בס"ד

טליה יהודה 316471978

ממ"ן 11 - מערכות הפעלה חלק עיוני :

שאלה 2 (5%)

א) מהי פעולת ה TRAP? תארו מתי ובשביל מה היא מתבצעת ומה קורה בעת ביצועה.

ב) הסבירו מה קורה בעת הקריאה לפונקצית write של ה C library . בפרט הסבירו כיצד עוברים הפרמטרים של ה write למערכת הפעלה Linux וכיצד המערכת מטפלת ב write.

ג) מה ההבדל בין write ל printf? תוכלו להיעזר בקבצי מקור של C library מ www.gnu.org/software/libc

תשובה :

א. פעולת ה TRAP מבצעת מעבר ממצב משתמש למצב קרנל ומתחילה את מערכת ההפעלה. בעצם הTRAP הוא חלק מהתהליך של קריאת מערכת – כאשר תהליך משתמש מבקש שירות כלשהו ממערכת ההפעלה קריאת מערכת מתבצעת ע"י השמת הפרמטרים הדרושים במחסנית וברגיסטר בהתאם והפעלת פעולת הTRAP. כאשר הTRAP מתבצעת, המעבד עובר למצב מיוחס - שינוי מצב ההרשאה ממשתמש לקרנל, כלומר היא גורמת למעבר השליטה למערכת ההפעלה. לאחר מכן כאשר מערכת ההפעלה מסיימת את העבודה שנדרשה השליטה מוחזרת לתוכנית המשתמש בהתאם להנחיות קריאת המערכת.

ב. נשים לב שבעת הקריאה לפונקציית write אנחנו שולחים אליה שלושת הפרמטרים הבאים ושומרים אותם במחסנית: מצביע לבאפר המכיל את התווים לכתיבה, מס' התווים שיש לכתוב ויעד הפלט שיש לכתוב אליו. לאחר מכן מועבר לתוך רגיסטר מספר השירות המבוקש שזה בעצם במקרה זה -write (מס' קריאת המערכת) המסופק ע"י מערכת ההפעלה . לאחר כל אלו מתבצעת הוראת TRAP שמעבירה את המעבד למצב מיוחס וגורמת למעבר השליטה למערכת ההפעלה ובמידה והכל היה תקין היא תבצע כתיבה בהתאם לפרמטרים אלו במחסנית וברגיסטר ולאחר מכן תחזיר את השליטה למשתמש בהתאם לקריאת המערכת.

ג. ההבדל ביניהם ש write היא פונקציית קריאת מערכת בסיסית (LOW LEVEL) שמקבלת פרמטרים לכתיבה (שפירטתי סעיף קודם) ויוצרת קריאת מערכת וTRAP ומעבירה את השליטה למערכת ההפעלה -לקרנל שיכתוב את רצף היבטים שהיו בבאפר. לעומתה הפונקציה printf היא פונקציה בשפה עילית יותר שמקבלת מחרוזת מבצעת לה פירמוט בהתאם לפרמטרים שהיא קיבלה ויוצרת קריאת לפונקציית write עם STDOUT ששווה ל1 שזה בעצם כתיבה עם המסך.

בס"ד

טליה יהודה 316471978

שאלה 3 :

א) ממשו את "**סמפור**" **בינארי יחיד**(ללא שם) על בסיס שימוש סינכרוני בסיגנל וללא שימוש במנגנוני סנכרון נוספים. ה"סמפור" מיועד לשימוש ע"י **מספר תהליכונים THREADS של אותו תהליך(לא תהליכים שונים). המירכאות** בגלל שה"סמפור" המיועד **אינו חייב לכלול את מבני הנתונים** הפנימיים הלא חיוניים לתיפקודו.

ממשו את הפונקציות הבאות **בשפת C:**

- void sem_init(int status) לאתחול הסמפור למצב מסומן(פתוח, = 1) או לא מסומן(סגור, = 0).
- void sem_down() להורדה(סימון כתפוס, המתנה) של סמפור.
- void sem_up() לשחרור(סימון כדלוק/פינוי) של סמפור.
- הכרזה ואתחול של משתנים שחייבים להיות גלובאליים עם ציון בצורה חופשית שהם גלובליים.

תשובה :

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

int sem_status = 1;//defult value to sem

int in_cs = 0; //to indicate if the is a thread in the cs

void sem_init(int status) {

   sem_status = status;

}

void sem_down() {

   while (sem_status <= 0 || in_cs) {

      sigset_t mask;

      sigemptyset(&mask);

      sigaddset(&mask, SIGUSR1);// init waiting list

      sigprocmask(SIG_BLOCK, &mask, NULL);// signal have to be blocked to ensures
                                          that the process won't be interrupted by this signal.

      sigsuspend(&mask);       // block the process until a signal is received

      sigprocmask(SIG_UNBLOCK, &mask, NULL);//Unblock the SIGUSR1 signal    }

   sem_status = 0;

   in_cs = 1;
 }

void sem_up() {

   sem_status = 1;

   in_cs = 0;

   kill(getpid(), SIGUSR1);    // Send the SIGUSR1 signal to wake up a waiting thread
 }
```

ב) האם בדרך דומה(ולא מסובכת משמעותית יותר) ניתן לממש את הסמפור המיועד לסנכרון בין מספר **תהליכים? תנו נימוק מילולי, אין צורך בכתיבת קוד.**

<u>תשובה :</u>

מימוש הסמפור לסנכרון בין מספר תהליכים ידרוש לשלוח סיגנלים לקבוצת תהליכים מה שדורש מאיתנו עבודה נוספת של שימוש במנגוני תקשורת בין התהליכים.בנוסף בין תהליכים אין לנו זיכרון משותף ולכן סינכרון ביניהם ע"י העברת סיגנלים בלבד יהיה מסובך יותר ופחות יעיל משום שהם בעקרון מפריעים לתהליך ולא לנהל מצב משותף בין כמה תהליכים – לעומת threadsשכן חולקים מרחב זיכרון משותף תחת אותו תהליך "מאחר", ולכן כן נרצה לדאוג למרחב זיכרון משותף שיהיה בין התהליכים שנוכל לתאם סנכרון בצורה הנכונה.

שאלה 4 (5%)

תקראו את מאמר שמסכם את הבדלים בין user threads ו kernel threads ואת מאמר שמסביר מודלים שונים של מימוש מנגנון threads . תענו לשאלות הבאות :

א. האם **M:1 model** מאפשר לנצל מספר ליבות במעבד CPU cores ? נמקו.

ב. האם ב **M:1 model** חסימת אחד מ user threads תגרום לחסימת כל התהליך? נמקו.

ג. מה המשמעות של מושגים user thread ו kernel thread ב **1:1 model** ?

<u>תשובה :</u>

א. לא. נשים לב כי במודל M:1 כל התהליכונים ברמת המשתמש ממופים יחד לתהליך אחד ברמת הגרעין. התהליך יכול להריץ כל פעם רק תהליכון אחד ברמת המשתמש בכל פעם משום שיש רק תהליך אחד ברמת הגרעין המשויך לתהליך. ולכן מודל M:1 לא מאפשר לנצל מספר ליבות במעבד CPU cores.

ב.כן. במודל M:1 כפי שהסברתי בסעיף א' כל התהליכונים ברמת המשתמש ממופים יחד לתהליך אחד ברמת הגרעין ולכן כאשר אחד מהתהליכונים ברמת המשתמש נחסם והקרנל חוסם גם את התהליך ברמת הגרעין אשר משמש בעצם לביצוע התהליכונים ברמת המשתמש.

ג. במודל 1:1 בעצם לכל תהליכון ברמת המשתמש יש תהליך יחיד ברמת הגרעין שניתן לתזמן. זאת אומרת שבמידה ויש לנו יותר ממעבד אחד תהליכונים ברמת המשתמש יכולים לפעול על מעבדים שונים.
אפרט :
User thread הוא תהליכון שנוצר ע"י המשתמש בעזרת ספריית משתמש ומתבצע במרחב של התהליך המארח, ללא התערבות של הקרנל. ניהול התהליכונים ברמת המשתמש באחריות ספריית פונקציות אשר מספקת ממשק ליצירה, להשמדה ולתזמון של התהליכונים, מספקת מנגנוני תקשורת ביניהם ודואגת לניהולה של טבלת התהליכונים המכילה מאפיינים של כל התהליכונים הנמצאים בתהליך המארח.
kernel thread הוא תהליך שניתן לתזמון ברמת הגרעין, כל thread ברמת הקרנל מיוצג על ידי מבנה נתונים כלשהו המכיל מידע הקשור אליו, והוא התהליך היחיד שמתזמן ה-CPU מתחשב בו.

שאלה 5 (5%)

הוכיחו כי בפתרון של Peterson ל 2 תהליכים, תהליכים אינם ממתינים זמן אינסופי על מנת להיכנס לקטע קריטי. בפרט הוכיחו כי תהליך שרוצה להיכנס לקטע קריטי לא ממתין יותר ממה שלקח לתהליך אחר להיכנס ולעזוב את הקטע הקריטי.

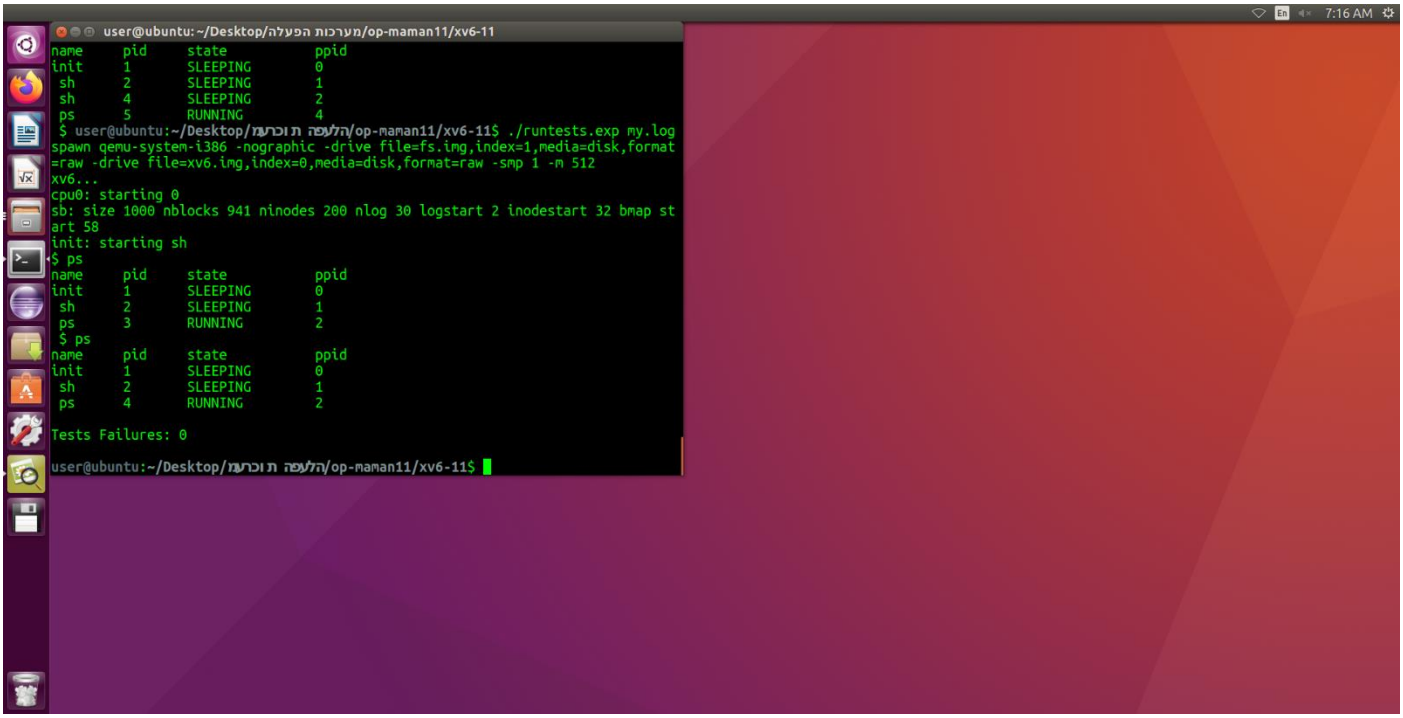
<u>תשובה:</u>

בפתרון פטרסון לשני תהליכים נניח תהליך 0 ותהליך 1. אם תהליך מעוניין להיכנס לקטע הקריטי הוא משנה את התא שלו במערך interested לtrue. אם רק אחד מהם רוצה להיכנס לקטע הקריטי אז הוא לא צריך לחכות ונכנס אליו ישר ומבצע אותו

(מניחים שזה זמן סופי)- ב.ה.ב נניח שזה תהליך 0 ולכן הוא משנה גם את הערך של turn =0.
ועל כן אם גם התהליך השני רוצה להיכנס לקטע הקריטי באותו הזמן הוא יעלה את התא שלו במערך interested[1]= true ואת הערך של turn =1 ולכן יחכה בלולאת ההמתנה שלו שלפני הכניסה לקטע הקריטי עד שהתהליך 0 יסיים את הקטע הקריטי שלו ויעדכן את interested[0]=false. כעת תהליך 1 יצא מלולאת ההמתנה שלו ויכנס לקטע הקריטי שלו, ותהליך 0 לא יוכל להיכנס לקטע הקריטי עד שתהליך 1 יסיים את הקטע הקריטי שלו וישנה את interested[1]= false.

בנוסף, בפתרון זה אין אפשרות שתהליך כלשהו למשל 0 ימשיך ויריץ את הקטע הקריטי מעל פעם אחת באותו "תור" כאשר התהליך השני מעוניין גם להיכנס לקטע הקריטי - משום שלפני כניסתו לקטע הקריטי הוא את ערך ה turn בהתאמה כך שהוא יכנס ללולאת ההמתנה כל עוד תהליך 1 לא יסיים את הקטע הקריטי וישנה את interested[1]=false.
נשים לב כי בפתרון של פטרסון זה יש שימוש בהמתנה פעילה זאת אומרת יש בזבוז זמן מעבד על התהליך השני עד שתהליך 0 יסיים את קטע הקריטי שלו ולהיפך. אך מכיוון שזה שני תהלכים בלבד זמן המתנה זה יהיה סופי בהכרח זאת אומרת שתהליך לא יחכה זמן אינסופי כדי להיכנס לקטע הקריטי באף מצב, ויתרה מזאת הוא לא יחכה יותר מהזמן שלוקח לתהליך אחר להיכנס ולצאת מהקטע הקריטי שלו כפי שפירטתי מעלה.

<u>שאלה 1 חלק מעשי צילומי מסך :</u>



בס"ד

טליה יהודה 316471978