**תרגיל בית מספר 2 – מערכות הפעלה**

**חלק יבש**

**פתרון שאלה מספר 1**

סעיף א':

למדנו כי הפונקציה schedule() בוחרת תהליך חדשה להרצה מה-runqueue של המעבד המקומי. מעבד נתון יכול להריץ אך ורק את התהליכים הניתנים להרצה שנתונים במסגרת ה-runqueue המתאים. תהליך הניתן להרצה תמיד יופיע בrunqueue אחד בלבד ולא במספר תורים ולפיכך כל עוד תהליך ניתן להרצה, הוא יהיה מוגבל למעבד אחד בלבד ולכן אין הבדל במקרה של מערכת מרובת מעבדים. הסיבה לכך שאין הבדל הינה בכך שבדר"כ דרך פעולה זו תהיה יעילה עבור ביצועי המערכת. זאת מכיוון שמטמון החומרה (hardware cache) של **כל** מעבד סביר להניח יתמלא במידע של התהליכים הניתנים להרצה מה-runqueue (כלומר ניצול טוב של כלל המעבדים). עם זאת, נדגיש כי במקרים מסוימים בכל זאת יכול להיות שתהליך הניתן להרצה יוגבל למעבד נתון וזה יגרום לפגיעה בביצועים. למשל, ע"י הקצאת מספר גדול של תהליכי batch המנצלים בצורה כבדה את המעבד ואם כולם יהיו על אותו runqeueu לבסוף אחד המעבדים יהיה עמוס בעוד שאר המעבדים כמעט ולא יעשו דבר.

סעיף ב':

כפי שצוין לעיל, להלן יתרון וחסרון של השיטה המתוארת.

יתרון: שיטה זו תורמת לביצועי המערכת מכיוון שמטמון החומרה (hardware cache) של **כל** מעבד סביר להניח יתמלא במידע השייך לתהליכים הניתנים לריצה במסגרת ה-runqueue הרלוונטי. כלומר, כל מעבד ינוצל בהתאם.

חסרון: כפי שצוין, לעיתים הגבלה של תהליך רץ למעבד מסוים עלול לגרום לפגיעה חמורה בביצועים. למשל, במקרה של מספר רב של תהליכי batch שגורמים לשימוש כבד של המעבד. אם רובם יהיו על גבי אותו runqueue, הרי שיכול להיות מצב שמעבד אחד במערכת יהיה עמוס בעוד האחרים כמעט ולא עושים דבר (כמעט idle).

סעיף ג':

Scheduling domain הוא בעצם קבוצה של מעבדים שעומס העבודה שלהם צריך להיות מאוזן ע"י ה-kernel. קבוצה זו משמשת במסגרת אלגוריתם איזון המשימות. כל scheduling domain מחולק בעצם, בהתאם לתור, לקבוצה אחת או יותר כאשר כל קבוצה כזו מייצגת בעצם תת-קבוצה של המעבדים של ה-scheduling domain. תהליך בעצם יעבור ממעבד אחד לאחר אם סך עומס העבודה של קבוצה מסוימת ב-scheduling domain הוא נמוך בצורה ניכרת מסך עומס העבודה של קבוצה אחרת באותו scheduling domain.

סעיף ד':

האלגוריתם מחליט האם יש צורך בהפעלת איזון משימות על המעבד ע"י שימוש בפונקציה שנקראת rebalance\_tick() אשר מופעלת ע"י הפונקציה המוכרת scheduler\_tick() בכל tick. הפונקציה מקבלת כפרמטרים אינדקס, אתהמעבד הנוכחי של המעבד המקומי ואת הכתובת של ה-rq הנוכחי ודגל מסוים שנקרא idle. הפונקציה קובעת תחילה את מספר התהליכים ב-rq ומעדכנת את ממוצע עומס העבודה וזאת ע"י גישה לשדות ה-nr\_running וה-cpu\_load של מתאר ה-rq. לאחר מכן, הפונקציה מבצעת לולאה על פני ה-scheduling domains בניתוב החל מה-base domain ועד ל-domain ברמה הגבוה ביותר ובכל איטרציה קובעת אם הגיע הזמן להפעיל פונקציה אחרת שנקראת load\_balance() שתפעיל את פעולת האיזון מחדש.

סעיף ה:

כפי שצוין, האלגוריתם מחליט כי יש צורך בהפעלת איזון משימות על המעבד כאשר קבוצה מסוימת של ב-scheduling domain מסוים מקיימת סך עבודה נמוך בצורה משמעותית מקבוצה אחרת באותו scheduling domain. באמצעות פונקציה הנקראת rebalance\_tick() ניתן לקבוע את מספר התהליכים ב-runqueue מסוים ולעדכן את ממוצע עומס העבודה על גבי אותו runqueue. הפונקציה עושה זאת תוך גישה לשדות ה-nr\_running וה-cpu\_load של מתאר ה-rq. פונקציה זו בעצם מבצעת לולאה המתקדמת על פני כלל ה- scheduling domains בניתוב מה-base domain אל ה-domain ברמה העליונה ביותר. בכל איטרציה, הפונקציה מחליטה האם הגיע הזמן להפעיל פונקציה אחרת שנקראת load\_balance() שזו תפעיל פעולת איזון מחדש על ה- scheduling domain. פונקצית ה-load\_balance() קובעת בעצם אם scheduling domain מסוים אינו מאוזן בצורה ניכרת. באופן מדויק יותר, ניתן לאמר כי היא בודקת אם חוסר האיזון ניתן להפחתה ע"י העברת מספר תהליכים מהקבוצה העמוסה ביותר (במסגרת ה-scheduling domain) ל-rq של המעבד המקומי (ובמידה וזה המצב תנסה לבצע לו 'הגירה' שכזו). הפונקציה הנוספת העיקרית במסגרת האלגוריתם היא פונקציית ה-move\_tasks() אשר מעבירה את התהליכים בפועל מrq המקור ל-rq המקומי.

**פתרון שאלה מספר 2**

**פתרון שאלה מספר 3**

**סעיף א':**

במהלך פעילותה, הפונקציה scheduler\_tick() מפעילה בין היתר את הפונקציה set\_tsk\_need\_resched() שמדליקה את דגל ה-NEED\_RESCHED של התהליך. דגל זה מאלץ בעצם להפעיל את הפונקציה schedule() כך שתהליך ה-current יוכל להיות מוחלף. הסיבה לכך שלא ניתן להשתמש בקריאה ישירה לפונקציה schedule() הינה שהפונקציה scheduler\_tick() נקראת כפונקציית טיפול של פסיקת שעון ופונקציה כזזו לא יכולה לגרום להפעלת הפונקציה schedule() אלא בסך הכל להדליק דגל. אחד המקרים להגעה לפונקציה schedule כפי שנלמד בתרגולים זה כתגובה על פסיקת שעון. במקרה זה, התהליך סיים את הtime\_slice שלו ופסיקת השעון מורידה אותו לאפס. אנו חייבים להגיע בסופו של דבר לפונקציה schedule ולבצע לעיתים החלפת הקשר. בניגוד לפונקצית כמו sleep\_on בהן קראנו מפורשות ל-schedule ישנם קטעי קוד שאסור להם לעשות זאת ואלה קטעי הקוד הנוגעים לפסיקות חומרה ובהן לא ניתן לבצע החלפת הקשר. פסיקת חומרה לעולם לא תוכל לקרוא לפונקציה schedule אולם ניתן לפנות למתאר התהליך ולהדליק את הדגל הרצוי, שהוא need\_resched.

**סעיף ב':**

המאקרו switch\_to מעורב בעצם בתהליך החלפת ההקשר בין תהליכים מסוימים שנסמנם כ-A ו-B. נדגיש כי בקוד, בטרם ביצוע פקודת ה-jmp אנו שומרים על גבי מחסנית הגרעין של תהליך B (אליו אנו מחליפים הקשר) את

next->thread.eip. בערך זה בעצם שאנו שומרים על המחסנית יש בעצם את המיקום בזיכרון של הלייבל "1" המופיע באותו קוד. רק לאחר מכן אנו מבצעים את הקפיצה. נבחין כי למרות שנהוג להפעיל פונקציה בשפת מכונה באמצעות הפקודה call אנו מבצעים כאן בעצם קפיצה עם jmp. הפקודה call כידוע לוקחת את כתובת החזרה, מכניסה למחסנית וקופצת לשורה הראשונה של הפונקציה הרצויה בעוד הפקודה jmp **רק קופצת.** כאשר הפונקציה תעשה return היא תלך אל המחסנית ותוציא את הכתובת אליה נחזור. הפונקציה תמצא במחסנית את הלייבל "1" ולכן אחרי ה-return נגיע לשורה הרצויה. הסיבה שלא השתמשנו ב-call במקרה זה הינה שההנחה שלנו הייתה שהתהליך B רץ בעבר ולכן ב-thread.eip היה לו את הלייבל "1". אולם, אם לא נניח זאת והתהליך רץ בפעם הראשונה, אז ב-thred.eip לא נמצא בעצם הלייבל "1" ולכן אנו נכניס לייבל אחר למחסנית ואז נבצע jmp שכשנחזור ממנו נגיע ללייבל אחר ולא ללייבל "1". בעצם, המטרה הכללית של השימוש ב-jmp במקום call הוא להתמודד עם המקרה של תהליך חדש בלבד (כלומר שעוד לא רץ). הסיבה שתהליך שרץ בפעם הלא ראשונה חוזר ללייבל "1" היא שאותו תהליך וויתר קודם לכן על המעבד ואז שמרנו את הלייבל ואנו משחזרים באמצעות הלייבל "1" את ה-thread.eip. אולם, עבור תהליך שרץ בפעם הראשונה, מה שכתוב ב-thread.eip אינו הלייבל "1" ונחזור ל-ret\_from\_fork.

**פתרון שאלה מספר 4**