פרק 1:

מערכת אצווה – מערכת היכולה לבצע באופן רצוף סדרה של משימות שהוכנו מראש, כל משימה מתבצעת מהתחלה עד הסוף.

קיפאון – קבוצה של תהליכים נמצאת במצב של קיפאון, כאשר כל אחד מהתהליכים לא יכול להתקדם אלא לאחר שחבר אחר בקבוצה יגרום לאירוע שישחרר אותו. שני תהליכים שכל אחד מהם מחזיק משאב שהאחר צריך.

קריאות מערכת – לרשות המשתמש הרגיל (שלא נמצא במצב מיוחס) אין אפשרות לבצע הוראות מיוחסות.לצורך כך ישנו ממשק שדרכו יכול לבקש ממערכת ההפעלה פעולות הדורשות הרשאות מיוחסות. קריאת מערכת מוציאה את השליטה מידי תוכנית המשתמש ומעבירה אותה למ"ה. בגלל שתוצאת הפעולה אינה ודאית, מערכת ההפעלה מחזירה אינדיקציה על תוצאות הביצוע. בגלל זה מבצעים בדיקה של הערך המוחזר לאחר חזרת התוכנית לידי המשתמש. שירותים כמו למשל פתיחת קובץ, יצירת תהליך חדש, שליחת מידע ברשת וכו'. קריאת מערכת מתבצעת ע"י השמת הפרמטרים הנדרשים במקומות מוגדרים(כמו רגיסטרים) ואז ביצוע פקודת TRAP שמעבירה את השליטה למערכת ההפעלה. קריאת מערכת בשפה עילית כמו read() בשפת C היא בעצם מעטפת שנראית כמו פרוצדורת ספריה, בעצם תפקידה העיקרי הוא לשים את הפרמטרים של הקריאה ברגיסטרים ולבצע TRAP.

מבנה של מערכות הפעלה:

1. מערכת מונוליתית – בעצם אוסף גדול של פונקציות
2. מערכות מרובדות – מאורגנת ברבדים שנמצאים ביחס היררכי, כאשר לכל רובד יש שכבת ממשק לצורך תקשורת עם הרבדים האחרים.
3. מכונה מדומה – שכבת תוכנה שמספקת לשכבות שמעל מספר העתקים של החומרה עם כמות משאבים פחותה מהכמות הפיסית.
4. מערכת exokernel – שירותים כמו מערכת קבצים, ניהול זיכרון, תזמון תהליכים ותקשורת הוצאו אל מחוץ לגרעין מ"ה. נובע מהסיבה שמ"ה לא כופה על המשתמש את צורת השימוש במשאבים.מ"ה רק דואגת להגנת המשאבים וניהולם.המערכת מקצה משאבים לתכניות משתמש שיכולות לעשות שימוש במשאבים כרצונן.
5. מערכות שרת-לקוח – מבוסס על רעיון של גרעין מינימליסטי micro-kernel, גרעין מ"ה מריץ בקשות של לקוחות אל שרתים המפוזרים במערכת, כאשר מתקבלת חזרה תשובה, הגרעין מחזיר אותה לתהליך המבקש.בשיטה זו, התוכנה מורכבת מיחידות עצמאיות שכל אחת ממלאת תפקיד מוגדר.שיטה זו מצמצת מאוד תלות בחומרה. ההבדל בין זה לבין exokernel הוא שבמודל זה, כל סוגי המשאבים מנוהלים ע"י תוכנת לקוח שמספקת הפשטה אחידה לכל המשתמשים.

גרעין \ KERNEL – החלק המרכזי והקריטי של מ"ה שאחראי על ביצוע התפקידים העיקריים שלה, לרוב הגרעין יכיל מרכיבים שאחראיים על ניהול פסיקות, תזמון תהליכים,ניהול זיכרון, ניהול שטחי אחסון משניים ועוד. הגרעין נמצא באופן קבוע בזיכרון הראשי של המחשב.

מצב ראשוני \ KERNEL MODE – מצב "מיוחס" של המעבד שבו ניתן לבצע כל פעולה אפשרית ללא הגבלות,חלקים של מ"ה שמחייבים שימוש בפעולות רגישות כמו למשל הכתיבה להתקני IO רצים במצב זה. תכניות רגילות תמיד רצות במצב משתמש.

פסיקות חומרה – מטרתן בעיקר להגדיל את ניצולת ה-CPU, כאשר יש תכנית שמבקשת לבצע IO, קריאה או כתיבה מההתקן, מושהית ע"י מערכת ההפעלה עד לסיום ה-IO.המעבד ינצל את ההשהיה לטובת ביצוע תוכניות אחרות.כאשר ההתקן עליו בוצע IO סיים את משימתו, יישלח סיגנל בקשה לפסיקה לCPU.המעבד משהה את הפקודות שמבצע כעת ומפעיל את השגרה לטיפול בפסיקות.לאחר סיום הטיפול בפסיקה, יכול ה CPU לחזור לבצע את התוכנית המקורית או כל תוכנית אחרת שהמתזמן יחליט.

ריסוק חיצוני – כאשר נוצרים קטעי זיכרון לא רציפים כתוצאה מהקצאות זיכרון מדויק, יוצר מצב של זיכרון חיצוני לא בשימוש

ריסוק פנימי – מצב שבו הזיכרון המוקצה מעט גדול יותר מהזיכרון הדרוש, יוצר מצב של זיכרון פנימי לא בשימוש

פרק 2 – תהליכים:

מה קורה כאשר בעת טיפול בפסיקה, מגיעה פסיקה חדשה? גישה אחת, לחסום כל הסיגנלים בעת טיפול בפסיקה. יתרון: פשטות. חסרון:מחייב טיפול מהיר (קריאת שעון).גישה שנייה (מועדפת) הענקת עדיפויות לסוגי פסיקות.

שגרה לטיפול בפסיקות חומרה: (חומרה)פסיקת חומרה, מעבד מסיים הוראה נוכחית, מעבד מסמן שהתקבלה פסיקה, מעבד מכניס את הפקודה הישנה למחסנית, מעבד טוען PC חדש לפי סוג הפסיקה--> (תוכנה) שמירת שארית הנתונים של התהליך, ביצוע הפסיקה, שחזור מצב תהליך ישן, שחזור PSW+PC.

הבדל בין תהליך לתוכנית? תכנית זה מתכון, תהליך זה ביצוע המתכון. דוגמא לפתיחת 2 מסמכי WORD יוצר 2 תהליכים נפרדים המריצים את אותו קוד.

מקרים שונים לצירת תהליכים: 1. בעת אתחול מערכת - תהליכי רקע Deamons: הבאת אימייל, המתנה לבקשה, הדפסה spooler; תהליכי משתמש: לתקשורת עם משתמש, הקלדת שם משתמש. 2. תהליך המבצע sys call כדי ליצור תהליך נוסף לצורך עזרה בביצוע עבודה; 3. יצירת תהליך ע"י משתמש – כגון פתיחת אפליקציה ; 4. במערכות אצווה – משתמשים שולחים בקשות למחשב ראשי שבעת הצורך יוצר תהליך שמבצע את העבודה.

שגרה לטיפול בפסיקות חומרה – יוצרת תקורה מכיוון שיש לבצע שמירת תוכן של התהליך, על מנת לאפשר חזרה לאותה נקודה.

יצירת תהליך ב-UNIX: רק ע"י fork שלאחר קריאה נוצר העתק זהה של התהליך האב, אותו מרחב זיכרון, קבצים פתוחים,מאפייני קבצים, רגיסטרים. לרוב תהליך הבן מפעיל את execve לצורך שינוי מרחב הזיכרון והרצה של תוכנית אחרת כרצונו. לבן יש מרחב כתובות משלו שהוא העתק של הכתובות של האב, אך שינוי באחד לא יגרור שינוי של השני.לאחר ביצוע fork האב מקבל את PID של הילד והילד מקבל PID=0.שיפור של fork בלינוקס הוא במקום להעתיק את כל ה-DATA של האב לבן, טבלת הדפים של הבן מפנה לזו של האב ללא הרשאת כתיבה, רק כאשר הבן מנסה לכתוב ונכשל, הקרנל יוצר עבורו עותק של דף ספציפי זה עם הרשאת כתיבה.

waitpid – כאשר האב רוצה להמתין עד לאחר שבן מסוים \ כולם מסיימים לרוץ.

מצב זומבי – כאשר בן מסיים ריצה, אך האב לא ביצע את waitpid עבורו.גורם למצב שבו הילד נחשב זומבי.כאשר האב יבצע את הפקודה, תהליך הבן יסתיים כרגיל.

יצירת תהליך ב-WIN – לא מחולק ל-2, קריאה ליצירת תהליך מיד יוצרת תהליך עם מרחב כתובות משלו ללא קשר לאב. היררכיה של תהליכים: ב-UNIX ישנה היררכיה, ב-WIN אין היררכיה.סיום תהליך – סיום רגיל, סיום עקב שגיאה(פרמטר שגוי), קריסה(התרסקות),סיום בעקבות הריגת תהליך.

תהליכונים ותהליכים –

חולקים: מרחב זיכרון זהה, משתנים גלובלים, קבצים פתוחים,תהליכי בנים,סיגנלים וטיפול בסיגנלים.

לא חולקים: PC,רגיסטרים,מחסנית, מצב.

מצבים של תהליכים: שיקולים להוספת מצבים "חדש" - לטובת תזמון תהליכים ועדיפויות. מצב "סיום" לצורך סטטיסטיקה ולצורך קבלת החלטות המסתמכות על אופן סיום של תוכנית(איחוי דיסק).

יתרונות למרחב כתובות משותף של תהליכונים בתוך תהליך – ביצוע כמה פעולות במקביל(שמירה+הקלדה בWORD);

יתרונות של תהליכונים – זמן יצירה \ זמן השמדה קטן הרבה יותר, תקשורת בין תהליכונים קלה בגלל שיתוף מרחב כתובות, לא דורשת התערבות גרעין מ"ה.

מימוש תהליכונים במרחב משתמש – גרעין לא מודע לקיומם, מימוש ע"י פונקציות ספריה המנהלת את טבלת התהליכונים.

יתרונות: 1. החלפת תהליכונים לא דורשת מעבר למצב ראשוני (מתזמן פנימי); 2. התאמת אלג' התזמון הפנימי לצרכי כל ישום בנפרד ; 3.אפשרות לרוץ ללא תמיכה של גרעין מ"ה, מאפשר לממש אפליקציות בעלות מס רב של תהליכונים גם כאשר מערכת ההפעלה לא תומכת בתהליכונים.

חסרונות: 1. אם אחד התהליכונים מבצע קריאה הגורמת לחסימת התהליך, יושעו יחד איתו גם כל שאר התהליכונים; 2. בדר"כ יש לתהליכונים אופצ' להפקיע מעבד מתהליכון אחר, פגיעה בעקרון חוטי ביצוע נפרדים.

מימוש תהליכונים בגרעין:

יתרונות:

1. מתגבר על הבעיות של הגישה הקודמת.
2. פרוצדורות של גרעין המערכת יכולות להיות ממומשות בתהליכונים.

חסרונות: בזבוז זמן הכרוך במעבר למצב ראשוני.

הפתרון הוא שילוב שני הגישות, גרעין מ"ה מספק תהליכונים ברמת הגרעין, בזמן שתהליכונים ברמת משתמש רצים ללא ידיעת הגרעין.

מכונת מצבים: מעבר **רץ--> חסום** כאשר תהליך לא יכול להתקדם,שחרור ע"י פסיקה. מעבר **חסום--> מוכן** ע"י פסיקה. מעבר **רץ <--> מוכן** מתזמן מערכת מפקיע מעבד.

מניעה הדדית הכוללת המתנה פעילה:

1. מניעת פסיקות לפני כניסה לקטע קריטי:

יתרונות – פשוט.

חסרונות – המתנה פעילה, כוח יתר לתהליכי משתמש,ירידה במקביליות תגובה איטית, ישים רק במערכת עם מעבד אחד.

סיכום: יעיל רק עבור גרעין מערכת.

2. משתנה נעילה יחיד (משותף):

חסרונות - המתנה פעילה,גורם לבעיית מרוץ כאשר לאחר קריאת ערך lock מתבצעת החלפת תהליכים.

3. פתרון התור (turn) –

יתרונות: מקיים מניעה הדדית.

חסרונות: המתנה פעילה ,יכול לגרום להרעבה, אם קיים הבדל בזמן ביצוע קטעים לא קריטיים ייתכן הפרה של תנאי שתהליך מחוץ לקטע קריטי יכול למנוע מתהליך אחר להיכנס אליו.

4. פטרסון (3 משתנים משותפים – מערך turn+interested):

הצהרה על רצון להיכנס לקטע קריטי+בדיקה האם תורי.

יתרונות – תהליך מחוץ לא יימנע מתהליך להיכנס לקטע, מבטיח הגינות, מי שיכנס לקטע הקריטי יהיה הראשון שעדכן את turn. פתרון סביר.

5. TSL:

פותר את בעיית ביטול הפסיקות במספר רב של מעבדים, פעולה אטומית. בודקת את תוכן המשתנה lock אם 0, משנה ל-1 ומחזירה ערך ישן, TSL נועלת את ערוץ הזיכרון למשך זמן ההוראה.

יתרונות: מקיים את 4התנאים.

חסרונות: המתנה פעילה,דרוש תמיכה של חומרה בפעולות אטומיות,אינו מסדיר את אופן הכניסה לקטע, יכול לגרום להרעבה במקרה של יותר מ-2 תהליכים.

מניעה הדדית ללא המתנה פעילה:

בכל הפתרונות המוצעים בהמשך, אנו מניחים שתהליך יכול להשהות את עצמו ע"י קריאת המערכת sleep ולהתעורר כאשר תהליך אחר מבצע קריאת מערכת wakeup.תהליכים שמבצעים sleep עוברים למצב חסום ולא מעסיקים את המעבד.

1.סמפורים – המשתנה S מכונה סמפור, בעצם מנגנון המסופק ע"י מ"ה או שפת תכנות.המנגנון משתמש בתור המנהל את התהליכים הממתינים על הסמפור במצב רדום.

פעולת DOWN – אם מקבלת '0', התהליך מוכנס לתור הממתינים ומועבר למצב חסום. אם מקבלת '1' יש אישור מעבר וערכו של התהליך יורד ל- 0.

פעולת UP – אם יש לפחות תהליך אחד רדום בתור, תעיר אותו ותחזור. אם אין תהליך רדום תעלה מונה ל-1.

יתרונות – מונע המתנה פעילה, אין הגבלה על מספר התהליכים,שירות הניתן ע"י מערכת ההפעלה,פותר בעיות הקשורות לקטע קריטי וסנכרון.

חסרונות: יעיל רק במצב של מעבד אחד+זיכרון אחד או מספר מעבדים+זיכרון משותף.לא ניתן להשתמש במערכות מבוזרות עם ניוד תהליכים.

2.מנעולים – משתנה במרחב זיכרון משותף המשמש לניהול הגישה לקטע קריטי, שני מצבים נעול\לא נעול. השימוש במנעולים יעיל יותר משימוש בסמפור בינארי כי אין צורך לעבור מרמת המשתמש לרמת הגרעין.

3.מבני פיקוח – זהו מצב שהטיפול במניעה ההדדית עובר לקומפיילר, מבנה פיקוח הוא ספרייה של שגרות,משתנים ומבני נתונים המוגדרים יחד במבנה של שפת תכנות.מבני פיקוח מספקים פתרון למניעה הדדית אך לא לסנכרון בין תהליכים. הפתרון הוא שימוש במשתנים מיוחדים condition variables, פעולת signal(c) מעירה אחד מהתהליכים שהיה חסום על ידי מבני פיקוח על התנאי C.פעולת wait(c) גורמת להשהיית תהליך כל עוד תנאי C לא מתקיים,מבנה פיקוח מבטיח שלאחר הקריאה ל-wait אחד מהתהליכים שהיה חסום על קטע קריטי יוכל להיכנס אליו.שימוש במבנה פיקוח פותר את המתכנת מהצורך לשחרר את הנעילה על קטע קריטי. שפות תכנות רבות לא מספקות מבני פיקוח, ברמה של מימוש בשפת תכנות קל יותר לספק סמפורים מאשר מבני פיקוח.

חסרונות: יעיל רק במצב של מעבד אחד+זיכרון אחד או מספר מעבדים+זיכרון משותף.לא ניתן להשתמש במערכות מבוזרות עם ניוד תהליכים.

דרכים להימנע משני תהליכים פעילים בתוך מבנה פיקוח הן:

1. לתת לתהליך ש-signal(c) העיר אותו לרוץ, תוך כדי השהייה של התהליך שקרא ל- signal.

2. לתת לתהליך שקרא ל-signal(c) לסיים את השגרה של מבנה הפיקוח ורק אז להעיר את אחד התהליכים שממתינים על C.

3. לדרוש ברמת התחביר של שפת התכנות, שהקריאה ל-signal(c) תתבצע רק בסוף השגרה של מבנה הפיקוח.

4. העברת הודעות – שיטת העברת הודעות פועלת גם במערכות מבוזרות וגם במודל של מספר מעבדים עם זיכרון משותף ולכן הוא כללי יותר.שתי השגרות send&receive יכולות להיות כאלו שחוסמות את התהליך השולח או לא חוסמות אותו. מיעון ישיר – מחייב להעביר כפרמטר את מזהה התהליך.מיעון עקיף – משתמש בתחנת ביניים שאליה מועברות ההודעות וממנה מתבצעת משיכתן.

יתרון: בעיית המניעה ההדדית נפתרת בקלות ע"י שימוש בשליחה\קבלה שהן חוסמות. ובעיית הסנכרון נפתרת בפשטות ע"י הגדרת כמות משאבים מתאימה.

הבדל בין הרעבה לקיפאון: מצב שבו תהליכים רצים אך לא מתקדמים במשימה נקרא **הרעבה,**יכול להימשך זמן בלתי מוגבל. ההבדל הוא שכל התהליכים הנמצאים במצב קיפאון ממתינים לאירוע מאחד מתהליכי הקבוצה, בעוד שבמצב הרעבה הם לא מקבלים משאב כלשהו(ללא קשר לקבוצה) לאורך תקופה לא מוגבלת וכתוצאה מזה לא יכולים להתקדם.

מושגים:

הגינות – fairness – כל תהליך יקבל את אותו זמן CPU
אכיפת מדיניות – policy enforcement – קביעת מדיניות הרצה לקבוצה של תהליכים (סדר ביניהם)
איזון – balance – איזון בין עומסי CPU במערכות עם מספר מעבדים
קצב עבודה - throughput – כמות תהליכים המסיימים באותה יחידת זמן
זמן השהייה – turnaround time – הזמן מהרגע שהתהליך הגיע עד שסיים לרוץ
ניצולת CPU – CPU utilization – אלג' תזמון צריך לדאוג שהמעבד יהיה עסוק רוב הזמן
זמן תגובה – response time – ממוצע הזמנים מרגע מתן הפקודה עד לקבלת תגובה ראשונה (לחיצה על עכבר)
יחסיות – proportionality – הגדרה של זמן תגובה טוב למערכת מסוימת
עמידה בזמנים – meeting deadlines – במערכות RT
ניתנות לחיזוי – predictability – מיועד ל- Soft RT (חיזוי תבניות עתידיות)

קריטריונים להערכה תזמון תהליכים:

כל המערכות: balance + policy enforcement + fairness
מערכות אצווה: CPU utilization+ turnaround time+throughput .
מערכות אינטראקטיביות:proportionality+ response time.
מערכות RT: predictability +meeting deadlines.
באים אחד על חשבון השני: CPU utilization&response time ; balance&fairness ; throughput&turnaround time .

אלג' לתזמון זמן קצר במערכות אצווה:

1. FCFS: או FIFO פשוט ביותר, מנהל תור תהליכים מוכנים לריצה,כאשר תהליך שרץ ב-CPU מבצע קריאת מערכת המובילה לחסימה, המתזמן נותן CPU לתהליך הבא בתור.כאשר התהליך חוזר ממצב רדום הוא מוכנס לסוף התור.

יתרון: פשוט למימוש.

חסרון:משך זמן השהייה ארוך.תהליכים עתירי קלט פלט יארכו הרבה זמן אם במערכת קיימים תהליכים עתירי חישוב.

2. SJF: מתגבר על חסרון של FCFS (זמן השהייה ארוך) בתנאי שזמני הריצה של התהליכים ידועים מראש.ובתנאי שמופעל על קבוצת תהליכים ידועה מראש.אופן פעולה, נותן זמן CPU לתהליך בעל זמן הריצה הקצר ביותר ומריץ אותו עד לסיום. לא מפקיע מעבד non-preemptive.

יתרונות: זמן שהייה מזערי בהנחה שהתהליכים ביצעו כמות זהה של פעולות I/O.

חסרונות:עלול לגרום להרעבה

3. SRTN – הוא שינוי של SJF, מבטיח שאם זמני ריצה ידועים+קבוצת התהליכים קבועה אז **זמן שהייה יהיה מזערי**, **ללא הנחות על I/O** , מבצע זאת ע"י הפקעת מעבד אם: תהליך חסום הפך מוכן, מגיע תהליך חדש שמוכן לריצה,אם תהליך רץ מבצע קריאת מערכת שגורמת להרדמתו.במקרה כזה, מתזמן בודק למי נשאר זמן קצר ביותר לריצה ומריץ אותו.

אלג' לתזמון זמן קצר במערכות אינטרקטיביות: אין מתזמן לטווח ארוך, בניגוד לאצווה, כל העבודות מתקבלות והופכות לתהליכים.

1. RoundRobin – חלוקת זמן CPU לquantum,לאחר סיום, מפקיעים מעבד,ומעבירים לתהליך הבא.מנוהל בתור מעגלי.אם תהליך הופסק בשל קריאת לפונקצית מערכת, יוצא מהתור המעגלי, רק לאחר שמגיע למצב ready חוזר לסוף התור (לפני התהליך המבוצע כרגע).

יתרונות: קל למימוש.
חסרונות:קביעת quantum מתאים קריטית, קצר מדי גורם לתקורת החלפת תהליכים.ארוך מדי, פוגע אנושות בתהליכים אינטרקטיבים.
צריך מנגנון דיפראנטיזציה (עדיפויות).

2.תזמון עם עדיפות(קדימות) – כל התהליכים המוכנים לריצה בזיכרון מדורגים לפי עדיפויות.תהליך רץ ב-CPU כל עוד עדיפותו גדולה משל שאר התהליכים, אם הוא מבצע קריאת מערכת המרדימה אותו, התהליך בעל העדיפות השנייה נכנס לריצה.
עדיפות קבועה – מתן עדיפות קבועה סטאטית.
יתרון: טוב לתהליכים הדורשים זמן תגובה מהיר, וקצר מאוד.
חסרון: עלול לגרום להרעבה,אם תהליך מקבל עדיפות גבוהה לזמן בלתי מוגבל, זה יגרום להרעבה של שאר התהליכים.
עדיפות דינאמית – עדיפות של תהליך נקבעת לפי החלק היחסי של ה-quantum שנוצל על ידו, ככל שהחלק המנוצל גדול יותר, עדיפות התהליך תהיה קטנה יותר. במידה וישנו תהליך שרוב זמנו רדום אבל דורש מהירות תגובה, הוא יקבל עדיפות גבוהה.
יתרון: לא גורם להרעבה מפקיע מעבד לאחר ה-quantum.

3.ריבוי תורים- פתרון לדיפרנציאציה בין קבוצות תהליכים שונות,כאשר כל התהליכים בקבוצה מצפים לאותו סוג שירות, לכל תור משויכת עדיפות, כל התהליכים הנמצאים באותו תור בעלי עדיפות שווה, יכולים לעבור מתור לתור לפי החלטת מתזמן.

4.shortest process next – בניגוד לSRTN במערכות אינטראקטיביות אין אפשרות לדעת זמן ריצה צפוי.משתמש בשיטה לניבוי הזמן הדרוש לפעולה הבאה,חשיבות הזמנים בעבר יורדת בצורה מעריכית, שיטת exponential average המבוססת על aging נוסחא: e(i+1)=a*t(i)+(1-a)*e(i).
יתרונות:מנסה לספק זמן תגובה מזערי.
חסרונות: עלולה לגרום להרעבה של תהליכים שזמן הפעולה האינטראקטיבית שלהם ארוכה מהזמן הממוצע במערכת. ריבוי תורים יכול לעזור להתגבר על בעיה זו.

5.תזמון מובטח- הבטחת אופי שירות למשתמשים או תהליכים שיקבלו כמות שווה של זמן CPU.מ"ה מנהלת בטבלת תהליכים רישום על משך זמן ריצה, כמה קיבל כל תהליך\משתמש ביחס לזמן שהובטח ולפי זה יבחר המתזמן את מי להריץ.
יתרון:חלוקה שווה של זמן המעבד.
חסרון:תקורה גדולה כאשר נוסף\יורד תהליך מהמערכת נדרש לחשב מחדש את היחס לכל התהליכים.

4.lottery Scheduling – מפחיתה את התקורה של חישוב המידות המובטחות ומביאה תוצאות דומות באופן סטט'.בעת כניסת תהליך חדש, הוא מקבל כרטיס הגרלה המזכה בנתח של CPU,בהחלטת תזמון מתבצעת הגרלה כשהמנצח מקבל את הCPU למשך הזמן.
יתרונות: אין צורך לחשב יחס שימוש במעבד, קל לתת ביטוי לחשיבות תהליך ע"י חלוקת כרטיסים נוספים ובכך להגדיל את סיכויו לרוץ יותר זמן, תהליכים שמשתפים פעולה יכולים להעביר כרטיסים זה לזה.

5.fair share scheduling- במערכת מחשב מרובת תהליכים, התהליכים אינם בלתי תלויים, אפליקציות הן אוסף של תהליכים קשורים ומנקודת המבט של המשתמש מעניין אותו זמן התגובה של האפליקציה בכללותה ולא של תהליך כזה או אחר. תזמון זה מתחשב בעובדה שישנם תהליכים קשורים לקבוצה מסוימת.

<u>אלג' לתזמון זמן קצר במערכות RT:</u> c(i) כמות יחידות זמן חישוב דרושה למשימה מחזורית. p(i)- מחזוריות המשימה, כמות הפעמים ביחידת זמן שהמשימה i צריכה להיות מתוזמנת.
C(1)/p(1)+…+C(m)/p(m)

$$\sum_{i=1}^{m}\frac{C_i}{P_i}\le 1$$

פרק 3 – ניהול זיכרון:

- הקצאת שטחי זיכרון פנויים באמצעות מפת סיביות – חלוקה של הזיכרון לקטעים הגודל קבוע.
  כל סיבית משקפת את מצב ההקצאה של הקטע (1 – תפוס, 0 פנוי), הקצאה ע"י חיפוש סדרת 0 רציפה בגודל המבוקש.
  יתרון - תקורה קבועה (מפת סיביות בגודל קבוע), חסרון – חיפוש איטי
- הקצאת שטחי זיכרון פנויים באמצעות רשימות מקושרות –
  (רשימה דו כיוונית או שתי רשימות - אחת לתפוסים אחת לריקים)
  אלגוריתמים להקצאת מחיצות עבור תהליך:
  - First-Fit: סריקה של הרשימה עד שמוצאים שטח המתאים לגודל המבוקש, פיצול השטח הקיים למה שנדרש לתהליך הנוכחי והשארית – חיפוש מהיר
  - Next-Fit: הסריקה מתחילה כל פעם מהמקום בו עצרה בסריקה הקודמת (סריקה מעגלית)
  - Best-Fit: סריקה של כל הרשימה עד למציאת השטח הפנוי הקטן ביותר המסוגל להכיל את התהליך
  - Worst-Fit: סריקה של כל הרשימה עד למציאת השטח הפנוי הגדול ביותר המסוגל להכיל את התהליך
    - מומלץ למיין את הרשימות ע"פ גודל כך ש- BF ו- WF יהיו יעילים כמו FF ו- NF.
    - FF הכי טוב בגלל פשטותו, מהירותו ותבניות הריסוק שנוצרות
    - NF מהיר כמו FF אך דורש יותר פעולות Compaction
    - BF הכי גרוע בגלל שיוצר שטחים פנויים קטנים אשר קשים לניצול
    - WF יוצר תבנית ריסוק אחידה כמו NF ולכן פחות טוב מ- FF
  - Quick-Fit: מספר רשימות מקושרות המחזיקות שטחים פתוחים, כל רשימה מחזיקה שטחים בגדלים שונים – בשחרור הזיכרון יש צורך לאחד בין שטחים ריקים והדבר גורם העברות בין רשימות – במקרה הגרוע יש לעבור על כל הרשימות
- מהו גודל טבלת הדפים:
  נתונים:
  כתובת וירטואלית הינה בעלת 32 bits
  יחידות המען הקטנות ביותר הן בגודל 1 byte
  גודל כל דף הינו 4 kb
  גודל שורה של טבלת דפים הינו 4 bytes
  פתרון:
  גודל המרחב הווירטואלי 2^{32}*1byte = 4GB
  גודל הדף הינו 4kb לכן כמות הדפים שאליהם מתחלק המרחב הווירטואלי הינה:
  4Gb / 4kb = 2^{32} bytes/2^{12} bytes = 2^{20} pages
  גודל שורת טבלת דפים הוא 4 bytes לכן גודל טבלת הדפים כולה הינו:
  2^{20} * 4 bytes = 2^{20} * 2^{2} = 2^{22} bytes = **4 Mb**
  לכן לא ניתן להחזיק את טבלת הדפים בדף זיכרון בגודל 4 kb.
- טבלת דפים בצורתה הפשוטה גדולה מדי, מספיק להחזיק חלקים של טבלת דפים המתייחסים לדפים שנמצאים בשימוש.
- תרגום המענים דורש פנייה לטבלת הדפים, אם הטבלה שוכנת בזיכרון הראשי, ביצוע הוראה ידרוש פנייה אחת לזיכרון לשם תרגום המען ופנייה נוספת לביצוע ההוראה עצמה – מקטין את מהירות המעבד בחצי מהיכולת שלו בפועל.
- מה קורה כאשר כל שורות טבלת TLB מנוצלות: ניתן לבצע החלפה של שורה ע"פ אלג' LRU (מדובר על מימוש בחומרה)
- כדי שהחיפוש ב- TLB יהיה מהיר, המעגלים החשמליים ב- TLB מבצעים חיפוש מקבילי בכל השורות יחד.
- מהו גודל טבלת הדפים המהופכת:
  נתונים:
  גודל הזיכרון הפיזי 1 GB
  גודל דף הינו 8 kb
  גודל שורה של טבלת דפים הינו 4 bytes
  פתרון:
  1 Gb / 8 kb * 4 bytes = 2^{30}bytes / 2^{13} bytes * 4 bytes = 2^{19} bytes = 512 kb

אלגוריתמים להחלפת דפים:

1. אלג' אופטימלי: האלג' יחליף את הדף שההתייחסות תקרה בעתיד הרחוק ביותר מבין כל הדפים הנמצאים בקבוצת דפים נתונה; למ"ה אין אפשרות לדעת את המידע הנ"ל ולכן **לא ניתן לממשו**.
2. NRU (Not recently used): שימוש ב- R (referenced) ו- M (modified); בפניה לדף, R הופך ל- 1, בכל פרק זמן נתון מ"ה מכבה את R. בעדכון הדף, M הופך ל- 1 ולאחר עדכון ה- HD מכבים את M.
   עדיפות פינוי דפים לאחר פסיקת דף:
   a. R=0, M=0
   b. R=0, M=1 (עדיף לכתוב ל- HD מאשר לפנות דף עם R=1 כי יכול להיות בשימוש בקרוב)
   c. R=1, M=0
   d. R=1, M=1
   קל למימוש, ביצועים סבירים אך לא אופטימליים.
3. FIFO: רשימה אשר בראשה נמצא הדף הוותיק ביותר ובסופה הצעיר ביותר (זה שהגיע אחרון); בפסיקת דף, מוחלף הדף שנמצא בראש הרשימה (זה שלא השתמשו בו מזמן), הדף החדש מתווסף לסוף הרשימה;
   פשוט למימוש, מתייחס רק לוותק ולא לנחיצות במערכת ולכן יכול להוציא דפים אשר בשימוש לעיתים קרובות.
4. Second Chance: שכלול של FIFO; נותן הזדמנות שנייה לדף ש- FIFO היה מפנה; כאשר דף נבחר לפינוי מראש הרשימה (הוותיק ביותר), נבחנת סיבית ה- R שלו, אם היא 0, הוא מפונה, אם היא 1, אז מכבים אותה ומעבירים את הדף לסוף הרשימה (כאילו הרגע הגיע), ממשיכים בסריקה.
5. שעון – Clock Page Replacement: כמו הזדמנות שנייה רק שמבנה הנתונים הוא רשימה מעגלית; המיון ע"פ וותק גם כן; סיבית R נבחנת, אם היא 0, הדף מפונה, אם 1 מכבים אותה והמחוג מתקדם לדף הבא.
   יעיל כי להבדיל מרשימה רגילה, כאן אין צורך להעתיק את הדף לסוף הרשימה, המחוג פשוט זז קדימה.
   האלג' לא מתייחס לסיבית M.
6. LRU (Last Recently Used): מחזיק את הדפים ממוינים ע"פ זמן התייחסות; מחליף את הדף הוותיק ביותר מבחינת התייחסות, כלומר את הדף שהפנייה אליו נעשתה בעבר הרחוק ביותר.
   יתרון - ביצועיו קרובים לאופטימלי; חסרון – עלות גבוהה בהחזקת דפים ממוינים לפי זמן התייחסות, כי בכל פנייה לדף צריך לעדכן את הסדר היחסי שלו בין שאר הדפים; מימוש בתוכנה ידרוש מבנה נתונים גדול; מימוש בחומרה מצריך חתימת זמן על כל מסגרת ומעגלים לחיפוש אסוציאטיבי (חיפוש בו זמנית בכל המבנה); השורה במטריצה שבה נמצא המספר הבינארי הקטן ביותר, מייצגת את הדף שהפנייה אליו הייתה המוקדמת ביותר.
7. Aging בתוכנה: זהו מימוש הדומה LRU אך בתוכנה; לכל דף מוחזק מונה הגדל ב-1 אם הייתה אליו פנייה בפרק זמן מסוים המוגדר ע"י האלג' (כלי להתעלם מהיסטוריה ישנה), לפני הגדלת המונה עושים לו הזזה, בהחלפת דף, יוחלף הדף בעל המונה הקטן ביותר; אם יש יותר מדף אחד כזה, יבחר אקראית;
   קיימת גם גרסה שלא מבצעת הזזה של המונה (NFU) אך הדבר גורם לשמירת היסטוריה, כלומר, ערך המונה לעולם לא יורד; סדר הפניות בין הדפים אינו ידוע ולכן במצב של מונה זהה, יבחר דף אקראי במקום הדף שאליו פנו קודם.
8. אלג' ע"פ קבוצת עבודה של תהליך: מסתמך על ההנחה כי בכל שלב של ביצוע תהליך, התהליך פונה לקבוצה מצומצמת של דפי זיכרון. (קבוצת עבודה של תהליך = קבוצת דפי זיכרון הנחוצים לריצת התהליך בשלב הנוכחי של הביצוע); עובד ע"פ טכניקת demand paging האומרת כי דפים מובאים לזיכרון לפי צורך.
   אם קבוצת העבודה ידועה, ניתן להביא אותה לזיכרון לפני תחילת ריצת התהליך, טכניקה זו נקראת prepaging והיא חוסכת הרבה פסיקות דף; קשה לנהל מעקב אחרי קבוצת העבודה; הקצאת קבוצת מסגרות לתהליך אשר קטנה מגודל קבוצת העבודה שלו, עלולה לגרום לסחרור (thrashing) = פסיקות דף קורות לאחר ביצוע מספר קטן של הוראות קוד ורוב זמן ה- CPU מבוזבז על החלפת דפים במקום קידום התהליך;
   ניתן לממש ע"י שמירת הזמן הווירטואלי של התהליך,
   תיאור האלג': כאשר יש צורך לפנות דף, עוברים על הדפים ובודקים את R, אם 1, מעדכנים את הזמן הווירטואלי של הדף לזמן הנוכחי ומכבים את R. (פנו לדף לאחרונה ולכן לא מועמד לפינוי); אם R=0, לא הייתה אליו התייחסות ולכן מועמד לפינוי, אם הגיל שלו גבוה מכמות יחידות הזמן

שנקבעו, זה אומר שהוא לא בקבוצת העבודה ולכן יפונה, אם הגיל שלו מתחת, הוא עדיין נמצא בקבוצת העבודה ולכן לא יפונה כרגע אלא אם יתגלה בסוף המעבר שהוא הוותיק ביותר מבין קבוצת העבודה שלו; הגיל מחושב ע"י הזמן הנוכחי פחות זמן השימוש האחרון בדף זה; אם כל ה- R=1 נבחר אקראית את הדף לפינוי אך רצוי עם M=0 כדי לא לכתוב לדיסק.

9. WSClock: גרסה מעגלית של האלג' הקודם; בכל מיפוי דף לזיכרון, מוסיפים איבר לרשימה; בפינוי זף מהזיכרון, הדף מוסר מהרשימה; באלג' זה, אם R=0 (הדף מיועד לפינוי) וגילו גבוה מהרצוי, אך מתגלה ש- M=1, מתזמנים כתיבה לדיסק וממשיכים לחפש הלאה דף העונה לאותו קריטריון אך ש- M=0; לאחר סיבוב שלם, במידה ולא נמצא R=0, העונה לשאר הקריטריונים (גיל ו- M=0), במידה ותוזמנה כתיבה, יתגלה הדף שכבר נכתב לדיסק ואותו ניתן להוציא, אם לא תוזמנה כתיבה, נוציא את הדף בעל הגיל הגבוה ביותר מאלו שבקבוצת העבודה.

הקצאת דפים: ניתן להקצות כמות קבועה של דפים לתהליך או כמות משתנה.

מדיניות החלפה לוקאלית: פינוי הדף מתבצע רק מקבוצת הדפים של התהליך הדורש דף נוסף. שימוש במדיניות זו עם הקצאת דפים קבועה, יכולה לגרום לסחרור או בזבוז (קבוצת עבודה גדלה או קטנה ואז יש דפים חסרים או מיותרים שהוקצו).

מדיניות החלפה גלובלית: פינוי הדפים ללא קשר לזהות התהליך הדורש; בגישה זו ניתן לממש רק את האפשרות של הקצאת דפים דינמית מכיוון שהקצאה קבועה לא הגיונית בגישה זו. קלה ליישום, ביצועים טובים אם יש מסגרות פנויות; כאשר אין מסגרות פנויות ההחלפה עלולה לפגוע בקבוצת העבודה של תהליך (אחר) שלא הוא זה שגרם לפסיקת הדף.

PFF (Page Fault Frequency): אלג' לניהול כמות הקצאת המסגרות; מודדים את תדירות פסיקות הדף של כל תהליך, במידה יש לו הרבה פסיקות = יש להוסיף לו דפים, במידה וכמעט אין פסיקות = יש לו יותר מדי דפים.

קביעת רמת מקביליות: אם גודל קבוצת העבודה של תהליכים הנמצאים במערכת > גודל הזיכרון שניתן להקצות, ניתן להקטין את רמת המקביליות (הוצאת תהליך לדיסק תפנה מסגרות לשימוש תהליכים אחרים); בחירת תהליך להוצאה: עדיפות נמוכה, גורם לכמות גדולה של פסיקות דף, תהליך שעבורו הוקצה מספר הכי נמוך / גבוה של מסגרות;

קביעת גודל דף בזיכרון וירטואלי:

יתרונות בדפים קטנים: פחות בזבוז בדף האחרון (ריסוק פנימי), חלוקה לכמות גדולה יותר של חלקים.

יתרונות בדפים גדולים: מקטין את גודל טבלת הדפים, העברת דף לדיסק או חזרה יעילה יותר כי יש חסכון בתקורה של חיפוש מיקום בדיסק.

שד מדפדף: תהליך רקע הדואג להחזיק מסגרות פנויות, מתעורר כל כמה זמן ובודק אם אחוז המסגרות הפנויות נמוך, מתחיל לבחור דפים לפינוי ע"פ מדיניות הפינוי; ניתן לממש זאת ע"י שעון עם שני מחוגים, המחוג הראשון מסמן את הדף כמיועד לפינוי, אם עד הגעת המחוג השני הדף עדיין מסומן כמיועד לפינוי, הוא יפונה, אם סומן כ- valid סימן שהוא בשימוש ואין לפנותו.

מעורבות מערכת ההפעלה בתהליך הדפדוף: (מקרים בהם יש התערבות מערכת ההפעלה)

1. יצירת תהליך: מ"ה קובעת מה יהיה גודל טבלת הדפים, מוקצה מקום מיוחד בדיסק (swapping area) שאליו יועתקו דפים מפונים, אתחול טבלת התהליכים של המערכת.
2. בהחלפת תוכן של תהליך: מ"ה דואגת שה- MMU יכיל מידע רלוונטי על התהליך החדש, ביצוע prepaging
3. בזמן פסיקת דף: זיהוי הכתובת הווירטואלית שגרמה לכך, ע"פ הכתובת מחושב המקום בדיסק ב- swapping area, חיפוש מסגרת פנויה, הבאת הדף לזיכרון הראשי, שינוי תוכן המעבד לתוכן התהליך שגרם לפסיקה.
4. בסיום תהליך: שחרור משאבים ע"י מ"ה (מסגרות, טבלת דפים, swapping area, חוץ מדפים משותפים שעדיין בשימוש של תהליך אחר)

אחסון בדיסק:

1. שיטת swapping בסיסית: אזור בדיסק המיועד להחזקת דפים, מנוהל ע"י רשימה מקושרת של שטחים פנויים
2. שיטה דינאמית: שטחי זיכרון לא מוקצים מראש, כשיש צורך לאחסן דף, מתבצעת הקצאה, בשיבוץ הדף לזיכרון הראשי הדף מפונה מהדיסק; קל לניהול.

חלוקת זיכרון לסגמנטים: אחסון חלקי תהליך בזיכרון; רק חלק מהתהליך אשר חיוני לביצוע נמצא בזיכרון הראשי, השאר נמצא בדיסק; הקטעים יכולים להיות בגדלים שונים; המיפוי נמצא בטבלת קטעים; הסגמנטים יכולים להיות במקומות שונים בזיכרון להבדיל משטית המחיצות המשתנות; להבדיל משיטת הדפדוף, השיטה ידועה למתכנת והוא יכול להגדיר סגמנטים ע"פ הצורך; החלוקה גורמת לריסוק חיצוני; ביצוע ציפוף הינו יקר בגדלי זיכרון אלו;

חלוקה לסגמנטים עם דפדוף: כל הסגמנטים מחולקים לדפים בגודל זהה; הכתובת המדומה מחולקת לשלושה שדות, שדה מספר הסגמנט בטבלת הסגמנטים, שדה מספר העמוד ושדה ה- offset; השיטה פותרת את בעיית הריסוק החיצוני.

תרגיל: גודל דף 1 kb, יחידת התייחסות 1byte, מה הכתובת הפיזית המתאימה לכתובת המדומה 1097 (בסיס 10), נתון כי דף מספר 1 ממופה למסגרת מספר 6.

פתרון: בייצוג בינארי = 1001 0100 100 = 1097, עשרת הסיביות הימניות הן ה- offset, לכן נוסיף 1 משמאל לכתובת. הכתובת הפיזית הינה 1100001001001 (בייצוג עשרוני $2^{12}+2^{11}+73 =$ 6217).

תרגיל: מהי כמות הדפים שאליו מתחלק כל סגמנט כאשר נתון כי:
מרחב זיכרון מדומה של תהליך מחולק ל- 4 סגמנטים בגודל זהה, הסגמנטים לא חופפים, הכתובת המדומה הינה בעלת 32 סיביות, גודל דף הינו 4 kbyte, אורך מילת זיכרון היא 1 btye.

פתרון: גודל מרחב זיכרון מדומה 2^{32} * 1 byte = 4Gb, מכיוון שהסגמנטים אינם חופפים, הגודל של הסגמנט הוא 1 Gb (כי מחולק ל- 4 סגמנטים), לכן מספר הדפים בכל סגמנט: 1Gb / 4 kb = 2^{20} kb / 2^{2} kb = 2^{18} pages

ריבוי תוכניות להרצה: שיטה המאפשרת הימצאות בו Multiprogramming - זמנית של יותר מתהליך אחד בזיכרון.

נעילת דפים בזיכרון: נעילת דף מונעת הוצאתו מהזיכרון הפיזי. מערכת ההפעלה משתמשת באפשרות הנעילה כדי לשמור בזיכרון דפים שאליהם מועברים נתונים כדי למנוע פינוי DMA (Direct Memory Access), על ידי דף ששייך לקבוצת העבודה של תהליך או כדי למנוע פינוי דפים מסוימים ששייכים למערכת ההפעלה עצמה.

פרק 4:

שמות לקבצים – סיומת הקובץ במערכת הפעלה WIN יכול להעיד על זיהוי קובץ הרצה BAT \ EXE, לעומת זאת בלינוקס זה לא מחייב שלקובצי הרצה תהיה סיומת מיוחדת.

3 סוגי מבנים פנימיים של קובץ –

1. **קובץ המורכב מאוסף בתים** – פעולות כתיבה \ קריאה מתבצעות ביחידות של בתים, וחיפוש בקובץ דורש מעבר סדרתי על רצף הבתים. זהו מצב שבו האחריות למבנה הקובץ מוטלת על התהליך או המשתמש עצמו. מבחינת מערכת ההפעלה זהו אוסף של בתים, בדר"כ רק היישום שיצר את הקובץ יודע לפענחו. זה המצב במערכות WIN ו-UNIX.
2. **קובץ המורכב מאוסף של רשומות** – פעולת קריאה\כתיבה של קובץ באופן לוגי, מבחינת המשתמש מבוצעות על רשומות. חיפוש דורש מעבר על רצף הרשומות וחיפוש בתוכן.רשומות יכולות להיות בעלות אורך קבוע \ משתנה. הארגון הלוגי הפנימי גם כן באחריות המשתמש. יתרון – ייעול פעולת קריאה\כתיבה של הקבצים.
3. **קובץ המורכב מרשומות המאורגנות במבנה לחיפוש מהיר (עץ עם מפתח חיפוש)** – מבחינת המשתמש פעולות קריאה\כתיבה מבוצעות על רשומות. רשומות בעלות אורך קבוע \ משתנה. לאפליקציה יש שליטה רק על תוכן הרשומות ולא על מיקומן. יתרון – פעולת חיפוש בקובץ לא דורשת מעבר סדרתי על הרשומות.

טיפוסי קבצים

1. **Block special files**: קבצים המשויכים להתקני IO שפועלים ביחידות של בלוקים.כאשר פונים לדיסק הקשיח דרך ממשק רגיל של קריאה\כתיבה של קבצים, מה שקורה בעצם זה שהפעולות מבוצעות על קובץ פנימי של מערכת הקבצים ומתורגמת לפעולות IO של הדיסק הפיסי המשויך לקובץ הפנימי.מערכת ההפעלה מבצעת הטמנה של בלוקים בזיכרון מטמון הנקרא buffer block buffer\cache.
2. **Character special files**: קבצים המשויכים להתקני IO, פועלים ביחידות של תווים כמו מדפסת, מסוף, מקלדת.אין הטמנה, כל הפעולות מבוצעות לאלתר.
3. **קבצי FIFO** – משמשים לתקשורת בין תהליכים המכונים named pipes.

גישה לקבצים:

1. **גישה סדרתית** – ניתן לגשת לתוכן הקובץ לפי הסדר, החל מתחילת הקובץ. זמני גישה ארוכים (טייפ מגנטי) כתיבה מבוצעת בלוק אחרי בלוק בצורה סדרתית, לכן ניתן לדעת מיקום הבלוקים לפני שנרשמו.
2. **גישה אקראית** – ניתן לגשת לתוכן הקובץ (בתים \רשומות) לאו דווקא לפי סדר הופעתם.דיסק קשיח ביתי.

Number of links – מציין את כמות התהליכים או ספריות המשתמשים כרגע בקובץ, כל שימוש נוסף מגדיל מונה פנימי. במידה ומונה פנימי לא שווה 0, לא ניתן לסגור את הקובץ.יצירת קישור סימבולי לא מעלה את המונה, וגם לא מצריכה קיומו של הקובץ, אלא רק בעת הפניה אליו.

יצירת קישור סימבולי על תיקיות (מדריכים) –אפשרות זאת ניתנת רק למנהל המערכת מכיוון ששימוש בלתי זהיר יכול לגרום ליצירה של מסלולים מעגליים בעץ הספריות.

סידור מערכת הקבצים –

MBR(master boot record) – מסייעת באתחול של מערכת ההפעלה הנמצאת במחיצה פעילה, המחיצה שמסומנת כזו שמכילה את מ"ה שצריכה לעלות כברירת מחדל.

טבלת מחיצות partition table – טבלה המכילה מידע על חלוקת הדיסק למחיצות, בנוסף יש אינדיקציה איזו מהמחיצות היא הפעילה.

מחיצות partitions – מחיצות מהוות חלוקה של דיסק למספר דיסקים מדומים.

תסדיר של מחיצה Partition ב-UNIX –

בלוק אתחול boot block – מכיל מידע המשמש לאתחול מ"ה אם זו נמצאת במחיצה.

סופר בלוק super block – מכיל מידע של מערכת הקבצים הקיימת במחיצה.עבודה מול מערכת קבצים מצריכה שהסופר בלוק יהיה טעון בזיכרון הראשי מכיוון שהוא מכיל את המידע על ארגון הקבצים הקיימים במחיצה.

בלוקים של דיסק המכילים מבני נתונים המשמשים לניהול שטחים פנויים במחיצה – מכילים מבני נתונים המשקפים את המצב של הבלוקים הפנויים במחיצה.

בלוקים של דיסק המכילים טבלאות i-nodes – מכילים טבלאות שמשקפות את מצב הקצאת הבלוקים של דיסק לטובת קבצים וספריות, כל אחד מהבלוקים המיועדים ל i-nodes מכיל מספר קבוע של טבלאות i-nodes.

בלוקים של ספרית השורש – בלוקים אלו מכילים רשומות המציינות את הקבצים ואת הספריות שספרית השורש מכילה.

בלוקים שיכולים לשמש להקצאה לקבצים וספריות – יכולים להיות תפוסים או פנויים, מצבם של הבלוקים התפוסים מצוין ע"י טבלאות i-nodes ומצבם של הבלוקים הפנויים רשום במבני נתונים המשמש לניהול בלוקים פנויים.

חשוב: מבחינת שיקולים הקשורים ליעילות הגישה לנתונים, לא תמיד נכון לשים מידע ניהולי רחוק מהבלוקים המנוהלים.ובגלל זה, במחיצות גדולות הבלוקים המכילים טבלאות i-nodes פזורים בין הבלוקים של המחיצה, ולאו דווקא ממוקמים ברצף בהתחלת המחיצה.כנ"ל גם לגבי הבלוקים שמכילים מידע על מצב הבלוקים הפנויים.

יישום קבצים-

הקצאה רציפה – הקצאה רציפה של תוכן הקובץ בדיסק, הסרה של בלוק מסוים מתוך הקובץ יוצר חור בגודל הזה.

יתרונות: הקובץ וגודלו ידוע.

חסרונות:ריסוק חיצוני גבוה, אין אפשרות גדילה לקובץ מעבר לגודל ההתחלתי.

רשימה משורשרת – בתחילת כל בלוק יש מצביע לבלוק הבא.

יתרון: לא דורש רציפות בזיכרון.

חסרון: אין גישה אקראית לבלוק N, וגודל בלוק לא חייב להיות בחזקת

רשימה משורשרת הממומשת בעזרת אינדקס בזיכרון הראשי – להבדיל מהשיטה הקודמת, המידע על קישור הבלוקים של הקובץ נשמר בטבלה הכוללת מידע על קישוריות הבלוקים של כל קבצי המערכת.רשומות של ספריה מצביעות רק לבלוק ההתחלתי של הקובץ בטבלת FAT.

I-NODES – זוהי השיטה המשמשת במערכות הפעלה דמויות UNIX, ספריות של מערכות קבצים מכילות רשומות המורכבות משם קובץ ומספר INODE.לפי מספר INODE ניתן להגיע לטבלת INODE עצמה המאגדת אינפורמציה על הקובץ ועל הקצאת הבלוקים שלו.

חשוב לציין, שגישה לקבצים גדולים דורשת כמות גדולה של פניות לדיסק מכיוון שצריך להביא לזיכרון בלוקים פנימיים שמכילים כתובות של בלוקים אחרים.יש להתפלגות הבלוקים של מערכת קבצים השפעה על הביצועים.כדי לחסוך בכמות הגישות לדיסק נשתמש בהטמנה.

יישום מדריכים – השמות של קובצי המידע המצביע על אופן הקצאת הבלוקים, נמצאים לרוב בספריות. שיטה אחת היא שתיקייה היא בעצם אוסף רשומות שמכילות את כל המידע הרלוונטי.

שיטה שנייה היא תיקייה שכל רשומה שלה מפנה ל-INODE. קיימות 2 שיטות לתמיכה בשמות קבצים ארוכים על ידי התיקיות.שיטה אחת היא שרשומה של מדריך היא רשומה באורך משתנה ומכילות את שם הקובץ.שיטה שנייה, רשומות התיקייה היא באורך קבוע והיא מכילה מצביע לשם הקובץ.

שיתוף קבצים בין משתמשים שונים – לכל תהליך קיימת טבלת דפים פתוחים שמכילה מצביע לטבלת הדפים הפתוחים של מערכת ההפעלה כולה.אם שני תהליכים פתחו את אותו קובץ, בטבלת הדפים הפתוחים של מערכת ההפעלה יופיעו 2 מופעים של אותו קובץ.בתוך רשומות אלו מצוין בדיוק באיזה מיקום נמצא התהליך בקובץ ע"י שדה ה-offset.כמו כן, בגלל שהקובץ פתוח פעמיים link count=2.

ניהול שטח דיסק – שטח הדיסק מחולק לבלוקים, מערכת קבצים יכולה להשתמש בגודל בלוק פיסי או לממש חלוקה בגודל שונה מעל החלוקה הפיסית. ככל שהבלוקים גדלים, כך גדל קצב העברת הנתונים.מצד שני, בלוקים גדולים גורמים לריסוק פנימי גדול יותר מבלוקים קטנים. 2 שיטות לניהול הבלוקים הפנויים שמערכת קבצים יכולה להקצותם לקבצים, ספריות או לצרכים פנימיים של ניהול. שיטה אחת ניהול בלוקים פנויים בעזרת מפת סיביות, שיטה אחרת היא ניהול בלוקים פנויים בעזרת רשימה מקושרת של בלוקים המכילים את רשימת הבלוקים הפנויים.

גיבוי מערכת הקבצים – גיבוי ברמה פיסית physical dump, עובר על כל הבלוקים של הדיסק ומבצע גיבוי של הנתונים שהשתנו ללא קשר למבנה של הנתונים, כלומר לרצף דפים גולמי.לעומת זאת, גיבוי הלוגי logical dump מתבצע בהתאם לשינויים שבוצעו במערכת הקבצים, כך מתבצע מעבר על עץ הקבצים. בגיבוי לוגי לא נשמרים הקבצים הפנימיים של מערכת הקבצים כמו FIFO ו-sockets ורשימת הבלוקים הפנויים.

עקביות מערכת הקבצים- מקרים בהם יש חוסר עקביות: בלוק חסר – לא מופיע ברשימת הבלוקים הפנויים אך לא מוקצה לאף קובץ(תיקון המצב – להוסיף אותו לרשימת בלוקים פנויים). בלוק חופשי מיותר – מופיע ברשימת בלוקים חופשיים יותר מפעם אחת. בלוק עם הקצאה עודפת – בלוק מוקצה ליותר מקובץ אחד.בלוק חופשי ומוקצה – בלוק המופיע גם ברשימת הפנויים וגם ברשימת המוקצים(תיקון המצב – להוציא את הבלוק מרשימת הבלוקים הפנויים)

זיכרון מטמון Block Cache\ Buffer Cache – בגלל הפער העצום בין מהירות גישה לדיסק למהירות עיבוד נתונים בזיכרון הראשי. הגישה לדיסק יכולה להיות צוואר הבקבוק של המערכת כולה.יש צורך ממשי לצמצם גישות לדיסק וזאת נעשה ע"י זיכרון מטמון.מערכת ההפעלה שומרת בזיכרון הראשי מסגרות המשמשות את מערכת ההפעלה לצורך הטמנה של בלוקים של דיסק.כל בלוק בדיסק ממופה ע"י פונקצית ערבול לרשימת בלוקים המנוהלים ע"י אלגוריתם LRU. בניגוד לניהול זיכרון במערכת קבצים, שבה יש חשיבות קריטית של זמן גישה לנתונים אין לנו בעיה להשתמש ב-LRU בעבודה עם דיסק מכיוון זמן הבאת בלוק ארוך הרבה יותר מאשר חישוב LRU. השימוש ב-LRU בצורתו הטהורה לא מתאים מכיוון שבצמצום נרחב מידי של פניות לדיסק נפגע עקרון מהימנות המידע. קבצים פתוחים הנמצאים בשימוש יכולים להתעדכן באופן רציף וכתוצאה משימוש ב-LRU יעבור זמן רב בין שמירות לדיסק. 2 צורות מימוש של זיכרון המטמון לבלוקים: Write-Through cache – מימוש שבו כל בלוק ששונה בזיכרון נכתב מיד אל הדיסק. Non-write–through cache הבלוקים לא נשמרים מיד לאחר השינוי, אלא במקרה שבלוק מפונה מזיכרון המטמון, או כאשר תהליך מיוחד שרץ ברקע שומר את כל הבלוקים ששונו, או כאשר המשתמש מבקש במפורש,

הבאת בלוקים מראש – בדיוק כמו במקרה של prepaging, ניתן גם כאן להביא בלוקים של דיסק מראש.

צמצום התנועות של זרוע הדיסק ע"י פיזור של נתוני שירות של מערכת קבצים – כדאי למקם בלוקים פנימיים של מערכת הקבצים כגון טבלאות INODE קרוב ככל האפשר לקבצים שהן מייצגות.צורת מיקום חכמה חוסכת מזרוע הדיסק לעבור פעמים רבות בין המסלולים של נתוני שירות ונתוני משתמש.

קובץ ממופה לזכרון – קובץ ממוקם זמנית בזיכרון הראשי, דבר המאפשר לגשת אל תוכנו באמצעות גישות זיכרון ולא בעזרת פונקציות גישה לקובץ.

פרק 5 – קלט / פלט:

מערכת ההפעלה מספקת ממש אחיד להתקני IO הנבדלים זה מזה.

קטגוריות של התקני IO:

1. התקן בלוקים: המידע מאוחסן בבלוקים בגודל קבוע; התייחסות ע"פ כתובת בלוק; HD ו- CD.
2. התקנים תוויים: פועלים על רצפים של תווים; לא ניתן לבצע פעולות חיפוש על תו מסוים; מדפסות, עכבר, כרטיסי רשת וכו'.

בקרים: הבקר מציג למ"ה ממשק לניהול פעולות IO האפשריות בהתקן זה;

שיטות מיעון של התקני IO: מתן פקודה מתבצעת ע"י כתיבה לאוגרים של בדר או לחוצצים מיוחדים המסוגלים להכיל את הנתונים; ניתן לבצע את המיעון בשלוש שיטות:

1. לבקר קיים מרחב כתובות נפרד מהזיכרון הראשי
2. הבקר משתמש במרחב כתובות משותף עם הזיכרון memory mapped I/O
   יתרונות: המיפוי המשותף מאשפר התייחסות לאוגרים כאל משתנים בשפת התכנות, מנגנון הגנה של תהליכים מסוגל להגן על גישה לרכיב שבו הוא משתמש מפני תהליכים אחרים, התייחסות לכתובות בזיכרון ולהתקן באותה הוראה בשפת סף.
   חסרונות: החומרה צריכה לאפשר ביטול caching, כל פנייה לזיכרון גורמת לתגובה גם של רכיב ה- IO כדי שהוא יוכל לדעת האם הפנייה הייתה אליו (יושבים על אותו האפיק).
3. שיטה מעורבת, חוצצים ממופים למרחב הזיכרון הראשי, הכתובות של אוגרי הבקרה ממופים למרחב נפרד.

גישה ישירה לזיכרון (DMA): בנוסף למיפוי הכתובות, יש צורך להעביר את הנתונים מהחוצצים לזיכרון הראשי לצורך שימוש התהליך, ההעתקות דורשות זמן CPU אלא אם בקר ה- IO תומך באפשרות של גישה ישירה לזיכרון; בקר כזה מקבל הוראה לבצע פעולת IO, ההוראה כוללת כתובת בזיכרון הראשי שאליה או ממנה צריך להעתיק נתונים, המבקר מבצע את ההוראה ללא CPU.

ה- DMA יודע להעביר נתונים בשיטות:

Word-at-a-time mode: העברת מילות זיכרון בזו אחר זו, לכל העברה הבקר גונב מחזור CPU אחד וגורם רק להאטה קלה (כדי למנוע תקורה של העברת כמויות גדולות).

Block mode: הבקר מעביר בלוק המורכב ממספר מילים. שיטה יעילה יותר מכיוון שנעילת האפיק מתבצעת רק פעם אחת, אך יכולה לגרום ל- CPU להמתין זמן רב יותר לנתונים מהזיכרון.

מטרות תוכנת IO:

- **אי-תלות בהתקנים (Device Independence):** מ"ה צריכה לספק ממשק אחיד ולהסתיר את ההבדלים בין ההתקנים השונים.
- **כינוי אחד (uniform naming):** כמו שם הנתיב במערכת הקבצים
- **טיפול בשגיאות:** למשל בדיקת בקר הדיסק האם הנתונים שהתקבלו מהדיסק תקינים, אם יש בעיה הבקר יכול לתזמן קריאה נוספת, אם יש עדיין בעיה ולא ניתן לפתור זאת ע"י החומרה, ניתן להעביר את השגיאה לתוכנת מתאם ההתקן.
- **העברת נתונים בצורה סינכרונית:** הדפסה למשל
- **Buffering:** לצורך העברה עתידית לדיסק
- **שיתוף אם אפשרי או להיפך שימוש בלעדי בלבד**

שכבות תוכנת ה- IO:

1. Interrupt Handler: מ"ה אחראית על הסתרה של פסיקות חומרה, לצורך הגנה על החומרה מפני המשתמש.
2. Device Drivers: מתאם התקן הוא תכנית שמספקת ממשק להפעלת התקן פיזי, ההפעלה דורשת גישה לאוגרים ולחוצצים של בקר ההתקן ולכן המתאם הינו חלק מגרעין המערכת. (מסופק ע"י יצרן ההתקן)
3. תוכנת IO המשוחררת מאילוצי חומרה: קביעת ממשק אחיד לשכבה של מתאמי התקנים, הטמנה של נתוני IO בחוצצי ביניים, טיפול בשגיאות המדווחות ע"י מתאמי ההתקן, הקצאה ושחרור של התקנים שלא ניתנים לשיתוף, לספק גודל בלוק אחיד.
4. תוכנת IO ברמת תהליכי משתמש: ספריות סטנדרטיות של IO (למשל stdlib המכילה את fprintf) ושדים האחראיים לריכוז פעולות IO הם ברמה של המשתמש (למשל ה-spooler של המדפסת, מדפסת מוקצית לספולר באופן בלעדי והשד מבצע בקשות של תהליכים הרוצים לשלוח נתונים להדפסה).

דיסקים מגנטיים: מורכב מתקליטים מסתובבים, כל תקליט מחולק למסלולים (tracks), כל מסלול מחולק לגזרות (sectors). אוסף המסלולים במרחק זהה מהציר נקרא צילינדר.

זמן העברת נתונים מושפע מזמן חיפוש המסלול (seek time), זמן סיבוב (rotation speed), (בממוצע חיפוש גזרה לוקח כמחצית סיבוב שלם), זמן העברת נתונים.

RAID: מערך דיסקים זה הוא שיטה לאחסון נתונים במספר דיסקים במקביל להגברת שרידות הנתונים והגדלת הקצבים, נראה כדיסק לוגי יחיד.

RAID L-0: הנתונים פזורים על פני דיסקים של מערך בפיסות בגודל קבוע על דיסקים שונים.
יתרון: גישה לנתונים מתורגמת למספר גישות מקביליות; היעילות תלויה בפיזור; ככל שגודל הפיסות קטן כך גדל קצב העברת הנתונים; השיטה לא תורמת לשרידות הדיסקים אלא מקטינה אותה, כשל באחד הדיסקים יגרום לאיבוד המידע;

RAID L-1: מכפילה את כמות הדיסקים יחסית ל- L-0, משתמשת בהם לגיבוי; הגדלת היעילות פי 2; שרידות גבוהה מאוד - בעת כשל, ניתן לעבוד מה- mirror; עלות שכפול דיסקים יקרה.

RAID L-2: כל 4 סיביות של נתונים מפוזרות על פני 4 דיסקים, 3 הדיסקים הנותרים משמשים לשמירה של קוד לתיקון שגיאות; ניתן לתקן סיבית אחת במקרה של בעיה; ניתן לגלות בעיה בשתי סיביות; התיקון מתבצע on-line ע"י הבקר; קצב העברה גבוה בגלל השימוש בפיסות מידע קטנות; בזבזני במונחי חומרה.

RAID L-3: ארגון דומה ל- L-2 אך למידע הנוסף משתמשים בדיסק אחד (Parity bit), במידה ויש בעיה, יודעים באיזה דיסק הייתה הבעיה וניתן לתקן זאת ע"י הדיסקים הנותרים; יעיל כמו L-2.

RAID L-4: שילוב של L-0 & L-3, יש דיסק של parity bit והמידע מחולק לפיסות קטנות; רמת שרידות זהה ל- L-3; איטי יותר מ- L-3;

RAID L-5: כדי לבזר עומסים ביחס ל- L-4 ה- parity strips מפוזרים על פני כל הדיסקים.

RAID L-6: כמו L-5 רק שיש שני דיסקים עודפים במקום 1; parity strips מחושבים פעמיים;

תזמון זרוע הדיסק:

- תור FCFS: מסלול ע"פ סדר הופעת הבקשות; הוגן ביותר; מונע הרעבה אם יהיה טיפול בפיסות מידע קטנות בכל פעם;
- Shortest Seek First (SSF): בוחר את המסלול הקרוב ביותר למיקום הזרוע; אם יש שתי בקשות במרחק זהה, יבחר כיוון אקראי; מדיניות חמדנית; מקטין את כמות תזוזות הזרוע; יכול לגרום להרעבה;
- מעלית (SCAN): הזרועה נעה מבפנים החוצה ומטפלת בבקשות לאחר מכן הכיוון משתנה והבקשות מבחוץ פנימה מטופלות; יוצא שיעבור פעמיים על המסלול האמצעי = קיפוח קיצוניים;
- סריקה חוזרת (Circular SCAN): כמו מעלית רק סריקה בכיוון אחד בלבד; מונע הרעבה אם יהיה טיפול בפיסות מידע קטנות בכל פעם;

פרק 6 – קיפאון:

קיפאון: קבוצת תהליכים נמצאת בקיפאון כאשר כל תהליך מצפה לאירוע הנמצא בשליטתו הבלעדית של תהליך אחר בקבוצה.

משאבים הניתנים לחלוקה – מסך (מספר חלונות), זיכרון, CPU (ע"י מקביליות מדומה)

משאבים בלעדיים – מדפסת, טבלאות, מאגר הודעות (משיכה ע"י שני תהליכים תגרום לבעיה)

קיפאון עוסק במשאבים בלעדיים בלבד.

Preemtable: משאבים הניתנים לחילוץ כפוי ללא גרימת נזק לתהליך כלשהו, אלה הם המשאבים אשר עלולים לגרום לקיפאון, מכיוון שאם ניתן לחלץ את המשאב ללא פגיעה, ניתן לצאת ממצב אשר עלול להביא לקיפאון.

ב- UNIX תהליך אשר ממתין במצב רדום עקב קריאה לפונ' מערכת יכול להתעורר בעקבות סיגנל. אם בעקבות סיגנל זה התהליך ישחרר חלק מהמשאבים, הדבר יכול לביא לסיום מצב קיפאון.

תנאים לקיפאון:

1. מניעה הדדית (משאב מוקצה לתהליך אחד לכל היותר)
2. החזקה והמתנה (תהליך המחזיק משאב יכול לדרוש משאב אחר בלי לשחרר את שלו)
3. אין חילוץ כפוי (רק תהליך המחזיק במשאב יכול לשחררו)

- תנאי מספיק לקיום קיפאון:
  מעגל המתנה (מעגל שבו כל תהליך מחזיק במשאב הנדרש על ידי תהליך הבא במעגל).

התמודדות עם קיפאון:

1. התעלמות (נניח וכמעט אין סיכוי שהוא יתרחש ולכן נתעלם מהטיפול בו)
2. גילוי והחלמה (מעת לעת מתבצע ניסיון לגילוי מצבי קיפאון, אם מתגלה הוא מטופל)
3. התחמקות (לא מונעת קיפאון לחלוטין אך מנסה להקטין את ההסתברות באמצעות הקצאת משאבים מבוקרת)
4. מניעה (שינוי מדיניות הקצאת משאבים המונעת קיפאון בכך שמונעת מקיום אחד מ- 4 התנאים ההכרחיים לקיום הקיפאון)

שיטת בת היענה: מחיר הפתרונות למניעת קיפאון הוא גבוה מאוד ולפעמים עלול להזיק יותר מאשר להועיל מכיוון שיכול להוות נטל על המערכת ולגרום לדיכוי חופש הפעולה של התהליכים.

היחלצות מקיפאון:

- החלמה ע"י חילוץ כפוי: חילוץ זה הוא התערבות שרירותית ולכן עלול לגרום נזק חמור לתהליך שוויתר עליו (למשל חילוץ מדפסת מתהליך יגרום לכך שיש צורך בהתערבות אנושית כדי להבין איפה היא נעצרה ולהמשיך הלאה).
  יש לפעול ע"פ שיקולי חשיבות התהליך על מנת לבחור לאיזה תהליך יילקח המשאב.
- היחלצות ע"י Rollback: שמירת checkpoints במהלך העבודה, כאשר מתקלה קיפאון אפשר לגלגל את התהליך שגרם לסגירת מעגל הקיפאון למצב העדכני האחרון.
  חסרונות – שמירה עלולה לדרוש הרבה זיכרון, שמירה פעם ב- X זמן תגרום לאיבוד המידע שחושב בזמן X, השיטה עלולה להביא נזק רב, אם מדובר על תהליכים המשמשים להעברת הודעות, גלגול לאחור עלול להביא לאיבוד מידע.
- היחלצות ע"י הריגת תהליכים: פעולה אלימה;
  יתרון – ההריגה גורמת לחילוץ מוחלט של המשאבים להבדיל מחילוץ כפוי המשחררת את המשאבים לתקופה קצרה;
  שיטה טובה לאחר שחילוץ כפוי או גלגול לאחור לא פתר את הבעיה והקיפאון התרחש שוב לאחר זמן קצר (תופעה בשם הצפה); תהליך שנבחר למות לא חייב להיות התהליך הקפוא, העיקר שהמשאבים ישתחררו.

התחמקות מקיפאון: הקצאת משאבים בדרך שתבטיח כי המערכת לא תיכנס לקיפאון.
ניתן לעשות זאת ע"י הקצאת מנהל משאבים לקבוצת תהליכים המשתפים ביניהם משאבים (קיפאון עדיין יכול להיגרם עקב בעיית שת"פ בין מ"ה למנהל).

מצב בטוח: אם התהליכים בו אינם קפואים ויש דרך להשלים את כל הבקשות של התהליכים, כך שכולם יסתיימו בוודאות ללא קיפאון. (מצב התחלתי שבו לאף תהליך אין משאב הינו בטוח).
מצב זה יכול להפוך למסוכן כאשר לפחות שני תהליכים מתחילים לפעול במקביל.

מצב מסוכן: יכול להיות שהתהליכים יסיימו את עבודתם ללא קיפאון אך מצב זה לא מבטיח את הסיום התקין.

מניעת קיפאון: תקיפת התנאים לקיפאון

1. תקיפת תנאי המניעה ההדדית: למשל ע"י הוספת שדון הדפסה ניתן להפוך את המדפסת לסוג של משאב הניתן לחלוקה, רק השדון יחזיק את המדפסת והבקשות יעברו דרכו.
   לא תמיד קיימת אפשרות זו, למשל לא ניתן לעשות זאת בקטע קריטי.

2. <u>תקיפת תנאי ההחזקה וההמתנה</u>: אם אין המתנה = אין קיפאון; כדי למנוע המתנה יש לספק לתהליך את כל המשאבים מראש, אך לא כל תהליך יודע מה הצרכים שלו מראש, הקצאת משאבים מראש הינה בזבזנית ואם תהליך אחד מחזיק במשאב, שאר התהליכים שוב לא יכולים להשתמש בו (פגיעה במקביליות).
   דרך נוספת היא לגרום לתהליך לפני בקשת משאב, לשחרר את כל המשאבים ואז לנסות לקחת את כולם בבת אחת (יכול לגרום לסדרה אינסופית של הקצאות ושחרורים)
3. <u>חילוץ כפוי</u>: מחירה כבד מדי. (מפורטת למעלה בסיכום)
4. <u>מניעת המתנה מעגלית</u>: קיימות מספר דרכים
   - הקצאת משאב יחיד (תהליך רשאי להחזיק רק משאב אחד) – לא מעשי
   - הקצאת משאבים בסדר קבוע (קשה למספר את כל המשאבים ולסדרם ע"פ חשיבות) – לא מעשי

<u>נעילה בשני מעברים</u>: פרוטוקול לצרכני מסדי נתונים; במעבר הראשון התהליך מנסה לנעול את הרשומות שלהן הוא זקוק בזו אחר זו, אם כל הנעילות מצליחות, במעבר השני התהליך מתחיל את העדכון ומשחרר את הנעילות. אם במעבר הראשון הנעילה נכשלה, התהליך משחרר את כל הרשומות וחוזר על השלב הראשון. גישה זהה לבקשת כל המשאבים מראש – לכן אינה מהווה פתרון סביר להתמודדות עם בעיית הקיפאון.

<u>הרעבה</u>: מצב שבו תהליך איננו יכול להמשיך בגלל המתנה ארוכה מאוד (או אינסופית) למשאב אחד או יותר. ההרעבה להבדיל מקיפאון היא מצב שבו התהליך לא זה שהביא את עצמו לכך אלא משאב חיוני נמנע ממנו שלא באשמתו (עקב מדיניות של סביבה חיצונית).
גורמים להרעבה: הקצאת משאבים במדיניות קבועה, עדיפות נמוכה לתהליך.
FIFO - היא מדיניות הקצאת משאבים אשר החסינה להרעבה.

- קיפאון בהכרח קשור למקביליות בין תהליכים להבדיל מהרעבה שיכולה להתבצע גם על תהליך אחד עקב תעדוף לא נכון.