חוברת המצורפת
לספר מערכות הפעלה

תירגול לפי נושאים

✐ פרק א: ניהול תהליכים – process management

א. כתבי תוכנית ב-c המייצרת תהליך סבא , בן ונכד. ועבור כל אחד מהם מדפיסה האם הוא סב , אבא או בן , את מס' ה-pid שלו ושל אביו.

תשובה

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>

void main()
{
	pid_t id1,id2;
	int status;
	id1=fork;()
	if (id1<0)
		printf ("err");
	else
		if (id1>0)
{
			printf ("i'm the grandfather and my id is:%d,and my father id
is:%d\n",getpid(),getppid());
			wait(NULL);
}
	else
{
		sleep;(1)
		printf("i'm the father and my id is:%d,and my father id is
:%d\n",getpid(),getppid());
	id2=fork;()
	if (id2==0)
		printf ("i'm the grandchild  and my id is:%d and my father idis %d\n",
getpid(),getppid());
}
}
```

ב. כתבי תוכנית ליצירת shell פשוט.

עליך לבצע לולאה אינסופית שתכיל:

- כתיבת Prompt של ה-shell
- קליטת שורת הפקודה
- שליחה לפונקצית Parser שתמיר את שורת הקלט מ- string לערכים במערך ע"מ לשלוח זאת לפקודת exec
- שליחה לפונקציה שתבצע את הרצת הפקודה : יצירת תהליך והרצת התוכנית של הפקודה החדשה עליו.
- חזרה ל- prompt של ה-shell

לפניך פונקצית Parser. תוכלי להעזר בה.

```
void  parse(char *line, char **argv)

{

    while (*line != '\0') {     /* if not the end of line ..... */

        while (*line == ' ' || *line == '\t' || *line == '\n')

            *line++ = '\0';   /* replace white spaces with 0    */

        *argv++ = line;        /* save the argument position     */

        while (*line != '\0' && *line != ' ' &&

              *line != '\t' && *line != '\n')

            line++;           /* skip the argument until ...    */

    }

    *argv = '\0';              /* mark the end of argument list  */

}
```

תשובה בעמוד הבא

תשובה ל-ב:

```
#include  <stdio.h>
#include  <sys/types.h>

void  parse(char *line, char **argv)
{
     while (*line != '\0') {       /* if not the end of line ....... */
          while (*line == ' ' || *line == '\t' || *line == '\n')
               *line++ = '\0';     /* replace white spaces with 0    */
          *argv++ = line;          /* save the argument position     */
          while (*line != '\0' && *line != ' ' &&
                 *line != '\t' && *line != '\n')
               line++;             /* skip the argument until ...    */
     }
     *argv = '\0';                 /* mark the end of argument list  */
}

void  execute(char **argv)
{
     pid_t  pid;
     int    status;

     if ((pid = fork()) < 0) {     /* fork a child process           */
          printf("*** ERROR: forking child process failed\n");
          exit(1);
     }
     else if (pid == 0) {          /* for the child process:         */
          if (execvp(*argv, argv) < 0) {     /* execute the command  */
               printf("*** ERROR: exec failed\n");
               exit(1);
          }
     }
     else {                                  /* for the parent:      */
          while (wait(&status) != pid)       /* wait for completion  */
               ;
     }
}

void  main(void)
{
     char  line[1024];             /* the input line                 */
     char  *argv[64];              /* the command line argument      */

     while (1) {                   /* repeat until done ....         */
          printf("Shell -> ");     /*   display a prompt             */
          gets(line);              /*   read in the command line     */
          printf("\n");
          parse(line, argv);       /*   parse the line               */
          if (strcmp(argv[0], "exit") == 0)  /* is it an "exit"?     */
               exit(0);            /*   exit if it is                */
          execute(argv);           /* otherwise, execute the command */
     }
}
```

ג. שאלות:

1. להלן תוכנית בשפת C:

```
Int main()
{
    int a=4;
    int *b=&a;   //  assume address of variable 'a' is 1234
    printf ("start: %d, %d, %d, %d\n", a, &a, *b, b);
    pid_t pid = fork(); // assume fork() succeeds
    if (pid==0)
    {
        *b=5
        sleep(5);
        printf ("child: %d, %d, %d, %d\n", a, &a, *b, b);
    }
    else
    {
      sleep(1);
      a+=2;
      sleep(10)
      printf ("father: %d, %d, %d, %d\n", a, &a, *b, b);
    }
    printf ("end: %d, %d, %d, %d\n", a, &a, *b, b);
    return 0;}
```

כמה שורות יוצגו על הצג לאחר ביצוע התוכנית הנ"ל?

1. שתיים
2. שלוש
3. ארבע
4. חמש

עפ"י התוכנית המוצגת לעיל, ניתן לומר בבטחה כי:

1. כל שורה בפלט שונה מן האחרות
2. ייתכנו שורות זהות בפלט.
3. כל שורות הפלט זהות
4. לא קיימות כלל שורות בפלט.

2. להלן תוכנית בשפת c:

```
Int main()
{
      int month=3;
      int year=2015;
      printf ("address of variables: %d, %d\n", &month ,&year);
      pid_t pid = fork();
      if ( pid == 0)
{
     Month++;
      Printf ("address of variables: %d, %d\n", &month ,&year);
Execv("/bin/cal","bin/cal,month,year");
Printf ("address of variables: %d, %d\n", &month ,&year);
}
Else
{
Month--;
printf("address of variables: %d, %d\n", &month ,&year);
}
Printf ("address of variables: %d, %d\n", &month ,&year);
Return 0;
}
```

כמה שורות יוצגו על גבי הצג לאחר ביצוע התוכנית הנ"ל?

1. אם הפקודות fork() ו-exec() יצליחו : ___4___
2. אם הפקודות fork() ו-exec() לא יצליחו : ____3_____
3. אם הפקודה fork() תצליח ו-exec() לא תצליח: ____6___

שאלות ממבחנים - משה"ח

א. קבע איזה מבין ההיגדים הוא נכון.
1. תהליך הוא אובייקט סטטי, ואילו תכנית היא אובייקט דינמי.
2. תהליך עשוי לשנות את נפח הזיכרון המוקצה לו, ואילו תכנית אינה יכולה לעשות זאת.
3. תהליך אינו יכול לפנות ישירות למערכת ההפעלה, ואילו תכנית יכולה לעשות זאת.
4. לתהליך מוקצים משאבים המנוהלים על-ידי מערכת ההפעלה, ואילו לתכנית לא.

ב. מתי נחשף המעבד לשדות ה-PCB – Process Control Block)- בלוק בקרת התהליך) של תהליך מסוים?
1. בעת השיגור (Dispatching)
2. בעת כניסת התהליך למצב פעיל (Ready/Active)
3. בעת כניסת התהליך למצב השהיה (Suspend)
4. בהתאם להחלטת המתזמן לטווח ארוך (Long Term Schedular)

ג. מתוך הסתכלות על הקוד (התוכנית) של תהליך בלבד ניתן לזהות נקודות בהן יעבור התהליך
1. ממצב רץ (running) למצב חסום (blocked/waiting)
2. ממצב רץ למצב מוכן (ready)
3. ממצב מוכן למצב רץ
4. ממצב מוכן למצב חסום

ד. כיצד נקראת הפעולה הממשת את המעבר בין תזמון (scheduling) תהליכים?
1. Context switch
2. DeadLock
3. Suspend
4. Blocked

ה. איזה נתון יש לשמור כדי שמערכת ההפעלה תוכל לתמוך במיתוג הקשר?
1. קוד התכנית של התהליך
2. מצב האוגרים ב-CPU
3. שם המשתמש
4. ההיסטוריה של מיקום התהליך בזיכרון

ו. מה חייב להופיע בגוש בקרת הקובץ- PCB- Process control Block ?
1. מספר מזהה בלבד
2. PC (Program Counter) בלבד
3. תוכן המחסנית ומספר מזהה
4. מספר מזהה, PC ותוכן המחסנית.

ז. איזה מידע אינו כלול ב-PCB?
1. Program counter
2. מידע על פעולות קלט/פלט של תהליך
3. מידע על אוגרים שהם בשימוש בזמן ביצוע תהליך.
4. מידע על מצב התהליך.

ח. מהו PPID?
1. מספר המציין את העדיפות של תהליך בנקודת זמן מסוימת.
2. מספר המזהה תהליכים בעלי הרשאה זהה.
3. מספר מזהה ייחודי של תהליך
4. מספר מזהה ייחודי של האבא של התהליך.

ט. נתונים שני תהליכים: p1 ו-p2. נתון כי p1 הוא תהליך האב של p2.
קבע איזה מבין ההיגדים הוא נכון.
1. p1.1 תמיד יסיים ריצתו לפני p2.
2. p2 תמיד יסיים ריצתו לפני p1
3. לאחר יצירת תהליךp2 תהליך p1 חייב להמתין לסיום ריצתו של p2
4. לא ניתן לקבוע בוודאות איזה משני התהליכים (p1 אוp2) יסיים קודם את ריצתו.

י. נתונה מערכת מחשב שמערכת ההפעלה שלה היא solaris . נתון התהליך P , וידוע כי הריצה של תהליך האב הסתיימה לפני הריצה של תהליך הבן) p(. נתון כי תהליך האב ותהליך הבן P אינם חולקים משאבים משותפים.

קבעי איזה מבין ההיגדים הבאים לגבי תהליך P הוא ההיגד הנכון:

א. מערכת ההפעלה תסיים את ריצתו של התהליך P מיד לאחר סיום ריצתו של תהליך האב.

ב. מערכת ההפעלה תעביר את התהליך P למצב המתנה.

ג. מערכת ההפעלה תוריד את עדיפותו של תהליך P

ד. התהליך P יאומץ) בכל מה שקשור ליחסי child- parent (ע"י תהליך מספר 0 (תהליך הinit-)

יא. להלן קוד:

```
main()
{
char *argv[]={"date",null};
execv("/bin/date",argv);
printf("hello");
}
```

מה יתרחש לאחר ביצוע הקוד?
1. אם ביצוע הפקודה execv() נכשל — יוצגו התאריך והשעה.
2. אם ביצוע הפקודה execv() הצליח — יוצג hello.
3. אם ביצוע הפקודה execv() הצליח — יוצגו התאריך והשעה.
4. אם ביצוע הפקודה execv() נכשל — יציאה מהקוד

יב. מה מבצעת הפונקציה wait()?

א. תהליך האב בודק מדי פעם האם הבנים שלו סיימו, ויכול בינתיים להמשיך לרוץ .

ב. תהליך האב נחסם וכשאחד הבנים מסיים הוא מתעורר ומטפל בענין.

ג. תהליך האב נחסם עד שכל הבנים מסיימים ואז מתעורר ומטפל בענין.

ד. תהליך האב ישן למספר השניות שנשלח כפרמטר לפונקציה

יג. משתמש הקצה מקליד את הפקודה ps. אפשר לקבוע בוודאות כי הפלט יכיל את השורה:

1. init
2. ps
3. login
4. csh

יד. כמה משתמשים מדווחים לך שמערכת ההפעלה פועלת לאט מאד. כדי לבודד את הבעיה, הפעלת את הפקודה המוצגת להלן.

```
#prstat-n 8
PID USERNAME SIZE   RSS    STATE  PRI  NICE TIME     CPU   PROCESS/
                                                          NLWP
975 user1    50m    22M    run    59   0    0:20:05  85%   run_away/1
720 root     4640K  4320K  cpu0   49   0    0:00:00  0.6%  prstat/1
451 root     23M    11M    sleep  59   0    0:00:31  0.3%  Xsun/1
711 root     7056K  4784K  sleep  59   0    0:00:00  0.1%  dtterm/1
534 root     9080K  6032k  sleep  59   0    0:00:05  0.0%  dtwm/3
654 root     2128K  1376K  sleep  100  -    0:00:00  0.0%  xntpd/1
561 root     3168K  1832K  sleep  59   0    0:00:00  0.0%  dtexec/1
560 root     328K   176K   sleep  49   0    0:00:00  0.0%  sh/1
558 root     6958K  3696K  sleep  49   0    0:00:00  0.0%  dtterm/1
Total: 73 processes, 135 lwps, load averages: 0.01, 0.01, 0.02
```

איזו פקודה תפעיל כדי לטפל בבעיה?

1. kill-KILL 975
2. kill-KILL 534
3. kill-KILL 561
4. kill-KILL 720

שאלות ממבחנים שונים

א. האם תיתכן מקביליות אמיתית (True Concurrency) במחשב חד- מעבד (Uniprocessor)?

א. כן. (במקרה של חישוב ו-IO)
ב. לא, זוהי אשליה הנוצרת בגלל פער המהירות בין פעולות ק/פ לבין מהירות מעבד.
ג. לא, המקביליות המדומה מתאפשרת בגלל היכולת לשחזר בדיוק מחדש תהליך שהופסק זמנית.
ד. לא, כי המעבד יכול לשרת רק תהליך אחד בכל "רגע".
ה. כל התשובות המפורטות צריכות להיות מסומנות.
ו. אף תשובה מפורטת לא צריכה להיות מסומנת.

ב. בגרף המצבים הבסיסי של תהליך במערכת UNIX , תהליך שמסיים עובר למצב zombie . הסיבה היא:

א. לשמור על מבנה הנתונים למקרה שנצטרך אותו שוב כשתהליך יבצע fork
ב. לשמור מידע על התהליך שסיים למקרה שהאבא שלו יבצע wait
ג. לשמור מידע על התהליך שסיים עד הפעם הבאה שהמערכת תאסוף מידע על הפעילות של משתמשים שונים לצורך חיוב (accounting) .
ד. זה מצב זמני כדי לחכות עד שנגמרות כל פעולות ה- I/O הכרוכות בסיום תהליך.

ג. מעבר תהליך ממצב "running" למצב "blocked" יכול להתרחש:
1. עקב קריאת מערכת (system call)
2. כאשר אלגוריתם תזמון ה-CPU הוא אלגוריתם מבוסס עדיפויות.
3. כאשר אלגוריתם תזמון ה-CPU הוא אלגוריתם שעובד עפ"י preemption
4. כאשר אלגוריתם תזמון ה-CPU הוא אלגוריתם מבוסס תור מעגלי (round robin) וה- time quantum הסתיים

ד. הפעולה suspend מאפשרת לתהליך אחד להשעות את ריצתו של תהליך אחר. אחד ההבדלים בין תהליך במצב suspended לתהליך במצב blocked הוא
א. התהליך יכול לבקש מעצמו לצאת מ-blocked, אבל לא מ-suspended
ב. תהליך אחר יכול במקרים מסוימים להוציא את התהליך מ-blocked, אבל לא מ-suspended
ג. רכיב חיצוני יכול במקרים מסוימים להעביר תהליך מ-blocked, אבל לא מ-suspended
ד. רכיב חיצוני יכול במקרים מסוימים להעביר תהליך מ-blocked ל-running, אבל לא מ-suspended ל-running.