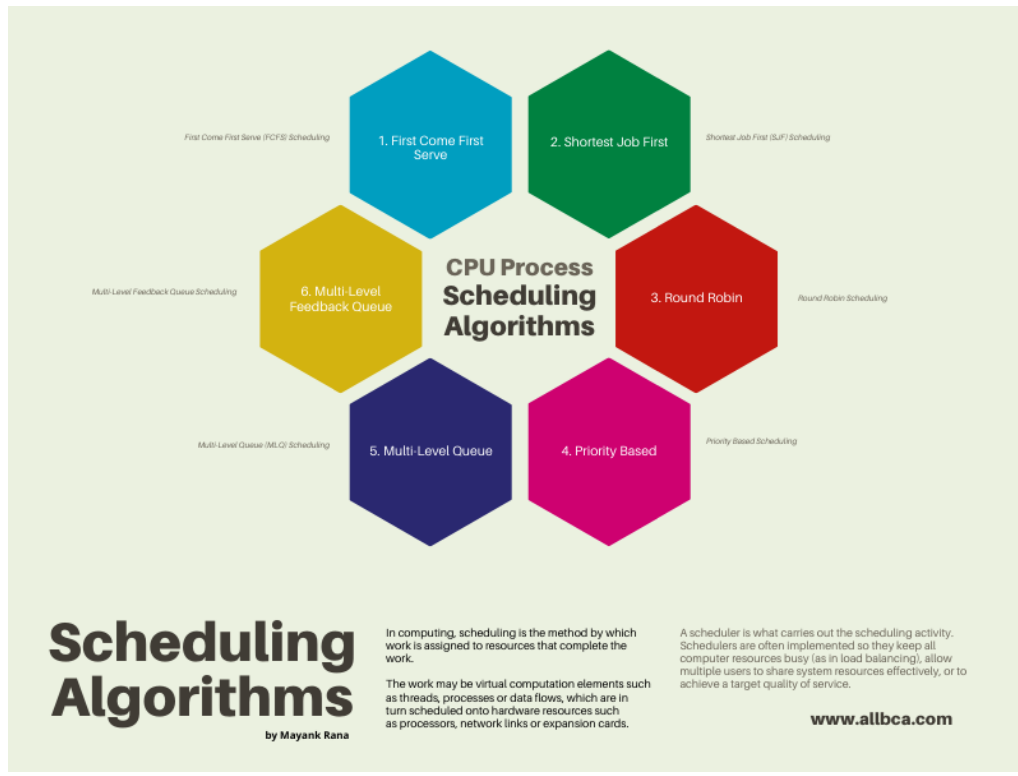


אלגוריתמי תזמון במעבד



שם: אוראל בריאנצב

ת"ז: 318264132

סמסטר: 2024א

תוכן עניינים

1.	מבוא -----	עמוד 3
2.	פרק 1 : הצגת אלגוריתם Priority Scheduling	
2.1	סקירה כללית -----	עמוד 4
2.2	פרמטר מרכזי -----	עמוד 5
2.3	השפעת האלגוריתם על ביצועי המערכת -----	עמודים 6-7
2.4	הדגמת ביצועי האלגוריתם -----	עמודים 8-14
3.	פרק 2 : הצגת אלגוריתם Round Robin	
3.1	סקירה כללית -----	עמוד 15
3.2	פרמטר מרכזי -----	עמוד 16
3.3	השפעת האלגוריתם על ביצועי המערכת -----	עמודים 17-18
3.4	הדגמת ביצועי האלגוריתם -----	עמודים 19-23
4.	פרק 3 : הצגת אלגוריתם Shortest Job First	
4.1	סקירה כללית -----	עמוד 24
4.2	פרמטר מרכזי -----	עמוד 25
4.3	השפעת האלגוריתם על ביצועי המערכת -----	עמוד 26
4.4	הדגמת ביצועי האלגוריתם -----	עמודים 27-29
5.	פרק 4 : השוואת אלגוריתמים -----	עמודים 31-41
6.	פרק 5 : סיכום -----	עמוד 42
7.	ביבליוגרפיה -----	עמוד 43

מבוא

בעבודה זו נסקור ונבחן ארבעה לאגוריתמי תזמון תהליכים מרכזיים : Round Robin, Shortest Job First, Non-Preemptive Priority, Preemptive Priority. כל אחד מהאלגוריתמים נועד להתמודד עם תזמון תהליכים במערכות הפעלה, כאשר לכל אחד מהם יש יתרונות וחסרונות בהתאם למטרות השונות של המערכת.

המטרה המרכזית בעבודה זו היא להערך את ביצועי האלגוריתמים בהקשר של ארבעה מדדים עיקריים: הוגנות (Fairness), קצב עבודה (Thought), תקורה (Overhead) וניצולת מעבד (CPU utilization). הוגנות מתייחסת לחלוקה שווה של משאבי המעבד בין התהליכים ומוודאת כי אף תהליך לא ממתין זמן רב לקבלת מעבד כתוצאה מהעדפת תהליכים אחרים. קצב עבודה הוא מדד לכמות התהליכים שהמערכת מסיימת בזמן נתון והוא משקף את היעילות של המערכת בביצוע משימות. תקורה נובעת בעיקר מהמעברים בין תהליכים כאשר החלפות תהליכים (context switch) גוזלים ממשאבי המערכת זמן. ניצולת המעבד מודדת את אחוז הזמן שבו המעבד עוסק בביצוע תהליכים ולא מבזבז זמן בהמתנה או בתפעול תקורות.

בכל אלגוריתם נבצע ניתוח של הפרמטר המרכזי המשפיע על ביצועי האלגוריתם, לדוגמה באלגוריתם Round Robin הפרמטר המרכזי הוא Quantum – נתח הזמן שכל תהליך מקבל במעבד עד שעובר לסוף התור. באלגוריתם Shortest Job First הפרמטר המרכזי הוא Burst Time – זמן הביצוע של כל תהליך. באלגוריתם Priority הפרמטר המרכזי הוא רמת העדיפות של כל תהליך כאשר האלגוריתם בוחר את התהליך המתועדף ביותר.

לאחר הצגת האלגוריתמים וניתוח הפרמטרים המרכזיים שלהם, נבצע השוואה מעשית בין האלגוריתמים באמצעות סימולטורים שפותחו במיוחד עבור עבודה זו. הסימולטורים המדמים את ריצות האלגוריתמים בסביבת לינוקס ומחשבים את זמני ה Completion Time (CT), Turnaround (TT), Waiting Time (WT) ו Response Time (RT) כמו גם את ניצולת המעבד והזיכרון ומידת ההוגנות של כל אחד מהאלגוריתמים. נשתמש במדדים אלו כדי להערך את ביצועי האלגוריתמים באופן כמותי ואיכותי.

בסיכום העבודה, נבצע השוואה כוללת בין האלגוריתמים על פי המדדים שהוזכרו וננתח את היתרונות והחסרונות של כל אחד מהם בהתאם לתוצאות הסימולציה. כך נוכל להצביע על האלגוריתם המתאים ביותר עבור סוגי מערכות ותהליכים שונים ולהבין כיצד בחירת האלגוריתם משפיעה על ביצועי המערכת.

פרק 1: אלגוריתם Priority Scheduling

1.1 סקירה כללית

אלגוריתם העדיפויות, Priority scheduling, הוא אלגוריתם מרכזי לתזמון תהליכים במעבד, בו מקצים זמן CPU לתהליך בעל העדיפות הגבוהה ביותר. כל תהליך במערכת מקבל עדיפות בהתאם למספר קריטריונים והאלגוריתם מקצה זמן CPU לתהליך בעל העדיפות הגבוהה ביותר.

האלגוריתם הוא רכיב משמעותי בניהול תהליכים במערכות הפעלה בהן יש חשיבות רבה לקביעת סדר עדיפויות בין התהליכים השונים על מנת להבטיח שתהליכים קריטיים יקבלו את המשאבים הדרושים להם בזמן, תוך שמירה על זמן תגובה נמוך, כפי שמקובל במערכות זמן אמת.

קיימות שתי גרסאות מרכזיות לאלגוריתם הזה:

○ Non-preemptive ללא הפקעת מעבד

בגרסה זו, כאשר תהליך מקבל זמן CPU, הוא ירוץ עד שיסיים את ריצתו ללא התחשבות בתהליכים חדשים המגיעים עם עדיפות גבוהה יותר משל התהליך שרוץ. רק בסיום ריצת התהליך הנוכחי, האלגוריתם יבחן את כל התהליכים הממתינים ויבחר בתהליך בעל העדיפות הגבוהה ביותר מבין כל התהליכים שנמצאים במערכת ויקצה לו זמן CPU.

גרסה זו מצמצמת את התקורה הנגרמת מהחלפת התהליכים היות ולא מתבצעת הפקעת מעבד כאשר תהליך מסוים מקבל זמן CPU ומתחיל את ריצתו. מנגד, גרסה זו מובילה לזמן המתנה גבוה ולעיתים גם עלולה לגרום להרעבה¹ עבור תהליכים בעלי תעדוף נמוך משום שהם צריכים להמתין זמן רב עד שכל התהליכים בעלי העדיפות הגבוהה יותר יסיימו את ריצתם.

○ Preemptive עם הפקעת מעבד

בגרסה זו, כאשר תהליך מקבל זמן CPU ומגיע תהליך בעל עדיפות גבוהה יותר, האלגוריתם יבצע החלפת תהליכים כך שהתהליך בעל העדיפות הגבוהה ביותר יקבל זמן CPU ויתחיל לרוץ. גרסה זו, יכולה לספק זמן תגובה מהיר יותר עבור תהליכים בעלי עדיפות גבוהה יותר ולכן עדיפה עבור מערכות הפעלה זמן אמת הדורשות זמן תגובה מיידי. מנגד, החלפת התהליכים המרובה גורמת לתקורה גבוהה שעלולה להוביל לניצולת מעבד נמוכה.

הגרסה המתאימה תיבחר בהתאם למטרות ולדרישות הספציפיות של המערכת הנתונה. למשל, מערכת הפעלה שמיועדת לשרתים של מרכזי נתונים תעדיף את הגרסה הראשונה על מנת למקסם יציבות ולמנוע תדירות גבוהה של החלפות תהליכים שעלולה לפגוע בביצועי המערכת. מנגד, מערכת הפעלה שמיועדת לשליטה ובקרה בסביבה תעשייתית, בה יש צורך בתגובה מיידי לאירועים, תעדיף את הגרסה השנייה על מנת להבטיח זמן תגובה מהיר.

¹ הרעבה – מצב בו תהליכים רצים אך אין התקדמות ממשית בריצתם.

1.2 פרמטר מרכזי

הפרמטר המרכזי באלגוריתם הוא העדיפות הנקבעת לכל תהליך, המשמשת כקריטריון להקצאת זמן המעבד לתהליכים במערכת. קיימות שתי אפשרויות להקצאת עדיפות לתהליכים:

- הקצאת עדיפות סטטית
העדיפות של כל תהליך מוקצת לתהליך במועד יצירתו ונשארת קבועה לאורך כל חיי התהליך, לא ניתן לשנות את עדיפות התהליך לאחר שנקבעה. העדיפות נקבעת לפי מדדים שונים כגון חשיבות התהליך, קלט מהמשתמש, משאבים הדרושים להרצת התהליך וכו'. גרסה זו נפוצה בעיקר במערכות זמן אמת בהן חשוב שתהליכים מסוימים יקבלו עדיפות עליונה ללא תלות במצב המערכת. כמו כן, גם במערכות משובצות יש שימוש בגישה זו שכן מערכות אלו מיועדות לסביבות עבודה בעלות דרישות ידועות מראש.
- הקצאת עדיפות דינמית
בניגוד לגרסה הקודמת, בגרסה הדינמית העדיפות שנקבעת לכל תהליך יכולה להשתנות בזמן ריצת התהליך. במקרה זה העדיפות מושפעים מהגורמים הבאים:
 - אלגוריתם ההזדקנות²: העדיפות של תהליכים שממתינים הרבה זמן תעלה על מנת למנוע הרעבה.
 - סוג: תהליכים הממתינים לקלט/פלט מהמשתמש יקבלו עדיפות גבוהה יותר משל תהליכים הממתינים לזמן CPU.
 - גורם חיצוני: משתמש או אדמינסטרטור עלולים לשנות את עדיפות התהליכים.גישה זו מספקת גמישות ויכולת להתאים את עדיפות התהליכים בהתאם לגורמים משתנים המשפיעים על המערכת. הקצאת עדיפות דינמית נדרשת במערכות הפעלה כלליות ובמיוחד במערכות הפעלה מרובות משתמשים ושיתופיות, במערכות אלו ישנו צורך לאזן בין דרישות שונות של תהליכים רבים בו זמנית.

הבחירה בין שתי הגרסאות מושפעת ממגוון שיקולים הקשורים למטרות המערכת, סוג היישומים בה היא תומכת ובצרכים של משתמשי המערכת. כאשר מערכת ההפעלה נדרשת לתמוך ביישומים שדורשים יציבות גבוהה וניתן לחזות את התנהגותם בצורה מדויקת, ייתכן שהגרסה הסטטית תתאים יותר היות וגרסה זו מבטיחה שעדיפויות התהליכים לא ישתנו ומאפשרת תכנון מראש וחיזוי טוב יותר של זמני התגובה והביצועים של המערכת. מנגד, כאשר מדובר במערכת הפעלה שמתמודדת עם סביבות דינמיות שדרישותיהן משתנות באופן תדיר, נעדיף את הגרסה הדינמית משום שבסביבות אלו, עדיפות התהליכים משתנות בזמן אמת כדי להתאים את המערכת למשימות ולצרכים המתעדכנים של משתמשי המערכת. הגרסה הדינמית יכולה להיות יעילה במיוחד גם עבור מערכות מרובות משתמשים בהן יש צורך בזמן תגובה מהיר לכל משתמש ובאיזון בין דרישות כל המשתמשים.

² אלגוריתם ההזדקנות (aging) - נועד למנוע הרעבה, עדיפות תהליכים ותיקים תעלה על מנת שיקבלו זמן CPU.

1.3 השפעת האלגוריתם על ביצועי המערכת

אלגוריתם Priority הוא אלגוריתם נפוץ במערכות הפעלה, במיוחד במערכות הפעלה בהן יש חשיבות לביצוע משימות קריטיות בטווח זמן מוגדר. היתרון הבולט של האלגוריתם הוא היכולת להבטיח שתהליכים בעלי עדיפות עליונה יקבלו גישה מהירה למעבד, כלומר לזמן CPU.

אם זאת, האלגוריתם יכול להוביל לחוסר הגינות ולהרעבה של תהליכים בעלי תעדוף נמוך. השפעת האלגוריתם על ביצועי מערכת ההפעלה תלויה בסוג הקצאת העדיפות.

נסקור את האלגוריתם ע"פ הגורמים שהגדרנו:

הגינות

- עדיפות סטטית: מורידה את מידת ההגינות משום שתהליכים בעלי תעדוף נמוך יותר ימתינו בזמן שתהליכים בעלי עדיפות גבוהה יותר יהיו בתור. גישה זו עלולה לגרום להרעבה של תהליכים עם תעדוף נמוך.
- עדיפות דינאמית: משפרת את מידת ההגינות היות וניתן לשנות את עדיפות התהליכים לפי זמני ההמתנה שלהם. גישה זו מאפשרת גם לתהליכים בעלי תעדוף נמוך לקבל זמן CPU ובכך בעצם למנוע הרעבה ולשפר את רמת ההגינות שהאלגוריתם מספקת.

קצב העבודה

- עדיפות סטטית: קצב העבודה תלוי בסוג התהליכים, כאשר מדובר בתהליכים שממתינים לזמן CPU בלבד, נצפה לקצב עבודה גבוהה יותר היות ותהליכים אלו מסיימים את ריצתם יחסית במהירות. לעומת זאת כאשר מדובר בתהליכים אינטרקטיביים הממתינים לפעולות I/O³, נצפה לקצב עבודה איטי יותר משום שזמן המעבד מתבזבז על המתנה לפעולה מגורם חיצוני.
- עדיפות דינאמית: קצב העבודה מהיר יותר משום שניתן לתזמן את התהליכים בהתאם לצרכי המערכת, לדוגמה במקרה של תהליכים אינטרקטיביים בעלי תעדוף גבוה, ניתן להנמיך את התעדוף שלהם לטובת תהליכים שצריכים זמן CPU בלבד.

תקורה

- עדיפות סטטית: התקורה נמוכה יותר משום שלאחר קביעת עדיפות התהליכים אין צורך (ואפשרות) בשינויים נוספים. גישה זו מורידה את העומס החישובי של המערכת.
- עדיפות דינאמית: התקורה גבוהה יותר משום שבגישה זו נדרש מעקב מתמשך ושינויי עדיפות תכופים של התהליכים. המערכת נדרשת להעריך ולחשב את עדיפות התהליכים כל הזמן ולכן התקורה גדלה. כמו כן, החלפת התהליכים המרובה מעלה את התקורה הכללית במערכת.

³ פעולות I/O – תהליכים שבמהלכם מקבלים מידע מהתקני הקלט או שולחים מידע מהתקני הפלט של המערכת.

ניצולת המעבד

- עדיפות סטטית: ניצולת המעבד יכולה לרדת במקרים בהם תהליכים אינטרקטיביים עם עדיפות עליונה ימתינו בתכיפות לפעולות I/O בזמן שהמעבד במצב ⁴idle בזמן שיש תהליכים בעלי תעדוף נמוך יותר שמוכנים לקבל זמן CPU.
- עדיפות דינאמית: ניצולת המעבד יכולה לעלות ע"י שינוי העדיפויות כך שהמעבד כל הזמן יהיה תפוס עם התהליך המתאים ביותר. לדוגמה, עבור כאשר תהליך עם עדיפות עליונה ממתין לפעולות I/O שיתבצעו, ניתן לעלות עדיפות עבור תהליכים שצריכים רק את המעבד ובעצם לעלות את ניצולת המעבד.

⁴ מצב idle – מצב בו המעבד אינו מבצע פעולות או מעבד תהליכים, נמצא במצב המתנה.

1.4 הדגמת ביצועי האלגוריתם

נדגים את ביצועי האלגוריתם עבור התהליכים $P_1 - P_5$, זמני הרצת התהליכים הם :

P	AT	BT	Priority
1	0	4	5
2	1	3	4
3	2	1	3
4	3	2	2
5	4	5	1

טבלה 1 – טבלת תהליכים לפי זמן הגעה וזמן ריצה

נדגים את ריצת האלגוריתם ללא הפקעת המעבד ועם הפקעת המעבד, העדיפות נקבעת בסדר הפוך לסדר הגעת התהליכים כלומר תהליך P_1 הוא בעל העדיפות הנמוכה ביותר ותהליך P_5 הוא בעל העדיפות הגבוהה ביותר.

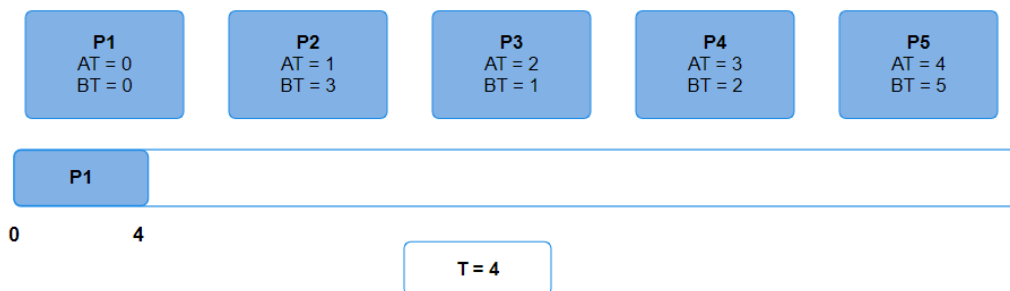
non preemptive – ללא הפקעת מעבד.

כאשר תהליך א' מקבל זמן CPU ומגיע תהליך נוסף ב' עם עדיפות גבוהה יותר, תהליך ב' ימתין עד שתהליך א' יסיים לרוץ ורק לאחר מכן יקבל זמן CPU.

זמן 0 עד 4

בזמן 0, מגיע תהליך P_1 , תהליך זה הוא בעל העדיפות הנמוכה ביותר אך בזמן 0 הוא התהליך היחיד שנמצא במערכת ולכן האלגוריתם יבחר להריץ אותו עד לסיום, כלומר $T = 4$.

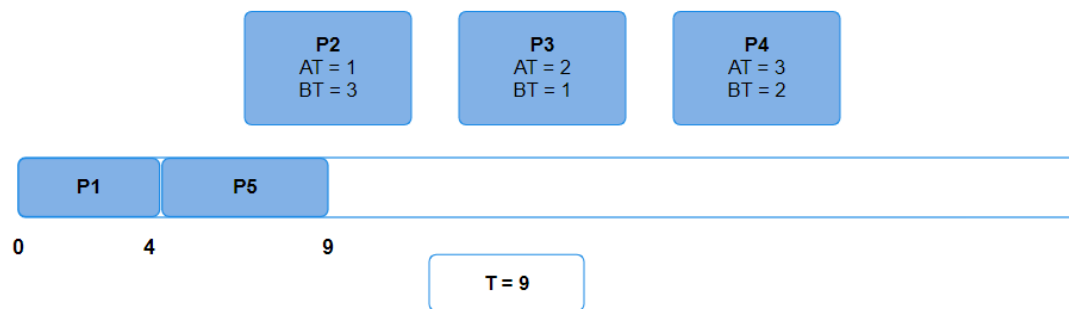
בזמן שתהליך P_1 מקבל זמן CPU, מגיעים תהליכים $P_2 - P_5$ וממתינים לסיום של P_1 .



תמונה 1.0 – אלגוריתם העדיפויות $NP, T=4$

זמן 5 עד 9

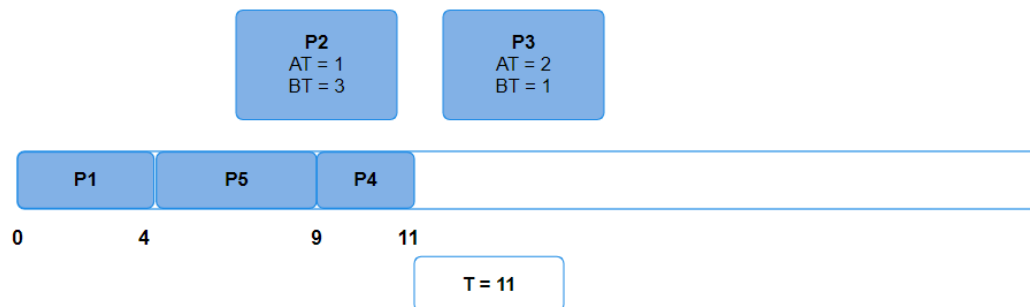
תהליך P_5 הוא התהליך בעל העדיפות הגבוה ביותר ולכן הוא יקבל זמן CPU עד שיסיים לרוץ.



תמונה 1.1 – אלגוריתם העדיפויות NP , $T=9$

זמן 9 עד 11

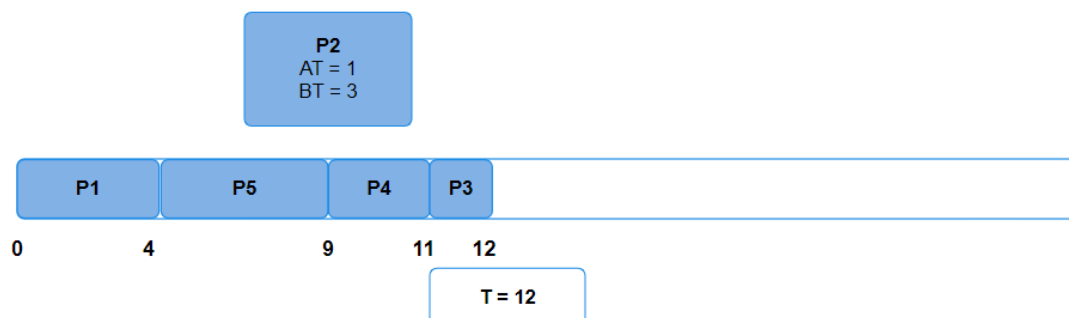
תהליך P_4 הוא התהליך בעל העדיפות הגבוה ולכן הוא יקבל זמן CPU עד שיסיים לרוץ.



תמונה 1.2 – אלגוריתם העדיפויות NP , $T=11$

זמן 11 עד 12

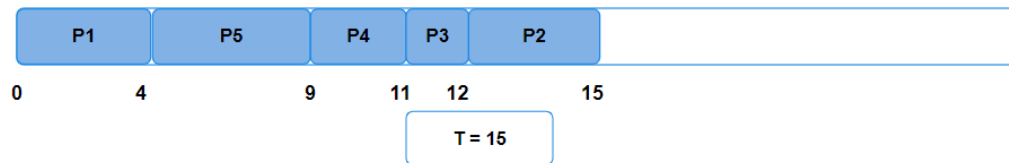
תהליך P_3 הוא התהליך בעל העדיפות הגבוה ולכן הוא יקבל זמן CPU עד שיסיים לרוץ.



תמונה 1.3 – אלגוריתם העדיפויות NP , $T=12$

זמן 12 עד 15

תהליך P_2 הוא התהליך בעל העדיפות הגבוה ביותר ולכן הוא יקבל זמן CPU עד שיסיים לרוץ.



תמונה 1.4 – Gantt chart for priority NP

כעת נחשב את ביצועי האלגוריתם על התהליכים הנ"ל לפי הנוסחאות הבאות:

○ CT – Completion Time

הזמן הדרוש להשלמת התהליך

○ TT – Turnaround Time

הזמן שחלף מהרגע שהתהליך התקבל במערכת ועד הרגע בו סיים לרוץ.

הנוסחה לחישוב היא $TT = CT - AT$

○ WT – Waiting Time

הזמן הכולל שבו תהליך המתין בתור.

הנוסחה לחישוב היא $WT = TT - BT$

○ RT – Response Time

הזמן שחולף מהרגע שהתהליך מגיע לתור עד לפעם הראשונה שהתהליך מקבל זמן CPU.

הנוסחה לחישוב $RT = FR - AT$ כאשר FR זה הזמן בו התהליך קיבל זמן CPU בפעם הראשונה.

נסכום את כל פרטי הרצת האלגוריתם ונקבל:

Non Preemptive Priority							
P	AT	BT	Priority	CT	TT	WT	RT
1	0	4	5	4	4	0	0
2	1	3	4	15	14	11	11
3	2	1	3	12	10	9	9
4	3	2	2	11	8	6	6
5	4	5	1	9	5	0	0

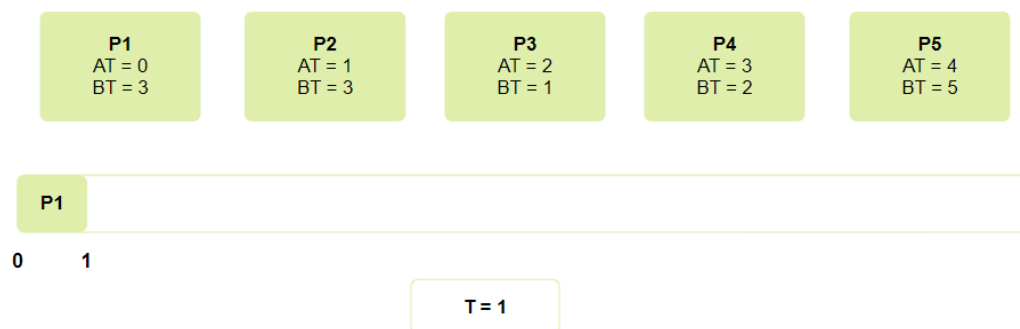
Preemptive – עם הפקעת מעבד

כאשר תהליך אי' מקבל זמן CPU ומגיע תהליך נוסף ב' עם עדיפות גבוהה יותר, תהליך ב' יקבל זמן CPU ותהליך א' יושעה עד שהעדיפות שלו תהיה הגבוהה ביותר.

זמן 0 עד 1

בזמן 0, מגיע תהליך P_1 , תהליך זה הוא בעל העדיפות הנמוכה ביותר אך בזמן 0 הוא התהליך היחיד שנמצא במערכת ולכן האלגוריתם יבחר להריץ אותו.

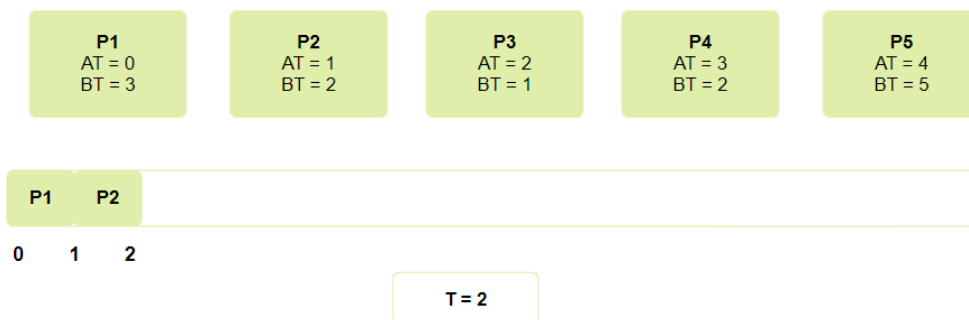
אחרי יחידת זמן אחת, כלומר ב- $T=1$ מגיע תהליך P_2 בעל עדיפות גבוהה יותר משל P_1 ולכן הוא מקבל זמן CPU. תהליך P_1 נותר עם $BT = 3$



תמונה 1.5 – אלגוריתם העדיפויות $T=1, P$

זמן 1 עד 2

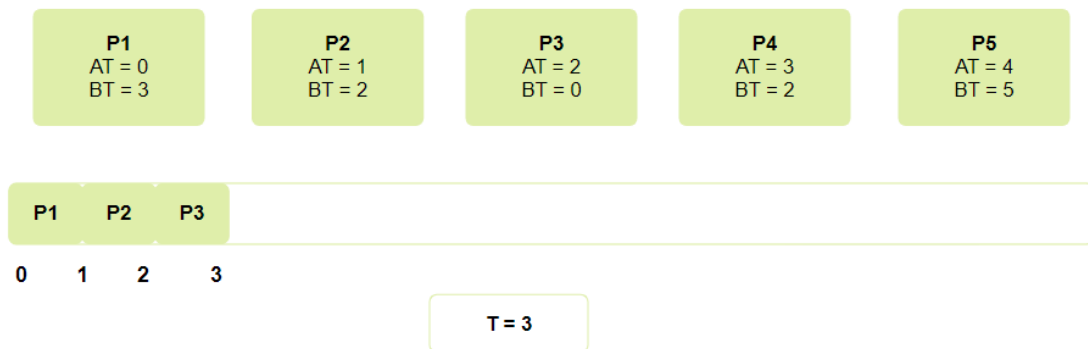
תהליך P_2 מקבל זמן CPU עד שמגיע תהליך P_3 ב- $T=2$ בעל עדיפות גבוהה יותר משל P_2 ולכן הוא מקבל זמן CPU. תהליך P_2 נותר עם $BT = 2$



תמונה 1.6 – אלגוריתם העדיפויות $T=2, P$

זמן 2 עד 3

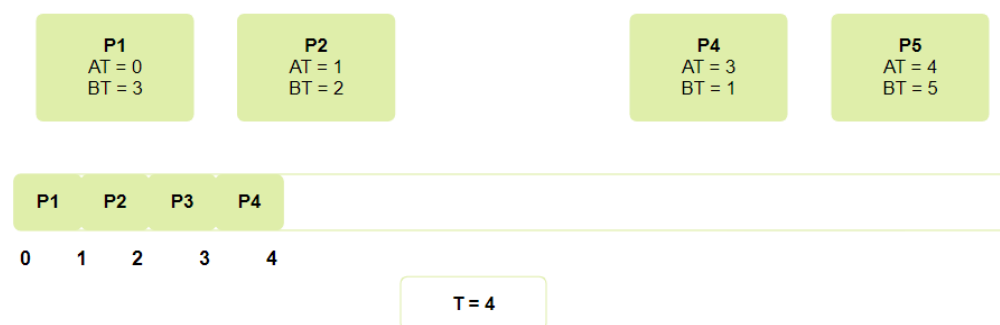
תהליך P_3 מקבל זמן CPU ומסיים את ריצתו ב- $T=3$. בזמן זה מגיע תהליך P_4 שהוא בעל עדיפות גבוהה יותר משל תהליכים P_1, P_2 ולכן הוא יקבל זמן CPU.



תמונה 1.7 – אלגוריתם העדיפויות P , $T=3$

זמן 3 עד 4

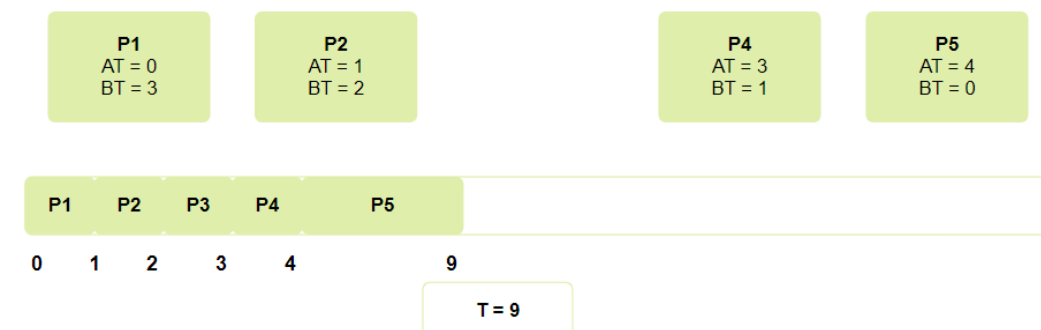
תהליך P_4 מקבל זמן CPU עד שמגיע תהליך P_5 ב $T=4$ בעל עדיפות גבוהה יותר משל P_4 ולכן הוא מקבל זמן CPU. תהליך P_4 נותר עם $BT = 1$.



תמונה 1.8 – אלגוריתם העדיפויות P , $T=4$

זמן 4 עד זמן 9

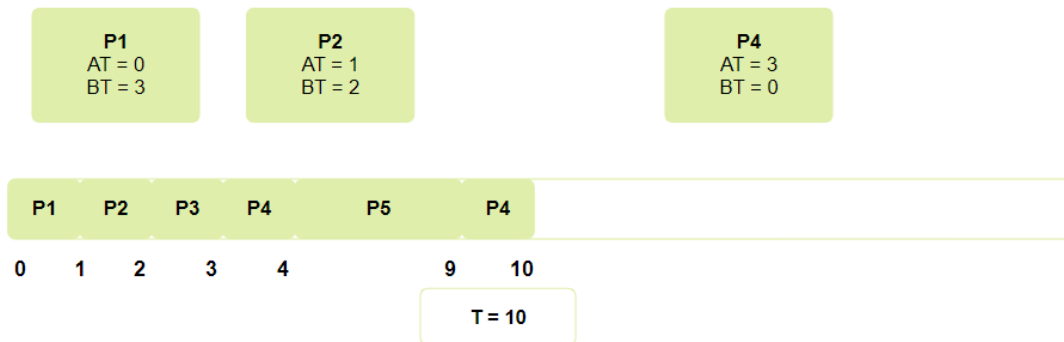
תהליך P_5 מקבל זמן CPU ומסיים לרוץ בזמן $T=9$. לאחר מכן ייבחר התהליך עם העדיפות הגבוהה ביותר, כלומר יבחר תהליך P_4 .



תמונה 1.9 – אלגוריתם העדיפויות P , $T=9$

זמן 9 עד 10

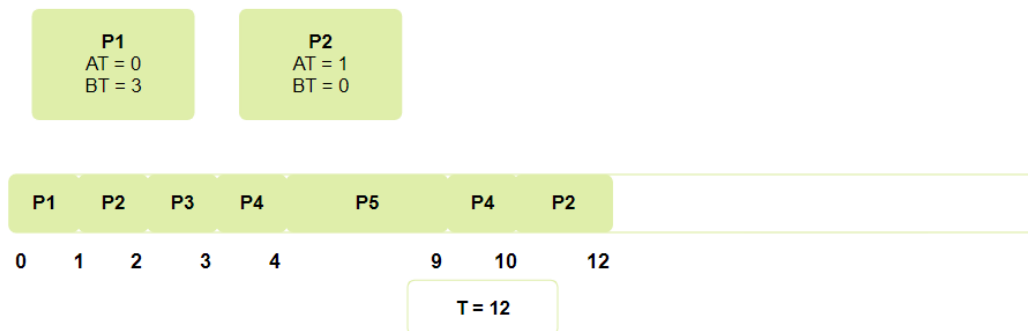
תהליך P_4 מקבל זמן CPU עד שיסיים לרוץ ב $T = 10$. לאחר מכן ייבחר התהליך עם העדיפות הגבוהה ביותר, כלומר יבחר תהליך P_2 .



תמונה 1.10 – אלגוריתם העדיפויות $P, T=10$

זמן 10 עד 12

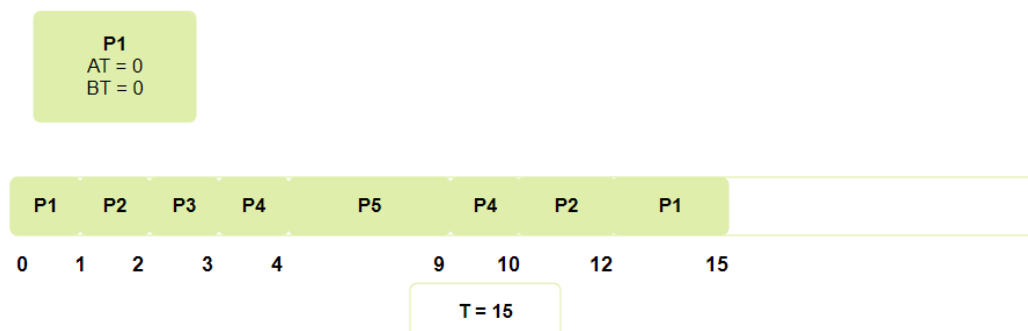
תהליך P_2 מקבל זמן CPU עד שיסיים לרוץ ב $T=12$. לאחר מכן יקבל תהליך P_1 זמן CPU משום שהוא התהליך הנותר.



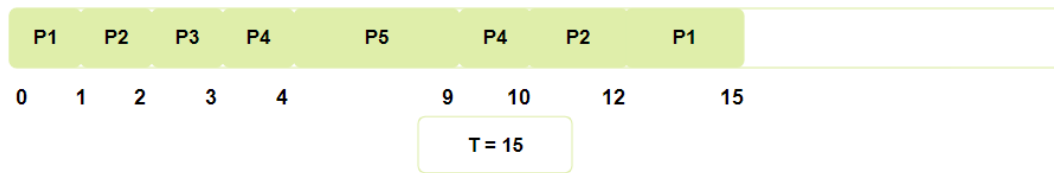
תמונה 1.11 – אלגוריתם העדיפויות $P, T=12$

זמן 12 עד 15

תהליך P_1 מסיים את ריצתו ב $T=15$.



תמונה 1.12 – אלגוריתם העדיפויות $P, T=15$



תמונה 1.13 Gantt chart for priority P

כעת נחשב את ביצועי האלגוריתם על התהליכים הנ"ל לפי הנוסחאות הבאות:

○ CT – Completion Time

הזמן הדרוש להשלמת התהליך

○ TT – Turnaround Time

הזמן שחלף מהרגע שהתהליך התקבל במערכת ועד הרגע בו סיים לרוץ.

הנוסחה לחישוב היא $TT = CT - AT$

○ WT – Waiting Time

הזמן הכולל שבו תהליך המתין בתור.

הנוסחה לחישוב היא $WT = TT - BT$

○ RT – Response Time

הזמן שחולף מהרגע שהתהליך מגיע לתור עד לפעם הראשונה שהתהליך מקבל זמן CPU.

הנוסחה לחישוב $RT = FR - AT$ כאשר FR זה הזמן בו התהליך קיבל זמן CPU בפעם הראשונה.

נסכום את כל פרטי הרצת האלגוריתם ונקבל:

Preemptive Priority							
P	AT	BT	Priority	CT	TT	WT	RT
1	0	4	5	15	15	11	0
2	1	3	4	12	11	8	0
3	2	1	3	3	1	0	0
4	3	2	2	10	7	5	0
5	4	5	1	9	5	0	0

פרק 2: אלגוריתם Round Robin

2.1 סקירה כללית

כעת נבחן את אלגוריתם ההחלפה המחזורית, הידוע בשם Round Robin (RR) הנחשב לאחד מהאלגוריתמים הנפוצים והבסיסיים ביותר בתזמון תהליכים במערכות הפעלה מודרניות.

האלגוריתם מתאפיין בגישה הוגנת לתזמון בה כל תהליך מקבל נתח שווה מזמן המעבד ובך מבטיח הוגנות בין כל התהליכים.

עקרון הפעולה המרכזי של האלגוריתם מבוסס על רשימת תהליכים מוכנים לריצה המאורגנים בתור מעגלי לפי עקרון FIFO על פי סדר הגעתם כלומר התהליך שהגיע ראשון הוא התהליך הראשון שיקבל זמן CPU. כאשר מגיע תהליך חדש הוא מצטרף לסוף התור וממתין לתורו לקבלת זמן CPU.

הפרמטר המרכזי באלגוריתם הוא quantum שהוא בעצם פרק זמן קצוב להקצאת זמן CPU. כל תהליך בתור מקבל הקצאה של המעבד לפי ה quantum המוגדר. על מנת לאפשר לתהליכים אחרים במערכת לקבל זמן CPU, כאשר תהליך מסיים את זמן ה quantum שהוקצב לו ובתנאי שהתהליך לא הסתיים, הוא מושהה ומועבר לסוף התור. המעבד מוקצה לתהליך הבא בתור והתהליך המחזורי ממשיך עד שכל התהליכים הממתנים בתור מסיימים.

האלגוריתם מתאים במיוחד למערכות בעלות מספר רב של תהליכים הדורשים זמן תגובה קצר, למשל במערכות הפעלה מרובות משתמשים בהן יש מספר רב של משתמשים שעשויים לבצע פעולות במקביל, האלגוריתם מבטיח שכל משתמש יקבל חלק הוגן מזמן המעבד וכך מתאפשרת חלוקה הוגנת של משאבים ונמנעים עיכובים בזמן התגובה לכל משתמש. גם במערכות הפעלה זמן אמת בהן חשוב לספק זמני תגובה קצרים, במערכות הפעלה עם תהליכים אינטרקטיביים ובמערכות הפעלה שיתופיות, האלגוריתם יתאים ויספק זמני תגובה קצרים יחסית וביצועי מערכת גבוהים.

אלגוריתם ההחלפה המחזורית ידוע ביכולתו לספק הוגנות, בכך שהוא מבטיח שכל תהליך יקבל זמן CPU שווה. המחזוריות של האלגוריתם, יחד עם זמן ה quantum, מאפשרות טיפול יעיל הן בתהליכים קצרים והן בתהליכים ארוכים. יתרה מכך, האלגוריתם יכול לשפר את זמן התגובה עבור תהליכים קצרים.

באופן כללי, אלגוריתם RR מספק איזון בין הגינות לתגובתיות, ויכול לשפר את התגובה לתהליכים אינטראקטיביים, אך עשוי להידרש לניהול מיטבי של פרמטרי זמן ה- Quantum כדי למנוע השפעות שליליות על תפוקה וזמן המתנה.

2.2 פרמטר מרכזי

ערך quantum הוא פרמטר מרכזי באלגוריתם Round Robin והוא משמעותי לקביעת אופן פעולת האלגוריתם וביצועי המערכת.

הquantum קובע כמה זמן CPU מוקצה לכל תהליך הנמצא בתור המעגלי הוא אחראי לקביעת כמה זמן CPU מוקצה לכל תהליך הנמצא בתור המעגלי ומשפיע ישירות על מדדים המערכת כגון ניצול CPU, תפוקה ותגובתיות המערכת. כמו כן, גודל quantum משפיע גם על פרמטרים אחרים, כמו זמן ההמתנה של התהליכים השונים וזמני השהייה של המערכת.

באלגוריתם Round Robin כל תהליך מקבל זמן CPU באורך quantum שהוגדר מראש, אם תהליך מסוים מסיים את הפעולות הנדרשות ממנו בזמן שהוקצב לו, הוא יוצא מהתור. אולם, מצב בו תהליך זקוק ליותר זמן quantum ממה שהוקצה לו, הוא ממשיך לרוץ עד תום זמן quantum ומועבר לסוף התור. בפעם הבאה שהתהליך יקבל זמן CPU, זמן הביצוע שלו יהיה קטן יותר כך שהזמן שיידרש לסיים את המשימה יתקצר באופן הדרגתי.

קביעת אורך quantum לערך אופטימלי הוא קריטי ביותר משום שיש לו השפעה ישירה על תפוקת האלגוריתם. קביעת אורך quantum קצר מידי עלולה להוביל לתקורה גבוהה בשל הצורך להחליף תהליכים בתדירות גבוהה. במקרה זה, למרות שהמערכת מעניקה זמן CPU לתהליכים רבים בטווח זמן קצר, חלק גדול מזמן ה CPU יתבזבז על ההחלפות המרובים בין התהליכים, מה שעלול להוריד את היעילות הכוללת של המערכת. יתרה מכך, במערכות בעלות זמן quantum קצר, זמני התגובה של התהליכים עשויים להיות מהירים יותר אך לעיתים זה עלול לבוא על חשבון ביצועי המערכת או על ניצולת ה CPU בצורה מיטבית. מצד שני, קביעת אורך quantum ארוך מידי עלול להוביל למצב שבו תהליכים ממתנים רמן רב עד לקבל זמן CPU, מה שעלול להוביל לזמן המתנה וזמן השהייה ארוכים יותר וכך בעצם לגרום להקטנת התגובתיות המערכת והשפעה על חווית המשתמש.

האתגר בקביעת גודל quantum הוא למצוא את האיזון הנכון בין תפוקה גבוהה לבין זמן תגובה נמוך. גודלו צריך להיות מותאם לסוג המשימות שהמערכת מבצעת ולאופי התהליכים במערכת. למשל עבור מערכת שמריצה תהליכים חישוביים ארוכים נבחר ב quantum ארוך יותר אך עבור מערכת שמבצעת תהליכים אינטרקטיביים נבחר ב quantum קצר יותר.

בנוסף, גודל quantum עלול להשפיע על תופעות כגון הרעבה של תהליכים בעלי דרישות משאבים נמוכות, במיוחד כאשר תהליכים אלו נדחקים לסוף התור ואינם מקבלים מספיק זמן CPU. במקרים כאלה, חשוב לבחור quantum המאפשר לכל תהליך להתקדם בקצב סביר, תוך שמירה על איזון בין תהליכים כבדי משאבים לתהליכים קלים יותר. בחירה מושכלת של גודל quantum תסייע לשמור על איזון זה ולמנוע עיכובים מיותרים בתהליכים קריטיים.

2.3 השפעת האלגוריתם על ביצועי המערכת

אלגוריתם ה Round Robin הוא מרכיב חיוני במערכות הפעלה מודרניות, בפרט במערכות בעלות timesharing⁵ בעיקר עקב הפשטות והביצועים שהוא מציע. היתרון המרכזי של האלגוריתם הוא ההגינות שהוא מספק, כלומר האלגוריתם מבטיח שכל תהליך מקבל חלק שווה מזמן ה CPU. הגינות במערכת הפעלה היא הכרחית בעיקר במקרים בהם מספר תהליכים צריכים זמן CPU בו זמנית. כך בעצם מבטיח האלגוריתם שאף תהליך לא מקבל זמן CPU באופן בלעדי ומפחית סיכוי להרעבה. מבחינת ביצועים, אלגוריתם ה Round Robin יעיל במקרים בהם עומס העבודה מורכב ממספר גדול של תהליכים קצרים ואינטרקטיביים כמו בסביבות desktop. האלגוריתם מבטיח זמן תגובה צפוי הנחוץ למערכות הפעלה, בפרט לאפליקציות המיועדות למשתמשים.

נבחן את יתרונות האלגוריתם במדדי המפתח שהגדרנו :

הגינות

האלגוריתם מספק אחוזי הגינות מאוד גבוהים למערכת ההפעלה היות וכל תהליך מקבל נתח שווה של זמן CPU. שום תהליך בתור לא מקבל גישה עודפת למעבד וכך בעצם האלגוריתם מונע הרעבה ומאפשר לכל התהליכים להתקדם בריצתם. אורך ה quantum אינה משפיע ישירות על מידת ההגינות של האלגוריתם אך נציין כי בחירת quantum ארוך מידי, תוביל לכך שתהליכים הנמצאים בסוף התור ימתינו זמן רב לקבל זמן CPU, ויפגע במידת ההגינות שהאלגוריתם מספק.

קצב עבודה

האלגוריתם מספק קצב עבודה די מהיר אך לעומת אלגוריתמים עם מספר החלפות תהליכים נמוך יותר, קצב העבודה של אלגוריתם ה RR נמוך יותר. אורך ה quantum משפיע במידה רבה על קצב העבודה, ככל שה quantum קצר יותר כך מספר החלפות התהליכים גדל וקצב העבודה קטן משום שהמעבד עסוק בהחלפת התהליכים ולא בהרצתם. ככל שה quantum ארוך יותר, מספר החלפות התהליכים קטן ולכן קצב העבודה גדל אך זמן התגובה עלול להתארך עבור תהליכים קצרים יותר.

תקורה

התקורה באלגוריתם היא די גבוהה בשל החלפת התהליכים המרובה. quantum קצר יותר יוביל לתקורה גבוהה יותר משום שיוביל למספר החלפות תהליכים גדול יותר, quantum ארוך יותר יוריד את התקורה אך יכול לגרום לפגיעה בהוגנות וזמן התגובה.

⁵ שיטה בה המערכת מאפשרת למספר משתמשים אינטרקטיביים לעבוד בו-זמנית.

ניצולת המעבד

האלגוריתם מביא לניצולת מעבד גבוהה היות והאלגוריתם פועל על מנת לשמור על המעבד עסוק ונמנע ככל שניתן ממצבים בהם המעבד במצב idle. הניצולת תלויה באיזון בין שימוש המעבד להרצת תהליכים לבין שימוש המעבד לטובת החלפת התהליכים. quantum קצר יותר יפגע בניצולת המעבד בגלל התקורה הגבוהה אך quantum ארוך מדי עלול לפגוע בניצולת המעבד כאשר מדובר בתהליכים אינטרקטיביים הממתינים לתגובת משתמש ובעצם מבזבזים את זמן ה quantum שהוקצה להם.

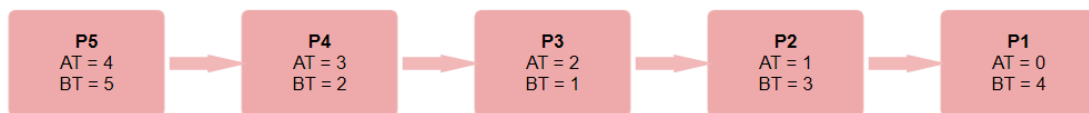
2.4 הדגמת ביצועי האלגוריתם

נדגים את ביצועי האלגוריתם עבור התהליכים $P_1 - P_5$ כאשר $\text{quantum} = 2$ וזמני הרצת התהליכים הם :

P	AT	BT
1	0	4
2	1	3
3	2	1
4	3	2
5	4	5

טבלה 2 – טבלת תהליכים לפי זמן הגעה וזמן ריצה

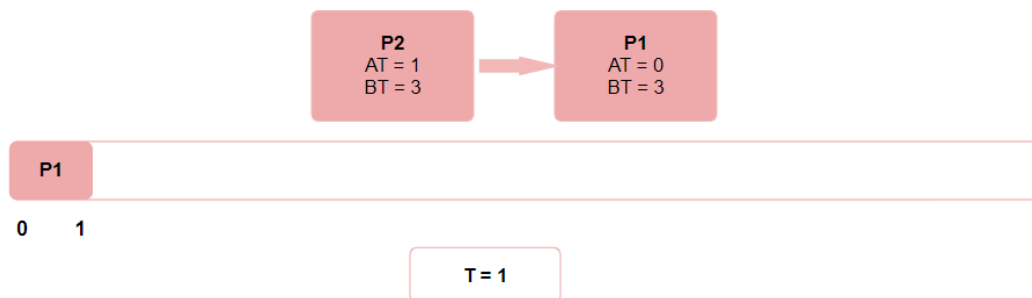
נסביר את ריצת האלגוריתם :



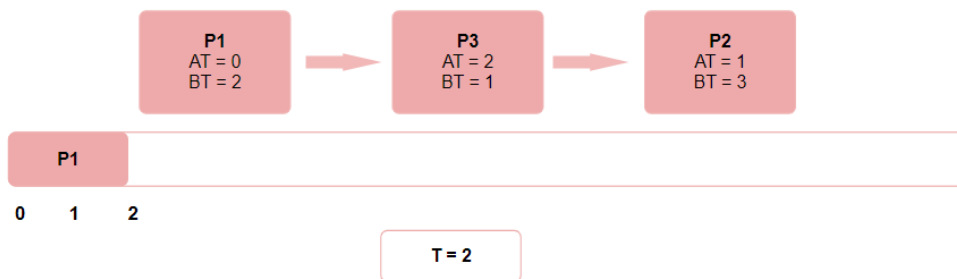
תמונה 2.0 – תהליכים לפי סדר הגעתם

בזמן 0 עד 2

תהליך P_1 מגיע ראשון לתור ומקבל זמן CPU למשך 2 יחידות זמן. בזמן ריצת התהליך תהליך P_1 כאשר ניצל כבר יחידת זמן אחת, נכנס תהליך P_2 לתור. לאחר שתהליך P_1 מסיים את ריצתו ומסיים את זמן הquantum שלו, הוא עובר לסוף התור עם $BT=2$ ונמצא מאחורי תהליך P_3 שמגיע בזמן $T=2$.



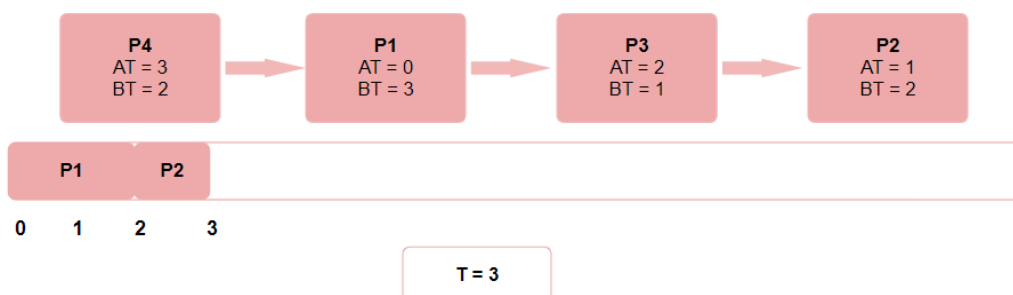
תמונה 2.1 – אלגוריתם $T=1, RR$



תמונה 2.2 – אלגוריתם $T=2, RR$

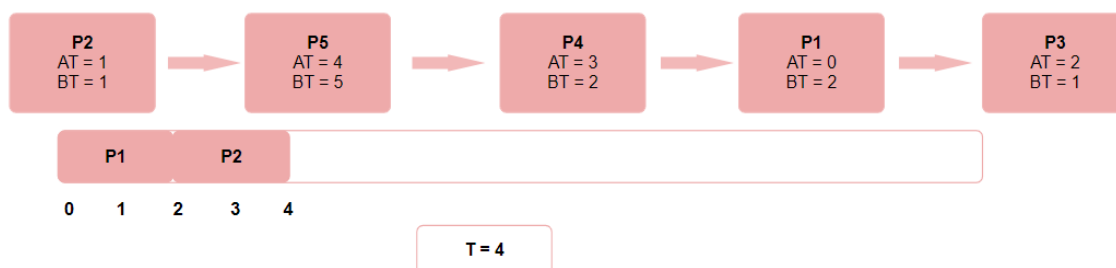
בזמן 2 עד 4

תהליך P_2 מקבל זמן CPU ולאחר יחידת זמן אחת מצטרף תהליך P_4 לתור מאחורי תהליך P_1 .



תמונה 2.3 – אלגוריתם $T=3, RR$

לאחר יחידת זמן נוסף ($T=4$), תהליך P_5 מצטרף לסוף התור. תהליך P_2 מסיים את זמן הquantum שהוקצה לו ונכנס לסוף התור מאחורי תהליך P_5 עם $BT=1$.

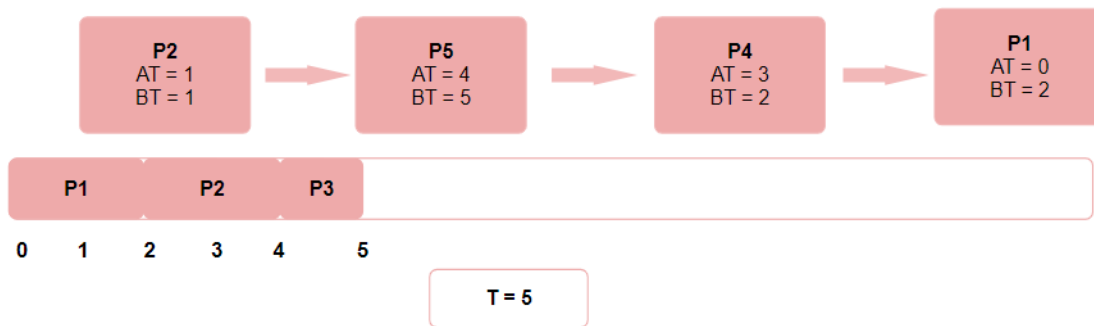


תמונה 2.4 – אלגוריתם $T=4, RR$

זמן 4 עד 5

לאחר מכן, בזמן $T=4$ מקבל תהליך P_3 זמן CPU ומסיים את ריצתו לאחר יחידת זמן אחת כלומר בזמן $T=5$.

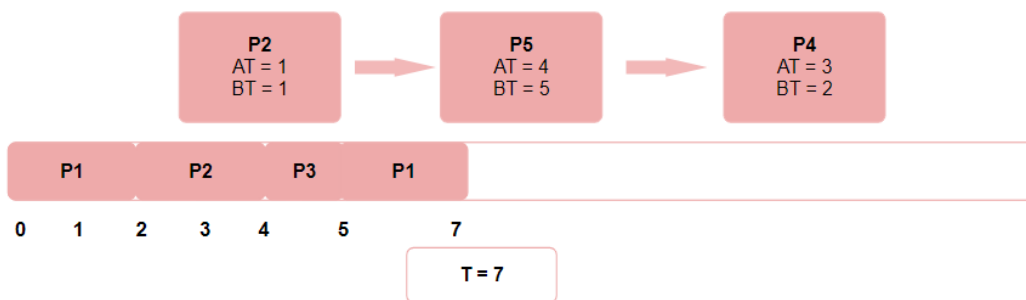
לאחר מכן, יועבר זמן הCPU לטובת המשך ריצת תהליך P_1 .



תמונה 2.5 – אלגוריתם $RR, T=5$

זמן 5 עד 7

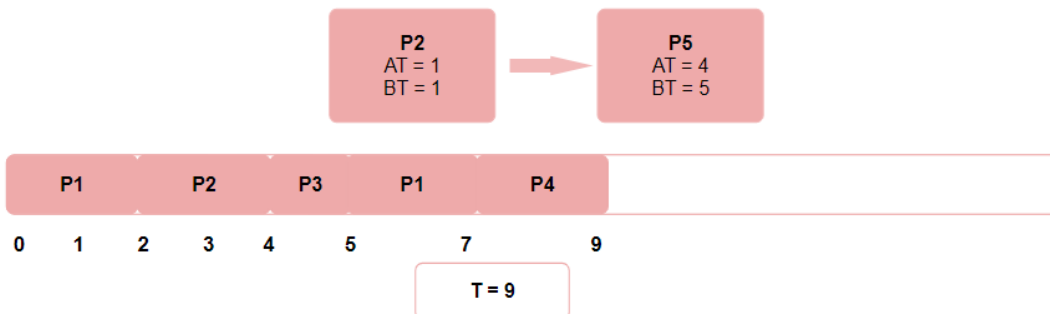
תהליך P_1 מקבל זמן CPU ומסיים את ריצתו ($T=7$) לאחר שרץ בכל הquantum שהקוצה לו ומוסר מהתור.



תמונה 2.6 – אלגוריתם $RR, T=7$

זמן 7 עד 9

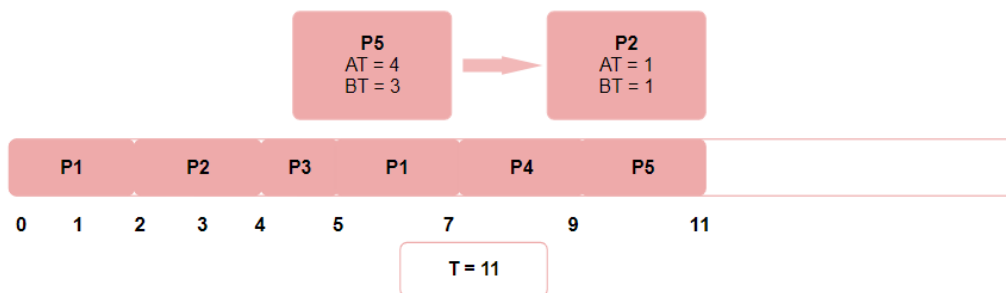
תהליך P_4 מקבל זמן CPU ומסיים גם הוא את ריצתו ($T=9$) ומוסר מהתור.



תמונה 2.7 – אלגוריתם $RR, T=9$

זמן 9 עד 11

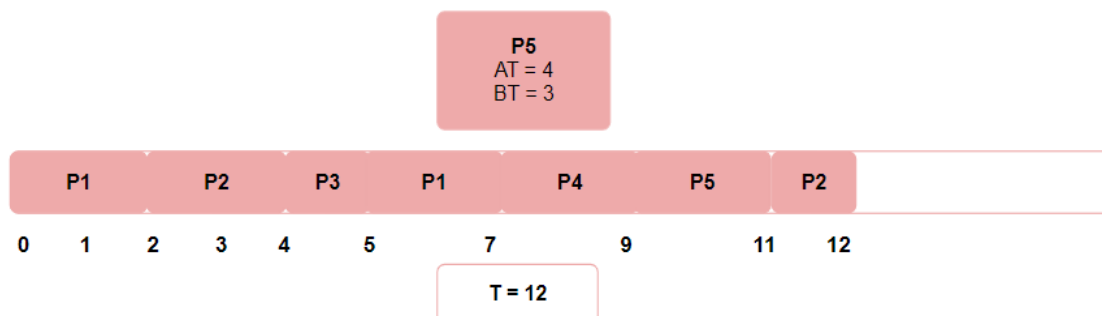
תהליך P_5 מקבל זמן CPU, לאחר שמנצל את זמן הquantum שהוקצה לו, תהליך P_5 מועבר לסוף התור עם $BT=3$.



תמונה 2.8 – אלגוריתם $RR, T=11$

זמן 11 עד 12

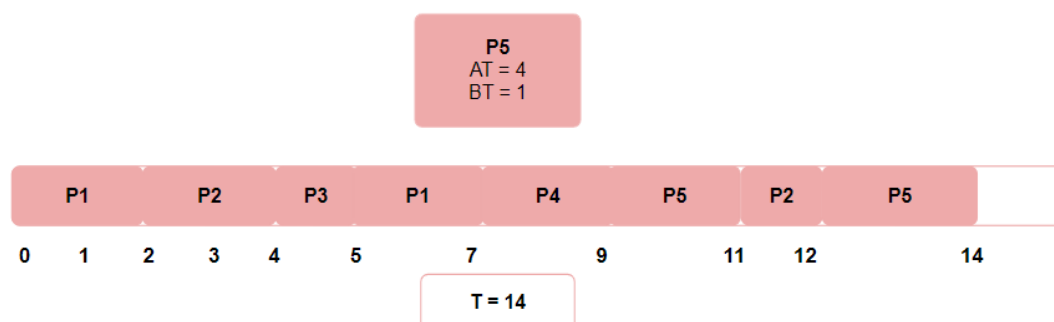
תהליך P_2 מקבל זמן CPU ומסיים את ריצתו בזמן $T=12$



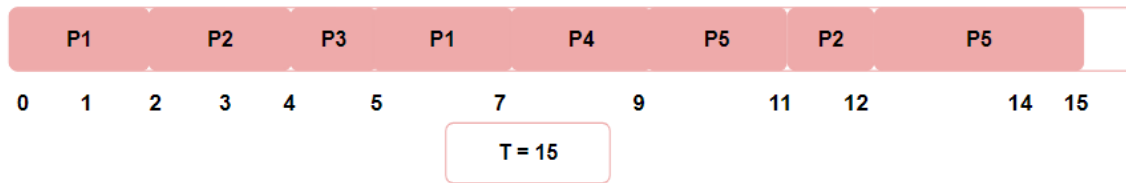
תמונה 2.9 – אלגוריתם $RR, T=12$

בזמן 12 עד 15

מקבל תהליך P_5 זמן CPU והוא מסיים את ריצתו בזמן $T=15$. כלומר תהליך P_5 מנצל את זמן הquantum שהוקצה לו ואז מקבל שוב זמן CPU ומסיים את ריצתו לאחר יחידת זמן נוספת.



תמונה 2.10 – אלגוריתם $RR, T=14$



תמונה 2.11 Gantt chart for RR

כעת נחשב את ביצועי האלגוריתם על התהליכים הנ"ל לפי הנוסחאות הבאות :

○ CT – Completion Time

הזמן הדרוש להשלמת התהליך

○ TT – Turnaround Time

הזמן שחלף מהרגע שהתהליך התקבל במערכת ועד הרגע בו סיים לרוץ.

הנוסחה לחישוב היא $TT = CT - AT$

○ WT – Waiting Time

הזמן הכולל שבו תהליך המתין בתור.

הנוסחה לחישוב היא $WT = TT - BT$

○ RT – Response Time

הזמן שחולף מהרגע שהתהליך מגיע לתור עד לפעם הראשונה שהתהליך מקבל זמן CPU.

הנוסחה לחישוב $RT = FR - AT$ כאשר FR זה הזמן בו התהליך קיבל זמן CPU בפעם הראשונה.

נסכום את כל פרטי ריצת האלגוריתם ונקבל :

Round Robin						
P	AT	BT	CT	TT	WT	RT
1	0	4	7	7	3	0
2	1	3	12	11	8	1
3	2	1	5	3	2	2
4	3	2	9	6	4	4
5	4	5	15	11	6	5

פרק 3: אלגוריתם Shortest Job First

3.1 סקירה כללית

אלגוריתם העבודה הקצרה ביותר, SJF, פועל על ידי בחירת התהליך בעל זמן הביצוע הקצר ביותר מבין התהליכים הממתינים והקצאת זמן CPU לתהליך זה. האלגוריתם מתמקד במתן עדיפות לתהליכים שיכולים לסיים את ריצתם בזמן הקצר ביותר, מה שמסייע לשפר את יעילות המערכת מבחינת ניצול ה-CPU וזמני המתנה. לאלגוריתם קיימות שתי גרסאות עיקריות:

○ Non-preemptive ללא הפקעת מעבד

כאשר תהליך מוגדר כבעל זמן הביצוע הקצר ביותר מבין כלל התהליכים הזמינים, האלגוריתם בוחר בו להקצאת זמן CPU ותהליך זה ימשיך לרוץ עד לסיומו. לא ניתן להפסיק את ריצת התהליך לטובת תהליכים בעלי זמן ריצה קצר יותר גם אם תהליכים אלו הגיעו לאחר תחילת הריצה של התהליך הנוכחי. סדר התורים נשמר לפי עקרון ה-FCFS כלומר כל התהליכים שמגיעים בזמן ריצת תהליך אחר, נכנסים לתור לפי זמן הביצוע שלהם. אם יש מספר תהליכים בעלי זמן ביצוע זהה, האלגוריתם יבחר את התהליך שהגיע ראשון לתור. גרסה זו של האלגוריתם פשוטה ליישום ונפוצה אך עלולה לגרום לתהליכים ארוכים להמתין זמן רב עד לקבלת זמן CPU, במיוחד עם תהליכים קצרים ימשיכו להגיע למערכת. גרסה זו תודגם במסגרת העבודה.

○ Preemptive עם הפקעת מעבד

בגרסה זו האלגוריתם מוכר בשם SRTF-Shortest Remaining Time First. כאשר תהליך מקבל זמן CPU ותהליכים בעלי זמן ריצה קצר יותר מגיעים לתור, המעבד יופקע מהתהליך הנוכחי ויעבור לתהליך הקצר ביותר מבין התהליכים החדשים שהגיעו. היתרון המרכזי בגרסה זו הוא היכולת של המערכת להתאים את עצמה לשינויים דינמיים בתור התהליכים ולתת עדיפות לתהליכים שיסיימו את ריצתם במהירות.

האלגוריתם נחשב לאחד האלגוריתמים היעילים ביותר בהפחתת זמן ההמתנה הממוצע וזמן Turnaround הממוצע ולכן מתאים במיוחד למערכות בהן ניתן לחזות בדיוק את זמני הביצוע של התהליכים כמו לדוגמה במערכות עיבוד אצווה בהן תהליכים מגיעים בקבוצות ולא נדרש זמן תגובה מהיר לכל תהליך בנפרד אלא לקבוצת התהליכים. במקרים אלו, האלגוריתם בשתי הגרסאות מציע שיפור ניכר בביצועים לעומת אלגוריתמים אחרים. אם זאת, אחד מחסרונות האלגוריתם הוא התלות בחיזוי מדויק של זמני הביצוע של התהליכים שכן חיזוי שגוי עלול להוביל להקצאות לא יעילות של זמן המעבד. בנוסף, האלגוריתם עלול להוביל להרעבה של תהליכים ארוכים משום שתהליכים קצרים מתועדפים באופן קבוע, ניתן לפתור מקרים אלו על ידי שימוש במדיניות נוספת כדוגמת aging.

3.2 פרמטר מרכזי

באלגוריתם Shortest Job First, הפרמטר המרכזי הוא זמן הביצוע כלומר Burst Time של כל תהליך. פרמטר זה משמעותי לקביעת אופן פעולת האלגוריתם והשפעתו על ביצועי המערכת. זמן הביצוע מייצג את משך הזמן הדרוש להשלמת כל תהליך עד תום ולכן יש לו השפעה ישירה על מדדים כגון זמני המתנה, זמני השהייה, ניצולת המעבד וכדומה. על ידי בחירת התהליך בעל זמן הביצוע הקצר ביותר בכל רגע נתון, ה-SJF נחשב לאחד האלגוריתמים היעילים ביותר בהפחתת זמן ההמתנה הכולל במערכת ובשיפור הביצועים הכלליים.

באלגוריתם SJF, התהליך בעל זמן הביצוע הקצר ביותר מבין כל התהליכים הממתינים הוא התהליך שייבחר לקבל זמן CPU. כאשר תהליך מסיים את זמן ה-CPU שהוקצה לו, הוא יוצא מתור התהליכים הממתינים והתהליך הבא שייבחר הוא התהליך בעל זמן הביצוע הקצר ביותר מבין התהליכים הנוותרים.

כך האלגוריתם מבטיח שהמערכת מתעדפת תהליכים שיכולים לסיים את פעולתם במהירות ובעצם למזער את זמן ההמתנה הכולל של התהליכים ומעלה את יעילות המערכת.

עם זאת, ישנן מספר בעיות פוטנציאליות הנובעות משיטת הפעולה של האלגוריתם. אחת הבעיות המרכזיות היא תופעת ההרעבה של תהליכים ארוכים משום שאלו עלולים להמתין זמן רב עד לקבלת זמן CPU במקרים בהם תהליכים קצרים יותר ממשיכים להגיע למערכת בכל עת.

בעיה זו מתבטאת בעיקר בגרסה שאינה מפקיעה את המעבד, בה תהליך לא מופסק ברגע שהוקצה לו זמן CPU, גם אם מגיעים תהליך קצרים יותר מהתהליך הנוכחי. בגרסה שמפקיעה את המעבד, הידועה גם בשם SRTF, בה האלגוריתם יכול לעצור תהליך אם מגיעים תהליכים קצרים יותר, נצפה לזמני תגובה קצרים יותר עבור תהליכים קצרים אך דחיית תהליכים ארוכים יותר עדיין מתרחשת.

בנוסף, האתגר המרכזי באלגוריתם טמון בכך שזמן הביצוע של התהליכים אינו תמיד ידוע מראש.

במקרים בהם לא ניתן לחזות את זמן הביצוע המדויק של התהליכים, האלגוריתם יתקשה לממש את יתרונותיו ויתכן מצב בו תהליכים בעלי זמן ביצוע ארוך ימתינו זמן ממושך ובעצם האלגוריתם יגרום להרעבה. אולם, כאשר ניתן לחזות את זמני הביצוע של התהליכים במדויק, האלגוריתם נחשב לאחד מהאלגוריתמים היעילים ביותר בהפחתת זמני המתנה והשהייה.

האלגוריתם יעיל במיוחד במערכות בהן התהליכים הם קצרים וקיימת דרישה לזמן תגובה קצר אך עבור מערכות בעלי אופי חישובי וזמן ביצוע ארוך יותר, האלגוריתם לא ייספק יתרון משמעותי.

כמו כן, בחירה באלגוריתם זה עשויה לגרום לחוסר איזון בין התהליכים הקצרים לבין התהליכים הארוכים במערכת, במיוחד במצב בו התהליכים הארוכים נדחקים לסוף התור ומקבלים זמן CPU לאחר המתנה ממושכת.

3.3 השפעת האלגוריתם על ביצועי המערכת

אלגוריתם SJF הוא מרכיב חשוב במערכות הפעלה, בפרט במערכות בהן קיים דגש על השלמת תהליכים בזמן המהיר ביותר. היתרון המרכזי של האלגוריתם הוא ההבטחה שהמערכת תתעדף תהליכים קצרים, מה שמוביל לקיצור זמני ההמתנה וההשהיה ההמוצעים ומשפר את היעילות של המערכת.

נבחן את יתרונות האלגוריתם במדדי המפתח שהגדרנו:

הגינות

ככל, האלגוריתם מתעדף תהליכים בעלי זמן ביצוע קצר יותר ולכן הוא פחות הוגן עבור תהליכים שזמן הביצוע שלהם ארוך יותר. במקרים בהם תהליכים קצרים יותר מגיעים שוב ושוב למערכת, ייתכנו עיכובים בתהליכים ארוכים יותר ואף הרעבה. במערכות בהן התהליכים הם קצרים יותר, האלגוריתם מספק הגינות במידה רבה.

קצב עבודה

האלגוריתם מספק קצב עבודה מהיר למדי משום שתהליכים קצרים מקבלים זמן CPU מהר יותר ומסיימים את ריצתם מהר יותר וכך בעצם מסתיימים יותר תהליכים בזמן קצר. הקטנת זמן הממתנה של התהליכים מובילה ליעילות גדולה יותר וקצב עבודה מהיר יותר. במקרה של מערכת עם הרבה תהליכים ארוכים, קצב העבודה ירד.

תקורה

התקורה באלגוריתם היא גבוהה משום שדורשת מהמערכת לחשב כל הזמן מי מבין התהליכים הוא האלגוריתם בעל זמן הביצוע הקצר ביותר, מה שמוביל לעומס חישובי על המערכת ולכן לתקורה גבוהה. ככל שזמני הביצוע של התהליכים שונים יותר זה מזה, התקורה גדלה משום שנדרש לשערך ולהשוות את זמני הביצוע של התהליכים. כאשר טווח זמני הביצוע של התהליכים קטן יותר, כלומר זמני הביצוע של התהליכים די דומים, התקורה בבחירת התהליך הקצר ביותר קטנה.

ניצולת המעבד

ניצולת המעבד באלגוריתם היא די גבוהה משום שבחירת התהליך הקצר ביותר מצמצמת את הזמן בו המעבד נמצא idle ובעצם דואגת להעסיק את המעבד בהרצת התהליכים. אם זאת, כאשר מדובר בתהליכים ארוכים מאוד, ניצולת המעבד עלולה לרדת משום שיש פערים בין ריצות התהליכים הנובעים מהצורך לחשב ולהשוות בין התהליכים למציאת התהליך בעל זמן הביצוע הקצר ביותר. במקרים בהם המערכת מכילה הרבה תהליכים הממתינים לפעולות I/O להתבצע, ניצולת המעבד תרד משום שהמעבד ממתין לפעולה מצד גורם חיצוני.

3.4 הדגמת ביצועי האלגוריתם

נדגים את ביצועי האלגוריתם עבור התהליכים $P_1 - P_5$, זמני הרצת התהליכים הם :

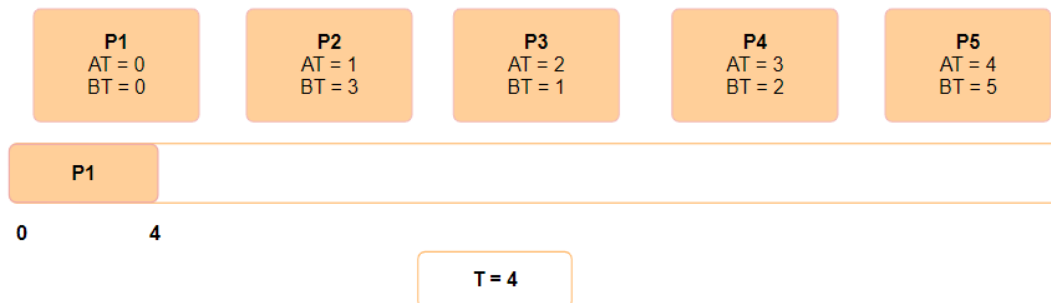
P	AT	BT
1	0	4
2	1	3
3	2	1
4	3	2
5	4	5

טבלה 3 – טבלת תהליכים לפי זמן הגעה וזמן ריצה

נסביר כיצד האלגוריתם פועל על התהליכים הנ"ל :
האלגוריתם בוחר לתת זמן CPU לתהליך בעל זמן הריצה הקצר ביותר ומריץ אותו עד לסיומו.

זמן 0 עד 4

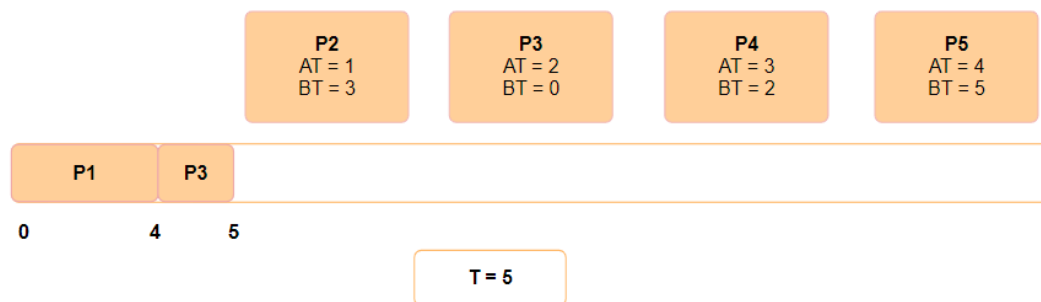
תהליך P_1 מגיע והוא היחיד שנמצא במערכת ולכן האלגוריתם מריץ אותו עד לסיומו.
התהליך מסיים את הריצה שלו בזמן $T = 4$.
בזמן שתהליך P_1 מקבל זמן CPU, מגיעים תהליכים $P_2 - P_5$ וממתינים לסיום של P_1 .



תמונה 3.0 – אלגוריתם SJF, $T=4$

זמן 4 עד 5

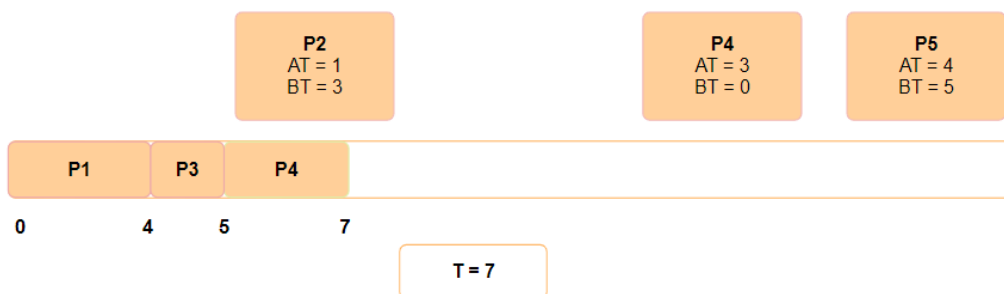
תהליך P_3 הוא התהליך בעל הBT הקצר ביותר ולכן האלגוריתם בוחר בו, התהליך רץ עד לסיומו ב $T = 5$.



תמונה 3.1 – אלגוריתם SJF, $T=5$

זמן 5 עד 7

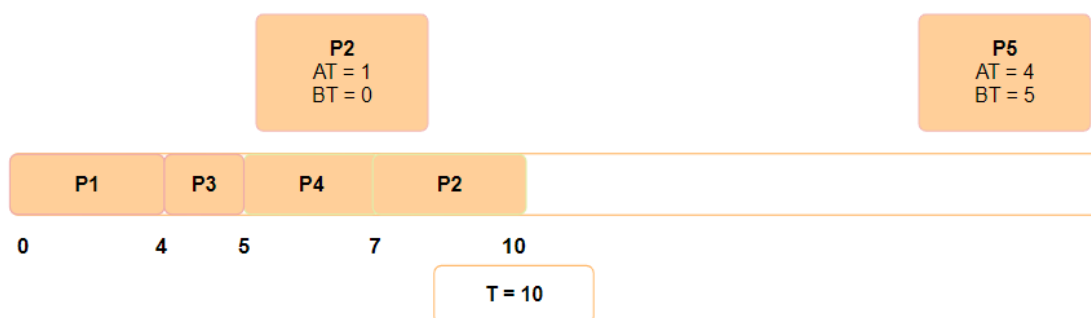
תהליך P_4 הוא התהליך בעל הBT הקצר ביותר ולכן האלגוריתם בוחר בו, התהליך רץ עד לסיומו בT = 7



תמונה 3.2 – אלגוריתם SJF, $T=7$

זמן 7 עד 10

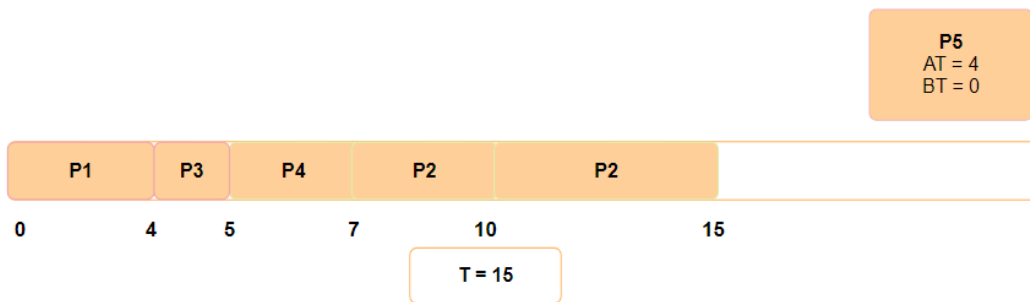
תהליך P_2 הוא התהליך בעל הBT הקצר ביותר ולכן האלגוריתם בוחר בו, התהליך רץ עד לסיומו בT = 10.



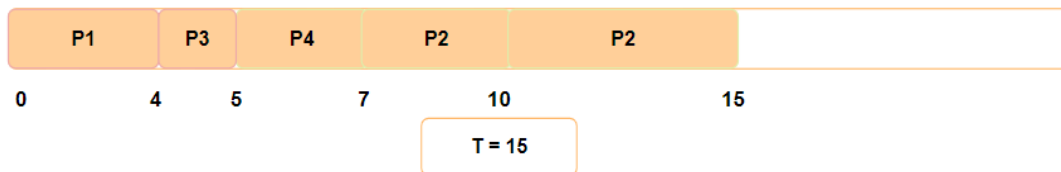
תמונה 3.3 – אלגוריתם SJF, $T=10$

זמן 10 עד 15

תהליך P_5 הוא התהליך בעל הBT הקצר ביותר ולכן האלגוריתם בוחר בו, התהליך רץ עד לסיומו בT = 15



תמונה 3.4 – אלגוריתם $T=15, SJF$



תמונה 3.5 – Gantt chart for SJF

כעת נחשב את ביצועי האלגוריתם על התהליכים הנ"ל לפי הנוסחאות הבאות:

○ CT – Completion Time

הזמן הדרוש להשלמת התהליך

○ TT – Turnaround Time

הזמן שחלף מהרגע שהתהליך התקבל במערכת ועד הרגע בו סיים לרוץ.

הנוסחה לחישוב היא $TT = CT - AT$

○ WT – Waiting Time

הזמן הכולל שבו תהליך המתין בתור.

הנוסחה לחישוב היא $WT = TT - BT$

○ RT – Response Time

הזמן שחולף מהרגע שהתהליך מגיע לתור עד לפעם הראשונה שהתהליך מקבל זמן CPU.

הנוסחה לחישוב $RT = FR - AT$ כאשר FR זה הזמן בו התהליך קיבל זמן CPU בפעם הראשונה.

נסכום את כל פרטי ריצת האלגוריתם ונקבל:

Shortest Job First						
P	AT	BT	CT	TT	WT	RT
1	0	4	4	4	0	0

2	1	3	10	9	6	6
3	2	1	5	3	2	2
4	3	2	7	4	2	2
5	4	5	15	11	6	6

פרק 4: השוואת האלגוריתמים

נשווה בין ביצועי האלגוריתמים Round Robin, Priority, SJF בסביבת Linux. לצורך ההשוואה נשתמש בסימולטורים ייעודיים שיחשבו את זמני ההמתנה (WT), זמני השלמה (CT), זמני ההשהייה (TT) וזמני התגובה (RT). בנוסף הסימולטורים יחשבו את ניצולת המעבד והזיכרון ואת מידת ההגינות של כל אלגוריתם.

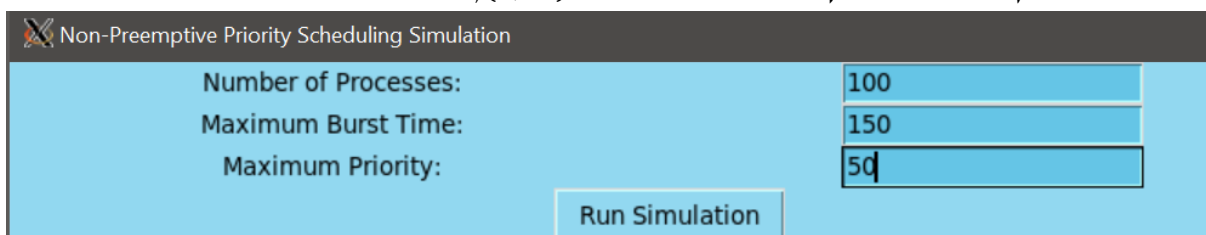
הסימולטורים נכתבו בשפת Python בסביבת WSL על מנת לדמות סביבת Linux ומשתמשים בVcxsrv על מנת לאפשר אינטרקציה נוחה מול התוכנה. כמו כן, על מנת לאפשר ניטור בזמן אמת על ניצולת המעבד והזיכרון, בוצע שימוש בספריית psutil. הסימולטורים מקבלים כקלט את מספר התהליכים הרצויים, זמן ריצה מקסימלי ומאפיינים נוספים כמו זמן quantum רצוי עבור הסימולטור של אלגוריתם RR וטווח עדיפויות מקסימלי עבור אלגוריתם Priority. הסימולטורים מייצרים את התהליכים לפי המספר המבוקש ומקצים זמן ריצה רנדומלי בטווח שבין 1 לזמן הריצה המקסימלי המבוקש. זמן ההגעה של כל תהליך יבחר רנדומלית בטווח (1,50) ויוקצה לכל תהליך. הטווח נבחר בצורה שרירותית. פלט הסימולטורים הוא פירוט ביצועי האלגוריתם עבור כל תהליך וממוצע ביצועי האלגוריתם לסך התהליכים, בנוסף הסימולטורים מחזירים את אחוז ניצולת המעבד והזיכרון של כל התהליכים עבור האלגוריתם הנבחר.

כל הסימולטורים יצרו 100 תהליכים, זמן הביצוע המקסימלי שהוגדר הוא 150ms, מאפיינים רלוונטים נוספים מוגדרים בהמשך לאלגוריתמים הרלוונטים.

תוצאות ריצות האלגוריתמים

Non-Preemptive Priority

האלגוריתם יקצה לכל תהליך עדיפות בטווח של (1,50), הטווח נבחר שרירותית.



Non-Preemptive Priority Scheduling Simulation	
Number of Processes:	100
Maximum Burst Time:	150
Maximum Priority:	50
<button>Run Simulation</button>	

תמונה 4.0 – קלט עבור Non-preemptive Priority

פלט הסימולטור – פירוט (חלקי) של זמני האלגוריתם עבור כל תהליך :

Non-Preemptive Priority Scheduling Simulation							
Number of Processes:				100			
Maximum Burst Time:				150			
Maximum Priority:				50			
				Run Simulation			
PID	Priority	BT	AT	WT	TT	CT	RT
0	6	40	7	856	896	903	856
1	4	21	6	537	558	564	537
2	5	69	10	771	840	850	771
3	38	143	1	6481	6624	6625	6481
4	4	40	3	500	540	543	500
5	23	60	2	3841	3901	3903	3841
6	25	92	1	4156	4248	4249	4156
7	2	44	9	244	288	297	244
8	39	30	6	6635	6665	6671	6635
9	41	51	5	7077	7128	7133	7077
10	42	103	4	7133	7236	7240	7133
11	9	145	3	1342	1487	1490	1342
12	3	130	7	366	496	503	366
13	9	133	5	1485	1618	1623	1485
14	34	65	7	5889	5954	5961	5889
15	25	96	4	4488	4584	4588	4488
16	27	136	3	4791	4927	4930	4791
17	1	145	1	44	189	190	44

תמונה 4.1 – פלט חלקי ביצועי אלגוריתם עבור Non-preemptive Priority

חישוב ממוצעי זמני האלגוריתם, ניצולת מעבד וזיכרון, תקורה והגינות :

Non-Preemptive Priority Scheduling Simulation							
Number of Processes:				100			
Maximum Burst Time:				150			
Maximum Priority:				50			
				Run Simulation			
92	14	150	8	2029	2179	2187	2029
93	17	29	2	2436	2465	2467	2436
94	47	30	10	7560	7590	7600	7560
95	16	12	4	2422	2434	2438	2422
96	17	124	9	2458	2582	2591	2458
97	37	4	8	6383	6387	6395	6383
98	35	19	7	6240	6259	6266	6240
99	10	45	0	0	45	45	0
Average Completion Time: 4152.46							
Average Turnaround Time: 4147.01							
Average Waiting Time: 4067.47							
Average Response Time: 4067.47							
Average CPU Usage: 0.15%							
Average Memory Usage: 6.30%							
Fairness (ms): 6014696.55							

תמונה 4.2 – פלט חלקי ביצועי אלגוריתם עבור Non-preemptive Priority

סיכום תוצאות האלגוריתם :

Non-Preemptive Priority						
avg WT(ms)	avg TT (ms)	avg CT(ms)	avg RT(ms)	CPU usage(%)	Memory usage(%)	Fairness(ms)
4067.47	4147.01	4152.46	4067.47	0.15	6.3	6014696

טבלה 4.0 – תוצאות אלגוריתם NPP

Preemptive Priority

האלגוריתם יקצה לכל תהליך עדיפות בטווח של (1,50), הטווח נבחר שרירותית.

Preemptive Priority Scheduling Simulation

Number of Processes: 100

Maximum Burst Time: 150

Maximum Priority: 50

Run Simulation

תמונה 4.3 – קלט עבור אלגוריתם Preemptive Priority

פלט הסימולטור – פירוט (חלקי) של זמני האלגוריתם עבור כל תהליך :

Preemptive Priority Scheduling Simulation

Number of Processes: 100

Maximum Burst Time: 150

Maximum Priority: 50

Run Simulation

PID	Priority	BT	AT	WT	TT	CT	RT
0	31	122	0	0	122	122	0
1	12	101	10	1338	1439	1449	1338
2	17	26	21	3086	3112	3133	3086
3	27	37	13	1983	2020	2033	1983
4	33	97	39	5811	5908	5947	5811
5	39	62	4	709	771	775	709
6	1	9	11	1647	1656	1667	1647
7	21	32	3	299	331	334	299
8	30	100	27	3965	4065	4092	3965
9	30	145	3	331	476	479	331
10	41	119	49	7271	7390	7439	7271
11	19	124	49	7036	7160	7209	7036
12	47	77	48	6911	6988	7036	6911
13	39	85	36	5355	5440	5476	5355
14	1	119	10	1219	1338	1348	1219
15	47	142	38	5670	5812	5850	5670
16	19	94	15	2164	2258	2273	2164
17	15	23	14	2026	2049	2063	2026

תמונה 4.4 – פלט חלקי ביצועי אלגוריתם עבור preemptive Priority

חישוב ממוצעי זמני האלגוריתם, ניצולת מעבד וזיכרון, תקורה והגינות :

Preemptive Priority Scheduling Simulation

Number of Processes: 100
Maximum Burst Time: 150
Maximum Priority: 50

Run Simulation

92	9	98	12	1655	1753	1765	1655
93	38	69	31	4321	4390	4421	4321
94	1	127	16	2370	2497	2513	2370
95	21	44	29	4149	4193	4222	4149
96	26	63	20	2890	2953	2973	2890
97	10	65	5	804	869	874	804
98	12	101	44	6308	6409	6453	6308
99	9	49	49	6987	7036	7085	6987

Average Completion Time: 3542.96
Average Turnaround Time: 3519.20
Average Waiting Time: 3443.63
Average Response Time: 3443.63

Average CPU Usage: 0.30%
Average Memory Usage: 8.70%
Fairness (ms) 4439380.92

תמונה 4.5 – פלט חלקי ביצועי אלגוריתם עבור preemptive Priority

סך תוצאות הרצת האלגוריתם :

Preemptive Priority						
avg WT(ms)	avg TT (ms)	avg CT(ms)	avg RT(ms)	CPU usage(%)	Memory usage(%)	Fairness(ms)
3443.63	3519.2	3542.96	3443.63	0.3	8.7	4439380

טבלה 4.1 – תוצאות אלגוריתם PP

Round Robin

זמן הquantum משתנה ממערכת למערכת ולכן נשתמש בערך החציוני בטווח זמני הquantum הממוצע, כלומר כאשר הטווח הממוצע הוא בין 10ms ל100ms, נבחר בזמן quantum של 55ms.

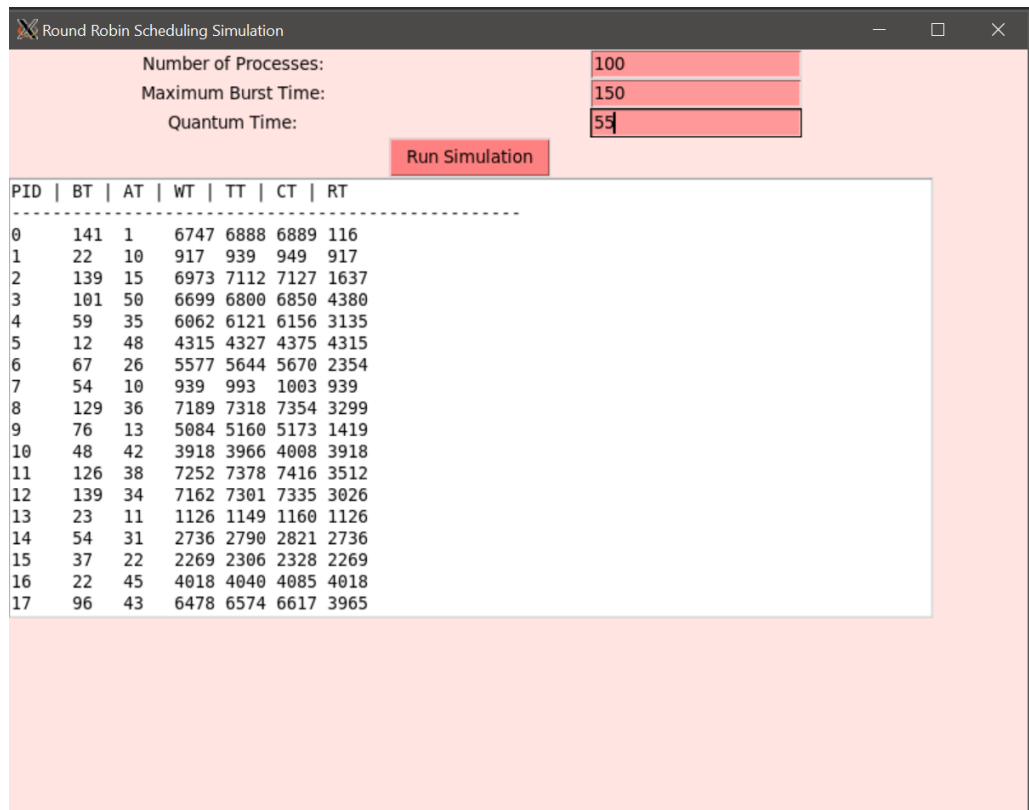
Round Robin Scheduling Simulation

Number of Processes: 100
Maximum Burst Time: 150
Quantum Time: 55

Run Simulation

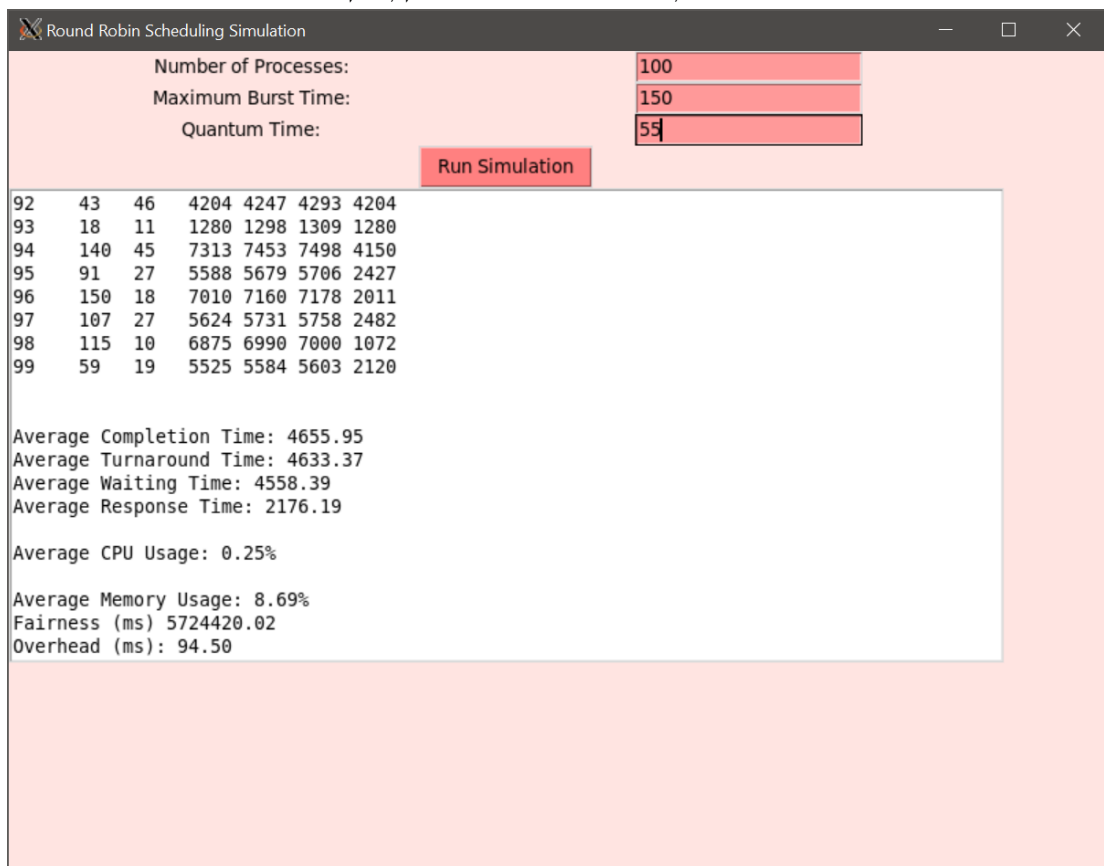
תמונה 4.6 – קלט עבור RR

פלט הסימולטור – פירוט (חלקי) של זמני האלגוריתם עבור כל תהליך :



תמונה 4.7 – פלט חלקי ביצועי אלגוריתם עבור RR

חישוב ממוצעי זמני האלגוריתם, ניצולת מעבד וזיכרון, תקורה והגינות :



תמונה 4.8 – פלט חלקי ביצועי אלגוריתם עבור RR

סך תוצאות הרצת האלגוריתם :

Round Robin							
avg WT(ms)	avg TT (ms)	avg CT(ms)	avg RT(ms)	CPU usage(%)	Memory usage(%)	Fairness(ms)	Overhead(ms)
5044.22	5123.45	5148.58	2253.64	0.76	8.11	5323078.58	97.5

טבלה 4.2 – תוצאות אלגוריתם RR

Shorstest Job First

האלגוריתם אינו דורש הגדרת מאפיינים נוספים פרט למספר תהליכים וזמן ריצה מקסימלי.

SJF Scheduling Simulation

Number of Processes:

100

Maximum Burst Time:

150

Run Simulation

תמונה 4.9 – פלט עבור SJF

פלט הסימולטור – פירוט (חלקי) של זמני האלגוריתם עבור כל תהליך :

SJF Scheduling Simulation

Number of Processes:

100

Maximum Burst Time:

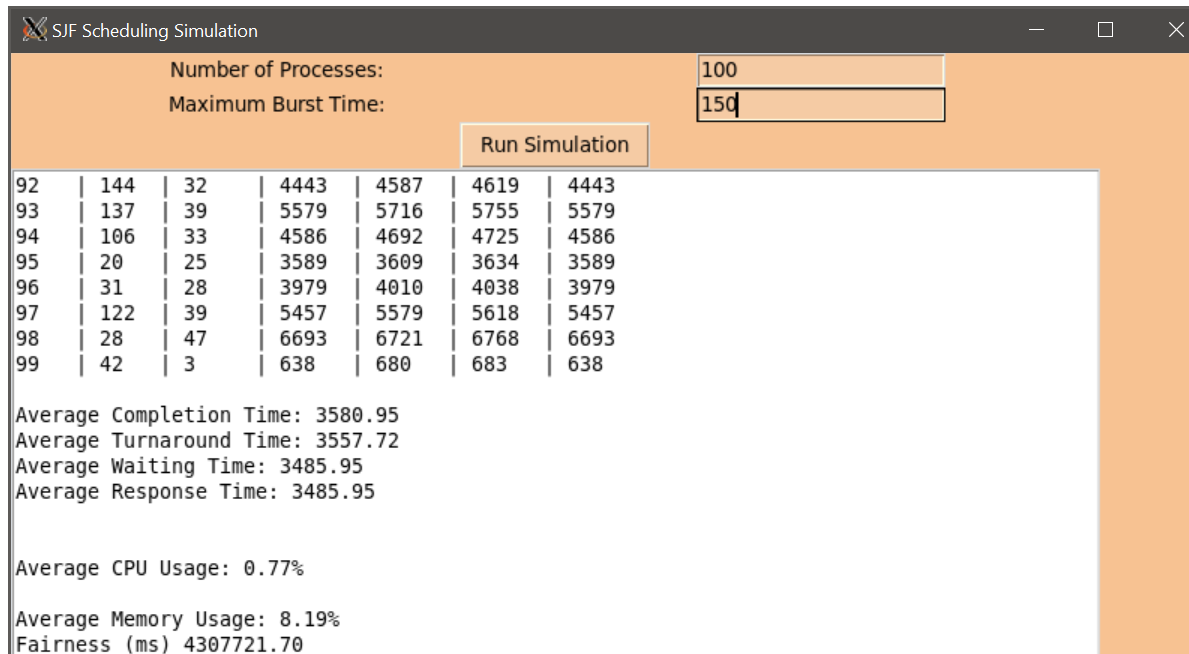
150

Run Simulation

PID	BT	AT	WT	TT	CT	RT
0	14	49	6945	6959	7008	6945
1	74	37	5286	5360	5397	5286
2	78	48	6868	6946	6994	6868
3	21	24	3494	3515	3539	3494
4	32	35	4910	4942	4977	4910
5	32	1	216	248	249	216
6	110	44	6347	6457	6501	6347
7	77	11	2358	2435	2446	2358
8	55	7	1539	1594	1601	1539
9	106	1	314	420	421	314
10	24	6	1225	1249	1255	1225
11	148	47	6721	6869	6916	6721
12	148	6	1381	1529	1535	1381
13	103	9	1985	2088	2097	1985
14	52	43	6296	6348	6391	6296
15	99	18	2767	2866	2884	2767
16	90	10	2087	2177	2187	2087
17	97	6	1284	1381	1387	1284

תמונה 4.10 – פלט חלקי ביצועי אלגוריתם עבור SJF

חישוב ממוצעי זמני האלגוריתם, ניצולת מעבד וזיכרון, תקורה והגינות :



תמונה 4.11 – פלט חלקי ביצועי אלגוריתם עבור SJF

סך תוצאות הרצת האלגוריתם :

Shortest Job First						
avg WT(ms)	avg TT (ms)	avg CT(ms)	avg RT(ms)	CPU usage(%)	Memory usage(%)	Fairness(ms)
3485.95	3557.72	3580.95	3485.95	0.77	8.19	4307721.7

טבלה 4.3 – תוצאות אלגוריתם SJF

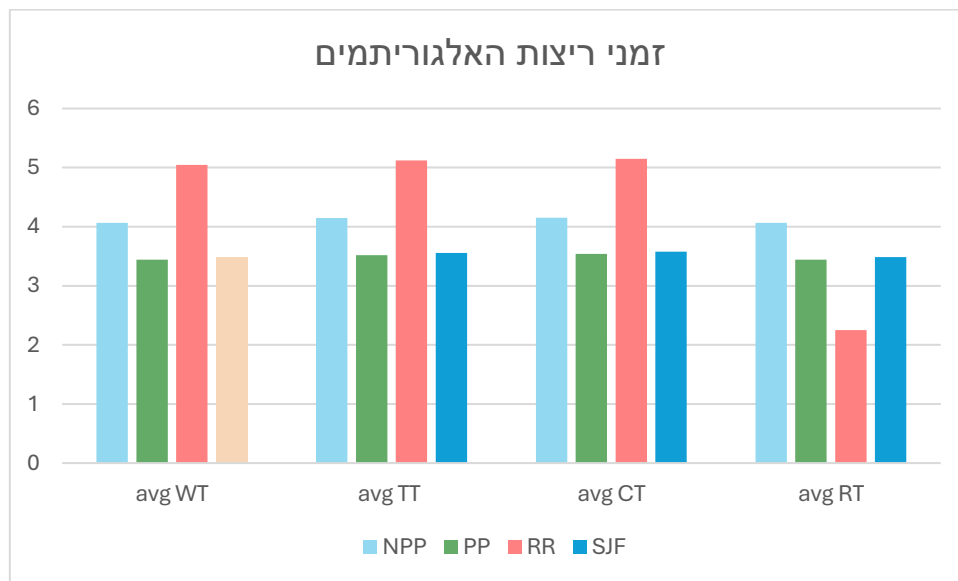
ניתוח ביצועי האלגוריתמים

כעת נבצע השוואה בין ארבעת האלגוריתמים Round Robin, SJF ו Priority (NPP & PP) ההשוואה תתבצע על בסיס מדדים מרכזיים כלומר על בסיס זמני המתנה, זמני השהייה, זמני סיום וזמני תגובה. בנוסף, נבחן את ניצולת המעבד והזיכרון עבוד כל אחד מהאלגוריתמים וכן את מידת ההגינות שלהם בחלוקת זמן המעבד בין התהליכים. לצורך ניתוח ברור יותר, תוצאות ריצות האלגוריתמים הומרו לשניות מה שמאפשר השוואה פשוטה ונוחה בין ביצועיהם.

	avg WT	avg TT	avg CT	avg RT	CPU usage(%)	Memory usage(%)	Fairness
NP Priority	4.067	4.147	4.152	4.067	0.15	6.3	6015
P Priority	3.443	3.519	3.542	3.443	0.3	8.7	4439
RR	5.044	5.123	5.148	2.253	0.76	8.11	5323
SJF	3.485	3.557	3.58	3.485	0.77	8.19	43078

טבלה 4.4 – תוצאות כלל האלגוריתמים

זמני ריצות האלגוריתמים



גרף 1 – זמני ריצות אלגוריתמים

זמני המתנה

אלגוריתם העדיפות בגרסה המפקיעה שלו (Preemptive priority) מציג את זמן ההמתנה הממוצע הנמוך ביותר (3.443sec) מבין ארבעת האלגוריתמים, כלומר תהליכים ממתנינים פחות זמן בממוצע לקבל זמן מעבד. מנגד, אלגוריתם Round Robin את זמן ההמתנה הממוצע הגבוה ביותר (5.044sec) תוצאה זו נובעת משום שהאלגוריתם מבצע סבבים בין התהליכים באמצעות זמן quantum שהוגדר (50ms).

שני האלגוריתמים הנותרים, NPP ו SJF מציגים ביצועים דומים כאשר אלגוריתם ה SJF מעט גבוהה יותר.

זמני השהייה

שוב, אלגוריתם העדיפות PP מציג את הביצועים הטובים ביותר, זמן ההשהייה הממוצע הנמוך ביותר (3.519sec).

בדומה לזמן ההמתנה הגבוה, אלגוריתם ה RR הוא בעל זמן ההשהייה הממוצע הגבוה ביותר (5.123sec), כאמור תהליכים ממתנים יותר זמן לקבלת נתח מזמן המעבד ולכן גם זמן השהייה שלהם בתור גבוה יותר.

שני האלגוריתמים הנותרים, NPP ו SJF מציגים ביצועים דומים כאשר אלגוריתם ה SJF מציג ביצועים טובים יותר (3.557sec) משום שבוחר בתהליכים בעלי זמני ביצוע קצרים יותר.

זמני השלמה

כצפוי, אלגוריתם ה PP מציג את זמן ההשלמה המוצע הנמוך ביותר (3.542sec) ואחריו אלגוריתם ה SJF (3.58sec).

בשני אלגוריתמים אלו, תהליכים קצרים או חשובים מתועדפים גבוה יותר ולכן מבוצעים מהר יותר.

מנגד, אלגוריתם ה RR מציג את זמן ההשלמה הגבוה ביותר (5.148 sec) עקב ההמתנה הממושכת של התהליכים הנובעת מהצורך להמתין לסיום זמן ה quantum המוקצה.

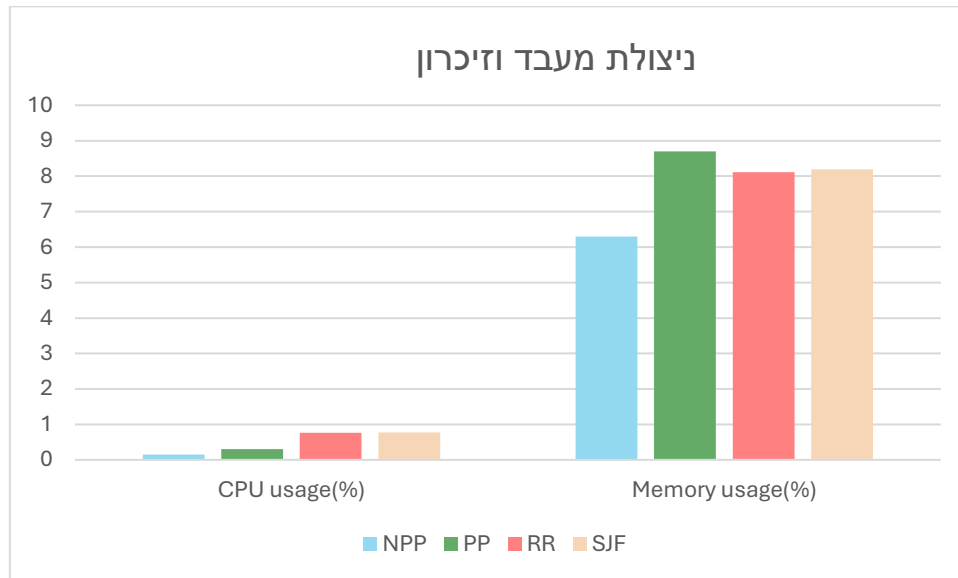
זמני תגובה

נופתע לגלות שמבחינת זמני תגובה, אלגוריתם ה RR שעד כה סיפק ביצועים פחות טוב משאר האלגוריתמים, הוא האלגוריתם בעל זמן התגובה הממוצע הנמוך ביותר (2.53sec). לאחר מכן, אלגוריתם ה PP (3.443sec) שמתעדף תהליכים חשובים יותר ולבסוף האלגוריתם בעל זמן התגובה הממוצע הגבוה ביותר הוא ה NPP (4.067sec).

מסקנות

- אלגוריתם ה Preemptive Priority הוא האלגוריתם שהציג את הביצועים הטובים ביותר עבור הדמיות הסימולטרים שבוצעו. היכולת להפקיע את המעבד לטובת תהליכים חשובים יותר מובילה לזמני המתנה, השהייה והשלמה נמוכים במיוחד.
- אלגוריתם ה Round Robin הפגין ביצועים טובים בזמני תגובה קצרים במיוחד, זוהי בדיוק חוזקו של האלגוריתם שכן הוא מבטיח שכל תהליך יקבל זמן מעבד במהירות.
- אלגוריתם ה SJF הציג ביצועים טובים יותר משל ה RR כמעט בכל המדדים. זמני ההמתנה וההשלמה נמוכים משום שהאלגוריתם בוחר את התהליכים הקצרים ביותר.
- אלגוריתם ה Non-Preemptive, הפגין ביצועים סבירים אך הוא נחות יותר בהשוואה לגרסתו השנייה של האלגוריתם.

ניצולת מעבד וזיכרון



גרף 2 – ניצולת מעבד וזיכרון באחוזים

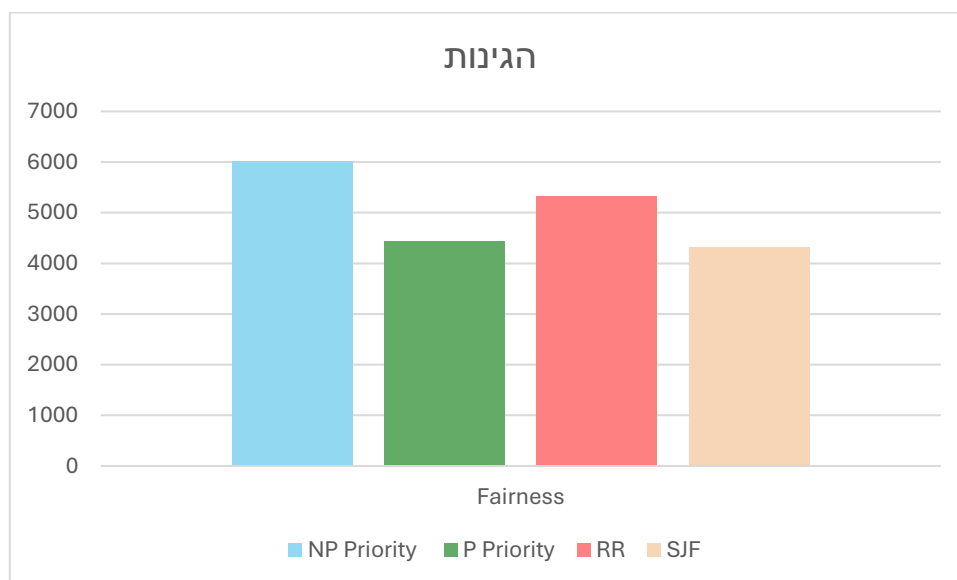
ניצולת מעבד

אלגוריתם NPP מציג את השימוש הנמוך ביותר במעבד מכל האלגוריתמים (0.15%) הסיבה לכך היא שהאלגוריתם אינו משהה ומפקיע תהליכים לטובת תהליכים אחרים ולכן מופחת העומס על המעבד ואת תדירות החלפות התהליכים. אחריו, אלגוריתם PP מציג שימוש גבוה יותר במעבד, הסיבה לכך היא הפקעת המעבד לטובת תהליכים מתועדפים יותר, כמות החלפת התהליכים מעלה את השימוש במעבד. לאחר מכן, אלגוריתם RR מציג שימוש די גבוה במעבד, הסיבה לכך היא החלפת התהליכים המרובה עקב סיום זמן הquantum. ולבסוף, אלגוריתם SJF מציג את השימוש הגבוה ביותר (0.77%), למרות שמספר החלפות התהליכים נמוך, ניהול תור התהליכים לפי זמן ביצוע מינימלי דורש משאבים משמעותיים מהמעבד.

ניצולת זכרון

אלגוריתם NPP מציג את השימוש הנמוך בזכרון (6.3%) מה שמעיד על פשטות יחסית בניהול המשאבים. אחריו מגיע אלגוריתם RR, האלגוריתם משתמש בזכרון לצורך ניהול מסודר של תור התהליכים. לאחר מכן, אלגוריתם SJF מציג שימוש גבוה בזכרון בשל הצורך הניהול מידע לגבי אורך התהליכים שנותרו לבצע. ואחרון, אלגוריתם PP בעל השימוש הגבוה ביותר בזכרון (8.7%) כיוון שהאלגוריתם דורש יותר משאבים לניהול התהליכים.

הגינות



גרף 3 – מידת ההגינות של תהליך

ההגינות בסימולטורים מבוססת על חישוב השונות של זמני ההמתנה של כל התהליכים. כלומר ההגינות בעצם מודדת כמה זמני ההמתנה של כל התהליכים באלגוריתם נבדלים זה מזה.

ככל שזמני ההמתנה בין התהליכים גדלים יותר כך השונות גדלה יותר וההגינות פוחתת.

נשים לב כי אלגוריתם הNPP הוא בעל ההגינות הגבוהה ביותר כלומר השונות בין זמני ההמתנה של התהליכים גדולה יחסית. מנגד אלגוריתם Priority הוא בעל ההגינות הנמוכה ביותר כלומר זמני ההמתנה של התהליכים די קרובים זה לזה.

למרות שהיינו מצפים מאלגוריתם הRR לספק את רמת ההגינות הנמוכה ביותר, כלומר השונות בין התהליכים אמורה להיות די קרובה, נראה כי האלגוריתם מספק הוגנות די סבירה ביחס לשאר האלגוריתמים.

הסיבה לכך יכולה לנבוע מבחירת הquantum שנבחרה עבור הסימולטורים, ייתכן כי הערך הנבחר קצר ולכן מספר החלפות התהליכים גדל וגרם לפיזור רחב של זמני ההמתנה. סיבה נוספת יכולה להיות קצב הגעת התהליכים, כאשר הקצב לא אחיד או במקרים בהם מגיעים הרבה תהליכים בטווח זמן קצר יחסית, ייתכן מצב בו התהליכים החדשים נכנסים לתור ומעכבים את התהליכים שכבר החלו את ריצתם וחזרו לסוף התור.

סיכום

בבחינת התוצאות שהתקבלו מהסימולטורים של האלגוריתמים השונים, ניתן לראות הבדלים ברורים בזמני ההמתנה, ההשהייה, ההשלמה והתגובה וגם בניצולת המעבד והזכרון. אלגוריתם העדיפות בגרסה המפקיעה שלו, כלומר אלגוריתם ה Preemptive Priority הראה את הביצועים הטובים ביותר מכלל האלגוריתמים, תוצאות אלו נובעת מהיכולת של האלגוריתם להפקיע את המעבד לתהליכים חשובים ביותר.

מנגד, אלגוריתם Round Robin הראה ביצועים יוצאי דופן מבחינת זמני תגובה, עם זמן תגובה ממוצע של 2.53 שניות. אך כאשר נמדדו זמני המתנה והשלמה, התוצאות היו גבוהות למדי. הסיבה לכך היא שהאלגוריתם מחלק את זמן המעבד בצורה שווה בין תהליכים באמצעות זמן quantum שנקבע מראש (50ms), דבר שככל הנראה גרם להמתנה ארוכה יותר עבור תהליכים שצריכים עוד זמן CPU על מנת לסיים.

אלגוריתם SJF סיפק ביצועים טובים במיוחד בזמני ההמתנה וההשלמה עם תוצאות של 4.5 ו-3.58 שניות בהתאמה. היתרון של האלגוריתם נובע מהתעדוף של תהליכים עם זמן ביצוע קצר, מה שמפחית את זמני ההמתנה הכוללים. בנוגע לניצולת המעבד והזכרון, אלגוריתם SJF הראה את השימוש הגבוה ביותר במעבד (0.77%) ואת השימוש הגבוה ביותר בזכרון (8.7%), עקב ניהול התור לפי זמן הביצוע הנמוך ביותר. לעומת זאת אלגוריתם Non-preemptive Priority הראה זמני המתנה והשהייה דומים אלו של SJF אך עם ניצול נמוך יותר של המעבד והזכרון עם תוצאות של 0.15% ו-6.3% בהתאמה. מבחינת הגינות, הNPP הציג את השונות הגבוהה ביותר בין זמני ההמתנה של התהליכים, בעוד שPP הראה את ההגינות הנמוכה ביותר עם זמני ההמתנה קרובים זה לזה. הדבר נובע מתעדוף התהליכים באלגוריתם PP שמוביל להפחתת השונות ביניהם.

מסקנות אלו מדגישות את היתרונות והחסרונות של כל אחד מהאלגוריתמים, ואת הצורך בהשוואה בין פרמטרים שונים כמו זמני המתנה, השהייה, השלמה ותגובה, יחד עם ניצולת המעבד והזכרון. חשוב לבחור את האלגוריתם המתאים בהתאם לצרכים הספציפיים של המערכת ולמטרות השימוש, תוך שמירה על איזון בין הוגנות, קצב עבודה, תקורה וניצולת.

חשוב לציין שהתוצאות שהתקבלו הן עבור הסימולטורים הספציפיים שנבנו לצורך עבודה זו, תוך שימוש במספר התהליכים וזמן הביצוע (BT) מקסימלי שהוגדרו מראש. ייתכן שבמקרים שונים, בהם מספר התהליכים משתנה, זמן הביצוע משתנה או פרמטרים נוספים משתנים, תוצאות הביצועים של האלגוריתמים עשויות להיות שונות. שינויים בכמות התהליכים או זמני הביצוע עשויים להשפיע על זמני ההמתנה, ההשהייה, ההשלמה והתגובה של התהליכים. בנוסף, המערכת בה רצות הסימולציות (כגון מערכת הפעלה, חומרה או משאבים זמין) עשויה להשפיע על התוצאות. גורמים כמו ביצועי המעבד, קיבולת הזיכרון, ותנאי הסביבה הכלליים יכולים להשפיע על התנהגות האלגוריתמים ולגרום לשינויים בביצועים. לכן, יש לקחת בחשבון את הקשר בין התנאים הספציפיים של הסימולציה לתוצאות המתקבלות, ולהבין שהתוצאות נוגעות למקרה הנסקר ואינן בהכרח משקפות תוצאות תחת תנאים שונים או במערכות שונות.

ביבליוגרפיה

1. ספרים

- ג'קי הרץ, דוד שריאל. **מדריך למידה מערכות הפעלה**. הוצאת האוניברסיטה הפתוחה.

Prentice A.S. Tanenbaum, *Modern Operating Systems*, 4th ed. (- Hall, 2015)

2. מאמרים

Andysah Putera Utama Siahaan(2016). Comparison Analysis of CPU - Scheduling : FCFS, SJF and Round Robin

-U.Saleem,M.Y.Javed (2012). Simulation of CPU scheduling Algorithms.

-Pushpraj Singh¹ , Vinod Singh² , Anjani Pandey³.(2014). Analysis and Comparison of CPU Scheduling Algorithms

-Hogar,K.Omar(2021). Comparative analysis of the essential CPU scheduling.

-Ali A. AL-Bakhrani¹ , Abdulnaser A. Hagar² ,Ahmed A. Hamoud³ and Seema Kawathekar(2021).Comparative Analysis Of Cpu Scheduling Algorithms: Simulation And Its Applications

-González-Rodríguez, Miguel; Otero-Cerdeira, Lorena; González-Rufino, Encarnación; Rodríguez-Martínez, Francisco Javier(2024). Study and evaluation of CPU scheduling algorithms.