שאלה 1 (5%)

מה הוא התפקיד העיקרי של system calls במערכת מחשב?

- ביצוע פעולות של מנהל המערכת
- ביצוע פעולות הקומפיילר בזמן ביצוע קומפילציה
- ביצוע פעולות ב kernel mode בעקבות בקשת תהליכי משתמש
- ביצוע פעולות ב user mode בעקבות בקשת תהליכי משתמש
- ביצוע פעולות שדורשות הרבה מאוד זמן, למשל, כפל מטריצות ענקיות

שאלה 2 (5%)

מה היא הדרך הנכונה **והסבירה** למניעת deadlock במערכת עם 4 RESOURSES?

- כל ה processes יוכלו לקבל את RESOURCES רק לפי סדר עולה ולשחרר לפי סדר יורד
- כל process ישחרר את כל RESOURCES שהוא **החזיק בהם** (כולל אמצעי סנכרון) לפני שיסתיים
- כל ה processes יוכלו לבקש ולהחזיר רק RESOURSE אחד בכל נקודת זמן של חיי התהליך
- כל process יוכל לבקש לא יותר מ 2 RESOURCES
- כל process יוכל לבקש לא יותר מ 3 RESOURCES

שאלות 3-5

בשביל כל אחת מ 3 בעיות ה synchronization שבהמשך (שאלות 3,4,5), בחרו את אמצעי ה synchronization הטוב (היעיל) ביותר בכדי לפתור את הבעיה.

אין צורך לממש אלגוריתם שלם.

באלגוריתם יכולים להיות מבני נתונים/משתנים שהכרחיים לפתרון, אין צורך להתייחס אליהם.

אין צורך לדאוג כיצד מתאפשרת הגישה לאמצעי ה synchronization, צריך להניח שיש גישה.

שאלה 3 (5%)

צוות מתכנתים מפתח מערכת דיגיטלית לקביעת תורים לרופא. המערכת אמורה לאפשר לכל המעוניינים ביצוע מקבילי של פעולות הבאות:

1) קביעת תור לשעה פנויה.

2) ביטול תור.

3) הוספת תורים ע"י רופא. רופא לא מבטל תורים וכשמוסיף, תמיד מוסיף מספר משמעותי של תורים.

כל מי שמתחבר למערכת מטופל כ THREAD נפרד ואם אין מקום פנוי, הלקוח מתנתק ו- THREAD שנוצר עבורו במערכת מחכה לתור שיתפנה ויודיע לו דרך שליחת מסרון טלפוני.

הציעו אמצעי synchronization שיאפשר פתרון יעיל לדרישות

- Lock and Condition variable with Condition broadcast
- 3 MUTEX (LOCK)
- 1 MUTEX (LOCK)
- 2 MUTEX (LOCK)
- Lock and Condition variable with Condition signal

שאלה 4 (5%)

במשחק מחשב קיימת בעיית מעבר בגשר צר שמעבר בו אפשרי רק בכיוון אחד.

מכוניות נעות במהלך משחק מחשב כשכל אחת היא THREAD נפרד. כשקיימות מכוניות הנוסעות בכיוון מסוים, מכוניות שרוצות לנוע בכיוון נגדי צריכות להמתין עד שייפסקו תנועת כל המכוניות בכיוון הנוכחי. ידוע מניסיון, שאף פעם לא מצטברות יותר מ 20 מכוניות שמחכות (אין צורך לטפל במקרה שמספרם יותר גדול), אבל מספרם יכול להיות גם נמוך יותר. כמו כן, אין צורך לדאוג למניעת התנגשויות בין המכוניות הנעות באותו כיוון.

בעזרת אילו אמצעי synchronization ניתן לפתור בקלות את הבעיה?

האמצעי/ם חייב/ים להיות יעילים ביותר.

עקרונית **מותר להשתמש בפתרון בנוסף גם במשתנים רגילים** ואין צורך להתייחס לזה.

- 2 Counting semaphore
- Lock and Condition variable with Condition broadcast
- 2 MUTEX (LOCK)
- Lock and Condition variable with Condition signal
- 2 Binary semaphore

שאלה 5 (5%)

צוות מתכנתים מפתח מערכת ניהול מקומות חניה במבנה רב קומתי. המערכת אמורה לאפשר לכל המעוניין ביצוע מקבילי של פעולות הבאות מכמה עמדות שמחוברות לאותה מערכת:

1) תפיסת מקום חניה.

2) רישום לחניה בקומה מסוימת. בכל קומה קיימים כמה מקומות חניה.

3) קבלת הודעה טלפונית מתי שמקום חניה מתפנה.

4) שחרור מקום ביציאה מן החניון, גם לזה קיימות כמה עמדות שגם כן מחוברות לאותה מערכת.

כל המתחבר למערכת מטופל כ THREAD נפרד שמטפל בהרשמה/ביטול/שחרור מקום, מחכה עד שמקום מתפנה ושולח הודעה טלפונית . עקרונית **מותר להשתמש בפתרון בנוסף גם במשתנים רגילים** ואין צורך להתייחס לזה.

הציעי אמצעי synchronization שיאפשר פתרון יעיל לדרישות?

- 3 Counting semaphores
- 2 Counting semaphores
- Lock and Condition variable with Condition broadcast
- 1 MUTEX (LOCK)
- Lock and Condition variable with Condition signal

שאלה 6 (5%)

נתונה התוכנית הבאה (pseudo code) שמיועדת ל SYNCHRONIZATION-בין מספר THREADS (1T ו 2T לדוגמא, אבל יכולים להיות יותר) שרצים במקביל. כל THREAD מבצע אותו פרוטוקול כניסה ויציאה ל CRITICAL SECTION.

התוכנית משתמשת במשתנים גלובליים משותפים שנגישים לכל ה THREADS

Protocol 1

```
bool lock = false; //global shared
```

```
T1:
while(true){

// non critical section

while (lock); // busy wait
lock = true;

   // critical section

lock = false;
// non critical section
    }
```

```
T2:
while(true){

// non critical section

while (lock); // busy wait
lock = true;

   // critical section

lock = false;
// non critical section
    }
```

בחרו את הטענה הנכונה:

- הפרוטוקול פותר את בעיית ה CRITICAL SECTION אבל רק אם יש 2 THREADS ולא יותר
- שני THREADS לכל היותר יכולים לשהות בו-זמנית ב CRITICAL SECTION
- הפרוטוקול פותר את בעיית ה CRITICAL SECTION.
- מספר threads יכולים לשהות בו-זמנית ב CRITICAL SECTION
- כמה threads עלולים להיכנס למצב DEADLOCK

שאלה 7 (10%)

נתונה התוכנית הבאה (pseudo code) שמיועדת ל SYNCHRONIZATION בין **שני** THREADS - T1 and T2 שרצים במקביל.

כל THREAD מבצע אותו פרוטוקול כניסה ויציאה ל CRITICAL SECTION.

התוכנית משתמשת במשתנים גלובליים משותפים שנגישים ל 2 ה THREADS

שאלה 7 (10%)

נתונה התוכנית הבאה (pseudo code) שמיועדת ל SYNCHRONIZATION בין **שני** THREADS - T1 and T2 שרצים במקביל.

כל THREAD מבצע אותו פרוטוקול כניסה ויציאה ל CRITICAL SECTION.

התוכנית משתמשת במשתנים גלובליים משותפים שנגישים רק ל 2 ה THREADS

Protocol 2

```
int turn = 1;   //global shared
```

```
T1:
while(true){
// non critical section

while (turn == 2); // busy wait

   // critical section
turn = 2;

// non critical section
    }
```

```
T2:
while(true){
// non critical section

while (turn == 1); // busy wait

   // critical section
turn = 1;

// non critical section
    }
```

בחרו את הטענה הנכונה:

- אף תשובה מבין המופיעות איננה נכונה
- שני תהליכים יכולים לשהות בו-זמנית ב CRITICAL SECTION
- הפרוטוקול פותר את בעיית ה CRITICAL SECTION ועומד בכל דרישות הפתרון
- הפרוטוקול מבטיח קדימות של T1 על פני המתחרה בכניסה ל CRITICAL SECTION
- הפרוטוקול מבטיח קדימות של T2 על פני המתחרה בכניסה ל CRITICAL SECTION

```
interest = true;                    interest = true;
turn = 1;                           turn = 0;

while(interest==true && turn==1);   while(interest==true && turn==0);
        // busy wait                        // busy wait

   // critical section                    // critical section
interest = false;                   interest = false;

// non critical section            // non critical section
      )                                   )
```

בחרו את הטענה הנכונה:

- שני ה threads עלולים להיכנס למצב DEADLOCK
- הפרוטוקול פותר את בעיית ה CRITICAL SECTION
- הפרוטוקול אינו תקין ויכול לגרום להקלה ברמת חומרה, בגלל שלשני ה THREADS יש אפשרות לגשת למשתנים משותפים בו-זמנית
- שני תהליכים יכולים לשהות בו-זמנית ב CRITICAL SECTION
- הפרוטוקול לא מאפשר שיהיה בו-זמנית בקטע קריטי גם מונע DEADLOCK, אבל הוא לא תקין כי THREAD מחוץ לקטע קריטי יכול למנוע מ THREAD אחר להיכנס אליו

שאלה 9 (10%)

מערכת הקבצים של מערכת הפעלה מסוימת משתמשת בשיטת ה I-node עם פרמטרים הבאים:

- גודל הבלוק במערכת הקבצים הוא 2 Kbytes
- כתובת הבלוק היא 4 בתים (bytes)
- 10 שדות של ה I-node יכולים להחזיק ישירות כתובת הבלוק בדיסק
- שדה נוסף אחד נועד להחזיק כתובת של ה single indirect block
- עוד שדה נוסף אחד נועד להחזיק כתובת של ה double indirect block
- ועוד שדה נוסף אחד נועד להחזיק כתובת של ה triple indirect block

חשבו מה הגודל המרבי של קובץ שניתן לאכסן ב 1514 בלוקים בסה"כ (כולל נתונים ומצביעים, אבל לא כולל את ה I-NODE עצמו) .

הערה: לצורך החישוב 1 Mbyte = 1024 Kbyte , 1 Kbyte = 1024 byte

- 3020 Kbytes
- 3028 Kbytes
- 3008 Kbytes
- 3030 Kbytes
- 3000 Kbytes

שאלה 10 (5%)

מה נכון לגבי HYPERVISOR מסוג 1 בתצורת bare metal?

- רק הקצאות זיכרון ראשי עוברות דרך hypervisor, אבל פעולות קלט-פלט מבוצעות **ישירות** ע"י ה OS האורחת **בלי שיתוף** של hypervisor
- כל משאבי החומרה hardware resources נשלטים **באופן ישיר** ע"י hypervisor
- רק הקצאות זיכרון ראשי ופעולות קלט-פלט עוברות דרך hypervisor, אבל הקצאת זמן המעבד CPU מבוצעת **ישירות** ע"י ה OS האורחת **בלי שיתוף** של hypervisor
- כאשר יישום המשתמש מבצע את system calls ל-OS שרצה על VMM , אותה ה OS **לא תנסה** לבצע את ה- system calls ב kernel mode
- HYPERVISOR מסוג 1 בעצמו פועל ב USER MODE

שאלה 11 (5%)

מה נכון לגבי ניהול זיכרון ראשי (RAM) מדפדף (PAGING)?

- **בכל פניה** לזיכרון ראשי RAM יהיה צורך בתרגום כתובת וירטואלית VIRTUAL ADDRESS לכתובת פיזית PHYSICAL ADDRESS.
- גודל הדף הווירטואלי תמיד צריך להיות שווה לגודל הדף הפיזי (מסגרת זיכרון פיזית), חוץ מהדף האחרון שיכול להיות יותר קטן
- זמני ביצוע גישות לכתובות וירטואליות VIRTUAL ADDRESSESS שונות תמיד אחידים
- כתובת וירטואלית VIRTUAL ADDRESS תמיד באותה כתובת פיזית לכל אורך חיי התהליך
- זמן טיפול בכל שגיאת דף בודדת PAGE FAULT תמיד אחיד

שאלה 12 (5%)

מה נכון לגבי מערכת הפעלה שמשתמשת ב COW – COPY ON WRITE כשיטת ניהול זיכרון ראשי RAM בזמן יצירת תהליך בן ע"י קריאת מערכת FORK לפני ביצוע קריאות מערכת נוספות?

- איזור הקוד של תהליך האב יהיה מועתק למרחב הזיכרון של הבן, אבל איזור הנתונים לא מועתק ויישאר משותף לאב ולבן.
- חלק ממרחב הזיכרון של תהליך האב המתחיל מהנקודה שבה נוצר תהליך הבן מועתק למרחב הזיכרון של הבן.
- בזמן יצירת תהליך הבן שום דבר לא מועתק, שני התהליכים משתפים את אותה טבלת הדפים שנמצאת במרחב הזיכרון של האב כשלבן ניתנת גישה אליה. העתקת טבלת הדפים תתבצע בזמן PAGE FAULT אצל הבן.
- ברגע יצירת תהליך הבן, טבלת הדפים של תהליך האב מועתקת למרחב הזיכרון של הבן ולפני ביצוע פעלה כלשהי ע"י הבן או האב, שתי הטבלאות מצביעות על אותן כתובות פיזיות.
- בזמן יצירת תהליך הבן, כל מרחב הזיכרון של תהליך האב מועתק למרחב של הבן.

13. (10 נקודות) בהשוואה בין pipeline / boss-workers models , נתונים 8 threads שעובדים במקבילות אמיתית מלאה ו 11 הזמנות ייצור. את הביצוע של כל הזמנה ניתן לחלק לכמה שלבים שצריך. ל boss-workers model לוקח 80 ms לבצע הזמנה אחת, ל pipeline לוקח 10 ms לכל שלב.

תחשבו זמן הממוצע לביצוע הזמנה אחת עד לסיומה(כולל זמני המתנה) לכל מודל.

	boss-workers	pipeline
1)	102	180
2)	160	180
3)	102	130
4)	80	80
5)	109	130

לה 14 (5%)

הסיבה העיקרית לשימוש ב-DMA (גישה ישירה לזיכרון) בביצוע פעולות קלט-פלט?

- הקטנת העומס על הזיכרון הראשי.
- למעבד CPU דרוש פחות זמן לביצועה פקודה בודדת.
- מאפשר להתקן, למשל דיסק, להשתמש ב RAM כמטמון.
- שיפור ביצועי המערכת ע"י הגדלת המקביליות
- מאפשר לאותו התקן, למשל דיסק, לבצע כמה פעולות קלט-פלט במקביל.

שאלה 15 (5%)

על ידי מי ניתן ליצור בפועל איזור של זיכרון משותף SHARED MEMORY בין שני תהליכים PROCESSES?

- ע"י התהליך הראשון שניגש לאזור שעתיד להיות משותף.
- ע"י מערכת הפעלה בעקבות בקשת התהליך.
- תלוי במספר ליבות CORES שיש בתוך המעבד.
- ע"י התהליך השני כשהוא ניגש לאזור השייך לתהליך הראשון.
- תלוי בגודל האזור של זיכרון משותף

שאלה 16 (5%)

מה נכון בהשוואת 2 ארכיטקטורות של מחשוב הענן: Homogeneous ו Heterogeneous, עם 100 nodes שצריכים להריץ מספר משתנה של משימות מאותו סוג:

- בעומס נמוך לא יהיה הבדל, אבל בעומס גבוה Heterogeneous תהיה יותר מהירה בגלל אפשרות של התאמה דינאמית במספר nodes המבצעים את העבודה
- לא יהיה הבדל במהירות הביצוע בין 2 הארכיטקטורות
- Heterogeneous תהיה יותר מהירה
- בעומס נמוך לא יהיה הבדל, אבל בעומס גבוה Homogeneous תהיה יותר מהירה בגלל אפשרות של התאמה דינאמית במספר nodes המבצעים את העבודה
- Homogeneous תהיה יותר מהירה