אוניברסיטת בן-גוריון בנגב, המחלקה למדעי המחשב

בוחן אמצע במערכות הפעלה (25 במאי 2011)

מרצים: דני הנדלר ורועי זיוון; מתרגלים: אלון גרובשטיין,איליה מירסקי, אמיר מנצ'ל ודולב פומרנץ

משך המבחן: שלוש שעות
חומר עזר: אסור
ענו על כל השאלות: סה"כ 100 נקודות.

1. (30 נקודות) תזמון (scheduling)

א. (5 נ') . תארו בקצרה אך באופן מדוייק וברור את אופן פעולתם של שני האלגוריתמים הבאים לתזמון: Round robin ו-Guaranteed scheduling. בפרט, תארו איזו "הגינות" (fairness) בתזמון מבטיחים שני האלגוריתמים.

פתרון:
Round robin – הינו אלג' preemptive, בעל פרמטר המגדיר את משך ה time slice אשר יקבל כל תהליך. האלג' בוחר תהליכים לריצה מבין התהליכים שהינם במצב ready על פי סדר מעגלי.
הוגנות: שואף לתת זמן ריצה שווה לתהליכים במצב ready. האלג' מבטיח שאין starvation מכוון שזמן ההמתנה של כל תהליך חסום במספר התהליכים הממתינים כפול גודל ה time slice.
Guaranteed scheduling – הינו אלג' preemptive, השואף לתת זמן ריצה שווה לכל התהליכים. עושה זאת על ידי שמירה של ערך המציין את היחס בין הזמן שהתהליך קיבל בפועל לבין הזמן שמגיע לו. האלג' יבחר להריץ את התהליך עם הערך הקטן ביותר.
הוגנות: האלג' בוחר בכל שלב את התהליך ה-"מקופח" ביותר, ובכך מבטיח שלא יהיה starvation. חישוב היחס מתחשב גם בהתהליכים שיצאו ל I/O ובכך יפצה אותם על הזמן שיאבדו.

ב. (5 נ') . נניח כי במערכת קיימים מספר חוטי קרנל (kernel threads) שהם CPU-bound ומספר חוטי קרנל שהם I/O-bound. איזה מבין שני האלגוריתמים מתאים יותר עבור מערכת כזו? נמקו בקיצור אך באופן מדוייק וברור.

פתרון:
במקרה זה נעדיף את אלגוריתם Guaranteed scheduling, וזאת מפני שהאלגוריתם יוודא שתהליכי ה IO-bound אשר לרוב ישתמשו בפחות זמן מעבד יקבלו אפשרות לרוץ באשר הם מוכנים. במובן כזה האלגוריתם שלנו יהיה קרוב יותר ל SJF מכוון שיעדיף תהליכים אשר יהיו זקוקים לזמן מעבד קצר יותר, ובנוסף יהיה הוגן יותר מבחינת הזמן הכולל שיקבל כל תהליך. תחת R.R. לעומת זאת, לאורך זמן תהיה העדפה ברורה (מבחינת זמן מעבד) לתהליכי ה CPU-bound.

ג. (8 נ') . כאשר עסקנו במוניטורים, תיארנו את המימושים הבאים. 1) מימוש מוניטור בסמנטיקה של Hoare (בו מובטח כי החוט המקבל את הסיגנל הוא הבא לרוץ במוניטור), ו-2) מוניטור של Java, בו לא מתקיימת הבטחה שכזו ובכל פעם החוטים (threads) המעוניינים להיכנס אליו ונמצאים במצב ready מתחרים על הכניסה.

נניח כי במערכת ישנם מספר חוטים המשתמשים במוניטור. האם הבחירה בין שני האלגוריתמים הנ"ל לתזמון תשפיע יותר על מימוש המשתמש במוניטור מטיפוס Hoare או על מוניטור של Java? נמקו תשובתכם בקיצור אך באופן מדוייק וברור.

פתרון:

הבחירה תשפיע יותר על מוניטור של Java, וזאת מכוון שבמצב זה אלגוריתם התזמון יחליט למעשה מי התהליך שיכנס למוניטור. אם יהיה זה Round robin, אז יהיה זה התהליך המעוניין הבא שניתקל בו במעבר המעגלי. אם יהיה זה Guaranteed scheduling, אזי התהליך שיכנס למוניטור יהיה התהליך המעוניין שקיבל את יחס הזמן הנמוך ביותר. במוניטור מטיפוס Hoare, המוניטור למעשה יבחר מי התהליך הבא שיוכל להכנס, ולכן לאלגי התזמון לא תהיה השפעה רבה על הריצה.

ד. (12 נ') . נניח כי במערכת בעלת מעבד (CPU) יחיד ישנה קבוצת חוטים אשר כל אחד מהם רץ תמיד משך זמן T לפני שהוא עובר להמתנה ל-I/O וכי context switch אורך זמן S. נניח גם כי במערכת מופעל אלגוריתם תזמון round robin עם time-quantum באורך Q. ניתן להניח כי תמיד ישנם במערכת חוטים המוכנים לרוץ (אינם ממתינים ל-I/O). חשבו את ניצולת ה-CPU במערכת כפונקציה של פרמטרים אלו עבור המקרים הבאים:

i) Q>T
ii) S<Q<T
iii) Q שואף לאפס.

פתרון:

I. במקרה זה Q חסר משמעות. כל תהליך ירוץ זמן T ואז נבזבז S זמן ב C.S. לכן ניצולת המעבד תהיה:

$$\frac{T}{T+S}$$

II. התקבלו תשובות בהן הונח ש $\frac{T}{Q} \in \mathbb{N}$. במצב כזה נקבל ניצולת מעבד:

$$\frac{T}{T+\frac{T}{Q}S} = \frac{1}{1+\frac{S}{Q}} = \frac{Q}{Q+S}$$

III. כאשר $Q \to 0$ נקבל על פי הנוסחה מהסעיף הקודם:

$$\lim_{Q\to 0} \frac{Q}{Q+S} = \frac{0}{S} = 0$$

2. (35 נקודות) תהליכים וחוטים

א. (15 נק׳) . מהם הפלטים האפשריים של התוכנית הבאה, על המסך ובקובץ a.txt? עבור כל פלט, ציינו האם הוא מודפס על ידי התהליך האב, תהליך בן או חוט. הניחו כי התוכנית רצה על מערכת עם מעבד יחיד, וכי פתיחת הקובץ מצליחה. הניחו כי בכל תהליך המזהים של החוטים נקבעים בסדר עולה, החל מ-1. בנוסף, הניחו כי כאשר תכנית מרובת חוטים מבצעת פעולת fork רק החוט אשר ביצע את הקריאה משתכפל. הניחו גם כי כל פעולת הדפסה היא אטומית (כלומר, היא לא תופרע באמצע על ידי context switch).

```
int x = -1;

void foo() {
        printf("x=%d\n", x);
        x = pthread_self(); // get thread ID
}

int main() {
        int fd;
        pthread_t tid[2];

        printf("STARTED\n");
        fd = dup(STDOUT_FILENO);
        close(STDOUT_FILENO);
        open("a.txt", O_WRONLY | O_CREAT | O_TRUNC);

        pthread_create(&tid[0], NULL, foo, NULL);
        fork();
        pthread_create(&tid[1], NULL, foo, NULL);
        sleep(10);  // sleep for 10 seconds
        printf("after fork x=%d\n", x);

        close(STDOUT_FILENO);
        dup(fd);
        printf("FINISHED\n");

        return 0;
}
```

On screen:	a.txt:
STARTED FINISHED FINISHED	x = -1 (parent) x = -1/1/2 (parent) x = -1/1 (child) after fork x = 1/2 (parent) after fork x = 1 (child) * first 3 lines can come in any order * last 2 lines could come in any order

ב. (10 נק'). הניחו כי ברשותכם שרת אינטרנט המספק בקשות משתמשים. לשרת מבנה נתונים בו הוא שומר ערכים שהובאו עבור בקשות שהתבצעו לאחרונה. בכל פעם שמתקבלת בקשה, מתבצעת פעולת חיפוש מהירה במבנה הנתונים. במידה והמידע הנדרש אינו מצוי במבנה הנתונים, מתבצע תהליך אחזור מתוך דיסק (I/O). הניחו כי לשרת מעבד יחיד. כמו כן הניחו כי לא ניתן לבצע שתי פעולות I/O לדיסק במקביל. נסמן ב-h את ההסתברות למציאת המידע הנדרש במבנה הנתונים, ב-c את הזמן הנדרש לאחזור מידע ממבנה הנתונים וב-t את משך הזמן הנדרש לאחזור מידע מהדיסק (ברור כי t>c). בפרט, כאשר המידע הנדרש לא נמצא במבנה הנתונים, משך האיחזור הכולל הוא t+c (ראשית יש לקרוא את המידע מן הדיסק למבנה הנתונים ואז לקראו ממבנה הנתונים). נתון כי שתי בקשות בלתי תלויות זו בזו, כל אחת מהן למידע שונה, מגיעות בו זמנית למערכת.

i. מהו משך הזמן הצפוי עד להשלמת הבקשות במערכת מבוססת חוטי משתמש (user space threads)?

ii. מהו משך הזמן הצפוי עד להשלמת הבקשות במערכת מבוססת חוטי קרנל (kernel threads)?

<u>פתרון:</u>

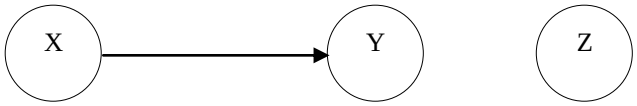
i. חוטי משתמש:
2ch^2+2(c+t)(1-h)^2+h(1-h)(4c+2t)
יש לשים לב כי המחובר השלישי סופר 2 מקרים א) הצלחה וכישלון וגם ב) כישלון והצלחה. אילו שני מקרים סימטריים שכל אחד דורש כ 2c+t זמן.

ii. חוטי קרנל:
2ch^2+ (c+2t)(1-h)^2+h(1-h)(2c+t+c+t)
יש לשים לב כי כאן המקרים אינם סימטריים. הצלחה וכישלון = 2c+t. כישלון והצלחה = c+t כי c<t.

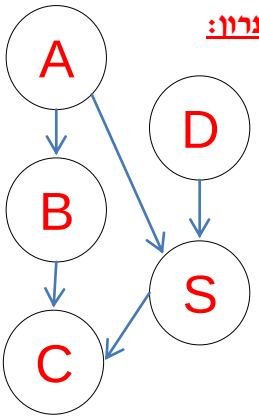
ג. (10 נק'). נתון קטע הקוד הבא:

```
void sigchld_handler(int s) {
        printf("S");
}
int main(){
        signal(SIGCHLD, sigchld_handler);
        signal_block(SIGCHLD);
        if (fork() != 0) {
                printf("A");
                signal_unblock(SIGCHLD);
                printf("B");
                wait ();
                printf("C");
        } else {
                printf("D");
        }
}
```

ידוע כי הפקודות signal_block וכן signal_unblock חוסמות ומשחררות חסימה לסיגנלים.
שרטטו גרף מכוון המתאר את הפלטים האפשריים לקוד זה. כל צומת בגרף תסמל הדפסה וכל קשת מכוונת תייצג יחס סדר מתחייב בין הדפסות. לדוגמא, אם עפ"י קוד מסוים ידוע כי יודפסו "X", "Y" ו- "Z" וכי ההדפסה של "X" תופיע בהכרח לפני ההדפסה של "Y" (אך "Z" יכול להופיע לפני או אחרי כל אחת מן ההדפסות האחרות), יתקבל הגרף הבא:



פתרון:



3. (35 נקודות) סינכרוניזציה – מניעה הדדית

א. (5 נ') . בכיתה הגדרנו את התנאים חופש מקיפאון (deadlock freedom) וחופש מהרעבה (starvation freedom) עבור אלגוריתמים למניעה הדדית. כתבו את ההגדרות **המדוייקות** של תנאים אלו.

פתרון:
חופש מקיפאון: אם קבוצת תהליכים מנסה להיכנס לקטע הקריטי, אזי לאחר מספר סופי של צעדים אחד התהליכים יצליח להיכנס אליו.
חופש מהרעבה: אם תהליך מנסה להיכנס לקטע הקריטי, אזי לאחר מספר סופי של צעדים הוא יצליח להיכנס אליו.

ב. (10 נ') . בכיתה הגדרנו תנאי הוגנות (fairness) עבור בעיית המניעה ההדדית הקרוי first-in-first-out (FIFO).

i. כתבו את ההגדרה **המדוייקת** של תנאי זה.

ii. האלגוריתם למניעה הדדית של פטרסון לשני תהליכים מובא בסוף שאלה זו. האם האלגוריתם מקיים תנאי FIFO? נמקו בקצרה אך במדוייק.

פתרון:
i. אם תהליך A ממתין (נמצא ב-waiting section) לפני שתהליך B התחיל לבצע את ה-doorway אזי B לא יכנס לפני A.
ii. אם אחד התהליכים נכנס להמתנה לפני שהשני החל לבצע את קטע הכניסה, אזי התהלך הראשון יכנס ראשון לקטע הקריטי, לפיכך אלגוריתם פטרסון אכן מקיים FIFO.

ג. (10 נ') . בשאלה זו עליכם לממש אלגוריתם למניעה הדדית עבור שלושה תהליכים – p, q ו-r. על האלגוריתם לקיים מניעה הדדית וחופש מקיפאון. בנוסף, עליו לקיים את הדרישה הבאה המבטיחה

עדיפות לתהליך p: **אם p מתחיל להמתין למנעול טרם שהתהליך המחזיק במנעול החל לבצע את קוד היציאה (exit section) שלו, אזי p הוא התהליך הבא שיכנס לקטע הקריטי.**

לצורך המימוש מותר להשתמש אך ורק במשתנים התומכים בקריאות ובכתיבות. אין להשתמש בסוגים אחרים של משתנים (בפרט, אין להשתמש בסמפורים או בפעולות כגון test-and-set). ניתן גם להשתמש באלגוריתם של פטרסון למניעה הדדית עבור שני תהליכים כאבן בניין. הסבירו בקצרה את האלגוריתם שכתבתם ונמקו בקצרה מדוע הוא מקיים את שנדרש (אין צורך בהוכחה פורמלית).

פתרון:

פתרון זה עושה שימוש באבן בניין מסוג פטרסון:

Algorithm for p	Algorithm for r	Algorithm for q

Peterson(p,qr).lock // as p0
CS
Peterson(p,qr).unlock // as p0
	Peterson(q,r).lock // as p1
	Peterson(p,qr).lock // as p1
	CS
	Peterson(p,qr).unlock // as p1
	Peterson(q,r).unlock // as p1
		Peterson(q,r).lock // as p0
		Peterson(p,qr).lock // as p1
		CS
		Peterson(p,qr).unlock // as p1
		Peterson(q,r).unlock // as p0

אם אחד מהתהליכים q או r בקטע הקריטי אזי התהליך השני לא יחל לבצע את Peterson(p,qr) טרם שהראשון יצא ואזי אם p כבר מחכה ב Peterson(p,qr) הוא הבא שיכנס. אופן אחר להסתכל על כך – p "מתחרה" תמיד רק עם אחד משני התהליכים האחרים. האלגוריתם הוא גם חסר הרעבה.

ד. (10 נ') . שלומציונה טוענת כי אלגוריתם המקיים את הדרישות של סעיף ג. בהכרח אינו מקיים חופש מהרעבה (starvation freedom). האם טענתה צודקת? נמקו תשובתכם בקצרה, אך באופן מדוייק וברור.

להלן האלגוריתם של פטרסון למניעה הדדית עבור שני תהליכים. שני הדגלים b[0] ו-b[1] מאותחלים לערך false. המשתנה turn מאותחל לערך 0 או 1 (אין זה משנה מבחינת נכונות האלגוריתם).

Algorithm for p_0	Algorithm for p_1
b[0]=true turn=0 await (b[1]=false OR turn=1) CS b[0]=false	b[1]=true turn=1 await (b[0]=false OR turn=0) CS b[1]=false

פתרון:

לא, ולראיה האלגוריתם לעיל. ומכל מקום, הדרישות של סעיף ג אינן גוררות שלא מתקיים חופש מהרעבה.

בהצלחה!