**מבנה מערכות הפעלה – 046209**

**תרגיל בית 1**

**איגור דה פאולה – 319407045**

**דניאל (חפון?) –**

**1.**

**1.1**

א. switch – המושג switch מתקשר לcontext של תהליך. הcontext של תהליך זה המרחב שהתהליך חי פה. כדי להפריד בין תהליכים שונים מערכת ההפעלה מגדירה לכל תהליך context, בו נמצא הזיכרון שלו, הPID,UID ושאר המאפיינים של תהליך. כאשר עושים context switch מערכת ההפעלה שומרת את כל הפרמטרים של תהליך אחד, ומחליפה את התהליך שרץ לתהליך אחר עם הפרמטרים שלו.

ב. Preemption – לקיחת משאב מתהליך שעדיין זקוק לו (לדוגמה תהליך שצריך זמן מעבד אך באמצעות Preemption לוקחים לו אותו, במקרה הזה יגרום לcontext switch)

ג. Interrupt - סימון למעבד שישנו ארוע הדורש את טיפולו המיידי ( נובע מהחומרה לכן אין החלפת הקשר וזה רץ על התהליך שכבר היה). יכול להיות קשור להתהליך עצמו (חילוק ב0) ויכול להיות קשור לרכיב חומרה אשר צריך לקבל טיפול. אחרי הטיפול ההמעבד חוזר לטפל בפעולות שהופסקו.

ד. ראה סעיף א.

ה. אופן זימון התהליכים של מערכת ההפעלה (בדרך כלל מתייחס לאלגוריתם בו מערכת ההפעלה משתמשת בשביל לקבוע אילו תהליכים ירוצו).

**1.2**

1.

א. זמן ביצוע כולל – זמן זה הוא הסכום של זמן הריצה וזמן ההמתנה. באמצעות הפקעה ניתן להוריד את זמן ההמתנה (באמצעות אלגוריתמים (לדוגמה שהקצר ביותר קודם)). לכן ניתן לשפר.

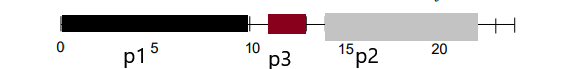
ב. תפוקה – ניתן האמצעות הפקעה לתת לתהליכים קצרים לרוץ קודם וכך לשפר את התקופה.

ג. ניצולת מעבד – באמצעות הפקעה ניתן להחליף תהליך שמחכה לתהליך אחר שיכול לרוץ בזמן שהוא מחכה. לכן המעבד אינו נח והניצולת גדלה.

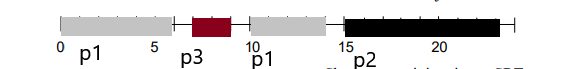
ד. slow down – קשור ישירות בזמן ביצוע כולל לכן גם זה יכול לשפר.

**2.**

SJF –



SRT –



א. אם יש תהליכים עתירי חישוב ואינטרקטיבים הייתי משתמש בSRT. כדי לא להמתין ולקבל תחושה שרצים על המעבד תהליכים במקביל ולקבל זמן תגובה מינימלי.

ב. אם יש תהליכים עתירי חישוב בלבד נשתמש בSFJ. זמן ההמתנה ירד ולא נקבל OVERHEAD כתוצאה מהחלפת הקשר תחופה.

**3.**

רצים  תהליכים. כל קריאה לfork מפצלת את התהליך ל2 (אב ובן). לאחר הפיצול כל תהליך ממשיך מהשורה הבאה לכן כל תהליך שנוצר גם יתפצל שוב. נקבל פיצול מעריכי לכן התוצאה היא 2 בחזקת מספר הקריאות. הפחות אחד נובע מכך שהתהליך הראשוני הינו יחיד.

**4.**

התוכנית תדפיס את מספר הקורס וירידת שורה עד שיגמר הזיכרון.

**5.**

א. סיבה שיכולה להסביר ששינוי הזמן לא השפיע על סדר הזימון הינו שאלגוריתם RR משתמש בקוונטה של זמן כדי להחליף את ההקשר. אלגוריתם CFS מקצה זמני ריצה בהתאם לעדיפות של כל אחד. כאשר הזמן המינימלי הינו משותף (minimal slice time). אם כל התהליכים הגיעו עם אותה חשיבות, כל תהליך יקבל את אותו קוונטה של זמן (אם הminimial time slice = quanta) כך שזימון התהליכים לא ישתנה בין האלגוריתמים.

ב. הגדלת זמן המחזור אומר שכל תהליך יקבל חתיכת זמן יותר גדולה לרוץ עליה. בהנחה שכל התהליכים הגיעו עם אותו ה(עדיפות) עדיין כל התהליכים יקבלו את אותו הזמן (כי המשקל בגלל העדיפות יהיה זהה). זמן התגובה ירד מפני שהזמן שכל תהליך יקבל על המעבד יגדל. התפוקה תגדל מפני שיהיה פחות overhead מהcontext switch. ההוגנות כנראה תרד מפני שהיא נמדדת ביחס לזמן מושלם שיודע להחליף הקשרים כך שכל תהליך מקבל במדויק את חלקו, כאשר נגדיל את זמן המחזור נקטין את הגמישות של הזמן ולכן ההוגנות תיפגע.

ג.

תהליך רץ slice\*(NICE\_LOAD\_0/load.waeight) בכל זמן מחזור.

ד.

הזמן IO שמשתמש באלגריתם FCFS אינו יעיל מפני שתהליך יכול להמתין על המעבד עד שהוא יסתיים וכך הניצולת יורדת. כמו כן אין סנכרון בין זמני התהליכים שמבקשים את הIO לבין הזמן של הIO בעצמו מבחינת זמן ריצה של התהליך. כדי להגדיל את נצילות המעבד ניתן לצרף את הבקשות IO לזמן של התהליך אך להשתמש בהפקעה אשר מחליפה את התהליך כאשר נוצרת בקשה כזו. כך המעבד יהיה ב100 עבודה.