מערכות הפעלה
89-231
תרגול חזרה

קרדיטים: אבשלום אלמלח, חן חג'ג', שני אלקובי, פריאל לוי, מיכל חבאני, גילעד מדמון, אופיר הניג, שמעון כהן, חן רוזנשטיין

שאלה

(1) מה יהיה פלט הקוד הבא? הניחו ש-puts מדפיסה את המחרוזת שהועברה אליה כפרמטר, בתוספת תו ירידה שורה.

a SIGUSR1 signal is pending

(2) איזו שורה מה-main השפיעה על תוצאת ההדפסה שמקורה בפונקציה check_pending? האם שורה 24 או 28? הסבירו מדוע.

שורה 28. נשים לב שקריאה ל-sigprocmask מגדירה חסימה מאותה נקודה והלאה, עד לשחרור החסימה. לעומת זאת, sigaction מגדירה חסימה שתשפיע רק בזמן ביצוע פונקציית הטיפול בסיגנל עבורו הוגדרה.

```
1  void catcher(int sig) {
2      puts("inside catcher() function");
3  }
4
5  void check_pending(int sig, char *signame) {
6      sigset_t sigset;
7
8      sigpending(&sigset);
9
10     if(sigismember(&sigset, sig))
11         printf("a %s signal is pending\n", signame);
12     else
13         printf("no %s signals are pending\n", signame);
14 }
15
16 int main(int argc, char *argv[]) {
17     struct sigaction sigact;
18     sigset_t sigset;
19
20     sigfillset(&sigact.sa_mask);
21     sigact.sa_flags = 0;
22     sigact.sa_handler = catcher;
23
24     sigaction(SIGUSR1, &sigact, NULL);
25
26     sigemptyset(&sigset);
27     sigaddset(&sigset, SIGUSR1);
28     sigprocmask(SIG_SETMASK, &sigset, NULL);
29
30     raise(SIGUSR1);
31
32     check_pending(SIGUSR1, "SIGUSR1");
33
34     // ...
35
36     return 0;
37 }
```

שאלה

```c
int main()
{
    pid_t pid1, pid2;
    int status, i, count = 0;
    pid1 = fork();
    if(pid1 == 0){
        pid2 = fork();
        if(pid2 == 0){
            for(i = 0; i < 100000; i++){
                count++;
            }
            printf("ppid is: %d\n", getppid());
            exit(0);
        }
        else{
            exit(0);
        }
    }

    waitpid(pid1, &status, 0);
    return 0;
}
```

נתון קטע הקוד הבא. הניחו שה- PID של התהליך הראשי הוא 100, ושערכי PID גדלים בקפיצות של 1. (כלומר – ה- PID של התהליך הבא שייווצר כתוצאה מהרצת קוד זה יהיה 101, וכן הלאה.)

הניחו ששורה מספר 21 מתבצעת לפני שלולאת ה- for שבשורה 9 מסיימת לרוץ.

הניחו הצלחה של קריאות מערכת/פונקציות ספריה.

לפי מה שלמדנו בתרגולים, **מה יהיה פלט הקוד?** הסבירו בקצרה.

ppid is: 1

הסבר - תהליך הבן (101) מסתיים מבלי שעשה wait לבנו (תהליך 102), לכן תהליך 102 יאומץ ע"י התהליך init (או systemd) שה-PID שלו הוא 1.

שאלה

נתון קטע הקוד הבא. הניחו שכל המשתנים הרלוונטיים הוגדרו ובמידת הצורך מאותחלים, וש- read לא נכשלה (לא החזירה 1-). הניחו שלא מתבצעות קריאות וכתיבות נוספות לקובץ במקביל.

```
int charsr = read(fdin,buf,SIZE);

int charsw = write(fdout,buf,SIZE);
```

מה הבעיה בשורת הקוד השניה?

יתכן ש-charsr יהיה קטן יותר מ-SIZE, למשל במקרה בו יש פחות מ-SIZE בתים לקריאה מהקובץ. במקרה כזה, לא נרצה לבקש לכתוב SIZE בתים, מכיוון שזה יותר ממה שקראנו לתוך buf בקריאה האחרונה מהקובץ.

הציעו פתרון לבעיה.

שינוי השורה השניה ל- int charsw = write(fdout,buf,charsr), כלומר נבקש לכתוב את מספר הבתים שהצלחנו לקרוא בקריאה האחרונה מהקובץ.

שאלה

```c
int fd;
char arr[2];

void *thread_function(void *arg){
    read(fd, arr, 2); // assume that: on success, read returns the number of bytes read
                      //and changes the second parameter passed.
                     // on error, read returns -1 and does not change the second parameter passed.
    printf("print: %s\n", arr);
    close(fd); // assume that: on success, close returns 0. on error, close returns -1.

    return NULL;
}

int main(){
    pthread_t tid;
    void *rv;

    fd = open("my_file.txt", O_RDONLY);
    pthread_create(&tid, NULL, thread_function, NULL);
    pthread_join(tid, &rv);

    read(fd, arr, 2);
    printf("print from main: %s\n", arr);

    return 0;
}
```

נתון קטע הקוד הבא. הניחו שהקובץ my_file.txt קיים, שתוכנו הוא hello world ושהתהליך מורשה לגשת אליו. מה יהיה פלט הקוד? הסבירו.

print: he

print from main: he

הסבר: ה-thread החדש שנוצר ידפיס he. לאחר מכן, הקריאה ל-read מה-primary thread תיכשל, מכיוון שה- thread השני כבר קרא ל-close. לאחר מכן, ה-primary thread ידפיס he. נשים לב שה-primary thread מדפיס ערך של משתנה גלובלי (arr) אשר נשאר ללא שינוי.

שאלה

```
1   int ctrl_c_count=0;
2   void ctrl_c(int);
3
4   void ctrl_c(int signum)
5   {
6       signal(SIGINT,ctrl_c);
7       ++ctrl_c_count;
8   }
9
10  int main()
11  {
12      signal(SIGINT,ctrl_c);
13      while(getchar()!='\n');
14      printf("ctrl-c count = %d\n",ctrl_c_count);
15      for(;;);
16      return 0;
17  }
```

הסבירו (בקוד או במילים) מהם השינויים הדרושים בקוד על מנת שהקשה על ctrl+c מה-shell לאחר ההדפסה שבשורה 14, תגרום ליציאה מלולאת ה-for שבשורה 15 ולסיום התהליך. אין לשנות את הקוד של הפונקציה ctrl_c. שימו לב שאין חובה לכתוב קוד.

דרך אחת – שמירה של פונקציית הטיפול המקורית בשורה 12, ורישומה מחדש כפונקציית הטיפול ב-SIGINT בין שורה 14 לשורה 15.

דרך שניה- רישום של פונקציית הטיפול הדיפולטיבית ב-SIGINT (כלומר ע"י SIG_DFL), בין שורה 14 לשורה 15.

שאלה

(1) מה הבעיה בקוד (בעיה שניתנת לפתרון על ידי השלמת שורות 29,45)?

(2) הציעו תיקון לבעיה ע"י השלמה של שורות 29,45. הסבירו למה התיקון שהצעתם יעבוד.

(3) האם בעבודה עם shared memory segment היינו נתקלים בבעיה דומה? הסבר מדוע.

```
1  union semun {
2      int val;
3      struct semid_ds *buf;
4      unsigned short *array;
5  };
6
7  int semid;
8  union semun semarg;
9
10 int main()
11 {
12     int err;
13     struct sembuf sops[1];
14
15     semid=semget(IPC_PRIVATE, 1, 0600);
16
17     semarg.val=1;
18     semctl(semid, 0, SETVAL, semarg);
19     sops->sem_num = 0;
20     sops->sem_flg = 0;
21     if (fork()==0) {
22         sops->sem_op = -1;
23         semop (semid, sops, 1);
24         printf("child got the lock\n");
25         sleep(5);
26         printf("child releases the lock\n");
27         sops->sem_op = 1;
28         semop(semid, sops, 1);
29         // ...
30         err=semctl(semid, 0, IPC_RMID, semarg);
31         if(err==-1)
32         {
33             printf("child delete semaphore error\n");
34         }
35     }
36
37     else {
38         sops->sem_op = -1;
39         semop(semid, sops, 1);
40         printf("parent got the lock\n");
41         sleep(5);
42         printf("parent releases the lock\n");
43         sops->sem_op = 1;
44         semop(semid, sops, 1);
45         // ...
46         err=semctl(semid, 0, IPC_RMID, semarg);
47         if(err==-1)
48         {
49             printf("parent delete semaphore error\n");
50         }
51     }
52     return 0;
53 }
```

תשובה

(1) מה הבעיה בקוד (בעיה שניתנת לפתרון על ידי השלמת שורות 29,45)?

מכיוון שמחיקה של semaphore set מתבצעת מיד, יתכן מצב בו תהליך האב מוחק אותו לפני שתהליך הבן סיים להשתמש בו, או להיפך. נרצה שהמחיקה תתבצע רק לאחר שכל התהליכים הרלוונטיים סיימו לעבוד עם ה-semaphore set.

(2) הציעו תיקון לבעיה ע"י השלמה של שורות 29,45. הסבירו למה התיקון שהצעתם יעבוד.

נוסיף את השורה

sleep(6)

בשורה 29 ובשורה 45.

מטרת שורה 29 היא למנוע מצב בו תהליך הבן מוחק את ה-semaphore set לפני שתהליך האב סיים להשתמש בו, ומטרת שורה 45 היא למנוע מצב בו תהליך האב מוחק את ה-semaphore set לפני שתהליך הבן סיים להשתמש בו.

(3) האם בעבודה עם shared memory segment היינו נתקלים בבעיה דומה? הסבר מדוע.

לא. בהקשר של System V Shared Memory שלמדנו בתרגול, מחיקה של shared memory segment תתבצע רק כאשר מספר ה-attached processes אליו הוא אפס ולאחר בקשה מפורשת למחיקה.

שאלה

- שנו את קטע הקוד הבא כך שלכל היותר 5 חוטים יבצעו את קטע הקוד שלהם. שאר החוטים ימתינו כל עוד יש 5 חוטים שמבצעים פעולה.
- אין למחוק שורות מהמימוש הקיים.
- אסור להשתמש ב-pthread_join.
- אם פחות מ- 5 חוטים במצב ריצה אסור לגרום לחוט אחר סתם להמתין.

threads.c, threads_sol.c

נכון/לא נכון

1. ניתן לבצע הן קריאה והן כתיבה ל-FIFO דרך אותו descriptor.
2. כאשר process נמצא במצב blocked ב-semaphore queue, הוא עסוק ב-busy waiting.
3. ניתן ליצור soft links בין partitions שונים.
4. הילד יורש את הסיגנלים הממתינים לתהליך האבא.
5. תהליך נוצר לראשונה עם חוט יחיד (החוט הראשי).
6. לכל thread יש את המחסנית שלו.
7. איתות הוא מאורע סינכרוני.

נכון/לא נכון

8. מספר חוטים של אותו תהליך יכולים לרוץ במקביל על מעבדים שונים.

9. קריאה ל exit ע"י חוט מסוים הורגת את התהליך כולו.

10. קריאת המערכת kill(...) בהכרח הורגת תהליך.

11. יכולים להיות מספר file descriptors של אותו תהליך המצביעים על מבנה open file description אחד.

נכון/לא נכון

12. ערך המונה של semaphore יכול להיות שלילי.

13. הפונקציה wait תחזור רק כאשר כל תהליכי הבנים של התהליך הקורא יסתיימו, לעומת הפונקציה waitpid (ללא שימוש בדגלים) שתחזור לאחר שתהליך ספציפי יסתיים.

14. מחיקה של pipe (שנוצר ע"י קריאת המערכת ()pipe) איננה צריכה להתבצע באופן מפורש (ע"י unlink/rm).

15. בכל נקודת זמן, חוט יכול להחזיק ב-mutex אחד לכל היותר.

16. תוכן של קובץ נשמר בתוך ה- inode שלו.

17. לאחר סיום "טבעי" של קוד של חוט ראשי מתבצעת קריאה אוטומטית ל-exit אשר גורמת לסיום ביצוע התהליך וכל החוטים בקבוצה של החוט הראשי.

שאלה

פרטו שלושה הבדלים בין soft link לבין hard link.

שאלה

פרטו הבדלים בין pipe לבין FIFO.

פרטו הבדלים בין pipe, FIFO ו-shared memory.

שאלה

כמה תהליכים **חדשים** (כלומר לא כולל התהליך הראשי ממנו הפונקציה נקראה) נוצרים במהלך הריצה של התוכנית הבאה? בהנחה שכל הקריאות ל fork() מצליחות. הסבירו תשובתכם.

```
void main()
{
    int a = 0;
    a += (fork() != 0) ? 2 : 3;
    if (a == 2) fork();
    a++;
    if (a == 3) fork();
}
```

שאלה

עליכם לכתוב תכנית אשר מקבלת כפרמטר ל main (ב argv) רשימה של תכניות.

על התכנית שלכם להגריל מספר אשר מסמל timeslot – זמן מקסימלי שניתן לתכנית לרוץ.

לאחר הגרלת ה timeslot על התכנית שלכם להריץ **סדרתית** את התכניות שקבלה כפרמטר למיין.

לכל תכנית ינתן זמן מקסימלי שמוקצה עבורה (ה timeslot שהוגרל לעיל).

אם תכנית לא סיימה את עבודתה בזמן שהוקצב לה על התכנית שלכם לעצור תכנית זו ולהריץ את התכנית הבאה בתור.

תכנית שסיימה את עבודתה יוצאת מהסבב.

על התכנית שלכם לחזור שוב ושוב על הסבב עד אשר כל התכניות סיימו את עבודתן.

כאשר תכנית מסוימת סיימה את עבודתה לפני הזמן שהוקצב עבורה על התכנית **שלכם להמשיך מיד** ולהריץ את התכנית הבאה בתור.

להזכירכם, כדי לעצור תהליך אפשר להשתמש באיתות SIGSTOP, וכדי להמשיך תהליך עצור אפשר להשתמש באיתות SIGCONT.

דוגמא: אם שם התכנית שלכם היא a.out, כך תורץ התכנית שלכם:

./a.out /u/grad/prog2.out /u/grad/prog3.out /u/grad/prong.out

להזכירכם, כדי להגריל מספר בין 0 לבין $2^{32} - 1$ עש להשתמש בפונקציה rand:

```
srand(time(NULL));

int r = rand();
```

שאלה

א. מה הן התוצאות האפשריות של כתיבה (פקודת write) לתוך pipe?

ב. מה הן התוצאות האפשריות של קריאה (פקודת read) מתוך pipe?

שאלה - המשך

ב. הפקודה pwd – **p**rint **w**orking **d**irectory מחזירה את הנתיב המלא מספריית השורש לספרייה הנוכחית (.). לדוגמא, הרצת pwd בספריית הבית תתן פלט דומה לפלט הבא:

```
student@exam:~$ pwd

/home/student
```

תארו בקצרה כיצד ניתן לממש את הפקודה pwd על בסיס הנלמד בכיתה:

שאלה - המשך

מלאו את הטבלה הבאה בקריאות המערכת (system calls)וקריאות הספרייה (C-library calls) הרלוונטיות בהתבסס על התיאור מהסעיף הקודם .שימו לב :אין צורך לכתוב את התחביר (syntax) המלא של הפקודה וניתן להסתפק בשמה ובתפקידה\תפקידיה .דוגמא למילוי הטבלה נמצאת מייד אחריה.

הפקודה	תפקידה\ים	פלט\ים מצופה\ים

<u>דוגמא למילוי הטבלה:</u>

הפקודה	תפקידה\ים	פלט\ים מצופה\ים
open	פתיחת קובץ	file descriptor

שאלה - המשך

ג. נתונה התוכנית הבאה:

```
1   #include <sys/types.h>
2   #include <sys/stat.h>
3   #include <unistd.h>
4   #include <fcntl.h>
5   #include <stdio.h>
6
7   int main(int argc, char **argv)
8   {
9     struct stat buf;
10    int i, j, k;
11    for (j=0, i=1; i<argc; ++i)
12    {
13      k = open(argv[i], O_RDONLY);
14      fstat(k, &buf);
15      j += buf.st_size;
16    }
17    printf("%d\n", j);
18  }
```

מה התוכנית אמורה לעשות?

שאלה - המשך

מה תהיה התוצאה של הרצת התוכנית במידה ויריצו אותה על הקבצים הבאים (הרשימה התקבלה כתוצאה מהרצת ls -l):

```
-rw-r--r-- 1 student os  307 2011-02-09 13:06 file3
prw-r--r-- 1 student os    0 2011-02-09 13:05 file6
-rw-r--r-- 1 student os  434 2011-02-09 13:06 file4
-rw-r--r-- 1 student os    0 2011-02-09 13:05 file7
-rwxr-xr-x 1 student os 8639 2011-02-09 13:03 file1
drwxr-xr-x 2 student os 4096 2010-10-25 14:31 file8
```

שאלה - המשך

כתוצאה מאיזה סוג קלטים התוכנית עלולה להיתקע?

אילו בעיות קיימות בתוכנית?

הציעו בקצרה דרכים לתקן את הבעיות הנ"ל.