אוניברסיטת בן-גוריון בנגב
מדור בחינות

תאריך הבחינה: 13.5.04
שם המרצה: פרופ' אהוד גודס
ומר רועי זיון
מבחן ב: מערכות הפעלה
מס' קורס: 202-1-301
שנה: תשס"ד סמ': ב' מועד: א'
משך הבחינה: 100 דקות
חומר עזר: אין

1. (30 נק')

תלמיד קיבל משימה לכתוב תכנית שמטרתה להריץ תכנית נתונה – prompt ע"י שימוש ב- fork ו- execvp. בנוסף נדרש התלמיד למנוע מן המשתמש "להרוג" את התכנית ע"י הקשת ctrl-c (שים/י לב כי התכנית prompt אינה מסתיימת לעולם). מצורף הפתרון שהוצע ע"י התלמיד (my_prog.c) וכן קוד התכנית prompt. (שתי התוכניות – prompt וזאת שכתב התלמיד מצורפים בדף הבא).

א. תאר במדויק את פלט התכנית כאשר הקלט הנו:
Good luck in the ^c midterm exam.
הערה: הסימן ↵ מתאר הקשת ENTER, והסימן ^c מתאר הקשת ctrl-c.

ב. האם הפתרון המוצע עונה על הגדרת התרגיל?

ג. אם תשובתך ל-ב' היא לא, כיצד היית משנה את התכנית my_prog.c (ניתן להוסיף/לשנות שורה או שתיים בקוד לכל היותר).

ד. אם בסעיף א' נחליף את ^C ב- ^Z bg מה יהיה פלט התוכנית.

my_prog.c

```
#include <stdio.h>
#include <sys/fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

void cntl_c_handler(int dummy){
  signal(SIGINT, cntl_c_handler);
}

main(int argc , char argv){
  int waited;
  int stat;
argv [o]="promt";
  signal(SIGINT, cntl_c_handler);

  if (fork() == 0){ /* son */
      execvp("prompt", argv [o]
  }
  else {            /* father -/
    waited = wait(&stat);
    printf("My son (%d) has terminated\n", waited);
  }
}
```

prompt.c

```
main(int argc , char argv){
  char buf[20];
  while(1){
    printf("Type something: ");
    gets(buf);
    printf("\nYou typed: %s\n", buf);
  }
}
```

2. (40 נק')

הנח/י 3 תהליכים שהתנהגותו של כל אחד מתוארת על ידי סדרת שניות cpu ו-i/o וחוזר חלילה. למשל, הסימון (2,1,1) מסמן משמאל לימין את הסדר 2 שניות cpu ; 1 שניה i/o; ושניה אחת cpu. ה- i/o מתבצע למסך השייך לתהליך הכותב בלבד.
זמני הגעה ודרישות התהליכים הם:

תהליך מס 1	זמן הגעה 0	דרישות (1,1,3)
תהליך מס 2	זמן הגעה 2	דרישות (2,2,2)
תהליך מס 3	זמן הגעה 3	דרישות (1,3,3)

מערכת ההפעלה מקימת שתי רמות של תור – בכל אחת מהרמות הנח/י מדיניות של – RR – Round robin לתור מספר אחד יש תמיד עדיפות על תור מספר שתיים.
תהליך הנכנס למערכת נכנס לתור מספר אחד. תהליך עובר מתור מספר אחד לתור מספר שניים אם הוא מגיע לסיום פרוסת הזמן שלו. תהליך המבצע i/o חוזר תמיד לתור מספר אחד, והוא בעדיפות על תהליך שעדיין לא התחיל.
צייר/י טבלת **Gantt** עבור תהליכים אלו (שורה אחת אופקית עבור כל תהליך, קו מוצק עבור CPU, קו מקווקו עבור i/o), חשב את זמן הסבב הממוצע (Average Turn-around Time -TA) ואת זמן התגובה הממוצע (Average Response Time -RT) עבור המקרה הבא:

א. פרוסת הזמן time-quantum = 1Sec. בתור מס' 1 ו- 2Sec בתור מס' 2 (יש במערכת Preemption).

לחישוב זמן התגובה של תהליך, הנח/י כי כל תהליך מבצע את ה-i/o שלו ככתיבה למסך שלו והמשתמש ממתין לאותה הדפסה. זמן ההדפסה הראשונה הוא תחילת פעולת ה- i/o.

ב. עכשיו הנח תור אחד בלבד עם פרוסת זמן 1 sec המתנהל לפי עדיפויות כאשר עדיפות P מחושבת לפי נוסחת Highest Response Ratio (HRR)

$$P = \frac{W + S}{S}$$

כאשר W - זמן המתנה.
S - זמן שרות מצופה המחושב על פי זמן CPU שהתהליך השתמש בו מה I/0 האחרון. אם התהליך לא השתמש ב- CPU אזי אם זה תהליך שעדיין לא התחיל S = 0.5 ואם זה תהליך שרק סיים I/0, אזי S = 0.1.
צייר את ה- GANTT עכשיו
וחשב את TA ו- RT במקרה זה.

ג. האם במקרה ב' מדיניות HRR זהה למדיניות RR? האם זה נכון תמיד? נמק או תן דוגמא נגדית.

שאלה 3
(30 נק) סעיף

נתון הקוד הבא (פתרון DEKKER) לפתרון בעיית האזור הקריטי.

```
Void main0 {
  boolean flag[2];
  int turn = 0;
   flag [0] = false
   flag [1] = false
   parbegin (P0, P1);
}
```

```
void P0()
{
    while (true)
    {
        flag [0] = true;
        while (flag [1])
                if (turn == 1)
                {
                    flag [0] = false;
                        while (turn == 1)
                                /* do nothing */;
                    flag [0] = true;
                }
        /* critical section */;
        turn = 1;
        flag [0] = false;
        /* remainder */;
    }
}
```

```
void P1( )
{
    while (true)
    {
        flag [1] = true;
        while (flag [0])
            if (turn == 0)
            {
                flag [1] = false;
                while (turn == 0)
                    /* do nothing */;
                flag [1] = true;
            }
        /* critical section */;
        turn = 0;
        flag [1] = false;
        /* remainder */;
    }
}
```

הוכח את שני התנאים הבאים:

1. Progress או Mutual Exclusion
2. No starvation.

בהצלחה!

שאלה 1:

א. פלט התכנית:

Type something: *Good luck*
You Typed: Good luck
Type something: *in the ^c*
My son 139 has terminated.

* קלט מהמשתמש מסומן באותיות נטויות

ב. הפתרון **אינו** עונה על הגדרת התרגיל כיוון שבביצוע- execvp(...) לא נשמרות תכונות הטיפול בסיגנלים (signal handling) למעט התעלמות. לכן התכנית prompt אינה מתעלמת מהקשת ^C וניתן להרוג אותה באופן זה.

ג. פתרון:

החלפת השורה:

```
signal(SIGINT, cntl_c_handler);
```

בשורה:

```
signal(SIGINT, SIG_IGN);
```

אפשרות נוספת:

הוספת השורה המודגשת:

```
if(fork() == 0){  /* son */
        signal(SIGINT, SIG_IGN);
        execvp("prompt", argv);
}
```

ד. פלט התכנית:

Type something: *Good luck*
You Typed: Good luck
Type something: *in the ^z*
This is a UNIX prompt~~>>*bg*
This is a UNIX prompt~~>>

הסבר:
^Z – מעביר את הקבוצה שב- foreground למצב- suspend.
bg – (כמו בתרגיל הראשון) מעביר את הקבוצה האחרונה ממצב suspend, לרוץ ברקע.
אבל, התכנית prompt מבקשת לבצע input, כזכור לא ניתן לבצע input מהרקע ובמצב זה נשלח סיגנל ע"י ה- tty והתהליכים חוזרים ל- suspend.

שאלה 2:

א.

CPU – ▬
I/O - ▪▪▪▪

P3					▬	▪▪▪▪	▪▪▪▪	▪▪▪▪	▬	▬		▬	
P2				▬				▬	▪▪▪▪	▪▪▪▪	▬		▬
P1	▬	▪▪▪▪	▬			▬	▬						
	0	1	2	3	4	5	6	7	8	9	10	11	12

TA = (7 + 11 + 9) / 3 = 27 / 3
RT = (1 + 6 + 2) / 3 = 9 / 3

התשובה הבאה התקבלה אף היא (הבדלים צבועים באדום):

P3					▬	▪▪▪▪	▪▪▪▪	▪▪▪▪	▬	▬	▬		
P2				▬				▬	▪▪▪▪	▪▪▪▪		▬	▬
P1	▬	▪▪▪▪	▬			▬	▬						
	0	1	2	3	4	5	6	7	8	9	10	11	12

TA = (7 + 11 + 8) / 3 = 26 / 3
RT = (1 + 6 + 2) / 3 = 9 / 3

ב.

P3					▬	▪▪▪▪	▪▪▪▪	▪▪▪▪	▬		▬		▬
P2				▬			▬	▪▪▪▪	▪▪▪▪	▬		▬	
P1	▬	▪▪▪▪	▬			▬		▬					
	0	1	2	3	4	5	6	7	8	9	10	11	12

TA = (8 + 10 + 10) / 3 = 28 / 3
RT = (1 + 5 + 2) / 3 = 8 / 3

ג. במקרה זה, בהנחה שתהליך החוזר מ- I/O הוא הראשון בתור, מדיניות HRR זהה למדיניות RR (ללא ההנחה מתקבלת מדיניות "כמעט" זהה).
התנהגות זו אינה קבועה לדוגמה:
נניח 3 תהליכים, כולם רוצים 10 שניות CPU. התהליך הראשון הגיע בזמן 0, והשניים הנותרים הגיעו בזמן 5.

P3					▬		
P2				▬		▬	
P1	▬	▬	▬				
	...	4	5	6	7	8	...

P3: (0 + 1) / 1 = 1

P2: (1+ 1) / 1 = **2**

P1: (2 + 5) / 5 = 7 / 5

בפרוסת הזמן 8 ניתן לראות כי תהליך P2 קיבל CPU בעוד שבמדיניות RR הינו מצפים שתהליך P1 יקבל CPU.

שאלה 3:

א. Progress - תהליך מחוץ ל- CS לא עוצר תהליך שמעוניין להיכנס:
תהליך שלא מעוניין להיכנס, מכבה את ה- *flag* שלו. מכאן שתהליך שמעוניין להיכנס ימצא את ה- *flag* הנגדי עם ערך *false*, ויכנס ישר ל- CS מבלי לבדוק את המשתנה *turn*.

Mutual exclusion – לא יהיו שני תהליכים ב- CS בו זמנית:
נניח ש- P0 בפנים. הפקודה האחרונה ששנתה את *flag[0]* הפכה אותו ל- *true* .
נניח ש- P1 בפנים. הפקודה האחרונה ששנתה את *flag[1]* הפכה אותו ל- *true* .
אבל, הפקודה האחרונה ששני התהליכים בצעו לפני הכניסה ל- CS הייתה פקודת ה- while,
ולפי ההנחה שני התהליכים היו "נתקעים" בלולאה ולא נכנסים ל- CS – **סתירה!**

ב. No starvation – תהליך המעוניין להיכנס לא יחכה לעד:
בה"כ נניח ש- P0 רוצה להיכנס ונמצא בתוך ה- while (אחרת הוא כבר בפנים).
P1 יכול להימצא באחד משלושה מקומות:
1) ב- CS.
2) בתוך קוד לולאת ה- while שלו עם *turn* = 0.
3) בתוך קוד לולאת ה- while שלו עם *turn* = 1.

מקרה 1: אם P1 ב- CS, בסופו של דבר הוא יצא ויציב *turn* = 0. אם יישאר בחוץ אנו חוזרים למקרה של progress. לכן נניח ש- P1 זריז ונכנס ללולאת ה- while שלו ולכן אנו במקרה 2 כאשר *turn* = 0.
מקרה 2: מכיוון ש- *turn* = 0 ואף אחד לא משנה אותו, אזי P1 יתקע בסופו של דבר בלולאה הפנימית לאחר שהציב *flag[1]* = *false*. כעת, כאשר P0 יקבל CPU, מכיוון ש- *turn* = 0 הוא יצא מהלולאה הפנימית ויגיע שוב ללולאת ה- while החיצונית. אבל *flag[1]* = *false* ולכן הוא יכנס ל- CS.
מקרה 3: אם *turn* = 1, ו- P1 מבצע את לולאת ה- while החיצונית, הוא יבצע אותה כל זמן ש- *flag[0]* = *true*.בסופו של דבר P0 יקבל CPU, התנאי בלולאת ה- while שלו יכשל והוא יכנס ל- CS. ואנו חוזרים למקרה 1.