות)

1. נתון הקטע הבא מתוך קוד הגרעין להחלפת הקשר:

```
01. movl prev, %eax
02. movl next, %edx
03. pushl %esi
04. pushl %edi
05. pushl %ebp
06. movl %esp, prev->thread.esp
07. movl next->thread.esp, %esp
08. movl $1f, prev->thread.eip
09. pushl next->thread.eip
10. jmp __switch_to
11. 1:
12. popl %ebp
13. popl %edi
14. popl %esi
```

א. מה יקרה אם נחליף את שורות 8-10 בשורה **call __switch_to** ?

הבעיה בפקודת call היא שהיא דוחפת למחסנית את כתובת החזרה של השורה הבאה לביצוע (תווית '1') תמיד, אך אנחנו רוצים לדחוף כתובת חזרה שונה בהתאם למקרה בו אנו נמצאים:

לראשונה לאחר fork: כתובת החזרה שנרצה לדחוף היא ret_from_fork.
ובשאר המקרים (המקרה הסטנדרטי): כתובת החזרה היא אכן התווית '1'.

ב. נניח כי ביצענו קריאה למאקרו current לפני שורה 1, לאחר איזה שורה קריאה נוספת למאקרו תחזיר תשובה שונה?

שורה מספר 7, כי שם מתבצעת החלפת המחסניות.

ג. מתרגל בקורס רצה לשפר את זמן הביצוע של החלפת ההקשר. מכיוון ששם לב כי המאקרו **switch_to** אינו עושה כלל שימוש ברגיסטר ecx ומאחר וידוע כי פעולות על רגיסטרים מהירות משמעותית מפקודות המערבות את הזיכרון, הציע להחליף את שורה **05** בשורה **mov %ebp, %ecx** ואת שורה 12 בשורה **mov %ecx, %ebp** האם הקוד יעבוד כראוי לאחר השינוי המוצע?

לא, מכיוון שערך הרגיסטר ecx משותף עבור שני התהליכים next ו prev בתוך הפונקציה switch_to ולכן השחזור של ebp לאחר ביצוע context_switch יהיה לערך של ebp של התהליך הקודם ולא החדש.

2. כפי שראינו בתרגול מצביע לבסיס מחסנית הגרעין של התהליך הנוכחי נשמר גם ב **thread->esp0** וגם ב– **tss->esp0** האם ניתן לוותר על אחד מהם? אם לא הסבר מדוע ואם כן פרט על מי וכיצד נשיג את המידע הדרוש.

בלתי אפשרי לוותר על tss->esp0 מכיוון שבעזרתו אנו ניגשים למחסנית הגרעין כאשר אנו עוברים בין מצב user mode ל- kernel mode. אין דרך אחרת בה נוכל לגשת אל מחסנית הגרעין של התהליך הנוכחי.
לעומת זאת, thread->esp0 אינו הכרחי אך חוסך חישוב אריתמטי. thread->esp0 שומר מצביע לתחילת מחסנית הגרעין של התהליך אליו הוא שייך ומשמש בעצם לעדכון של tss->esp0 בעת החלפת הקשר.
ניתן לוותר עליו ולעדכן את tss->esp0 באמצעות המצביע לתהליך next בהחלפת הקשר ע"י הוספת 8K לכתובת next וכך נגיע לכתובת תחילת מחסנית הגרעין.

שאלה 3 – זימון תהליכים (30 נקודות)

שאלה זו עוסקת במדיניות זימון התהליכים של לינוקס כפי שנלמדה בתרגולים.

לנוחיותכם מצורף חלק מהקוד כפי שנלמד בתרגולים:

```
#define MAX_PRIO 140
#define MIN_TIMESLICE (10 * HZ / 1000) /* 10 msec */
#define MAX_TIMESLICE (300 * HZ / 1000) /* 300 msec */
#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
if (prio < MAX_RT_PRIO)
prio = MAX_RT_PRIO;
if (prio > MAX_PRIO - 1)
prio = MAX_PRIO - 1;
BONUS(p) = 10*(SleepAvg/MaxSleepAvg-1/2)
DELTA(p) = 5*TaskNice(p)/20+2
```

א) (5 נק') האם ייתכן מצב בו תהליך אינטראקטיבי **A** יהיה באותה העדיפות הדינאמית כמו תהליך חישובי **B** ? אם כן תארו מצב כזה (תארו 2 תהליכים, את עדיפותם הסטטית, ואת עדיפותם הדינאמית - והוכיחו מספרית בעזרת חישובי **bonus** ו **DELTA**-שהתהליכים אכן מקיימים את הדרישות), אם לא נמקו מדוע.

ייתכן, נניח כי המצב נתון המצב האפשרי הבא:

$$A \rightarrow prio = B \rightarrow prio = 116$$

$$A \rightarrow static_{prio} = 120\,;\; B \rightarrow static\_prio \;=\; 116$$

$$Task_{Nice(A)} = 0, Bonus_{(A)} = 4\,, \qquad Task_{Nice(B)} = -4, Bonus_{(B)} = 0$$

ולכן במקרה זה אכן מתקבל כי תהליך A הוא אינטרקטיבי כי מתקיים עבורו כי ה- Delta שלו היא 2 וההפרש בין העדיפות הסטטית לדלתא הוא 118 והוא גדול ממש מ-116 ולכן הוא אינטרקטיבי.
בעוד שלתהליך B הדלתא היא 1 ולכן ההפרש בין העדיפות הסטטית לדלתא הוא 115 שהוא קטן ממש מהעדיפות הדינמית ולכן התהליך לא אינטרקטיבי.

ב) (2 נק') בתרגול מצוינות **4** סיבות בגינן עשוי תהליך להגיע לפונקציה schedule. בחנו את הקוד של **sys_sched_yield** ו- **interruptible_sleep_on**. בכל אחת מהפונקציות הנ"ל ישנה קריאה ישירה לפונקציה **schedule** . אם היינו משנים קריאה זו ל-
set_tsk_need_resched(current) כיצד זה היה משפיע אם השינוי היה מתבצע רק ב-
sys_sched_yield? כיצד זה היה משפיע אם השינוי היה מתבצע רק ב-
interruptible_sleep_on ? נמקו (בתשובתכם התייחסו לנכונות הביצוע - כלומר האם התהליך אכן מוצא מהקשר? האם הקרנל ימשיך לעבוד בצורה תקינה? וכו')

Sched_yield – הקריאה ל-schedule מתבצעת בסוף פונקציית המערכת ולא מתבצעת שום פקודה נוספת מלבד יציאה מהפונקציה. לכן הנ"ל שקול להדלקה של הדגל need_resched אשר בסוף הפונקציה (לפני החזרה ל-user) יעביר אותנו ל-schedule ותבוצע החלפת הקשר באופן זהה. והקרנל ימשיך לעבוד באופן תקין.

Interruptible_sleep_on – במקרה זה לפני הקריאה ל-schedule מתבצעת הכנסה לתור ההמתנה ולאחר מכן הוצאה מתור הריצה והחלפת הקשר. כאשר התהליך יתעורר, הוא יבצע הכנסה לתור הריצה והוצאה מתור ההמתנה. במידה ונחליף את הקריאה ל-schedule ב-need_resched, תתבצע הכנסה לתור ההמתנה ולאחריה הדלקה של הדגל need_resched ומיד לאחר מכן הוצאה של התהליך מתור המתנה! לאחר מכן רגע לפני החזרה ל-user אנו נכנס ל-schedule ונוציא את התהליך גם מתור הריצה!! ונבצע החלפת הקשר. בעצם איבדנו את התהליך! לא ניתן להעיר אותו בחזרה. תפקוד לא תקין של ה-kernel.

ג) לצורך שאלה זו נזכיר כיצד מחושב **sleep_avg** של תהליך בגרעין:

```
#define MAX_SLEEP_AVG (2*HZ)
sleep_time = jiffies - p->sleep_timestamp;
p->sleep_avg += sleep_time;
if (p->sleep_avg > MAX_SLEEP_AVG)
p->sleep_avg = MAX_SLEEP_AVG;

#define EXPIRED_STARVING(rq) \
((rq)->expired_timestamp && \
(jiffies - (rq)->expired_timestamp >= \
STARVATION_LIMIT * ((rq)->nr_running + 1))
```

1. נניח שקיים תהליך אינטראקטיבי **A** בעדיפות סטטית 100 ובעל **sleep_avg=MAX_SLEEP_AVG**.

בנקודת זמן t=0 התהליך החל לבצע משימה חישובית ארוכה. מהו הזמן המקסימלי (במילישניות) שהתהליך ירוץ לפני שיעבור ל **expired** בהנחה שהוא התהליך היחיד ב runqueue?

תהליך יקבל TIME_SLICE כשיגמר לו הTIME_SLICE כל עוד הוא מוגדר כאינטראקטיבי.
לכן נבדוק מתי התהליך מפסיק להיות אינטראקטיבי.
על מנת שתהליך יהיה אינטרקטיבי נדרש ש $Bonus \geq \Delta$ עבורו.
מהנתונים עולה כי ה-Bonus שלו הוא בהתחלה 5 ו-ה-Delta שלו היא 3- לאורך כל הדרך.
<u>הסבר לחישוב הדלתא:</u> העדיפות הסטטית היא 100 (נקבעת ע"י TASK_NICE + 120) ולכן TASK_NICE = -20 ומכאן Delta = -3.
כעת,

1. נשים לב שכיוון שה-bonus תלוי ב-sleep_avg לפי הקשר: $Bonus = 10 * \left(\frac{sleep_{avg}}{MAX_{SLEEP_{AVG}}} - \frac{1}{2}\right)$

נקבל כי התהליך יפסיק להיות אינטראקטיבי כאשר $Bonus < \Delta$:

$$10 * \left(\frac{sleep_{avg}}{MAX_{SLEEP_{AVG}}} - \frac{1}{2}\right) < -3$$

$$sleep_{avg} < \frac{2}{10} * MAX_{SLEEP_{AVG}} = \frac{4}{10} * HZ$$

ולכן כאשר זמן ההמתנה יגיע ל $400\ msec$ התהליך כבר לא יהיה אינטראקטיבי.
וזה קורה אחרי 1600 msec (כיוון ש-MAX_SLEEP_AVG = 2000 msec בהתחלה).
בנוסף, נכנס לתנאי שבודק אם התהליך אינטרקטיבי (התנאי השני) רק אחרי שיגמר ה-TIME_SLICE של התהליך הנ"ל (בפונקציה scheduler_tick אשר נקראת בפסיקת שעון) כלומר אחרי כפולה של TIME_SLICE שהוא במקרה זה 300 msec – חישוב ע"י המאקרו בסעיף הבא.
=> והזמן המינימלי בו שני התנאים קורים הוא לאחר 1800 מילי שניות.
ולכן לאחר 1800 מילי שניות נעביר את התהליך ל-expired.

2. ביצד תשובתכם לסעיף הקודם הייתה משתנה אם נתון שקיים תהליך **B** ב **expired** והחל מהרגע t=0 שתואר קודם, נותרו 1000 מילישניות עד כילוי ה**timestamp_expired** ? כלומר עד שהמאקרו **EXPIRED_STARVING** מתחיל להחזיר **TRUE**).

כיוון שהתהליך מוגדר כאינטרקטיבי ע"י המאקרו הרלוונטי אזי נכנס לתנאי אשר יעביר אותו ל-expired רק אם ה-time slice שלו הוא 0 וגם EXPIRED_STARVING יחזיר true (בניגוד לסעיף 1 ששם נכנסנו לתנאי כיוון שהתהליך הפך להיות לא אינטרקטיבי) , זה יקרה אחרי c*TIMESLICE(A) כאשר c>0 קבוע כלשהו מינימלי עבורו c*TIMESLICE(A)>=1000.

מהנתונים של השאלה ומכיוון שה TIME_SLICE המוקצה כל פעם הוא 300msec נקבל כי c=4.
ונקבל כי נכנס לתנאי הראשון וגם לתנאי השני עם ה-EXPIRED_STARVING אחרי 1200 מילישניות.

<u>חישוב ה-TIMESLICE עפ"י המאקרו:</u>

```
#define TASK_TIMESLICE(p) (MIN_TIMESLICE + \
((MAX_TIMESLICE - MIN_TIMESLICE) * (MAX_PRIO-1-(p)->static_prio)/39))
```

ד) מהו פרק הזמן המקסימלי שיכול לעבור מהרגע שהודלק הדגל **need_resched** בתהליך שרץ ועד שהוא מגלה את הצורך בהחלפת הקשר (במערכת מרובת מעבדים)?

***בעייתיות בשאלה.**

ה) האם אפשר לתת דוגמה בה תהליך יכול להתחיל לרוץ קצת אחרי פסיקת שעון ולעזוב את ה CPU קצת לפני פסיקת השעון הבאה (פחות מtick)? אם הדבר בלתי אפשרי הסבירו למה, אחרת תנו דוגמה מפורטת.

כן. נניח והובא תהליך חדש עקב פסיקת שעון (למשל נגמר לתהליך הקודם ה-time slice) וכעת נרצה שהתהליך החדש ירוץ במשך זמן שקטן מ-2msec (הזמן עד פסיקת השעון הבאה) ואז יסתיים/יוחלף. זמן החילוף ב-schedule אינו נספר מכיוון שאנו חוסמים פסיקות. ומכיוון שה-time_slice המינימלי הוא 10msec, המקרה הנ"ל יכול לקרות רק במידה ולאחר החלפת ההקשר, בחזרה לתהליך החדש ב-user התהליך מסתיים או מבצע החלפת הקשר עקב קריאה ל-sched_yield/wait או פסיקת חומרה שאינה פסיקת שעון אשר גוררת החלפת הקשר. ובעצם נחזור לkernel ונחליף את התהליך שוב לפני פסיקת השעון הבאה.