** מחלקה להנדסת תוכנה**

**מערכות הפעלה סמסטר ב תשפ"ב**

**תרגיל בית 2**

ניתן להגיש עד 10.04.2022 עד השעה 23:55

|  |  |  |  |
| --- | --- | --- | --- |
| **מספר ת"ז:** | **316109115** | **שם:** | **ליאור ז'יגלו** |
| **מספר ת"ז** | **318423324** | **שם:** | **שיראל גולדנברג** |

1. **(30 נקודות) לכל אחת מהטענות הבאות, נא לציין האם הטענה נכונה.   
   נא לנמק את תשובתך.**
2. (5 נקודות) במערכת הפעלה לינוקס, לכל תהליך יש PID ייחודי במערכת ולכל חוט TID ייחודי במערכת.

נכון / לא נכון

|  |
| --- |
| כל חוט מקבל TID ייחודי בתוך **התהליך** שיצר אותו כלומר יכולים להיות 2 |
| חוטים על אותו TID באותה מערכת **בתהליכים** שונים. |
|  |
|  |

1. (5 נקודות) כל כתיבה לזיכרון הראשי ( RAM) של המחשב מחייבת מעבר למצב קרנל.

נכון / לא נכון

|  |
| --- |
| ב User mode אין אפשרות לתקשר עם חומרה, לכן על מנת לתקשר עם חומרה |
| עלינו להשתמש ב api של מערכת ההפעלה המאפשר לנו לתקשר |
| עם החומרה או הזיכרון RAM הפיזי של המעבד. |
|  |

1. (5 נקודות) תהליך יכול לעבור למצב BLOCKED ממצב RUNNING בלבד.

נכון / לא נכון

|  |
| --- |
| תהליך שנמצא במצב blocked מחכה ל event כלשהו |
| לדוגמה קלט/פלט, לכן תהליך יכול להגיע למצב שהוא מחכה לevent |
| רק אם הוא רץ והגיע לקטע שמצריך event לכן הוא יכול לעבור למצב |
| blocked רק ממצב running כי אחרת הוא לא הספיק לעשות שום דבר. |

1. (5 נקודות) תהליכים בעלי אותו Parent Process ID (PPID) בהכרח נוצרו על ידי אותו תהליך אב.

נכון / לא נכון

|  |
| --- |
| כאשר נוצר תהליך הוא מקבל PID שלו ו PPID של התהליך שיצר אותו. |
| ולכן אם תהליכים נוצרו ע"י אותו תהליך אז בהכרח יהיה לכם את אותו ה PPID. |
| ניתן לחשוב על זה כמו עץ שה PID של האב זה ה PPID של הבנים. |
|  |

1. (5 נקודות) במעבד בעל ליבה אחת, שימוש בריבוי חוטים הממומשים במרחב הגרעין (מודל 1:1) תמיד יגרום לשיפור בביצועים של תוכניות.

נכון / לא נכון

|  |
| --- |
| במודל 1:1 כאשר המשתמש יוצר חוט אז לכל חוט במרחב המשתמש יש חוט |
| במרחב הגרעין, בהינתן שיש לנו יותר ממעבד אחד זה מאפשר לנו לשפר את |
| ביצועי התכנית. אבל נתון שיש לנו מעבד אחד בעל ליבה אחת אז לא נראה שיפור |
| בביצועי התכנית כי מצב זה יהיה זהה למודל 1:N. |

1. (5 נקודות) אין תועלת במימוש קוד מרובה-חוטי-משתמש (user threads) לביצוע קלט-פלט וחישובים.

נכון / לא נכון

|  |
| --- |
| כאשר נבקש קלט מהמשתמש התכנית תמתין לקלט ולכן ללא קוד מרובה חוטים |
| לא נוכל לנצל את זמן הציפיה לקלט לצורך חישובים. |
| לכן קוד מרובה חוטים כן יעיל במצב הנתון. |
|  |

1. **(20 נקודות) נתונה התוכנית הבאה:**

int Mat[N][N];

int count[N];

void \* check(void \*arg) { // 🡨 arg = int

int i ;

int row = \*(int \*)arg;

count[row] = 0;

for (i = 0; i < N; i++) if(Mat[row][i]>0) count[row]++;

pthread\_exit(0);

}

int main() {

int i, x=0, arr[N];

init\_Mat(); //Initialize Matrix

pthread\_t threads[N];

for (i = 0; i < N; i++) {

arr[i] = i; /// \*\*

pthread\_create(&threads[i], NULL, check, &arr[i]);

}

for (i = 0; i < N; i++) {

pthread\_join(threads[i], NULL);

x += count[i];

}

if(x > N\*N/2)printf (“YES\n”);

else printf(“ NO\n”);

}

1. מה הקוד הנ"ל מבצע ? מתי יודפס YES ומתי יודפס NO

|  |
| --- |
| אם יותר ממחצית האינדקסים במטריצה גדולים מ-0 אז התכנית תדפיס yes, |
| אחרת no. |
|  |
|  |

1. האם ניתן למחוק את השורה המסומנת ב \*\* ולשנות את השורה אחרי ל:

pthread\_create(&threads[i], NULL, check, & i);

כתוב כן / לא **והסבר**.

|  |
| --- |
| לא, הכוונה פה היא לשלוח את i ישירות כפרמטר, מצב זה הוא בעייתי מהסיבה |
| שכאשר אנו יוצרים את החוטים החוט הראשי לא בהכרח מפנה להם זמן מעבד |
| באותו הרגע, ואז נקבל תוצאות לא צפויות כי כולם חולקים את אותו הזיכרון |
| והמשתנה i משתנה בערכו כל הזמן לכן כל חוט כאשר ירוץ לא בהכרח יקבל את |
| הערך שרצינו לשלוח אליו. |

1. כתוב main חדש המבצע את אותו הקוד בצורה סדרתית (ללא חוטים).

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#define N 3

int mat[N][N];

int check();

int main() {

init\_mat();

if (check() >= N\*N/2)

printf("YES\n");

printf("NO\n");

}

int check() {

int i = 0, j = 0, sum = 0;

for (i; i < 3; i++) {

for (j; j < N; j++) {

if (mat[i][j] > 0)

sum += 1;

}

}

return sum;

}

1. בהינתן שהתוכנית הנתונה רצה במערכת בעלת מעבד אחד בעל ליבה אחת.  
   האם התוכנית שכתבת בסעיף ב יעילה יותר (מבחינת זמן ריצה) להשגת המטרה?   
   בחר תשובה נכונה והסבר.
   * + 1. הרצת התוכנית המקבילית הנתונה יעילה יותר מכל תוכנית סדרתית.
       2. הרצת התוכנית הסדרתית שכתבתי בסעיף ב יעילה יותר.
       3. אי אפשר לדעת.

|  |
| --- |
| התשובה הנכונה היא שהרצת התוכנית הסדרתית שכתבנו בסעיף ב יעילה יותר. |
| מכיוון, שכאשר יש לנו הרבה חוטים אך רק מעבד בעל ליבה אחת התוכנית לא |
| תעבוד בצורה יעילה כי אין מקביליות וכל פעם נעשה contex switch ולכן |
| במקרה כזה עדיף להריץ תוכנית סדרתית. |
|  |
|  |

1. **(25 נקודות) נניח שבתוכנית מסוימת מוגדר משתנה גלובלי int\*p .  
   תהליך A המריץ תוכנית זאת מבצע את שורת הקוד:  
   p = (int\*) malloc(sizeof(int));  
   לאחר ביצוע שורה זאת, מתקיים p=1000 כלומר הזיכרון הוקצה בכתובת 1000.  
   סעיף א:   
   תהליך A לאחר malloc מבצע : \*p=0 ולאחר מכן מבצע fork.**

* האם ערך המצביע p הוא 1000 גם אצל הבן? הסבר

|  |
| --- |
| נכון, מכיוון שכשמבצעים fork, מועתק לתהליך הבן אותו זיכרון שהיה לתהליך האב |
| ולכן P יקבל כתובת 1000. |
|  |

* נתון שתהליך הבן ביצע \*p=1 , האם תוכן זיכרון תהליך האב יושפע כתוצאה מפעולה זאת?הסבר

|  |
| --- |
| לא, לתהליכים שונים איזורים שונים בזיכרון לכן התוכן של p בתהליך האב |
| לא מושפע מפעולות בתהליך הבן. |
|  |

* נתון שתהליך האב ביצע \*p=1 לאחר ה fork, האם תוכן זיכרון תהליך הבן יושפע כתוצאה מפעולה זאת?הסבר

|  |
| --- |
| לא, לתהליכים שונים איזורים שונים בזיכרון לכן התוכן של p בתהליך הבן |
| לא מושפע מפעולות בתהליך האב. |
|  |

* נתון שתהליך האב ביצע free(p) לאחר ה fork, אילו תהליכים יושפעו מפעולה זאת?האם כעת חוקי לגשת לזיכרון שהוקצה בכתובת 1000 בתהליך הבן?   
  האם חוקי לגשת לזיכרון שהוקצה בכתובת 1000 בתהליך האב?

|  |
| --- |
| חוקי לגשת לכתובת בתהליך הבן כי p עדיין קיים בתהליך הבן, כלומר הוא שוחרר |
| רק אצל האב. |
| לא חוקי לגשת ל p מתהליך האב כי אז ניגש לזיכרון שכבר לא שייך לנו. |

**סעיף ב:   
  
תהליך A לאחר malloc מבצע : \*p=0 ולאחר מכן מבצע pthread\_create.**

* האם ערך המצביע p הוא 1000 גם אצל החוט שנוצר ? הסבר

|  |
| --- |
| כן, חוטים חולקים את אותו איזור הזיכרון כמו של התהליך שיצר אותם. |
|  |
|  |

* נתון שהחוט שנוצר ביצע \*p=1 , האם עבור החוט הראשי \*p ==1?הסבר

|  |
| --- |
| כן, החוט שנוצר חולק את אותו זיכרון כמו של התהליך שיצר אותו ולכן כשהחוט |
| שנוצר משנה משתנים בזיכרון החוט הראשי גם כן רואה את זה. |
|  |

* נתון שהחוט הראשי ביצע free(p) לאחר pthread\_create.   
  האם כעת החוט שנוצר יכול לגשת לזיכרון שמוצבע ע"י p?

|  |
| --- |
| לא, שני החוטים חולקים את אותו זיכרון ולכן אם אחד משחרר איזור כלשהו גם השני לא יוכל לגשת אליו יותר. |
|  |
|  |

1. **(25 נק') מהם כל הפלטים האפשריים של התוכנית הבאה:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

char \* str;

int main( int argc, char \*argv[] ) {

str = malloc( 80 );

strcpy( str, "First" );

switch( fork() ) {

case 0:

printf( "Child: %s\n", str );

strcpy( str, "Second" );

printf( "Child: %s\n", str );

free( str );

break;

case -1:

printf( "Error with fork()\n" );

break;

default:

printf( "Parent: %s\n", str );

sleep(1);

printf( "Parent: %s\n", str );

free( str );

}

return 0;

}

**יש להסביר כל פלט באופן מפורט.**

|  |
| --- |
| פלט 1: |
| מצב בו fork נכשל, והפלט יהיה: |
| Error with fork() |

|  |
| --- |
| פלט 2: |
| מצב בו ניכנס לתהליך הבן קודם ואז לתהליך האב והפלט יהיה: |
| Child: First |
| Child: Second |
| Parent: First |
| Parent: First |
|  |
| פלט 3: |
| מצב בו ניכנס לתהליך האב, נגיע לsleep, נבצע context switch, ירוץ תהליך הבן |
| ואז ימשיך תהליך האב והפלט יהיה: |
| Parent: First |
| Child: First |
| Child: Second |
| Parent First |

|  |
| --- |
| פלט 4: |
| מצב בו ניכנס לתהליך האב, נגיע ל sleep אבל לא ניכנס לתהליך הבן אלא נחזור |
| לתהליך האב והפלט יהיה: |
| Parent: First |
| Parent First |
| Child: First |
| Child: Second |
|  |
| פלט 5: |
| מצב בו ניכנס לתהליך הבן אבל אחרי ההדפסה הראשונה נבצע context switch |
| וניכנס לתהליך האב ונחזור לתהליך הבן רק לאחר סיום תהליך האב |
| והפלט יהיה: |
| Child: First |
| Parent: First |
| Parent First |
| Child: Second |
|  |
|  |
|  |
|  |
|  |
| פלט 6: |
| מצב בו ניכנס לתהליך האב, נגיע לsleep נבצע context switch וניכנס לתהליך |
| הבן, ולאחר ההדפסה הראשונה נבצע context switch ונחזור לתהליך האב |
| והפלט יהיה: |
|  |
| Parent: First |
| Child: First |
| Parent First |
| Child: Second |
|  |
| פלט 7: |
| מצב בו ניכנס לתהליך הבן ולאחר ההדפסה הראשונה נבצע context switch וניכנס |
| לתהליך האב ובו כשנגיע ל sleep נבצע context switch לתהליך הבן, נבצע |
| הדפסה שנייה ואז נחזור לתהליך האב ונבצע הדפסה שנייה גם בו והפלט יהיה: |
| Child: First |
| Parent: First |
| Child: Second |
| Parent: First |

**בהצלחה !**