**מבוא למערכות הפעלה, תרגיל מס' 1**

**תזמון תהליכים**

כללי:

מועד ההגשה: ה-17 באפריל 2020, 23:59 בלילה.

אופן הגשה: מערכת ההגשה האוטומטית של המחלקה - הוראות מפורטות יינתנו בהמשך בפורום.

שאלות הקשורות לעבודה יש לשאול **אך ורק** בפורום המתאים.

רקע:

בתרגיל זה תממשו מערכת לתזמון תהליכים. המערכת תכלול סימולטור להרצת קוד פשוט, ומערכת הפעלה פשוטה התומכת בריבוי תהליכים. בתרגיל זה לא תעבדו עם תהליכי מערכת אמיתיים או threads. נתונה לכם מערכת ממומשת חלקית ועליכם להוסיף למערכת זו חלקים נוספים.

ניתן לפתור את המטלה מבלי לעשות כל שינוי במחלקות הבאות, אך מומלץ ורצוי להבין כיצד הן עובדות על מנת לפתור את המטלה. שתי המחלקות הקשורות ל Console מיועדות על מנת לייצר ממשק משתמש שתוכלו לעבוד עמו במהלך ההרצה.

"ReadTokenRequest", " ProcessTableEntry", " ProcessConsole", "OutputConsole", "OutputConsole.Designer", "DISK", "AddressSpace", "Code".

במהלך העבודה אין לשנות את חתימות השיטות הנתונות, אם כי מותר להוסיף מחלקות, שיטות, ושדות כרצונכם.

שפת התכנות בה תומך הסימולטור : חלק זה כבר ממומש. הסימולטור יבצע הרצה באמצעות Interpreter (כלומר, ללא קומפילציה) של פקודות קוד פשוטות. הפקודות בהן תומך הסימולטור הן:

1. variable x – הכרזה על משתנה. כל המשתנים בהם תומך הסימולטור הינם משתנים נומרים (double). יש להכריז על משתנים לפני השימוש בהם.
2. x = 5 – שורת השמה. ניתן לשים מימין לפעולת ההשמה גם ביטויים מורכבים כגון x = y + 3 / 2. אין להשתמש בסוגריים, והביטוי יחושב תמיד משמאל לימין. יש להקפיד על רווחים בין תתי ביטויים.
3. if x < 5 – תנאים. הסימולטור תומך באופרטורים <,>,<=,>=,=,!=. השורה הבאה אחרי התנאי תבוצע רק אם ערכו של התנאי הוא true, אחרת תבוצע השורה שמיד אחריה. אין תמיכה במשפטי else או בתנאים מקוננים.
4. write “message” – רישום הודעה לחלון ה-Console של התהליך. לכל תהליך חלון Console משלו אליו יש לרשום הודעות. “message” יכול להיות מחרוזת כלשהי, או שרשור באמצעות האופרטור + של מחרוזות ומשתנים, לדוגמה write x + “=” + y.
5. goto 12 – מעבר לשורה 12 בקוד. ניתן להשתמש ב-labels בקוד. השורה LABEL: מגדירה תווית ששמה הוא LABEL. ניתן להשתמש בקוד ב-goto LABEL במקום להכניס מספרי שורות. שורות ה-LABEL: מתורגמות לערכים מספריים בזמן קריאת הקוד מהדיסק ולכן אינן שורות קוד "אמיתיות" אותן צריך להריץ ה-CPU. דוגמה ניתן לראות בקובץ d.code"".
6. read file token\_number variable – קריאה מקובץ. שורות אלו פונות לדיסק על מנת לבקש token מתוך קובץ. הסימולטור תומך רק בקבצי טקסט המכילים מספרים עם רווחים ביניהם. הפקודה מבקשת את ה-token ה-token\_number מתוך הקובץ file ושמה את ערכו (המספרי) בתוך המשנה variable. דוגמא: read data.txt 10 result קוראת את המספר העשירי מהקובץ data.txt ושמה את ערכו במשתנה result. במידה והקובץ נגמר מוחזר במשתנה הערך EOF הממומש בסימולטור על ידי double.NaN. כלומר, ניתן לאחר ביצוע פעולת הקריאה לבדוק if result != EOF על מנת לראות האם ישנם 10 מספרים בקובץ data.txt בדוגמא להלן. הפקודה read מעבירה את התהליך הקורא למצב חסום (blocked) עד לסיום הפקודה.
7. yield – פקודה זו משחררת את ה-CPU מבלי להעביר את התהליך למצב חסום.

כחלק מהחבילה ניתנו כמה דוגמאות של קבצי קוד תחת התיקייה code files. ניתן ליצור קבצי קוד חדשים כרצונכם לפי החוקים על מנת לבדוק את עבודתכם.

**שלבי העבודה: ניהול תהליכים**

1. מימוש החלפת תהליכים על ה-CPU (context switch) – ממשו את השיטה context switch במחלקה OperatingSystem המחליפה את התהליך הנוכחי על ה-CPU אם קיים כזה (אם CPU.ActiveProcess != -1) בתהליך החדש iEnteringProcessId. המידע הנדרש קיים בטבלת התהליכים m\_dProcessTable, ויש לעדכן את הטבלה בפרטי התהליך היוצא. לאחר מימוש חלק זה ניתן להריץ את פונקציית הדוגמא Example1 מה-main. השיטה מחזירה את כניסת הטבלה " ProcessTableEntry" של התהליך היוצא מה-CPU. בהמשך תאלצו להרחיב את השיטה על מנת לתמוך במדיניות התזמון השונות.
2. ממשו את השיטה Interrupt במחלקה OperatingSystem - שיטה זו מופעלת על ידי ה-Disk עם סיום פעולת קריאה, בדומה לפסיקות חומרה. על השיטה לבצע את הפעולות הבאות:
   1. תרגום ה-Token המוחזר בתוך ה-ReadTokenRequest ל-double. במידה ומוחזר null המשמעות היא שהגענו לסיום הקובץ ולכן יש להחזיר את הערך double.NaN המשמש לסימון end of file (EOF).
   2. שמירת הערך בתוך המשתנה המתאים (על פי ה-ReadTokenRequest) בתוך ה-AddressSpace של התהליך שיצר את בקשת הקריאה.
   3. באם יש בקשות נוספות לקריאה הממתינות בתור m\_lReadRequests יש להפעיל את הדיסק על הבקשה הבאה (הראשונה בתור).
   4. לבסוף, אם ה-scheduler דורש הפעלה לאחר interrupt, יש לקרוא לו (הקוד כבר קיים בסוף השיטה).
3. הוספת תהליך idle – הריצו את פונקציית הדוגמא Example2. שימו לב כי הריצה מסתיימת לפני שכל התהליכים סיימו לקרוא מהקובץ כרגע.
   1. ממשו את המחלקה IdleCode כך שתממש תהליך idle שמריץ לולאה ריקה (goto 0).
   2. הוסיפו לבנאי של המחלקה OperatingSystem אתחול של תהליך המריץ את ה- IdleCode (התבוננו בשיטה CreateProcess כדי להבין מה נדרש בכדי ליצור תהליך).
   3. הריצו כעת את הדוגמא Example2 וודאו כי כל התהליכים מסיימים לקרוא מהקובץ לפני שמסתיימת ריצתם.
4. הוסיפו תמיכה בפקודה yield - הפקודה גורמת לתהליך הפעיל לשחרר את ה-CPU מבלי להפוך לחסום. במחלקה CPU, בשיטה ExecuteLine ממשו את המקרה yield. שימו לב לערכי ה-ProgramCounter בכדי להבטיח שהתוכנית הבאה שמתוזמנת ממשיכה באופן תקין. שנו את הקוד של IdleCode כך שיכלול שתי שורות – yield ו-goto 0. הריצו את Example3 ובדקו שהיא רצה באופן תקין.

**תזמון:**

כעת נממש מספר שיטות שונות לתזמון תהליכים. כל השיטות יממשו את המחלקה האבסטרקטית SchedulingPolicy. בכדי לבחור שיטת תזמון נעביר אותה למערכת ההפעלה בבנאי. אין צורך לכתוב קוד יעיל או חסכוני במיוחד למימוש התזמון. כמו כן, שיטות התזמון השונות דומות זו לזו ורצוי לעשות שימוש חוזר (reuse) של הקוד על ידי תורשה. דוגמה למימוש של מדיניות אפשר לראות במחלקה HighestIndexPolicy

1. ממשו את שיטת התזמון FirstComeFirstServedPolicy (על ידי מימוש השיטות במחלקה המתאימה). מדיניות תזמון זו מחזיקה תור קבוע של תהליכים כאשר לאחר שתהליך יורד מה-CPU (בגלל סיום, חסימה, או yield) מופעל התהליך הבא בתור שמוכן לרוץ (כלומר לא חסום ולא סיים). שיטה זו הינה **non-preemptive**, כלומר התהליך הפעיל ממשיך לרוץ כרצונו בלי quantum וללא יכולת של מערכת הפעלה להפסיק את ריצתו.
2. ממשו את שיטת התזמון RoundRobin הפועלת כמו FirstComeFirstServed אך מוסיפה quantum לתהליך (ה-quantum מועבר כפרמטר לבנאי). עם סיום ה-quantum מבוצעת החלפת תהליכים, גם אם התהליך הנוכחי רוצה להמשיך לרוץ (preemption). בכדי להפעיל את השעון יש לקבוע את שעון ה-CPU בשדה ה-RemainingTime ל-quantum המבוקש לפני הפעלת התהליך. יש להשתמש בשדה Quantum שבטבלת התהליכים.
3. ממשו את שיטת התזמון PrioritizedScheduling הפועלת כמו RoundRobin אך מוסיפה עדיפויות לתהליך (העדיפות מוגדרת בעת יצירת התהליך). גם שיטה זו מבצעת preemption - עם סיום ה-quantum מבוצעת החלפת תהליכים, גם אם התהליך הנוכחי רוצה להמשיך לרוץ. בשיטה זאת יופעל התהליך המוכן עם העדיפות הגבוהה ביותר. השיטה תבצע round robin בין תהליכים עם עדיפות שווה.
4. לסיום, עליכם לממש שני מדדים שיסיעו להדגמת ההבדלים בין שיטות התזמון.
   1. ממשו במחלקה OperatingSystem את השיטה **AverageTurnaround** המחשבת את הזמן הממוצע מרגע הגעת תהליך למערכת ועד לסיומו. יש להשתמש בשדות StartTime ו-EndTime בטבלת התהליכים כדי לשמור את הערכים הנדרשים.
   2. ממשו את השיטה **MaximalStarvation** המחשבת את הזמן המקסימלי בו תהליך כלשהו היה מוכן לרוץ אך לא קיבל זמן CPU (באופן פרטני עבור העבודה -זמן זה יכלול גם את הזמן שתהליך נמצא במצב blocked) כלומר פשוט כל זמן שהוא לא על הCPU. השתמשו בשדה MaxStarvation בטבלת התהליכים כדי לשמור את הערכים הרלוונטיים.

השדה TickCount במחלקה CPU מייצג את שעון המערכת – מספר הטיקים שחלפו מאז ההפעלה.

לנוחיותכם, ניתן להשתמש בתכונה Debug של המחלקה CPU בכדי להציג את השורה הנוכחית והתהליך הפעיל.

הגינות – את העבודות יש לממש **לבד (לא בזוגות).** אין לשלוח קטעי קוד בין סטודנטים. אין להעתיק קטעים מעבודות קיימות או מסטודנטים אחרים. אל תעתיקו!