קודם כל, סיכום קצר. אנחנו יוצרים מדיניות זימונים חדשה בשם short. סדר העדיפויות מבחינת קריאה לתהליכים:

1. זמן אמת
2. Short
3. Other
4. Overdue

כדי למנוע באגים, אנחנו נעבוד כל הזמן ככה שshort עדיף פחות מother ואז נחליף ביניהם. כרגע עושה רושם שנצטרך בrunqueue להחזיק active בשביל short (ולפי תשובה מהמתרגל נדע אם אנחנו צריכים גם expired). אנחנו מתכננים גם להחזיק רשימה של תהליכים שהגיעו לoverdue, בגלל שבחירת התהליכים שלהם מתבצע לפי FIFO.

**שאלות למתרגל:**

1. האם במהלך הריצה אפשר לשנות את trial? למשל, להגדיל את המספר הזה?! כביכול התשובה הברורה היא לא, אבל זה לא כל כך מובן לפי התרגיל.
   1. **אי אפשר לשנות אותו בכלל**
2. האם מריצים את אותה עדיפות עד שגומרים אותה או שמריצים בסגנון round robin כמו other? אם זה כמו בother אנחנו כנראה נצטרך גם להחזיק expired.
   1. **RR, בלי לשאול את המתרגל... קראתי שוב. נצטרך לתחזק גם רשימת expired\_short בדומה לרשימת expired הרגיל כדי באמת לגמור את ה-timeslice שלהם. כלומר, אם יש 2 תהליכי SHORT (נניח, P ו-Q) בעדיפות גבוהה ועוד תהליך SHORT אחד בעדיפות נמוכה L, נריץ למשך slice אחד את P ואז נריץ למשך slice אחד את Q, ואז עוד פעם P... עד ש-P,Q מסיימים או הופכים ל-Overdue**
3. כאשר תהליך יוצר תהליך חדש האם הוא מוריש לו את requested (רלוונטי לdo\_fork)?
   1. **כן**
4. מה קורה לתהליך overdue שלא הסתיים אבל עצר. (בשביל קלט למשל)? ממשיכים הלאה? מחכים?
   1. **ממשיכים הלאה, הוא עובר לסוף התור (כלומר, זה כאילו הוא חיכה 0 זמן)**

**הערות:**

1. לשים לב בקריאת מערכת 245 שזה trials לא trails.
2. אם לא מוצאים פונקציה ששמה מתחיל ב-sched\_, כדאי לנסות להשמיט את sched\_ ולנסות שוב.

**קבצים ופונקציות שיהיו רלוונטיים עבורנו.**

(דורי, אתה זוכר אולי לכתוב תיאור קצר על הפונקציות, ומי קורא להן?)

כל הקבצים כנראה נמצאים בתיקיית kernel\sched.

פונקציית setscheduler (נמצא ב-sched.c):

* צריך לשנות כדי לטפל במקרה שמשנים לshort
* צריך למנוע מעבר מ-short לכל דבר אחר
* Priority נשאר אותו הדבר
* אם trial (או requested) לא תקין צריך להחזיר שגיאה עם -1
* צריך למנוע יכולת שינוי מספר ה-TRIALS לאחר אתחול (אפשר לקרוא לפונקציה הזו שוב גם על SHORT, אבל אסור לשנות את מספר ה-TRIALS)

sched.h:

אותו הקובץ עם כל ההגדרות ששינינו בתרגיל הקודם. צריך להוסיף פה define חדש עבור short (מספר 4). בשלב זה אנחנו לא רואים סיבה להוסיף גם לoverdue.

task\_struct (נמצא ב-entry.S):

להוסיף requested, trial, slice, current\_trial.

do\_fork (נמצא ב-fork.c):

* רלוונטי כשתהליך short (או overdue) יוצר תהליך חדש.
* לזכור תמיד לעדכן את current->need\_resched להיות 1.
* לערוך את השדות הרלוונטיים (trials, time\_slice)
* לטפל במקרה של overdue מתפצל (מבחינת הוספה לתור למשל)
* בעת יצירת תהליך, להתחיל לתעד את 30 התהליכים הבאים
* אם האב הופך כאן ל-Overdue צריך לטפל בזה
* לטפל בשינוים ב-runqueue – להכניס את הבן למקום הנכון ואם צריך לשנות את המיקום של האב

do\_exit (נמצא ב-exit.c):

* בעת יציאת תהליך (לא צריך לחכות ל-wait) להתחיל לתעד 30 תהליכים קדימה
* לטפל בשינוים ב-runqueue (שליפה)

reschedule\_idle (נמצא ב-sched.c):

* לקרוא ולהבין מה צריך לעשות פה

schedule() (נמצא ב-sched.c):

* צריך למצוא איך לתעד את החלפת ההקשר (הבדיקה האם צריך תתבצע בקריאה לפונקציה המתעדת).

scheduler\_tick() (נמצא ב-sched.c):

* פונקצית טיפול בפסיקות שעון.
* להוריד ב-1 את ה-time\_slice (זה כבר אמור לקרות) ואם הגענו ל-0 להוריד אחד מ-trials, לחשב את הslice הבא ולהעביר ל-expired או אם הפכנו ל-Overdue, להעביר לתור שלהם. יש מצב שזה לא קורה כאן אבל נראה אם זה קורה לדברים אחרים ונזרום.
* לבדוק האם נשאר זמן. אם נגמר צריך גם להוריד את הtrial. זה נעשה פה?
* לבדוק האם trial=0 ואז לשלוח לoverdue

runqueue (נמצא ב-sched.c):

* להגדיר תור נוסף לטיפוס SHORT
* להגדיר תור expired נוסף לטיפוס SHORT (כדי שנוכל להריץ קבוצה של SHORTים באותה עדיפות בשיטת RR)
* להגדיר רשימה מקושרת פשוטה ל-Overdue-SHORTים
* מצביע לתהליך ה-Overdue-SHORT הראשון בתור (כשאנחנו מוסיפים תהליכים לתור הזה מוסיפים אותם לסוף, ולכן צריך לשמור מצביע להתחלה)

sched\_{get,set}\_scheduler/param (הכל ב-sched.c):

* צריך לזכור לבדוק שהערכים שקיבלנו חוקיים: מספר הניסיונות לא משתנה, ובאופן כללי 2 הפרמטרים בטווחים המותרים. בנוסף, ה-sched\_priority חייב להיות 0, ואסור לשנות תהליך SHORT למשהו אחר או תהליך שהוא לא OTHER ל-SHORT.