אוניברסיטת בן-גוריון בנגב, המחלקה למדעי המחשב

בוחן אמצע במערכות הפעלה (13 במאי 2011)

מרצים: דני הנדלר ואמנון מייזלס; מתרגלים: אלון גרובשטיין, אמיר גרשמן, אמיר מנצ'ל ודולב פומרנץ

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

1. (45 נקודות)

א. (25 נ') . כתבי קוד עבור n>1 תהליכים שיבצע עבורם Barrier Synchronization. כלומר, כל התהליכים מגיעים עד למחסום (Barrier) ומחכים לתהליך האחרון. כאשר מגיע התהליך ה-n, כולם עוברים. כתבי את הקוד כך שלאחר השלמת המעבר של כל התהליכים מתאפשרת הצבת אותו מחסום במקום הבא בקוד לכל אחד מן התהליכים. במלים אחרות, מצב המשתנים המרכיבים את המחסום לאחר שכל התהליכים עברו אותו זהה בדיוק למצב בהתחלה. השתמשי בשני סמפורים בינאריים in ו- out ובמשתנה counter מסוג integer המשמש כמונה. כל התהליכים יודעים את מספרם הכולל n.

הדרכה: השתמשי בסמפור in המאותחל ל-1 כבמחסום הכניסה ל- Barrier ובסמפור out המאותחל ל-0 כבמחסום המעבר ל"צד השני" של המחסום. הניחי כי הקוד מתחיל בשורות הבאות:

```
down(in);
counter++;
....
....
```

השלימי את הקוד בלא יותר מ-8 שורות נוספות. כמובן שנדרש להשתמש במספר התהליכים n.

ב. (20 נ') להלן הקוד של אלגוריתם ה-Bakery אשר ראינו בכיתה:

```
int numer[n] initially {0,..., 0}
boolean choosing[n] initially {false,...,false}

Code for procss i∈{1,...,n}

1   choosing[i]:=true
2   number[i]:=1+max{number[j] | 1 ≤ j ≤ n}
3   choosing[i]:=false
4   for j:=1 to n do
5     await choosing[j]=false
6     await (number[j]=0  OR  (number[j], j) ≥ (number[i],i))
7  od
8  Critical Section
9  number[i]:=0
```

נניח כי נחליף את שורה 5 בקוד לעיל בשורה הבאה:

```
5    if j<i then await choosing[j]=false
```

האם האלגוריתם החדש עדיין נכון? נמקו במדוייק את תשובתכם.

2. **(30 נקודות)** נתונות שתי התוכניות הבאות (נסמן את השמאלית כתוכנית A והימנית כתוכנית B):

A

```
int a = 0;

void* funcZ(void* p) {
1. int i = *(int*)p;
2. int b = 0;
3. a++; b++;
4. printf("child says: id = %d a = %d
          b = %d\n", i, a, b);
5. while (1); // endless loop
}

int main(){
  pthread_t t1, t2;
6. int x1 = 1;
7. int x2 = 2;
8. a++;
9. printf("Parent says a: %d\n", a);
10 pthread_create(&t1, NULL, funcZ,
                  (void *)&x1);
11 pthread_create(&t2, NULL, funcZ,
                  (void *)&x2);
12 pthread_join(t1, NULL);
13 pthread_join(t2, NULL);
14 printf("Parent says: all done\n");
}
```

B

```
int a = 0;

void* funcZ(void* p) {
1. int i = *(int*)p;
2. int b = 0;
3. a++; b++;
4. printf("child says: id = %d a =
          %d b = %d\n", i, a, b);
5. while (1); // endless loop
}

int main(){
6. int x1 = 1;
7. int x2 = 2;
8. Int pid1, pid2;
9. a++;
10 printf("Parent says a: %d\n",
          a);
11 if (0==(pid1=fork())) {
12     funcZ((void *)&x1);
  }
13 if (0==(pid2=fork())) {
14     funcZ((void *)&x2);
  }
15 wait(&status);
17 wait(&status);
18  printf("Parent says: all
           done\n");
```

עבור כל אחד מהמקרים הבאים ציינו עבור <u>כל אחת</u> משתי התוכניות האם קיים פלט (הדפסות) אפשרי אחד או יותר. אם קיים רק פלט אפשרי אחד – כתבו את הפלט. אם קיימים לפחות שני פלטים אפשריים – כתבו שניים מהם. הסבירו את תשובתכם.

הניחו כי במערכת יש preemtion. הניחו כי כאשר תהליך אב יוצר תהליך בן, תהליך הבן נכנס ל–ready queue ואילו תהליך האב ממשיך. בדומה, הניחו כי כאשר חוט יוצר חוט חדש, החוט החדש נכנס ל-ready queue.

א. משתמשים באלג' תזמון מסוג round robin כאשר ידוע שפרוסת הזמן מספיקה להרצת כל שורות הקוד של `funcZ` וכמו כן נתון שסדר תזמון התהליכים הוא FIFO. **(15 נ')**

ב. משתמשים באלג' תזמון מסעיף א' ובנוסף השורות "while(1)" נמחקו מהתוכניות. **(15 נ')**

3. (25 נקודות) מערכת התרגילים המעשיים – Xv6

להלן 3 פונקציות מרכזיות הלקוחות מתוך קוד המתזמן (scheduler) של Xv6:

```
01      void scheduler(void){
02        struct proc *p;
03        for(;;){
04          sti();
05          acquire(&ptable.lock);
06          for(p = ptable.proc; p <&ptable.proc[NPROC]; p++){
07            if(p->state != RUNNABLE)
08              continue;
09            proc = p;
10            switchuvm(p);
11            p->state = RUNNING;
12            swtch(&cpu->scheduler, proc->context);
13            switchkvm();
14            proc = 0;
15          } // for
16          release(&ptable.lock);
17        } // for
18      }

19      void sched(void){
20        int intena;
21        if(!holding(&ptable.lock))
22          panic("sched ptable.lock");
23        if(cpu->ncli != 1)
24          panic("sched locks");
25        if(proc->state == RUNNING)
26          panic("sched running");
27        if(readeflags()&FL_IF)
28          panic("sched interruptible");
29        intena = cpu->intena;
30        swtch(&proc->context, cpu->scheduler);
31        cpu->intena = intena;
32      }

33      void yield(void){
34        acquire(&ptable.lock);
35        proc->state = RUNNABLE;
36        sched();
37        release(&ptable.lock);
38      }
```

א. **(3 נ')** איזה אלגוריתם תזמון משמש את Xv6?

ב. **(6 נ')** הסבירו מהו אלגוריתם תזמון preemptive ומהו אלגוריתם תזמון non-preemprtive. האם האלגוריתם המשמש את Xv6 הוא אלגוריתם preemptive? נמקו במדויק את תשובתכם.

ג. **(16 נ')** בשאלה הבאה הניחו כי המערכת כבר רצה עם מספר תהליכים בה (כלומר, כל תהליך קיבל זמן ריצה לפחות פעם אחת).

1. סטודנט א' טען בפני סטודנט ב' כי לבו של אלגוריתם התזמון מצוי בשורה 12 שלאחריה ירוץ הקוד הקשור בתהליך החדש (התהליך לאחר ההחלפה). האם צדק סטודנט א'? במידה וכן רשמו מהי השורה שתרוץ לאחר שורה 12. אחרת רשמו איזה חלק (או חלקים) בקוד אחראים על החלפת התהליך. **(8 נ')**
2. סטודנט ב' חושד שהמימוש הנתון אינו יעיל. לטענתו, המנעול הנתפס בשורה 5 משתחרר רק בשורה 16, כלומר רק לאחר שהתהליך המתוזמן מסיים את פרוסת הזמן שלו. משמעות הדבר היא שבמערכת עם חומרה מרובת מעבדים תזמון התהליכים אינו יעיל – כל מעבד אחר נאלץ להמתין כל עוד לא שוחרר המנעול. האם צדק סטודנט ב'? במידה וכן הציעו דרך להגברת המקביליות בעת ריצת המערכת על חומרה מרובת מעבדים. במידה וטענתו שגויה הסבירו כיצד דואגת המערכת לשחרור הנעילה. תארו תרחיש המצדיק את תשובתכם. **(8 נ')**

בהצלחה!