**יבש 2**

**מגישים :**

**חן פרי 313283657**

**גיל קפל 305323776**

**חלק 1 :**

1. מה עושה פקודת yes בלינוקס ? מה הארגומנטים שהיא מקבלת ?

מקבלת כקלט (אופציונלי) מחרוזת ומדפיסה אותה אינסוף פעמים עד שהיא נהרגת או נעצרת . אם הפקודה לא מקבלת קלט אז היא מדפיסה y אינסוף פעמים עד שהיא נהרגת או נעצרת.

1. מדוע השתמשנו בפקודת yes עם מחרוזת ריקה במהלך הפקודה הבאה ?

>> yes '' | make oldconfig

Make oldconfig זה תהליך ארוך יחסית ששואל את ה-user "האם לאפשר הגדרות קונפיגורציה חדשות שלא יהיו זמינות בקרנל קודם", המשתמש יצטרך לענות על השאלות האלה כל פעם ,לכן נשתמש בפייפ שהפקודה yes " תהיה ערוץ הפלט לפקודה make config ותשלח לה מחרוזת ריקה שזה שקול ללחוץ אנטר.

לחיצת אנטר כתשובה בוחרת ערכים דיפולטיבים בשאלות שהמשתמש נשאל, אופציה בטוחה יותר מכיוון שתשובות דיפולטיביות בדרך כלל האופציה הנכונה.

בעצם המשתמש לא יצטרך להיות נוכח ולענות על השאלות , הפייפ עושה לו את העבודה ועושה זאת במהירות .

1. מה משמעות הפרמטר TIMEOUT\_GRUB בקובץ ההגדרות של GRUB ?

GRUB\_TIMEOUT=5

מייצג את מספר השניות בו הGRUB מחכה שהמשתמש יבחר איזו מערכת הפעלה לטעון מהדיסק.

אם הוא שווה 0 הכוונה שתטען מיידית מערכת ההפעלה הדיפולטית ללא הצגת התפריט, אם הערך הוא -1 הכוונה שהתפריט יהיה מוצג עד שתבחר מערכת הפעלה מתאימה על ידי המשתמש.

הסבירו מה היתרונות ומה החסרונות בהגדלת הפרמטר .TIMEOUT\_G

יתרונות - משפר את חווית המשתמש שרוצה לבחור בעצמו את מערכת ההפעלה, מאפשר לו זמן לבחור את הבחירה המתאימה לו ביותר.

חסרונות – מאריך את זמן הטעינה של מערכת ההפעלה.

עשוי לפגוע במשתמש שיש לו מערכת הפעלה אחת על המחשב או מערכת הפעלה אחת שהוא עובד עליה.

המשתמש צריך להיות בזמן טעינת מערכת ההפעלה כדי לבצע בחירה.

עשוי לבלבל משתמשים שלא יודעים איזה מערכת הפעלה לבחור.

1. מדוע הפונקציה process\_init\_run אשר נמצאת בקובץ c.main/init בקוד הגרעין קוראת לפונקציה execve\_do במקום לקריאת המערכת?execve

המודול init/main הוא חלק מקוד הגרעין כלומר הוא יכול לקרוא רק לפונקציות גרעין, אין לו תמיכה של פונקציות ספרייה שקיימות רק במרחב המשתמש כמו הפונקציה execve() שממומשת בספרייה libc .בעצם פונקציה שנמצאת בקוד המשתמש מבצעת קריאת מערכת כדי שקוד הגרעין יפעיל את הפונקציה המתאימה בקוד הגרעין כמו במקרה הזה, do\_execv() זאת פונקציה בקוד הגרעין שנקראת כאשר בקוד המשתמש משתמשים בפונקציה execve() .

אם ננסה לקרוא ל- execve() מתוך קוד גרעין, הוא לא יזהה את הפונקציה כי אינו מכיר אותה ולכן הקוד לא יתקמפל.

1. מה עושה קריאת המערכת syscall ?כמה ארגומנטים היא מקבלת ומה תפקידם? באיזו ספריה ממומשת קריאת המערכת syscall ?

קריאת המערכת syscall() היא פונקציית ספרייה glibc והיא מצבעת קריאת מערכת ישירה המבקשת באופן ישיר (ללא פונקציית מעטפת) מהקרנל של מערכת ההפעלה איזה פעולה לבצע לפי המספר שמועבר כפרמטר. המספר שמועבר הוא מסוג integer ויכול לקבל את הערכים 1-256.

לכל קריאת מערכת מקושר מספר ש-syscall מקבלת כפרמטר – זהו פרמטר חובה.

בנוסף, syscall מקבלת פרמטרים נוספים שיועברו בהתאם לקריאת מערכת שרוצים לבצע, לפי סדרם, אם יועברו יותר מידי פרמטרים ממה שהפונקציה מקבל, syscall תתעלם מהם.

ערך החזרה מוגדר לפי ערך החזרה של קריאת המערכת שמתעוררת, אם מוחזר 0 זה מציין הצלחה אם הוא מוחזר -1 זה מציין שגיאה ומספר השגיאה מאוחסן בernno .

1. מה מדפיס הקוד הבא? האם תוכלו לכתוב קוד ברור יותר השקול לקוד הבא?

קריאת מערכת syscall(39) זאת קריאת מערכת שמחזירה את ה-pid של התהליך שרץ.

זה שקול לפונקציה שמבקשת ממערכת ההפעלה את ה-pid של התהליך שרץ , קוד הכתוב באופן הבא:

#include <unistd.h>

Int main() {

pid\_t r = getpid();

printf(“sys\_hello returned %ld\n”, r);

return 0;

}

1. התבוננו בתוכנית הבדיקה c.test1 שסופקה לכן והסבירו במילים פשוטות מה היא בודקת.

התוכנית מדפיסה את המשקל של התהליך הנוכחי, אם הוא שווה אפס היא משנה אותו למשקל ששווה 5, אחרת יוצאת מהתהליך. לאחר השינוי ל-5 בודקת אם השינוי הצליח, אם לא הצליח, יוצאת מהתהליך, אחרת שומרת במשתנה x את המשקל החדש, אם הוא שווה ל-5 מדפיסה SUCESS אחרת יוצאת מהתוכנית.

**חלק 2 – זימון תהלכים :**

**שאלה 1:**

**סעיף א :**

**Convoy effect :** מאט את כל התהליכים (יכולים להיות קצרים) בעקבות תהליך ארוך מאוד שרץ לפני.

1. **FCFS** – יש אפקט שיירה - זה אלגוריתם מבוסס FIFO ולכן מה שנכנס ראשון ירוץ ראשון , אם יכנסו תהליכים ארוכים או תהליך ארוך בהתחלה זה יאט את כל שאר התהליכים שיכנסו אחרי.
2. **RR** – אין אפקט שיירה, כל תהליך שממתין מקבל את הquantum בתוך הepoch הבא .
3. **SRTF** – אין אפקט שיירה – אלגוריתם זה מחשב מחדש את התהליך שצריך לרוץ ,כל פעם שמגיע תהליך למערכת ונותן לתהליך הקצר ביותר לרוץ , כלומר מבצע החלפת הקשר כשיש צורך ולכן תהליך ארוך לא יכול לתקוע ריצה של תהליכים אחרים לרוץ .
4. כל מדיניות זימון עם הפקעה ( preemption) – אין אפקט שיירה, כדי לפתור את בעיית אפקט השיירה השתמשו בהפקעה.
5. כל מדיניות זימון בלי הפקעה ( preemption) – ראינו דוגמא בסעיף 1 לאלגוריתם כזה.

לא למדנו את כל האלגוריתמים האפשריים בלי הפקעה , אבל כל אלגוריתם עם מעבד יחיד שיכול באיזשהו שלב לתת לתהליך ארוך לרוץ בלי החלפת הקשר יכול לגרום לאפקט שיירה.

1. לא נכון .

**סעיף ב :**

כפי שלמדנו, תחת תנאים מסוימים אלגוריתם SJF הינו אופטימלי עבור מדד זמן תגובה ממוצע. מהם שלושת התנאים?

1. מעבד אחד.
2. כל התהליכים מגיעים ביחד.
3. זמן הריצה שלהם ידוע.

**שאלה 2:**

**Text, letter

Description automatically generated**

נסתכל על epoch חדש ,קיימים n תהליכים בתור.

מנתון b הזמן הוירטואלי של כל התהליכים שווה בתחילת הepoch ולכן מתקיים כי תוספת הזמן הוירטואלי של כל התהליכים שווה בכל epoch (על מנת שבתום הepoch הזמן הוירטואלי ימשיך להיות זהה).

עבור תהליך i עם קוואנטום Qi מתקיים :

*כאשר הוא תוספת הזמן שתתווסף לזמן הוירטואלי של תהליך i בסוף הקוואנטום Qi .*

*מכיוון ש- שווה עבור כל התהליכים נקבל:*

*לכל תהליך אחר j כאשר :*

*לכן .*

*לפי נתון c: , נציב לכל במקום את הנוסחה שמצאנו :*

**שאלה 3:**

A screenshot of a computer

Description automatically generated with low confidence

1. פגיעה בהוגנות של CFS שיכול לגרום להרעבה של תהליכים אחרים.

אם יש שני תהליכים במערכת A ו-B , נניח ולתהליך A זמן וירטואלי נמוך , אז הוא נותן לו לרוץ ואז שהוא חוזרת מהריצה יש לו את אותו זמן וירטואלי (הנמוך משל B) ולכן הוא ירוץ שוב ושוב , אם A זה תהליך ארוך ,B ימתין הרבה זמן.

1. מציאת מינימום ברשימה יקח O(n) במקום log(n) , נצטרך לעבור על כל הרשימה כדי למצוא את התהליך עם זמן הריצה הוירטואלי הנמוך ביותר שירוץ על המעבד.
2. אם נסיר את המינינמום על גודל הקוונטום ומספר התהליכים N במערכת גבוה, המערכת עלולה לסבול מהחלפות הקשר תכופות ופגיעה בביצועים. החלפות הקשר תכופות מובילות לתקורה גבוהה של מערכת ההפעלה ולפגיעה בביצועים (הרבה משאבים וזמן יבוזבזו על החלפת הקשר).
3. הepoch של האלגוריתם הוא סכום כל הקוואנטומים של התהליכים, כלומר הepoch יהיה לא קבוע. אם במערכת יהיה מס' גבוה של תהליכים, הepoch יכול להיות מאוד גדול, וזה יגרום לתהליכים להיות פחות אינטרקטיביים ופחות תגובתיים, כי הם רצים פעם אחת בכל epoch. אם במערכת יהיה מס' נמוך של תהליכים, הepoch יכול להיות מאוד קטן, דבר שיגרום להחלפות הקשר תכופות ולתקורה גבוהה של מערכת ההפעלה.
4. בשיטה המקורית, זמן הריצה הוירטואלי גדל ביחס ישר ל־ כאשר זמן הריצה הוירטואלי גדל ביחס ישר ל־ המשקל של תהליך מקבל פחות משקל בחישוב הזמן הוירטואלי . המשמעות מכך היא שהיחס בין צבירת הזמן הוירטואלי של תהליכים עם משקל קטן יותר לבין הצבירה בתהליכים עם משקל גדול יותר תקטן, תהיה הבחנה פחותה בין תהליכים בעלי עדיפויות שונות.
5. פגיעה בהוגנות האלגוריתם והרעבה של התהליכים עם זמן וירטואלי גדול. תהליכים חדשים יכנסו לאמצע התור במקום לסוף התור ובמידה ויגיעו תהליכים חדשים רבים אז תהליכים ישנים שהזמן הוירטואלי שלהם גבוה יותר ירעבו כי כל התהליכים החדשים שהגיעו ירוצו לפניהם.

**שאלה 4:**

Text

Description automatically generated

לפי שירי, חוט ירוץ עד לסיומו ואם זה תהליך ארוך אז הוא ירעיב את שאר התהליכים, כי נמתין עד לסיום ריצת כל החוטים של התהליך.

תהליך יכול לייצר מספר גדול של חוטים, חוטים חולקים את אותו מרחב זיכרון לכן כל החוטים של אותו תהליך יקבלו עדיפות על פני תהליכים אחרים ונצטרך להמתין לסיום הפעולה של כל החוטים של אותו תהליך ומכאן תהיה הרעבה של החוטים של התהליכים שאינם שייכים לחוטים של התהליך שרץ . לכן עומר צודק.

**שאלה 5:**

Table

Description automatically generated

Chart, line chart

Description automatically generated

1. הקשר שקיבלנו הוא : .

לכן

לכן יש קשר הפוך בין ערך ה-nice לבין במשקל של התהליך.