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

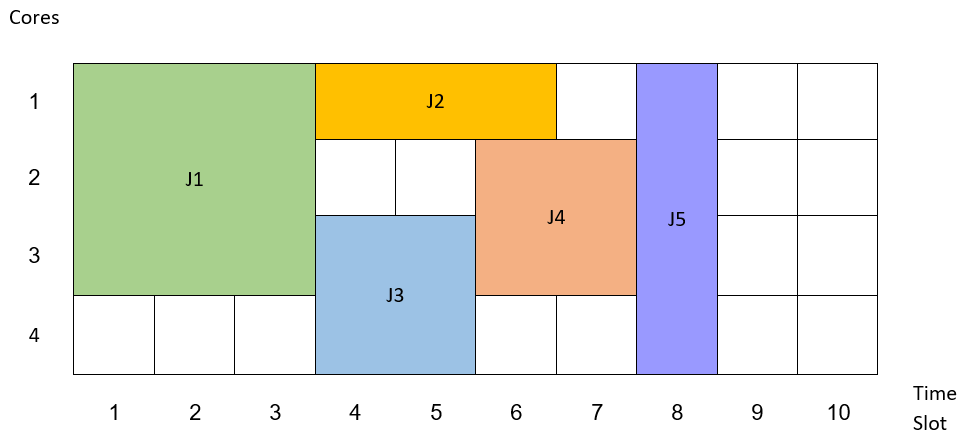
Dry part submission instructions:

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:
   1. The first page should contain the details about the submitters - Name, ID number and email address.
   2. 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):

ניעזר בהגדרה של זמן המתנה: .  
ונרצה לחשב את זמן ההמתנה הממוצע עבור כל התהליכים:

* 1. (6 נק') מדד זמן התגובה הממוצע (average response-time):

ניעזר בהגדרה של זמן תגובה: .  
ונרצה לחשב את זמן התגובה הממוצע עבור כל התהליכים:

* 1. (6 נק') מדד הניצולת (utilization):

ניצולת האלגוריתם היא אחוז הזמן שמנוצל לריצת תהליכים מתוך כל הזמן במסגרת זמן הריצה

1. (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**)

על ידי שימוש במדיניות SJF (התהליך הקצר ביותר נכנס קודם) נזמן את התהליכים (למשל) בסדר הבא:

ואז הניצולת תהיה:

נדגים בציור:



J5

J3

J4

J1

J2

1. (8 נק') מנו 2 יתרונות ו-2 חסרונות של מדיניות זימון מסוג batch-scheduling על פני מדיניות Round-robin. נמקו.

**יתרונות:**

1. במדיניות Batch Scheduling ניתן להשיג ביצועים אופטימליים בזמן המתנה ממוצע (עם שימוש במדיניות זימון SJF) – הוכחנו בהרצאה.
2. במדיניות Batch לא מפקיעים תהליכים מהמעבד (non-preemtive) ולכן יש פחות overhead בביצוע context switch כיוון שתהליכים מתחלפים רק כאשר תהליך אחד מסיים לרוץ והתהליך הבא מזומן

**חסרונות:**

1. במדיניות Batch Scheduling, כל תהליך שמזומן ירוץ עד שהוא מסתיים. זה יכול ליצור מצב שתהליכים כבדים מאוד מרעיבים תהליכים קלים ומגדילים משמעותית את פקטור ההאטה שלהם (slowdown).
2. מדיניות Batch Scheduling לא מתאימה לתהליכים אינטראקטיביים שמעוניינים לקבל את המעבד הרבה פעמים לזמן קצר (למשל: מעבדי תמלילים ותהליך האחראי על צג המחשב) ולכן לא אופטימלית לשימוש במחשבים ביתיים עבור כל התהליכים.

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.

1. (8 נק') האם BSAF סובל מ-convoy effect? נמקו (עליכם להמחיש בעזרת הטבלה הנתונה).

**תשובה**: BSAF סובל מconvoy effect, יכול להיווצר מצב שתהליך מאוד ארוך דורש רק מעבד אחד, ומספר תהליכים קצרים מאוד דורשים את כל מעבדי המחשב, התהליך הארוך יגדיל משמעותית את הslowdown של כל התהליכים הקצרים כיוון שלא ניתן למלא את הדרישות עבור ריצתם עד שהוא יסתיים. למשל בציור הבא:



1. (8 נק') האם אלגוריתם הזימון BSAF הינו אופטימלי מבחינת מדד הניצולת? הוכיחו או הפריכו (עליכם להמחיש בעזרת הטבלה הנתונה)

**תשובה**: אלגוריתם הזימון אינו אופטימלי מבחינת מדד הניצולת. נראה דוגמה עבורה התהליכים יכולים להיות מזומנים בדרך שתגדיל את הניצולת. נשים לב שניתן לזמן את התהליכים הצהוב והאפור לפני הכתום והכחול, ואז נקבל ניצולת טובה יותר.



# שאלה 2 - החלפת הקשר (20 נקודות)

.1נתון 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 10-8 בשורה **call \_\_switch\_to** ?  
  
**תשובה**: אם נחליף את השורות הנ״ל בפקודת הקפיצה הסטנדרטית call, המעבד ידחוף תמיד למחסנית את כתובת החזרה של השורה הבאה לביצוע (במקרה זה השורה של התווית 1). זהו המקרה הסטנדרטי, אך קיים מקרה נוסף, בו התהליך מזומן בפעם הראשונה למעבד (כאשר התבצע fork ליצירת התהליך ואז context switch עבור תהליך זה). במקרה כזה אנחנו רוצים שערך החזרה של fork עבור התהליך הבן יהיה 0, ולכן נקפוץ ל-ret\_from\_fork.

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

**תשובה**: לאחר שורה 7 משתנה ערך הרגיסטר esp. כיוון שהמאקרו current מבצע חישוב bitwise על הרגיסטר esp, הוא תלוי אך ורק בו, ובפרט לאחר שורה 7, המאקרו יחזיר ערך שונה.

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?

**תשובה**: לא בהכרח, הרגיסטרים משותפים לכל התהליכים הרצים על מעבד מסוים בעוד הזיכרון אינו משותף. לכן, יווצר מצב שבהחלפת הקשר, התהליך הבא לביצוע ידרוס את המידע של התהליך הקודם.

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.

**תשובה**: ניתן לוותר על שמירת המבנה tss לכל מעבד. ניתן לגשת לesp0 בשדה הthread של התהליך הנוכחי שרץ על המעבד. כאשר תתבצע החלפת הקשר, המעבד יריץ את התהליך הבא, וניתן יהיה לגשת לesp0 בשדה ה-thread. שמירת המבנה tss נובעת משיקולי מהירות, שכן גישה לזיכרון המעבד מהירה פי כמה סדרי גודל מגישות לזיכרון, ובגלל שמשתמשים בesp0 מספר רב של פעמים (למשל, בחישוב המאקרו current) הגישה הזו צריכה להיות מהירה.

# 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**

1. (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.  
     
   **תשובה**: ייתכן מצב כזה. ניקח לדוגמה 2 תהליכים :  
   A->static\_prio = 120, nice(A,-8)  
   B->static\_prio = 116, nice(B,-8)  
   נסמן ב- את זמן ההמתנה המקסימלי של תהליך (), ונניח שזמן ההמתנה הממוצע של A הוא וזמן ההמתנה הממוצע של B הוא . אזי מתקיים   
   עבור התהליך A מתקיים התנאי ולכן הוא אינטראקטיבי. עבור התהליך B מתקיים התנאי ולכן הוא חישובי. ובפרט העדיפות הדינאמית שלהם זהה כפי שהראנו (118). נשים לב כי שני התהליכים אינם תהליכי זמן אמת, שכן
2. (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')  
   **תשובה**: ההבדל בין קריאה ישירה לפונקציה schedule לבין הדלקת הדגל הוא שקריאה ישירה תבצע מיד החלפת הקשר, והדלקת הדגל תאפשר לתהליך לרוץ במצב גרעין, ותבצע schedule רק בעת החזרה ממצב גרעין למצב משתמש. אם השינוי היה מתבצע ב-**sys\_sched\_yield** בלבד, הוא לא היה משפיע על נכונות הקוד, כיוון שקריאת המערכת **sched\_yield** גורמת לתהליך לוותר על המעבד מרצונו. לכן בסוף קריאת המערכת ובעת חזרה למצב משתמש, הדגל ייבדק והמעבד אכן יופקע מהתהליך ותתבצע קריאה ל-**schedule**, והקרנל ימשיך לעבוד בצורה תקינה. הפונקציה**-interruptible\_sleep\_on** אחראית להפוך תהליך ל-INTERRUPTIBLE, כלומר הוא ממתין לביצוע פעולה כלשהי, אך ניתן לקטוע את ההמתנה ע״י שליחת סיגנל. קריאה זו מוסיפה אותו לתור ההמתנה, מבצעת reschedule, ולאחר שהתהליך חוזר מהחלפת הקשר (כלומר, הוא סיים להמתין והזמן החליט שהוא התהליך הבא לביצוע) הוא מוציא את עצמו מתור ההמתנה. נשים לב שם היינו מחליפים שם את הקריאה לschedule ב-set\_need\_resched(), אז התהליך היה מוסיף עצמו לתור המתנה, **ממשיך לרוץ** במצב קרנל, מעיר את הדגל ואז מסיר עצמו מתוך ההמתנה. למעשה, ברגע שהתהליך יחזור ממצב גרעין, תתבצע החלפת ההקשר, אבל כעת התהליך אינו בתור ההמתנה יותר, על אף שהוא ממתין, וזו התנהגות שגויה (הגרעין יכול לזמן אותו לביצוע בהחלפת הקשר כלשהי לפני שהוא יסיים להמתין).
3. 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=0התהליך החל לבצע משימה חישובית ארוכה. מהו הזמן המקסימלי (במילישניות) שהתהליך ירוץ לפני שיעבו ר ל **expired** בהנחה שהוא התהליך היחיד ב?runqueue   
**תשובה**: נבחין כי Hz = 1000. כלומר, MAX\_SLEEP\_AVERAGE =2000ms. נחשב את הדלתא של התהליך: (הnice של התהליך הוא 20- כי nice = static\_prio – 120). נרצה לבדוק את התנאי לסיווגו של התהליך כאינטראקטיבי:  
כלומר, כל מילי שנייה נשלחת פסיקת שעון ו-sleep\_avg = T קטן באחד. לכן, לאחר 1600ms התנאי כבר לא יתקיים והתהליך לא יסווג כאינטראקטיבי. כעת הtimeslice שלו יחושב באופן הבא:  
כלומר, התהליך יסיים את הtimeslice לאחר 1900 מילי-שניות סך הכל, וזה גם הזמן שהוא ירוץ לפני שיועבר לתור expired.

2. כיצד 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**).

1. מהו פרק הזמן המינימלי שיכול לעבור מהרגע שהודלק הדגל **need\_resched** בתהליך שרץ ועד שהוא מגלה את הצורך בהחלפת הקשר (במערכת מרובת מעבדים)?
2. האם אפשר לתת דוגמה בה תהליך יכול להתחיל לרוץ קצת אחרי פסיקת שעון ולעזוב את ה CPU קצת לפני פסיקת השעון הבאה (פחות מ ?(tickאם הדבר בלתי אפשרי הסבירו למה, אחרת תנו דוגמה מפורטת.