System calls - process

עבודה עם קומפיילר

- פקודה nano – עורך קבצים.
- לדוגמא nano lab2.c
- בתוך הקובץ נוכל לרשום תוכנית בשפת C.
- לדוגמא :

```
#include <stdio.h>
void main(){
printf(“hello world”); }
```

- הרצה:

```
gcc lab2.c
./a.out
Or
gcc –o lab2 lab2.c
./lab2
```

תהליכים

- אחת המשימות העיקריות של מע' הפעלה – ניהול תהליכים.
- כל תהליך מריץ תכנית.
- תהליך יכול להיות במצב חסום, מוכן לריצה או במצב ריצה.
- יתכן מצב בו ירוצו כמה תהליכים במקביל.
- מע' הפעלה עוקבת אחרי כל התהליכים שרצים בה ואוכפת את מדיניות הגישה של התהליכים למשאבי מערכת.
- בעת יצירת התהליך הוא יכיל – קוד, רשימת משאבים הזמינים לו (זיכרון וכו), הרשאות, מצב מעבד ועוד.
- לכל תהליך יש מספר מזהה משלו (processid (pid
- כל תהליך יכול ליצור תחתיו תהליכי בנים.

סיום תהליך

- סיום רגיל – תהליך מסיים את התכנית ומבצע פינוי משאבים שתפס.
- סיום בעקבות שגיאה – סיום מתוכנן בעקבות שגיאה במהלך ריצת התכנית לדוגמא- תכנית שקיבלה מס פרמטרים לא נכון, דוא"ל שנכשל בשליחה וכו.
- סיום בעקבות שגיאה פטאלית- שגיאה שלא נצפתה מראש ולא היה לה מענה בקוד- התכנית קורסת.
- סיום בעקבות "הריגת" תהליך – מע' הפעלה שולחת בקשה לסיום תהליך כלשהו.

קריאות מערכת לעבודה עם תהליכים

- **pid_t getpid(void)** – פקודה המחזירה pid של התהליך הנוכחי (תהליך בו הפעלנו את הפקודה)
- **pid_t getppid(void)** – פקודה המחזירה pid של תהליך אב לתהליך הנוכחי.
- **pid_t fork(void)** – פקודה זו יוצרת תהליך חדש
  - תהליך המפעיל את הפקודה נקרא תהליך אב
  - תהליך הנוצר על ידי הפעלת הפקודה נקרא תהליך בן

הערך המוחזר של הפקודה:

- תהליך בן יקבל ערך 0
- תהליך אב יקבל:
  - אם תהליך בן נוצר בהצלחה אזי pid של תהליך בן
  - אחרת -1

קריאות מערכת לעבודה עם תהליכים

- לעבודה עם קריאות מערכות הקשורות לתהליכים

נוסיף את הספריות הבאות:

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

הפונקציה fork()

- נתונה התכנית הבאה :

מה הפלט ?

```
void example2()
{
        fork();
        printf("Hello\n");
}
```

הפונקציה fork()

- תרגיל 1 - נתונה התכנית הבאה :

מה הפלט ?

```
void example3()
{
        printf("Hello");
        fork();
        printf("Bye");

}
```

הפונקציה Fork()

- פתרון תרגיל 1 –

תהליך אבא

```
1 printf("hello")
2 fork()
3 printf("bye")
```

תהליך בן

1 X אי אפשר מבלבל

2 X את השורות קוד

3 V לאחר ה- fork

תשובה - פלט:

אבא מדפיס { hello, bye

בן מדפיס { bye

הפונקציה Fork()

- תרגיל 2 – נתון הקוד הבא – מה הפלט?

```
void example4()
{
        fork();
        pid_t myPID = getpid(), pPID = getppid();
        printf("My PID is %d and my parent's PID is %d\n", myPID, pPID);
}
```

הפונקציה Fork()

תרגיל 3 – נתונה התכנית הבאה :

מה הפלט ?

```
void example() {
    fork();
    fork();
    fork();
    printf("hello\n")
}
```

הפונקציה Fork()

פתרון תרגיל 3 –

- n - מספר ה forks בתכנית

(יש לשים לב , לפעמים יש תנאים נוספים ואז הנוסחה לא מתאימה)

כמות הילדים שנוצרים : 2^n-1

לכן התשובה לתרגיל – כמות הforks = 3

כמות הילדים שנוצרו 2^3-1=7

כמות הפעמים שתודפס המילה hello : 8 (7 ילדים + תהליך אב ראשי)

הפונקציה wait

pid_t wait(int *status);

- משהה תהליך נוכחי עד שתהליך בן יסיים את עבודתו.
- status – מצביע למשתנה בו יאוחסן סטטוס נוכחי של תהליך בן.
  - ערך 0 – הבן סיים.
- ערך מוחזר מהפונקציה:
  - pid של תהליך שסיים את עבודה.
  - 1- בעת שגיאה.

דוגמא:

לדוגמא: האבא מחכה לסיום של אחד מבניו

```
wait(null)
wait(&status) != -1 // תהליך אב ממתין שהבן יסתיים
                    ומעוניין לקבל מידע כלשהו לגבי הבן
```

הפונקציה waitpid

```
pid_t waitpid(pid_t pid, int *status, int options);
```

- פונקציה משהה תהליך עד לרגע שתהליך בן עם pid ספציפי יסיים את עבודתו.
- משתנה pid יכול לקבל:
  - 1- המתנה לבן כלשהו (שקול ל ()wait)
  - 0 < המתנה לתהליך עם pid נתון.
- משתנה options יכול לקבל:
  - WNOHANG - דגל שבודק אם הבן סיים אז לא להמתין לו.
  - WVNTRACED - זיהוי תהליך שעצר וסטטוס סיומו אינו ידוע.

הפונקציה waitpid

- הסטטוס שתהליך אב מקבל על תהליך בן מאפשר להפעיל הרבה אפשרויות

* status - יכול להראות על:
- סיום תקין/לא תקין
- סיבה לסיום
- סטטוס יציאה

בשביל להשתמש בקריאות מערכות הקשורות ל wait נוסיף לתוכנית
#include <wait.h>

דוגמאות שימוש ב wait

לדוגמא:

```
#include <unistd.h>
#include <wait.h>

int main(int argc, char *argv[])
{
        int pid, status;

        if (fork() == 0)
                printf("I am the son process and my pid is %d.\n", getpid());
        else
        {
                pid = wait(&status);
                printf("I am the father process.\n");
                printf("My child with pid %d was exited and returned status %d\n", pid, status);
        }

        return 0;
}
```

ספריה המגדירה את פקודות wait

אם תהליך בן - נדפיס pid

תהליך אב - מדפיס שהוא אבא + pid וסטטוס של בן שסיים.

Fork יוצרת

תהליך בן – הבן מתחיל

לרוץ ובינתיים תהליך אב

מחכה שהבן יסיים

ולאחר מכן מדפיס פלט.

מהו הפלט ?!

WEXITSTATUS(status)

בדיקת ערך הסטטוס

דוגמאות שימוש ב wait

פלט לדוגמא –

I am son , my pid is 1111

I am father

My child with pid 1111 was done and return 0

הערה חשובה

wait(&status) - כאשר הבן יסתיים נרצה לדעת עוד פרטים על התהליך.

wait(null) – הבן מסתיים וזהו , שאר המידע לא מעניין את תהליך האב.

דוגמה ליצירת תהליכי בנים

- בדוגמה יהיה תהליך אב יחיד לכל תהליכי הבנים שניצור.
- דוגמא לקוד שמייצר N בנים לאב אחד.

```
void example7(){
        for(int i=0;i<5;i++){
                if(fork()==0){
                printf("[son] pid %d from [parent] pid %d\n", getpid(), getppid());
                //exit(0);
                }
        }
        for(int i=0;i<5;i++){
                printf("I am pid %d pid %d\n", getpid(), getppid());
                wait(NULL);
        }
}
```

מה יקרה
אם נמחק את
exit(0) ?

יצירת תהליכי בנים בצורת עץ - tree

תשובות לשאלות -

- ניצור 2^5-1=31 בנים במקום 5 בנים.

- האבא נתפס ב FOR השני וממתין עד שכל חמשת ילדיו ייווצרו. אם נוריד את השורות קוד -> יכול להיות מצב שתהליך האב (התכנית הראשית) תסתיים לפני שכל חמשת הבנים יספיקו להיווצר.

יצירת תהליכי בנים בצורת עץ – tree

```
int main()
{
        fork(); // f1
        fork(); // f2
        fork(); // f3
        printf("hello.\n");
}
```

f1 – נוצר תהליך 1

f2 - נוצרים 2 תהליכים

1 ע"י תהליך אבא (סה"ש)

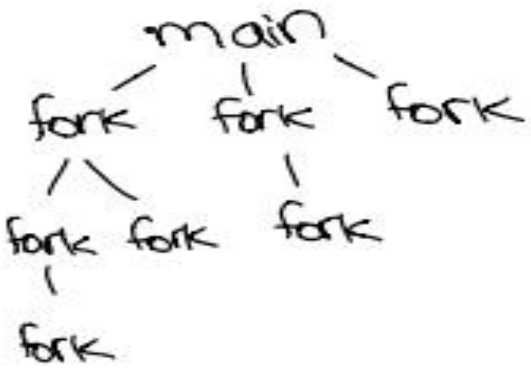
1 ע"י התהליך שנוצר ע"י f1

f3 - נוצרים 4 תהליכים

1 ע"י תהליך אב

1 ע"י תהליך שנוצר מ- f1

2 ע"י תהליכים שנוצרו מ- f2



סה"כ 3 fork's $\Leftarrow$ $2^3 - 1 = 7$ $\Leftarrow$ 7 תהליכים בנים.

מטלה – מעבדה מספר 3

- **<u>תרגיל 1</u>** – יש לכתוב תכנית שיוצרת תהליך בן. עבור כל תהליך שרץ יש לבדוק :
- אם התהליך הנוכחי בן -> להדפיס I’m son
- אם התהליך הנוכחי אבא -> להדפיס I am father

מטלה – מעבדה מספר 3

- **<u>תרגיל 2 –</u>** נתון קטע הקוד הבא : כמה אפשרויות לפלט יש ? יש לפרט מהם הפלטים האפשריים.

```
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
    if (fork()== 0)
        printf("HC: hello from child\n");
    else
    {
        printf("HP: hello from parent\n");
        wait(NULL);
        printf("CT: child has terminated\n")
    }

    printf("Bye\n");
    return 0;
}
```