העתק של חלק מאוסף השקפים להוראת מערכות הפעלה.

Operational Systems 01 Saul Coval - Computers Systems

ננסה להשתלט על המתרחש בתוך המחשב !!!

Operational Systems 01 Saul Coval - Computers Systems

הגדרה כללית

יחידה המנהלת חומרת המחשב,
את התהליכים, את הנתונים
ואת התקשורת עם המשתמשים
בהתאם למדיניות שנקבעת
על-ידי הארגון האחראי על
מערכת המשב.

8/18/2004 6

Operational Systems 01 Saul Coval - Computers Systems

בסיס לניהול מפעל

8/18/2004 7

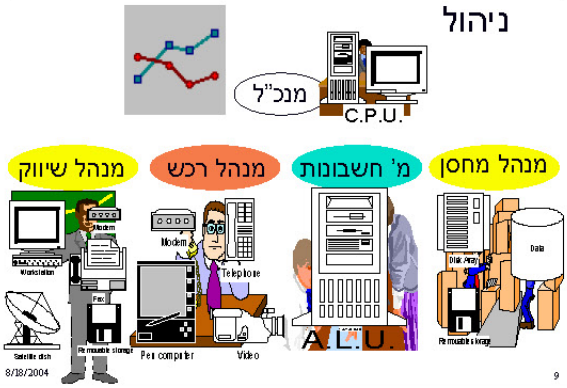
Operational Systems 01 Saul Coval - Computers Systems

ניהול המפעל

8/18/2004 8

Operational Systems 01 Saul Coval - Computers Systems

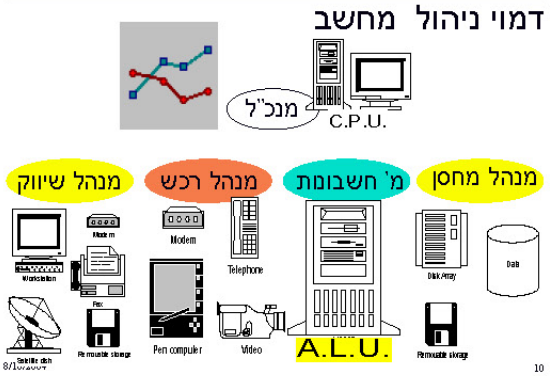
ניהול



8/18/2004 9

Operational Systems 01 Saul Coval - Computers Systems

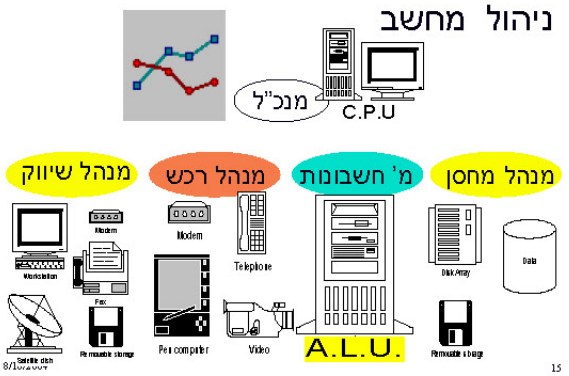
דמוי ניהול מחשב



10

Operational Systems 01 Saul Coval - Computers Systems

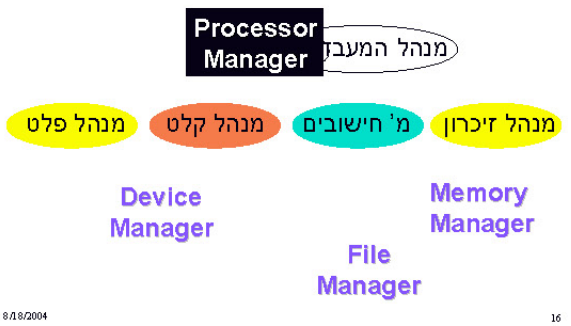
ניהול מחשב



15

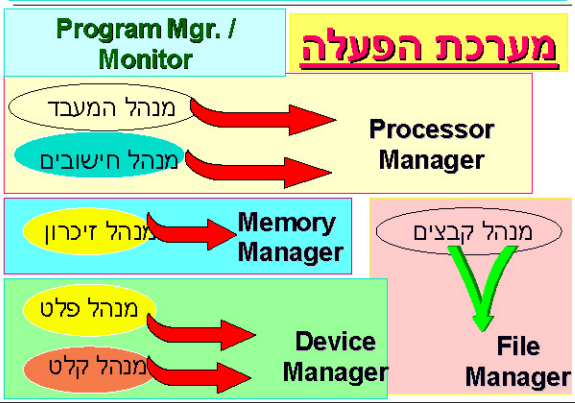
Operational Systems 01 Saul Coval - Computers Systems

ניהול מחשב מערכת הפעלה



8/18/2004 16

מערכת הפעלה



מושגים במערכות הפעלה

מערכת הפעלה (Operational System)

היא אוסף של תוכניות שנועדה לנהל הקשר בין חומרת המחשב או רשת מחשבים לבין המשתמש ותכניותיו.
היא קובעת את סדר העבודה, התקשורת,סידור הזיכרון ותוכניות שרות.היא כוללת קבוצת תוכניות שרות המבקרת את המערכת הממוחשבת.
יעדה העיקרי של מערכת הפעלה הוא למנוע את המצב המכונה "idle".

מושגים במערכות הפעלה

מצב "idle"

הוא המצב בו המעבד לא עובד בגלל פעולה אחרת של חלקים ואו ציוד היקפי של המחשב.

מושגים במערכות הפעלה

משאבי מחשב (computer resources)

הם כל ההתקנים במערכת המחשב, אשר עומדים לרשות המחשב כדי שיוכל לבצע את מטרותיו (הרצת תוכניות וכד'). דוגמאות להתקנים ,השייכים למערכת מחשב: מעבד, זיכרון פנימי ,זיכרון מטמון (cache memory), כוננים דיסקטים, דיסקים ומסך.

מושגים במערכות הפעלה

תוכנית (program)

היא אוסף של הוראות המוכרות על ידי המחשב.
ההוראות באות ברצף והן מאוחסנות בזיכרון .(הפנימי של המחשב במלאו או בחלקו).

מושגים במערכות הפעלה

משימה/עבודה (job)

,היא תוכנית יישום או מספר תוכניות יישום בצירוף הנתונים הדרושים עבורם, המוכנים לעיבוד. מכלול הדברים המתבצע עבור משתמש מסוים נקרא job המילה באה על שם קבוצת הכרטיסים שהיו מביאים בעבר להרצה במחשב.

מושגים במערכות הפעלה

תהליך (task/process)

זה תוכנית בזמן ביצוע / ריצה / פעולה.
ברגע שהתוכנית מבוצעת במחשב היא הופכת לאובייקט לוגי של מערכת ההפעלה/תהליך. תהליך יכול להיות גם פעולה כלשהי של נתונים שנועדה לבצע עבודה מוגדרת מראש.
תוכנית יכולה להיות תהליך אם היא מתבצעת תוך כדי ביצוע של תוכנית אחרת. תוכנית היא דבר סטטי שאינו פועל.
מתייחסים לתוכנית ל- job לפני הבאתה להרצה במחשב ולפני כניסתה למערכת המחשב. ברגע שתוכנית מגיע לזיכרון הפנימי של המחשב ומתחילה להתבצע, היא הופכת לתהליך.

מושגים במערכות הפעלה

תהליך (task/process) - המשך

תהליך הוא צרכן של המעבד. מערכת ההפעלה שומרת נתונים עבור כל תהליך, צרכן, או מעבד, מרגע כניסתו למערכת ועד לסיום פעולתו.
קיימים מקרים בהם מספר תהליכים יכולים לבצע אותה תוכנית. תוכניתכזו קרויה קוד שיתופי או תוכנית הבנויה (re entrant code). קוד טהור מקבוצת הוראות המכילהערכים שאינם משתנים במהלך ביצוע הקוד.

מושגים בניהול נתונים

נתונים (data)

הם עובדות גולמיות בלתי מעובדות.
הנתונים הם ייצוג המציאות כפי שהיא באה לידי ביטוי במחשב.
הנתונים הם חומר הגלם בתהליך עיבוד הנתונים.

מושגים בניהול נתונים

מידע (information)

זהו נתון שיש לו משמעות.
המידע מופק על ידי פעולה של עיבוד נתונים, וזאת על מנת שניתן יהיה לקבל החלטות ולפתור בעיות.

מושגים בניהול נתונים

עיבוד נתונים (data processing)

זהו תהליך ההופך נתון למידע.
העיבוד כולל סיווג, מיון, מיזוג, ביצוע חישובים, סיכומים וכד'.
ניתן להפעיל תהליכי עיבוד שונים על אותם נתונים לקבלת מידע שונה.

מושגים בניהול נתונים

מערכת מידע ממוחשבת (computerized information system)

הם מערכות המנהלות ומעבדות נתונים בעזרת מחשב.
מערכות המידע מכילות אוסף נתונים המאוחסנים ביחידות אחסון חיצוניות, ואוסף של תוכניות המבצעות את העיבוד ומפיקות את המידע. אחד המאפיינים החשובים של מערכות מידע ממוחשבות הוא זמן התגובה של המערכת.
מבחינים בשלוש צורות עיבוד נתונים:
(*);עיבוד אצווה (batch processing)
(*);עיבוד מקוון (on line processing)
(*).עיבוד בזמן אמת (real time processing)

מושגים בניהול נתונים

מערכות אצווה (Batch Processing)

מערכת הצוברת את האירועים ומעבדת את כולם במכלול אחד, ברצף.

מידע ⇦ עיבוד ⇦ ניקוב ⇦ טפסים ⇦ צבירה ⇦ אירוע
⇧ ⇦ נתונים

עלות העיבוד במערכת מסוג זה נמוכה מאשר בשיטת עיבוד שנזכיר בהמשך. זמן התגובה ארוך יחסית בגלל תהליך הצבירה לדוגמה, צבירת תנועות ניפוק ממחסן ועיבודם פעם בשבוע, יגרום לכך שנגלה רק כעבור שבוע שהפריט ירד מתחת לרמת מלאי מינימום. מסיבה זו למשל, עוברים במערכות ניהול מלאי ובעיקר במערכות בנקאיות לשיטות עיבוד אחרות, שבהם זמן התגובה קצר יותר.

מושגים בניהול נתונים

מערכות מקוונות (On Line Processing)

עיבוד ⇦ ⇨ ⇦ ⇨ ⇦ ⇨ אירוע
⇧ ⇦ נתונים

מערכות המגיבות בזמן קצר ביותר לכל אירוע. האירועים מוזנים למערכת העיבוד בזמן היווצרותם באמצעות מסופים ומערכת תקשורת. מערכות אלה הן יקרות בגלל התשתית הדרושה לרשת התקשורת ואחסון הנתונים (דיסקים). לדוגמה, מערכת בנקאית, משיכת כסף מהחשבון של הלקוח או הפקדה תגרום לעדכון מיידי של היתרה. מערכות אלה מורכבות יותר בגלל זמני התגובה הקצרים וריבוי המשתמשים הניגשים באותו זמן לנתונים.

מושגים בניהול נתונים

מערכות לבקרת זמן אמת (Real Time Processing)

מערכת המטפלת בכמויות קטנות של נתונים מיד עם הופעתם.
לאחר הטיפול ניתן להתעלם מהנתונים או למחקם.

מערכות מסוג זה מתמחות רק בסוג הפעלה שהוגדר להן, הן אינן גמישות והדגש בהן הוא על ביצוע מהיר ואמין של מספר קטן של תפקידים, המוגדרים היטב.

לדוגמה: מערכת להנחיית טילים, מערכת טייס, מערכת בקרה של מזל"ט.

מערכת מידע ממוחשבת (Computerized Information System)

BATCH - הקשר בין מערכות הפעלה לבין שיטות העיבוד

עיבוד אצווה היא הצורה היחידה לעיבוד נתונים שהייתה מוכרת עד שנות ה-60 לכן כל מערכות הפעלה שנכתבו אז הותאמו לשיטת עיבוד זו ומקראו Operational System.
בתחילת שנות ה-60, עם התפתחות החומרה, הפך העיבוד ל- On Line
ולכן פותחו מערכות הפעלה חדשות שלא נתונות למגבלות הישנות המערכות החדשות מבוססות על שתוף משאבי מחשב בין משתמשים שונים (שיתוף זמן) ומאפשרים עבודה בדמן אמת.
רוב מערכות ההפעלה כיום אינן כופות על המשתמש צורת עיבוד מסוימת, ויכולות לעבד תוכניות גם בצורה מכוונת וגם באצווה.

ניהול-זיכרון

הקדמה:

תהליך יכול להתבצע בלא שיוקצו לו מדפסת או כונן, אך אינו יכול בלא הקצאת זיכרון.
הנושא שנדון בו יהיה אפוא ניהול זיכרון.
מנהל הזיכרון הוא אחד מחלקי המערכת , הקובע אילו תהליכים ימתינו בתור החסומים.
כאשר תהליך מבקש זיכרון ואין די זיכרון פנוי, התהליך המבקש ימתין בתור החסומים.
שום נושא בחקר מערכות הפעלה לא זכה קרוב לוודאי, לתשומת לב כל כך כמו נושא ניהול הזיכרון.
תשומת לב זו מובנת , משום חשיבותו הגדולה של ניהול הזיכרון.
אנו נציג את נושא ניהול הזיכרון תוך כדי תיאור מהלכו ההיסטורי, מכיוון שמרבית התפתחויות בתחום זה חלו בזו אחר זו.

ניהול זיכרון במערכת של משתמש אחד:

במערכות המחשב הראשונות, עמד המחשב כולו לרשות משתמש אחד, שהריץ את כניתו.
ניהול הזיכרון לא היה אלא הקצאת הזיכרון כולו לתכנית המורצת.
"כל הזיכרון" כלל גם את הזיכרון שנתפס על-ידי מערכת ההפעלה.
צורת עבודה כזו גרמה לכך שטעות בתכנית המשתמש שקרתה במהלך הריצה גרמה להפלת מערכת ההפעלה.
במקרה כזה היה צורך לטעון מחדש את מערכת ההפעלה כדי להמשיך בעבודה, והדבר היה כרוך בבזבוז זמן רב.
היה צורך לטעון כרטיסים או סרט מגנטי ולקרוא לזיכרון מספר קטן של שגרות מערכת.
מכיוון שהיה רק משתמש אחד במערכת, לא היו תכניות שרצו בו זמנית ושפעולתן עלולה הייתה להיפגע.
קבצים נשמרו גם הם בדרך כלל על כרטיסים או על סרט מגנטי, והגישה אליהם הייתה מבוקרת באופן ידני על ידי מפעיל או על ידי המשתמש עצמו, ולא על ידי מערכת הפעלה.
המידע הניהולי שנצבר היה מועט, וגם הוא נשמר כתדפיס, שהופק באמצעות מסוף דמוי מכונת כתיבה או על גבי סרט מגנטי.
לכן נפילת מערכת ההפעלה אמנם הייתה אירוע לא נעים, אך בהחלט לא דבר נורא.

הפרדת זיכרון המשתמש מזיכרון מערכת ההפעלה:

עם התפתחותן של מערכות מחשבים התעורר הצורך לשמור על שלמותן של מערכות הפעלה. אמנם עדיין תכנית משתמש אחת בלבד הורצה במחשב, כך שלא היו תכניות שרצו בו זמנית ויכלו להיפגע ישירות.
אך תכנית אחת שהורצה יכלה להשפיע בצורה לא ישירה על משתמשים אחרים ועל מערכת ההפעלה עצמה.
למשל, קבצים אוחסנו באמצעי אחסון מקוון דוגמת תקליטים, וכך הפכו נגישים לתכנית המשתמש, בלא צורך בהתערבות מפעיל.
נפילת מערכת ההפעלה עלולה הייתה להשאיר קבצים אלו פתוחים לגישה של משתמש, שלא מורשה לגשת אליהם. כמו כן, מרכזי מחשבים רבים החלו לדרוש תשלום עבור שימוש במשאבי מחשבים, ומאחר שתשלום זה הסתמך על רישום מדויק של ניצול

משאבים שנעשה באמצעות מערכת ההפעלה, נפילת המערכת עלולה הייתה לגרום לאיבוד מידע ניהולי חשוב.
גורמים אלו ואחרים הביאו לפיתוח חומרה להגנה על מערכת ההפעלה.
פותחו כמה גישות להגנה על חלקים מהזיכרון מפני שינוי מכוון או מקרי.
ישנן כמה שיטות:

- שינוי כתובת אוטומטי, מערכת ההפעלה נטענה בחלק התחתון או בחלק העליון של הזיכרון הראשי.
- פנייה לזיכרון שונתה באופן המבטיח שלא תתבצע גישה לחלק הזיכרון במערכת ההפעלה.

למשל, במערכת עם זיכרון ראשי של 12 k , שבה מערכת ההפעלה נטענה ב- 4k העליונים, כל פנייה של המשתמש לכתובת Rבזיכרון שונתה לכתובת R ומדולו 8k.
לחלופין אם מערכת ההפעלה נטענה ב-4k התחתונים, אפשר היה להשתמש במרחב כתובות של 13 סיביות, ולהוסיף 4k לכל כתובת בעת הפנייה לזיכרון.
לשיטות אלו יש מספר רב של גרסאות, אך קיימת שיטה אחת שברצוננו להזכיר, כאשר מקצים למערכת ההפעלה מחצית הזיכרון, ולמשתמש את המחצית השנייה, אפשר להפריד בין שני החלקים על ידי הוספת 1 לפני כל כתובת שנוצרת על ידי תכנית המשתמש.
ניתן גם לשנות את תוכן אוגר הגבול כדי לשנות את גודל הזיכרון המוקצה לכל אזור.
ברור כי גישה כזו מגדילה את התקורה, מכיוון שכל גישה של המשתמש לזיכרון כורכה בביצוע פעולת השוואה שאינה מהירה כאחת השיטות שתיארנו מקודם.
לסיום סעיף זה נעיר, שאי אפשר להפריד לגמרי את תכנית המשתמש ממערכת ההפעלה.
לעתים תכנית המשתמש צריכה לתקשר עם מערכת ההפעלה, ולשם כך יש צורך להשתמש בצורה זו או אחרת של קריאה לשירותי מערכת ההפעלה.
מנגנון תקשורת זה יכול להיות קוד- פעולה מיוחד, או קוד-פעולה רגיל שנעשה בו שימוש מיוחד.
מערכת ההפעלה יכולה להיענות לבקשת המשתמש אם היא תקינה, בהתאם לפסיקה, לקוד-הפעולה, לכתובת ההוראה הפונה ולכתובת היעד, אך אם הבקשה אינה תקינה, מערכת ההפעלה יכולה להפסיק את תכנית המשתמש.

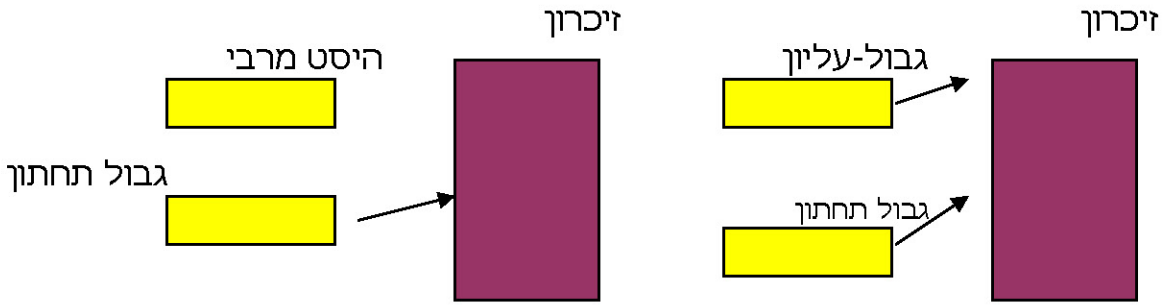
מערכת מרובת משתמשים:

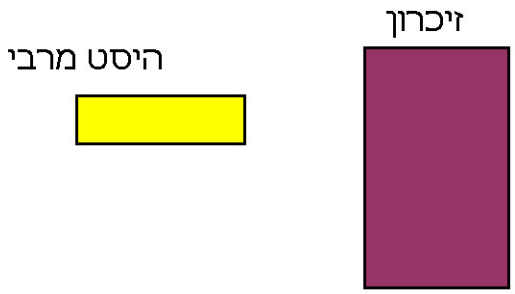
ניהול הזיכרון קיבל מימד חדש עם פיתוחן של מערכות מרובות משתמשים.
במערכות אלו היה צורך לקבל החלטות לגבי אופן חלוקת הזיכרון.
אופן ההקצאה של האזורים המתקבלים מהחלוקה, ומנגנון ההגנה על כל אחד מתהליכי המשתמש מפני התהליכים האחרים.
בטרם נמשיך, נבהיר את המונחים שנשתמש בהם.

- יש מחברים הקוראים לקטע של הזיכרון מחיצה (partition).
- אנו מעדיפים לקרוא לקטע זיכרון אזור(region), ולמקום החלוקה בין שני אזורים כאלו נקרא גבול (boundary).
- הפעולה של חלוקת הזיכרון לאזורים נקראת חציצה (partitioning).
- מושג נוסף הוא רציפות (contiguity).
- כאשר תהליך נשמר ברציפות (contiguously), הוא מאוחסן כיחידה אחת במקומות עוקבים בזיכרון. כאשר תהליך אינו נשמר ברציפות, חלק ממנו מאוחסן באזור אחד, וחלק אחר, או חלקים אחרים, מאוחסנים באזורים אחרים.

ניהול גבולות קבועים:

הרעיון הראשון שהוצע לניהול זיכרון במערכת מרובת משתמשים היה לחלק בפשטות את הזיכרון לאזורים בגדלים שונים, בעת אתחול המערכת, ולהקצות לכל משתמש אחד מהקטעים הללו.
במונחים שהצגנו לעיל, נוכל לומר שבגישה זו, מחלקים את הזיכרון לאזורים בעלי גודל קבוע, עם גבולות קבועים, ולכל משתמש מקצים אזור נפרד.
העובדה ששני גבולות קובעים את אזור המשתמש הובילה לרעיון הראשון להגנה על תהליכים.
כאשר מתבצע תהליך של משתמש מסוים, המיקומים של שני גבולות הזיכרון שלו מוכנסים לשני אוגרי גבול.
בעזרת נתונים אלה אפשר להגביל גישות רגילות לזיכרון כך שיהיו בין שני ערכי האוגרים, גישות לאזורים שמחוץ לגבולות אלו תגרום לפסיקה, הנקראת פגיעה בהגנת הזיכרון.
ניתן לשנות מעט שיטה זו, כאשר החומרה משתמשת במיעון עם היסט בכתובות שהן היסט ים.
אם מותירים היסט ים שליליים, הרי שבמקום שני אוגרי גבול אפשר להשתמש באוגר שבו מאוחסן ערך הגבול התחתון, ובערך של ההיסט המרבי.
אם מותירים רק היסט ים חיוביים, יש צורך רק בהיסט המרבי.





בדיון שערכנו עד כה בגבולות קבועים, התעלמנו משאלה חשובה:
כמה אזורי זיכרון צריכים להיות, ומה צריך להיות גודלם?
גישה אחת היא לדחות החלטה זו, ולקבוע את הגבולות ואת גודל האזורים באופן דינמי, בהתאם לתכונות התהליכים הרצים במערכת.
אולם בתקופה שאנו סוקרים, פתרון כזה נראה יקר ומסובך מדי.
גישה זו קשורה לא רק למנהל התהליכים, יידרש לטפל במספר כלשהו של תהליכים, ולא יוכל להסתמך על מספר מרבי מסוים .

יתר על כן שימוש גישה כזו יש לפתח אלגוריתם החלטה דינמי לחציצת זיכרון, שיניב תוצאות טובות באופן ניכר מאלו המתקבלות מקביעה סטטית של אזורי הזיכרון.
חלק מהמערכות מרובות המשתמשים , חלוקת הזיכרון לאזורים בוצעה בזמן קביעת הפרמטרים של המערכת.
במערכות אחרות, החלוקה בוצעה בזמן טעינת המערכת, על ידי מפעיל שהגדיר את מספר האזורים ואת גודלם.

גבולות נעים ומספר משתנה של תהליכים:

בעיית הפיצול הובילה לצעד ההגיוני הבא בניהול הזיכרון, והוא שימוש בגבולות נעים לאזורים המוקצים, כלומר הקצאת אזורים בגדלים שונים.
כדי לבחון אם מספר האזורים צריך להיות קבוע או משתנה, נתבונן בדוגמא הבאה:
נתונה מערכת עם זיכרון של 256k בתים.
במקרה של מספר אזורים קבוע, נגדיר, למשל, שמונה אזורים בזמן טעינת המערכת.
נניח כי לאחר שהמערכת פעלה במשך פרק זמן מסוים, מוקצים 192k לשבעה תהליכים, ואילו 64k נותרים ללא שימוש.
במערכת שבה מוקצים אזורים עם גבולות נעים, מתעוררת בעיה בניהול העידכון:במהלך העבודה נוצר פיצול של הזיכרון, כלומר, מתקבלים גושי זיכרון פנויים ותפוסים לסירוגין.
קיימות שלוש שיטות עיקריות לבחירת אזור שיוקצה לתהליך:
(1) הכנסת התהליך לגוש שבו שארית הזיכרון הלא מנוצלת תהיה מזערית (שיטת המתאים ביותר-best fit):
(2) הכנסת התהליך לגוש הפנוי הגדול ביותר בזיכרון, (שיטת הבלתי מתאים ביותר - worst fit):
(3) סריקה סדרתית של הגושים הפנויים בזיכרון, ובחירת הגוש הראשון שהוא גדול דיו (שיטת המתאים הראשון-first fit):
לפי הגדרתה, שיטת "המתאים ביותר" תגרום ליצירת גושים לא מנוצלים קטנים יותר מאשר שאר השיטות.
אולם, במרוצת עבודת המערכת, השימוש בשיטה זו גורם להיווצרותם של גושי זיכרון רבים, הקטנים מכדי להכיל תהליך כלשהו.
שיטת "הבלתי מתאים ביותר" משאירה בכל הקצאה את חלק הזיכרון הגדול ביותר, אך גורמת לפיזור של החלקים הלא מנוצלים של הזיכרון על פני שטחים לא רציפים, ולכן מקשה על מציאת שטח פנוי לעבודות גדולות יותר.
שיטת "המתאים ראשון" גורמת למיצוע של "המתאים ביותר" ושל "הבלתי מתאים ביותר" , בנוסף לכך היא מאופיינת בנדידה של זיכרון פנוי לאזור כתובות הזיכרון הנמוך ביותר.
יתר על כן, בשיטת "המתאים הראשון" ניתן לשמור את השטחים הפנויים בסדר כלשהו, בהסתמך על מדיניות ההחזרה של המשאבים, במקום למיינם לפי גודל, כפי שנהוג בדרך כלל בשיטות "הבלתי מתאים ביותר" ו-"המתאים ביותר".
בשלוש השיטות יש לנהל רשימה, הנקראת רשימת פנויים (free list), של הגושים הפנויים וגודלם.
במקום להשתמש בזיכרון נפרד נוסף, משתמשות מערכות רבות בזיכרון הפנוי עצמו להחזקת רשימת הפנויים.
לסיום הסעיף נדגיש שתי עובדות חשובות.

- האחת-כאשר אזור זיכרון מוחזר לרשימת הפנויים, מנהל הזיכרון אינו שם אותו ישירות במקום המתאים לו.
- השנייה-להימנע מיצירת אזורי זיכרון קטנים פנויים.

מערכת מרובת משתמשים:

במשך הזמן חלה עלייה מתמדת במספר המשתמשים במערכת אחת, וגדל קצב השינוי של תוכן אזורי הזיכרון.
עקב כך, אי יכולת להזיז תהליכים לאחר התחלת ביצועם, הפכה לצוואר הבקבוק בניצול הזיכרון.
הזיכרון התחלק לאזורים מנוצלים ולגושים פנויים שנמצאו בין האזורים המנוצלים.
בעיה זו נקראת פיצול חיצוני(external fragmentation).
ההשפעה הכוללת של בעיה זו הייתה ירידה ביעילות המערכת.
בעיה זו גרמה למתכנני מערכות הפעלה לחפש דרכים שיאפשרו להעביר תהליכים למקומות כלשהם בזיכרון.
השיטה הכללית ביותר והמעניינת ביותר מבין השיטות שהוצעו לפתרון הבעיה, הייתה שיטת אוגר ההזזה(relocation register).
ההבדל בין תהליכים שאפשר להזיז לבין תהליכים שאי אפשר להזיז טמון ביכולתו של המשתמש להגיע ישירות למקומות בזיכרון הראשי.
הרי שבשלב כלשהו יוחלף x בערך הכתובת הפיזית שלו, וביצוע ההוראה יגרום להעברת כתובת זו לאוגר 7. כתובת זו תישמר בזיכרון.
אם תהליך יועבר במועד מאוחר יותר לאזור אחר בזיכרון הכתובת כבר לא תהייה כתובת חוקית.
עתה נניח כי כאשר הדרנו את התכנית שלנו, או תרגמנו אותה באסמבלר, x לא הוחלף בכתובת פיזית, אלא הוגדר כהיסט ביחס לנקודת ההתחלה של התכנית.
כאשר תכנית נטענת, נשמרת הנקודה שבה החלה טעינת התכנית, וכאשר התכנית מבוצעת, נקודת ההתחלה מוכנסת לאוגר מיוחד הנקרא – "אוגר הזזה".
גישה זו מקילה את טעינת התכנית, מאחר שניתן להשאיר את שדות הכתובות להיסט ים, במקום להפוך אותן לכתובות פיזיות בזיכרון.
יתר על כן גישה כזו מאפשרת להזיז תהליכים ממקום למקום, מכיוון שלאחר הזזה, מערכת ההפעלה יכולה להכניס לאוגר הזזה את כתובת ההתחלה החדשה.
היכולת להעביר תהליכים ממקום למקום בזיכרון נקראת ניוד דינאמי (dynamic relocation).
הניוד מאפשר למערכת להימנע מבזבוז זיכרון כתוצאה מפיצול חיצוני.
אולם יש להיזהר, מכיוון שניוד תהליכים הוא משימה הדורשת זמן.
כאשר תעסוק בניהול זיכרון, תיתקל רבות במונח מיפוי (mapping).
מיפוי הוא המרת כתובות לוגיות בכתובות פיזיות, באמצעות שימוש בתוכן אוגר ההזזה.

גישות שונות לחלוקת זיכרון:

הבעיה המרכזית בכל השיטות לניהול זיכרון שדנו בהן עד כה היא הדרישה כי התכנית תיטען למקומות רצופים בזיכרון.
כלומר, הצורך לטעון את התכנית לאזור אחד של כתובות זיכרון עוקבות.
גישה אחרת היא לאפשר חלוקת התהליך לחלקים, וטעינת כל אחד מחלקים אלה לאזור בזיכרון הראשי.
השאלה היא כיצד אפשר לממש פתרון כזה.
היה צורך לפתח שיטה, שבה מהדרי השפה יוכלו ליצור כתובות חוקיות לכל חלק של תהליך, למרות שכל חלק אינו נמצא בזיכרון ברצף עם חלקים אחרים.
נוסף לכך, מאחר שבמקרה כזה, הזיכרון הראשי מורכב מאוסף לא רציף של חלקי תהליכים, שיטת המימוש צריכה לכלול אמצעי להגנה על מערכת ההפעלה ועל המשתמשים מפני משתמשים אחרים.

נחזור לרגע לרעיון של אוגר הזזה.
בגישה זו, הכתובות נוצרות כהיסט ים מכתובת התחלת התכנית.
מאוחר יותר, בזמן הביצוע, היסט ים אלו נלקחים כמרחק מכתובת הטעינה של התכנית.
את חלוקת התהליך לחלקים אפשר לממש בשיטה דומה, כלומר: אפשר ליצור את הכתובות בכל קטע תכנית כהיסט ים מתחילת הקטע.
אז אפשר לטעון כל חלק בכל מקום בזיכרון הראשי.
אך כיצד נממש את ההגנה מפני משתמשים אחרים? לשם כך ניתן להשתמש באוגרי גבול, אך הדבר גורם לביצוע מספר רב של פעולות נוספות.
כל כתובת A מוגדרת תחילה על ידי החלק של התהליך שבו היא נמצאת:
נניח שחלק זה נקרא P .
הדבר מחייב את המערכת להשתמש בטבלת גבולות לכל חלק, ולטעון את האוגר עם הערכים המתאימים לחלק P .
לאחר מכן יש לחבר את ההיסט של הכתובת A לנקודת ההתחלה של P.
כדי לקבל את הכתובת הסופית, שאותה יש להשוות לאוגרי הגבול.
ניתן לקבוע את גודלו של כל חלק באופן שרירותי, ולהטיל על מהדרים לחלק את תכנית המשתמש.
לכל משתמש ניצור טבלה, ובה נקודות ההתחלה של כל חלקי התכנית.
במקרה זה אין צורך באוגרי גבול, מכיוון שניסיונות, מכוונים או אקראיים לגישה מחוץ לחלק P, יגרמו לגישה לחלק אחר של אותה תכנית, Q למשל.
הדבר יקרה לפני גישה לטבלת המיפוי של התכנית, ולכן תתבצע גישה לחלק Q , ולא לאזור בזיכרון שבו נמצאת תכנית של משתמש אחר.
גישה אחרת לשילוב של טבלאות הסתכלות ובדיקה מאפשרת למשתמש להגדיר את מספר החלקים בתוכניתו ואת גודלם, ולהכניס מידע זה לטבלת המיפוי.

דיפדוף (Pager):

בדיפדוף המערכת מחלקת את תהליך המשתמש לחלקים בגודל קבוע מראש הנקראים דפים (PAGES).
במרבית המערכות, "הגודל הקבוע מראש" הוא גודל קבוע של מערכת ההפעלה.
עקרונית, גודל הדף יכול להיות גודל משתנה, אך שימוש בגודל דף משתנה מעורר שוב את הבעיות הכרוכות הניהול זיכרון ללא דיפדוף.
נניח אפוא שגודל הדף קבוע, ונבדוק כיצד קובעים אותו.
אחד הגורמים הראשונים בקביעת גודל הדף הוא בסיס המיעון של החומרה.
כך למשל, אם המחשב שלנו הוא בינארי, ונבחר דף בגודל 600, אנו יכולים להיתקל בבעיות.
ראשית, המעדר אן באסמבלר יצטרכו לבצע פעולות לוגיות נוספות כדי לקבוע אם יש צורך לעבור דף.
שנית, נצטרך להשתמש השדה היסט שגודלו 10 סיביות, אך כמעט מחצית הערכים לא ינוצלו.
לעומת זאת, אם נבחר דף בגודל 1024 בתים לאותו מחשב בינארי, לא נזדקק לחישובים נוספים כדי לקדם את מונה הדפים, ועדיין נשתמש בהיסט של 10 סיביות.
קיימים גורמים נוספים חשובים לקביעת גודל הדף.
למשל, לעתים תכופות, גודל הדף מותאם לגודל הפיסי של הרשומה בתקליט שהמערכת משתמשת בו.
ללא התאמה כזאת, היה מתבזבז מקום על התקליט, וגם זמן ההעברה היה ארוך יותר עקב קיומם של "חלקי הדפים" לא מנוצלים.

דפים גדולים יותר גורמים לבזבוז מקום, מכיוון שבדרך כלל, תכניות ונתונים אינם מתאימים בדיוק למספר דפים מסוים.
מצד שני, דפים קטנים יותר דורשים ניהול רב יותר, מכיוון שבממוצע יהיו לכל תהליך דפים רבים יותר.
משקבענו את גודל הדף במערכת, אנו צריכים להחליט כיצד יעשה המיפוי מהכתובת הלוגית של המשתמש לכתובת פיסית בזיכרון.
כפי שהזכרנו קודם לעתים קרובות נעשה הדבר באמצעות טבלה, הנקראת טבלת מיפוי דפים (PAGE MAP TABLE).
כל פנייה של המשתמש לזיכרון כרוכה בשתי פעולות זיכרון: אחת לאיתור הדף, והשנייה לביצוע קריאה\כתיבה, אם היינו מבקשים לכלול ביע"מ מספר מספיק של אוגרים להכלת הטבלה, הי ע"מ הייתה יקרה, והיה נדרש זמן רב כדי להחליף תהליכים בי ע"מ.
ניתן להגיע לפשרה על ידי שמירת חלק מהטבלה באוגרים, והחלק האחר שלה בזיכרון הראשי, או אפילו על ידי שימוש בזיכרון מטמון (CACHE) מהיר כזיכרון ביניים בין האוגרים לזיכרון הראשי.
אפשר גם להשתמש בזיכרון אסוציאטיבי–שהפנייה למיקום מסוים בו נעשית על פי התוכן, בזיכרון האסוציאטיבי, ניתן למצוא את נקודת ההתחלה של דף באמצעות מספר הדף.
שאלה נוספת הנוגעת לדיפדוף היא מדוע יש לכתוב את כל הדפים לזיכרון המשני עם סיום התהליך.
ואכן, אין הכרח לכתוב את כולם.
ניתן להוסיף בטבלה סיבית אחת לכל דף, וסיבית זו תציין אם הוכנסו שינויים בדף אם לא, לזיכרון המשני יש לכתוב רק דפים ששונו.
נוסף על כך, אין צורך לטעון לזיכרון הראשי את כל הדפים לפני תחילת הביצוע.
כתובת של הוראה או של נתונים ממופה לדף אחד בכל מחזור.
מכאן ניתן להגיע לגישה קיצונית שלפיה יש צורך לטעון דף אחד בלבד לזיכרון הראשי, כאשר יתר הדפים נמצאים הזיכרון המשני.
בדרך כלל החזקת דף אחד לתהליך אינה הגיונית, מכיוון שהיא תגרום תנועה רבה בין התקליט לזיכרון הראשי.
הגיוני יותר להחזיק רק חלק של תהליך הזיכרון.
גישה כזו קרויה בדרך כלל דיפדוף על פי דרישה(DEMAND PAGING), והיא הבסיס למערכות מרובות משתמשים.
מה היה קורה לתכנית זו אם הייתה מתבצעת במערכת מרובת משתמשים עם דיפדוף?
אם המערכת מאפשרת שלכל תהליך יימצאו בזיכרון הראשי חמישה דפים, אין בעיה.
אם יוקצו לתהליך שבעה דפים, יהי שיפור מועט, אם בכלל.
אולם, אם תהליך היה מקבל רק ארבעה דפים, הייתה נגרמת ירידה חריפה בביצועים, מכיוון ש-20% מהפניות לזיכרון היו גורמות לכשל דף (PAGE FAULT), כלומר פונות לדף שאינו נמצא בזיכרון הראשי.
קבוצת חמשת הדפים נקראת קבוצת העבודה של התהליך בפרק הזמן הנתון במהלך הביצוע.
נחזור ונדגיש כי קבוצת העבודה משתנה בהתאם לתהליך המתבצע, ובהתאם למצב של התהליך: היא תלויה בנתוני התכנית ובמספר רב של גורמים אחרים.
המושג של קבוצת עבודה הוא חשוב ביותר, מכיוון שאם מקצים לתהליך
דפים במספר יותר גדול מגודל קבוצת העבודה שלו s, גורמים בדרך כלל
לבזבוז, בעוד שהקצאת מספר דפים קטן מ-s תפגום בביצועים.
במבט ראשון, הפילוסופיה שמאחורי קבוצות עבודה נראית טובה, אך עולה השאלה אם המערכת יכולה לתחזק בצורה יעילה את מידע הפנייה לזיכרון.
גישה ישירה אחת למימוש קבוצות עבודה היא מימוש באמצעות מונים מיוחדים בחומרה.

מערכת דיפדוף על פי דרישה יכולה להשתמש ברעיון של קבוצות עבודה, ויכולה גם להשתמש באלגוריתם אחר לקביעת מספר הדפים שיוקצו לכל תהליך (למשל מספר קבוע, או חלק ממספר הדפים הפנויים).
בכל מקרה יכול להיווצר מצב, שבו תהליך זקוק לדף חדש, אך המערכת אינה יכולה להקצות זיכרון נוסף.
איזה דף יש להחליף במקרה זה?
שאלה זו הייתה ונותרה נושא למחקר.
ברור כי ניתוח תבניות הפנייה לדפים באלגוריתם החלפת הדפים הוא בעל השפעה מכרעת על ביצועי המערכת, ובמיוחד על תקורת המערכת בהחלפת דפים בזיכרון.
מערכת, שבה החלפות הדפים תכופות מדי, עלולה לפעול באיטיות רבה.
חלק מאלגוריתם להחלפת דפים שהצגנו עד כה מעוררים בעיות מיידיות.
כדי לפתור אותן ניתן להציע להחליף את הדף הותיק ביותר במערכת (זהו שימוש של אלגוריתם FIFO).
הבעיה בפתרון כזה היא, שדף יכול להיות הוותיק ביותר במערכת מכיוון שהוא מכיל שגרה חיונית המבקרת את יתר התהליך.
גם החלפת הדף האחרון שהתוסף (אלגוריתם-LIFO) אינה נראית כפתרון טוב, אם זה עתה ניגשנו לראשונה לדף כלשהו, רבים הסיכויים כי בקרוב ניגש אליו שוב.
עתה משנוכחנו לדעת כי הן FIFO והן LIFO אינם מספקים פתרונות טובים נציע פתרון אחר:
החלפת הדף שמספר הגישות אליו הוא הקטן ביותר מאז ההחלפה האחרונה.
שיטה זו נקראת הנדיר ביותר בשימוש (LEAST FREQUENTLY USED) או LFU.
זהו רעיון טוב, אך הוא דורש החזקת מונה לכל דף, ועדכונו עם כל פנייה לדף , וכן מיון ערכי המונים, כדי למצוא את הדף שיש להחליף.
דרישות מיון וחיפוש אלו חלות גם בעת שימוש בדף שבוצעה אליו פנייה לפני הזמן הרב ביותר.
שיטה זו נקראת הקודם ביותר בשימוש (LEAST RECENTLY USED) או LRU.
בהמשך נראה כי ניתן למצוא פתרון טוב לבעיית הדף המוחלף באמצעות שיטה הקרובה ל-LRU.
עד כה הנחנו כי האלגוריתם להחלפת דפים מיושמים על כל תהליך הנזקק לדף נוסף, במצב שבו אין מקום פנוי בזיכרון.
אך לא תמיד זהו המצב. למשל, במקצת המערכות המשתמשות בקבוצות עבודה, האלגוריתם להחלפת דפים מופעל על התהליך בעל הקדימות הנמוכה ביותר.
מכאן, שכל התהליכים נוטלים דפים מהתהליך בעל הקדימות הנמוכה ביותר עד שתהליך זה מאבד את כל דפיו, ומוכנס להשעיה.
עד כה דנו בשיטות הבסיסיות לדיפדוף.
עתה נדון בשני נושאים אחרים: מרחב כתובות פיסי לעומת מרחב כתובות וירטואלי, ושיתוף בקוד.
במהלך הפרק הפרדנו בין הכתובת של משתנה x, כפי שהמשתמש רואה אותה, לבין המיקום הפיסי של x בזיכרון ברגע כלשהו.
כמו כן אמרנו כי בדיפדוף על פי דרישה, אפשר להסתכל על הכתובת הלוגית של x ככתובת הווירטואלית של x .
נשאלת השאלה מה הקשר בין מספר הדפים שניתן להתייחס אליהם בזיכרון וירטואלי, v, לבין מספר הדפים הנמצאים בפועל בזיכרון הראשי.
v יכול להיות שווה ל-p, ואז מרחב הכתובות של המשתמש שווה לגודלו של הזיכרון הראשי שהוא תופס.

זה היה המצב במרבית המחשבים הגדולים הראשונים, ובמיני מחשבים מהדור הראשון.

אין פרוש הדבר שכל דפי המשתמש נמצאים כל הזמן בזיכרון, אלא שאורך כתובת בזיכרון הווירטואלי שווה לאורך כתובת בזיכרון הפיסי.
v גם יכול להיות גדול בהרבה מ - p, וזהו אכן המצב במרבית המחשבים עם אורך מילה גדול. זוהי שיטת התכנון הנפוצה כיום ליע"מ.
שיטה זו מאפשרת למשתמש להתייחס לזיכרון גדול בהרבה מזה שיש במחשב למעשה, גם אילו היה המשתמש עובד על המחשב לבדו.
שיטה זו נוחה, למשל, לטיפול באזורי נתונים גדולים כבמערך אחד, או לכתיבת תכניות גדולות מאוד בלא צורך לחלקו לחלקים אחדים.
חשיבותו של הזיכרון הווירטואלי נובעת בעיקר מיחס זה שבין p ל-v.
לבסוף, v יכול להיות קטן מ-p .
למשל, במיני מחשב או מיקרו מחשב עם מרחב כתובות וירטואלית של 16 סיביות וזיכרון ראשי בגודל 1 מגה-בית.
הנושא השני הוא שיתוף קטעי קוד או שיתוף נתונים במערכת דיפדוף לא תמיד פשוט כל כך. לדוגמה, נניח כי כל אחת מהתכניות מכילות מערך ששמו BOOK וגודלו 10.240 בתים. הם רוצים להריץ את התכניות בו זמנית, ולשתף את התכניות באמצעות מיפוי שני המערכים לאותן כתובת בזיכרון.
הדבר נראה פשוט, מכיוון שהם יכולים להשתמש בשתי כניסות בטבלת המיפוי המצביעות על אותו דף פיסי.
הבעיה היא שהכתובת הפיסי של הדף נקבעת על פי המקום היחסי של התכנית.
מכאן, שהתכנית עשויה לטעון את המערך החל בדף 5 היסט 200, ועד דף 15 היסט 199.
לעומת זאת, בתכנית המערך עשוי להיטען החל מדף 11 היסט 0, ועד דף 20 היסט 1023.
הדבר מדגים את הקושי במיפוי הפניות לאיברי המערך, לאותו מקום שאליו נעשות הפניות לאיברים אלה.

קיטוע:

דיפדוף הוא טכניקה חשובה ושימושית, אך יש לו כמה חסרונות.
באופן כללי, חסרונות אלו מקורם בחלוקה הפיסית השרירותית של התהליך על ידי מערכת ההפעלה, במקום חלוקה לוגית על ידי המתכנן שלו.
אנו נראה שני מקרים שבהם חלוקה כזו יכולה לגרום בעיות.
ראשית יש בעיה באכיפה של מגבלות גישה.
נניח כי תכננו מערכת עבור מחלקות אחרות בחברה שאנו עובדים בה, ואנו רוצים שהגישה לחלק מאזורי הנתונים תהיה בלתי מוגבלת, בעוד שהגישה לאזורים אחרים תהיה לקריאה בלבד. קשה לבצע זאת בדיפדוף.
עלינו לחלק את אזורי הנתונים ולהבטיח כי הם יועברו לדפים עם הגבלות הגישה המתאימות.
שנית, איננו מנצלים גורם שיכול להיות משמעותי לגבי יעילות המערכת: הידע שיש למשתמש על מבנה התכנית שלו ועל ביצועיה.
בדיפדוף אנו משתמשים בדרך כלל אך ורק באלגוריתם החלפת דפים קבוע מראש להסקת מסקנות בנושאים אלו.
כדי לענות על בעיות אלו, נחזור לשיטה אחרת קיטוע (SEGMENTATION).
ההבדל העיקרי בין קיטוע לבין דיפדוף הוא זה: בקיטוע, המשתמש מחלק את התכנית לחלקים לוגיים במקום שהמערכת תחלק אותה לחלקים פיסיים שרירותיים.
למשל, המשתמש יכול להגדיר קטעי קוד בשם INPUT, SORT ו- ANALYZE, וקטעי נתונים בשם – PREVIOUS ו- TRANSACTIONS .
הכתובות הסמליות בקיטוע מורכבות משם הקטע ומתווית של ההוראה או שם של משתנה באותו קטע.

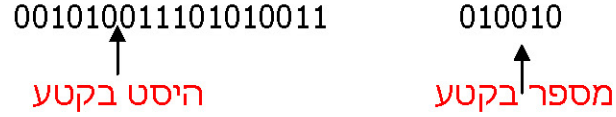
אלו מתורגמים בדרך כלל למספר קטע והיסט בתוך הקטע.
מובן שינן גם מוסכמות, המקילות על המשתמש.
אם שום קטע אינו מוגדר, מניחים כי הכוונה לקטע הקוד ולקטע הנתונים הנוכחיים.
גם במקרה זה אפשר לחשוב על טבלה למימוש המיפוי.
במקום ה-PMT המכילה את מקומות ההתחלה של הדפים, אפשר להשתמש בטבלת מיפוי קטעים.
שדה ההיסט המרבי נכלל בכניסה של SMT, משום שבניגוד לדפים, הקטעים יכולים להיות בעלי גודל שונה.
בסעיף הקודם דנו באופן שבו הדיפדוף מקל את השימוש בקוד משותף עם אזורי נתונים אישיים.
תכונה חיובית זו יש גם לקיטוע.
לשני משתמשים (או יותר) יכולות להיות כניסות ב-SMT המצביעות על מהדר אחד, הנמצא פיסית במקום אחד בזיכרון.
לכל משתמש יכול להיות אזור נתונים משלו, המכיל את קוד המקור (SOURCE) ואת קוד המטרה (OBJECT): נקרא להם בפשטות מקור ומטרה.
כאשר משתמשים במהדר, ה-SMT שלו פעיל. מכיוון שהמיפוי נעשה באמצעות ה-SMT, בכל פעם שהמהדר יפנה למקור ולמטרה הוא יפנה לעותק הפרטי של אזורים אלו.
כאשר מסתיים פלח הזמן והתהליך מתחיל להתבצע, הוא ישתמש באותו עותק של מהדר.
אך הוא יפנה לנתונים שלו דרך ה-SMT שלו.
למעשה ניתן להשתמש בקיטוע לשיתוף נתונים גם ברמה נמוכה יותר.
בעבודה עם קיטוע אפשר להגדיר קטע בשם BOOK, שאליו יהיה מצביע בכל SMT, לאותו מיקום פיסי.
כאשר BOOK מוגדר כקטע עצמאי, אין צורך לדאוג שההיסט בדפים יהיה זהה.
יתירה מזו, ניתן לשמור SMT כללי של מערכת, ובו מידע על כל הקטעים הפעילים בכל זמן, מי המשתמש בהם, ואילו מהם אינם מצויים בזיכרון הראשי.
לסיום אנו רוצים להדגיש את ההבדל בין קיטוע לדיפדוף במונחים של העברה לזיכרון (של הקטע או של הדף). בגרסה זו של קיטוע אנו עוסקים באזורים בעלי גודל שונה, מכיוון שהקטעים הם בגודל שונה.
לכן ייתכן שנצטרך לחזור לסעיפים הראשונים בפרק, כדי לבדוק אלגוריתמים כמו "המתאים הראשון" או "המתאים ביותר". אנו אומרים כי ייתכן שנצטרך לעשות זאת, מכיוון שישנו הבדל שאותו נציג מייד.

קיטוע משולב בדיפדוף:

קטעים הם ישויות לוגיות נפרדות.
עובדה זו מאפשרת לבצע את השינוי שעליו רמזנו בסוף הסעיף הקודם, והוא שימוש בדיפדוף על כל קטע.
נציג דוגמה שתמחיש הצעה זו, ונראה כי כאשר עוקבים אחר מיפוי צעד אחר צעד, קל להבין כיצד הוא מתבצע.
נתבונן לדוגמה בהוראה:

MOVE 0,BOOK/AUTHORS;

ההוראה מתייחסת למקום AUTHORS בקטע BOOK.
ההוראה תתורגם למספר קטע ולהיסט בתוך הקטע:



כאשר מתבצעת פנייה לכתובת זו, המערכת פונה לטבלת מיפוי הקטעים.
לאחר בדיקת זכויות הגישה, היא עוברת לשדה הבא, שבגרסתנו הקודמת לקיטוע קראנו לו "מיקום".
אלא שכאן, המיקום אינו כתובת תחילת הקטע, כי אם התייחסות לטבלת מיפוי הדפים (PMT) של אותו קטע.
לאחר שהמערכת בודקת את ה-PMT, היא מפעילה את שיטות מיפוי הדפים, כמתואר בסעיף. כך אנו נהנים מיתרונות הקיטוע, ונמנעים מעיסוק בניהול הזיכרון של אזורים בעלי גודל שונה.
המתנגדים לשיטה יכולים לטעון כנגד התקורה הגדולה, הכרוכה בתהליך המיפוי המורכב, ואכן תהליך זה אינו פשוט.
לכן, במערכות המשתמשות בקיטוע עם דיפדוף, חלק גדול מהמיפוי מתבצע ישירות באמצעות החומרה.

סיכום:

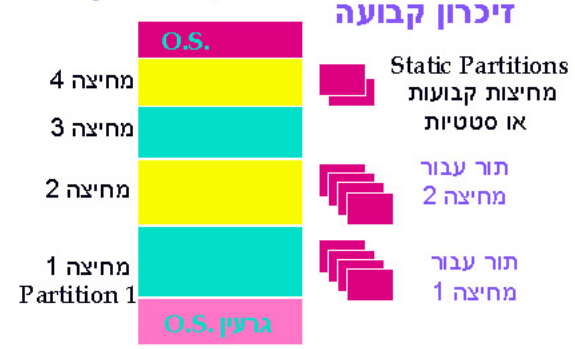
ניהול זיכרון הוא נושא חשוב ומעניין.
זהו פרק ארוך, ולמרות זאת נגענו רק במקצת הנושאים, ותיארנו גישות כלליות בלא להיכנס לפרטי האפשרויות השונות.
יתירה מזאת, לחלק מהשאלות לא מצאנו תשובה מלאה: הסיבה לכך היא, שמתכנני מערכות הפעלה עדיין אינם יודעים את כל התשובות לבעיות אלו.
ניהול זיכרון הוא דוגמה טובה למקרה שבו המימוש קרוב יותר לאמנות מאשר למדע, ואשר יש בו הזדמנויות רבות למחקר נוסף.
המחשב ATLAS היה מוקד למחקר במשך 25 שנים.
היבט מעניין במיוחד במחשב זה הוא מערכת הזיכרון הווירטואלי שלו.
הזכרנו את השימוש בזיכרון מטמון במערכות לניהול זיכרון.
קיימים ניתוחים מתמטים רבים ומחקרים ניסויים בנושא מערכות לניהול זיכרון.

הסברים מתוך עבודה של התלמיד אופיר מרחבי.

העתק של חלק מאוסף שקפי המורה שאול קובל להוראת **ניהול זיכרון.**

Operational Systems 03 *Saul Coval - Computers Systems*

מערכת רבת-תוכניות בעלת חלוקת זיכרון קבועה



Operational Systems 03 *Saul Coval - Computers Systems*

Static Partitions מחיצות קבועות

היא השיטה הפשוטה ביותר לניהול זיכרון במערכת ריבוי-תכניות. מספר המחיצות וגודליהן נקבעת על-ידי המפעיל. מבנה הנתונים היחיד שזקוק לו מנהל הזיכרון הוא טבלה, שיש בה שורה לכל מחיצה.

מען	גודל	שיוך
250	100	Job 1
350	150	Job 2
500	200	free
700	300	job3

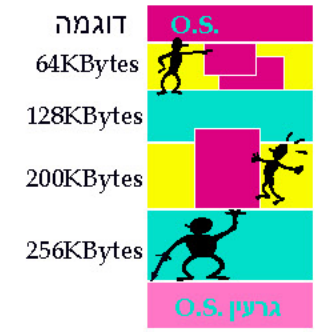
Operational Systems 03 *Saul Coval - Computers Systems*

Static Partitions חלוקת זיכרון קבועה

מנהל המערכת קובע הגודל של כל מחיצה

Operational Systems 03 *Saul Coval - Computers Systems*

Static Partitions חלוקת זיכרון קבועה



כללים

**בכל מחיצה אפשר להכניס רק תוכנית אחת

**תוכנית אחת לא תיכנס בתוך יותר ממחיצה אחת

**תוכניות שגודלן גדול מהמחיצה הגדולה ביותר לא יפעלו אף פעם

Operational Systems 03 *Saul Coval - Computers Systems*

Static Partitions חלוקת זיכרון קבועה

מדיניות לריצה או מדיניות להקצאת הזיכרון

First Fit - תואם ראשון

התוכנית הראשונה ברשימה תכנס במקום הריק הראשון, על פי הכללים הקודמים

Best Fit - התואם ביותר

התוכנית שגודלה מתאימה ביותר למקום הריק הראשון, היא זו שתיכנס ראשונה לעבודה

Worst Fit - הבלתי תואם ביותר

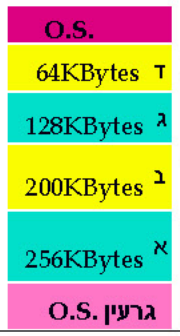
הבקשה נענית תמיד מהשטח הפנוי הגדול ביותר

לא ידוע על מערכות המשתמשות בה

Operational Systems 03 *Saul Coval - Computers Systems*

Static Partitions חלוקת זיכרון קבועה

דוגמה: ריצה לפי המדיניות



First Fit:

1	2	3	4	5
212K	312K	128K	196K	96K
4sec	2sec	6sec	5sec	3sec

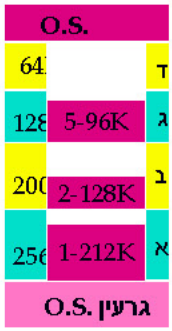
Best Fit:

1	2	3	4	5
212K	312K	128K	196K	96K
4sec	2sec	6sec	5sec	3sec

Operational Systems 03 *Saul Coval - Computers Systems*

Static Partitions חלוקת זיכרון קבועה

דוגמה: שניות ריצה 1,2,3



First Fit:

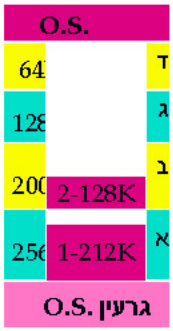
2	4
312K	196K
2sec	5sec

= שטח לא מנוצל

Operational Systems 03 *Saul Coval - Computers Systems*

Static Partitions חלוקת זיכרון קבועה

דוגמה: שניית ריצה 4



First Fit:

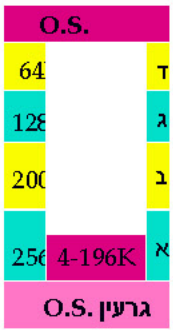
2	4
312K	196K
2sec	5sec

= שטח לא מנוצל

Operational Systems 03 *Saul Coval - Computers Systems*

חלוקת זיכרון קבועה Static Partitions

דוגמה: שניות ריצה 5,6



First Fit:

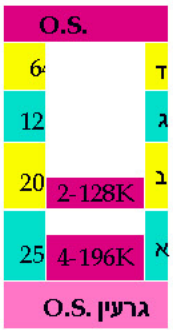
2
312K
2sec

שטח לא מנוצל =

Operational Systems 03 *Saul Coval - Computers Systems*

חלוקת זיכרון קבועה Static Partitions

דוגמה: שניות ריצה 5,6



First Fit:

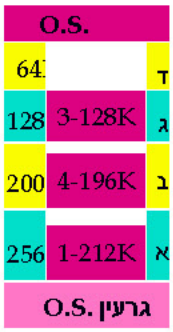
2
312K
2sec

שטח לא מנוצל =

Operational Systems 03 *Saul Coval - Computers Systems*

חלוקת זיכרון קבועה Static Partitions

דוגמה: שניות ריצה 1,2,3,4



שטח לא מנוצל =

Best Fit:

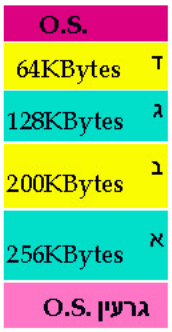
2
312K
2sec

5
96K
3sec

Operational Systems 03 *Saul Coval - Computers Systems*

חלוקת זיכרון קבועה Static Partitions

דוגמה: לאחר שניית ריצה 7



First Fit:

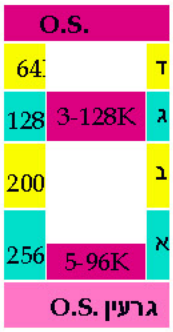
2
312K
2sec

תוכנית זו לא תופעל
אף פעם, היתר
סיימו ריצתם ב-7 שניות

Operational Systems 03 *Saul Coval - Computers Systems*

חלוקת זיכרון קבועה Static Partitions

דוגמה: שניית ריצה 6



שטח לא מנוצל =

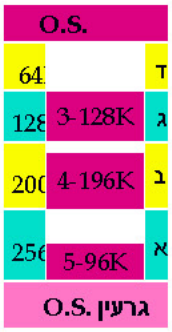
Best Fit:

2
312K
2sec

Operational Systems 03 *Saul Coval - Computers Systems*

חלוקת זיכרון קבועה Static Partitions

דוגמה: שניית ריצה 5



שטח לא מנוצל =

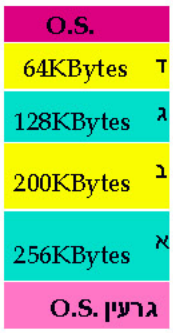
Best Fit:

2
312K
2sec

Operational Systems 03 *Saul Coval - Computers Systems*

חלוקת זיכרון קבועה Static Partitions

דוגמה: לאחר שניית ריצה 7



Best Fit:

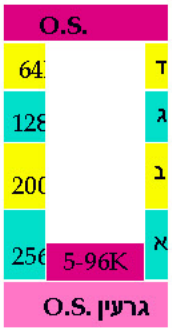
2
312K
2sec

תוכנית זו לא תופעל
אף פעם, היתר
סיימו ריצתם ב-7 שניות

Operational Systems 03 *Saul Coval - Computers Systems*

חלוקת זיכרון קבועה Static Partitions

דוגמה: שניית ריצה 7

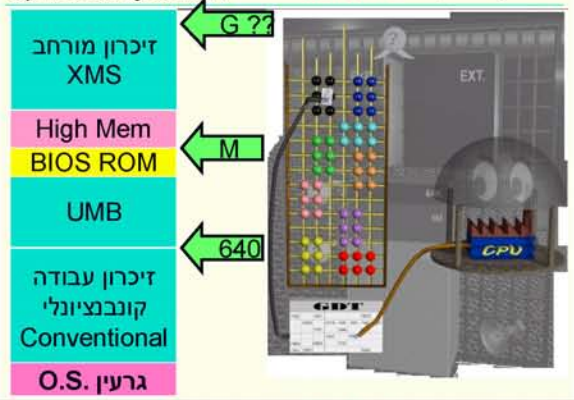


שטח לא מנוצל =

Best Fit:

2
312K
2sec

Operational Systems 04 — Saul Coval - Computers Systems



Operational Systems 04 — Saul Coval - Computers Systems

OPERATIONAL SYSTEMS
מערכות הפעלה

Operational Systems 04 — Saul Coval - Computers Systems

לשימוש עבור מדיניות שיחרר דף תכנית שבזיכרון, וכרגע לא פעילים				First	Least	Most >>	Bit for
Release policy to release non working pages from XMS				IN	Recently	Frequently	Protected
memory management (Page Demand) [64KByte each]				First OUT	Used	Used	Mode (1,0)
מספר דף בזיכרון	כתובת פיזית הדף Physical Address		שם תהליך	דף תהליך	זמן כניסה לזיכרון	זמן סיום שלב אחרון	ס"כ יחידות זמן פעיל	סיבית הגנה נגד מחיקה
PAGE	Begins	Ends	Task	File PAGE	Begin @	last End @	Work time	Prot. Bit
0	100000	10FFFF						
1	110000	11FFFF						
2	120000	12FFFF						
3	130000	13FFFF						
4	140000	14FFFF						
5	150000	15FFFF						

Operational Systems 04 — Saul Coval - Computers Systems

GLOBAL DESCRIPTOR TABLE (GDT) on Extended Memory - XMS

טבלה לתיאור חלוקת זיכרון מורחב XMS

לשימוש עבור מדיניות שיחרר דף תכנית שבזיכרון, וכרגע לא פעילים					First	Least	Most >>	Bit for	Not	Free-שופש
Release policy to release non working pages from XMS					IN	Recently	Frequently	Protected	Used	Work-פעיל
memory management (Page Demand) [64KByte each]					First OUT	Used	Used	Mode (1,0)	Recently	StdBy-רזרוב
									(0,1)	(Flag)
מספר דף בזיכרון	כתובת פיזית הדף Physical Address		שם תהליך	דף תהליך	זמן כניסה לזיכרון	זמן סיום שלב אחרון	ס"כ יחידות זמן פעיל	סיבית הגנה נגד מחיקה	סיבית מחוחדת	מצב בזיכרון	הערות
PAGE	Begins	Ends	Task	File PAGE	Begin @	last End @	Work time	Prot. Bit	N.U.R-ל	State	Notes
0	100000	10FFFF									
1	110000	11FFFF									
2	120000	12FFFF									
3	130000	13FFFF									
4	140000	14FFFF									
5	150000	15FFFF									
6	160000	16FFFF									
7	170000	17FFFF									
8	180000	18FFFF									
9	190000	19FFFF									
A	1A0000	1AFFFF									
B	1B0000	1BFFFF									
C	1C0000	1CFFFF									
D	1D0000	1DFFFF									
E	1E0000	1EFFFF									
F	1F0000	1FFFFF									
10	200000	20FFFF									
11	210000	21FFFF									
12	220000	22FFFF									
13	230000	23FFFF									
14	240000	24FFFF									
15	250000	25FFFF									

Operational Systems 04 — Saul Coval - Computers Systems

מצב אכשיי של הזיכרון לצורך			הישן ביותר	זמן ללא שימוש	Least <<	ללא שיחרר	Not	חופשי-Free
לשימוש עבור מדיניות שיחרר דף תכנית			First	Least	Most >>	Bit for	Used	פעיל-Work
ase non working pages from XMS			IN	Recently	Frequently	Protected	Recently	רזרוב-StdBy
t (Page Demand) [64KByte each]			First OUT	Used	Used	Mode (1,0)	(0,1)	(Flag)
כתובת פיזית ddress	שם תהליך	דף תהליך	זמן כניסה לזיכרון	זמן סיום שלב אחרון	ס"כ יחידות זמן פעיל	סיבית הגנה נגד מחיקה	סיבית מחוחדת	מצב בזיכרון
Ends	Task	File PAGE	Begin @	last End @	Work time	Prot. Bit	N.U.R-ל	State
10FFFF	G - Emm386	G 1	20:23:21.3	21:13:36.7	256	1	0	Stand-By
11FFFF	D - Explorer	D 2	20:33:45.8	20:57:28.9	2380	1	0	Stand-By
12FFFF	A - Excel	A 3	21:02:01.4	21:03:11.7	110	0	0	Stand-By
13FFFF	E - ViruSafe	E 2	20:56:01.7	21:01:11.4	420	0	0	WORK
14FFFF	F - Ms Mouse	F 1	20:12:31.4	20:13:56.7	25	0	0	Free-חופשי
15FFFF	A - Excel	A 5	21:07:01.4		115	0	1	WORK
16FFFF	C - Solitaire	C 1	21:01:03.9	21:01:45.5	42	0	0	Stand-By

Operational Systems 04 — Saul Coval - Computers Systems

לשימוש עבור מדיניות שיחרר דף תכנית שבזיכרון, וכרגע לא פעילים					First	Least	Most >>	Bit for	Not	Free-שופש
Release policy to release non working pages from XMS					IN	Recently	Frequently	Protected	Used	Work-פעיל
memory management (Page Demand) [64KByte each]					First OUT	Used	Used	Mode (1,0)	Recently	StdBy-רזרוב
									(0,1)	(Flag)
מספר דף בזיכרון	כתובת פיזית הדף Physical Address		שם תהליך	דף תהליך	זמן כניסה לזיכרון	זמן סיום שלב אחרון	ס"כ יחידות זמן פעיל	סיבית הגנה נגד מחיקה	סיבית מחוחדת	מצב בזיכרון	
PAGE	Begins	Ends	Task	File PAGE	Begin @	last End @	Work time	Prot. Bit	N.U.R-ל	State	
0	100000	10FFFF									
1	110000	11FFFF									
2	120000	12FFFF									
3	130000	13FFFF									
4	140000	14FFFF									

Operational Systems 04 — Saul Coval - Computers Systems

ספר עזר: עקרונות מערכות הפעלה - אוניברסיטה פתוחה
" ספר שני – י"ח 8-5 " - פרק/יחידה חמש

Operational Systems 04 — Saul Coval - Computers Systems

דף תהליך	זמן כניסה לזיכרון	זמן סיום שלב אחרון	ס"כ יחידות זמן פעיל	סיבית הגנה נגד מחיקה	סיבית מחוחדת	מצב בזיכרון
File PAGE	Begin @	last End @	Work time	Prot. Bit	N.U.R-ל	State
G 1	20:23:21.3	21:13:36.7	256	1	0	Stand-By
D 2	20:33:45.8	20:57:28.9	2380	1	0	Stand-By
A 3	21:02:01.4	21:03:11.7	110	0	0	Stand-By
E 2	20:56:01.7	21:01:11.4	420	0	0	WORK
F 1	20:12:31.4	20:13:56.7	25	0	0	Free-חופשי
A 5	21:07:01.4		115	0	1	WORK
C 1	21:01:03.9	21:01:45.5	42	0	0	Stand-By
D 5	20:59:45.7	21:00:45.9	110	0	0	Stand-By
G 2	20:33:21.3	21:18:36.7	156	1	0	Stand-By
E 1	20:59:01.6	20:59:12.3	10	1	0	Stand-By
D 4	21:03:02.5	21:04:05.2	97	0	0	Stand-By
D 6	21:03:37.9	21:02:45.8	11	0	0	Stand-By
F 3	20:13:21.3	20:13:36.7	15	0	0	Free-חופשי
C 3	21:01:38.7	21:01:53.6	18	0	0	Stand-By
D 1	20:52:04.6	20:54:13.3	45	0	0	WORK
A 7	21:03:11.8	21:03:21.6	10	0	0	Stand-By
E 3	21:09:03.2	21:10:07.1	10	0	0	Stand-By
G 5	20:58:25.3	20:73:36.7	108	0	0	Stand-By
H 2	21:07:51.6		45	0	0	WORK
C 4	21:08:00.6	21:08:51.5	50	0	1	WORK
A 9	21:05:07.1	21:05:08.6	1	1	0	Stand-By
D 7	20:48:59.6	20:51:02.1	38	0	0	Stand-By

Operational Systems 04 — Saul Coval - Computers Systems

דף תהליך	זמן כניסה לזיכרון	זמן סיום שלב אחרון	ס"כ יחידות זמן פעיל	סיבית הגנה נגד מחיקה	סיבית מיוחדת	מצב בזיכרון
File PAGE	Begin @	last End @	Work time	Prot. Bit	N.U.R-ל	State
G 1	20:23:21.3	21:13:36.7	256	1	0	Stand-By
D 2	20:33:45.8	20:57:28.9	2380	1	0	Stand-By
A 3	21:02:01.4	21:03:11.7	110	0	0	Stand-By
E 2	20:56:01.7	21:01:11.4	420	0	0	WORK
F 1	20:12:31.4	20:13:56.7	25	0	0	Free-חופשי
A 5	21:07:01.4		115	0	1	WORK
C 1	21:01:03.9	21:01:45.5	42	0	0	Stand-By
D 5	20:59:45.7	21:00:45.9	110	0	0	Stand-By
G 2	20:33:21.3	21:18:36.7	156	1	0	Stand-By
E 1	20:59:01.6	20:59:12.3	10	1	0	Stand-By
D 4	21:03:02.5	21:04:05.2	97	0	0	Stand-By
D 6	21:03:37.9	21:02:45.8	11	0	0	Stand-By
F 3	20:13:21.3	20:13:36.7	15	0	0	Free-חופשי
C 3	21:01:38.7	21:01:53.6	18	0	0	Stand-By
D 1	20:52:04.6	20:54:13.3	45	0	0	WORK
A 7	21:03:11.8	21:03:21.6	10	0	0	Stand-By
E 3	21:09:03.2	21:10:07.1	10	0	0	Stand-By
G 5	20:58:25.3	20:73:36.7	108	0	0	Stand-By
H 2	21:07:51.6		45	0	0	WORK
C 4	21:08:00.6	21:08:51.5	50	0	1	WORK
A 9	21:05:07.1	21:05:08.6	1	1	0	Stand-By
D 7	20:48:59.6	20:51:02.1	38	0	0	Stand-By

Operational Systems 04 — Saul Coval - Computers Systems

GLOBAL DESCRIPTOR TABLE (GDT) on Extended Memory - XMS

טבלה לתיאור חלוקת זיכרון מורחב XMS

						הישן ביותר	זמן ללא שימוש	Least <<	ללא שיחרור	Not	חופשי-Free	
					לשימוש עבור מדיניות שיחרור דפי תוכניות שבזיכרון, וכרגע לא פעילים	First	Least	Most >>	Bit for	Used	פעיל-Work	
					Release policy to release non working pages from XMS	IN	Recently	Frequently	Protected	Recently	רדום-StdBy	
					memory management (Page Demand) [64KByte each]	First OUT	Used	Used	Mode (1,0)	(0,1)	(Flag)	
מתוך הזיכרון המורחב ..	מספר דף בזיכרון	כתובת פיזית הדף Physical Address		שם תהליך	דף תהליך	זמן כניסה לזיכרון	זמן סיום שלב אחרון	ס"כ יחידות זמן פעיל	סיבית הגנה נגד מחיקה	סיבית מיוחדת	מצב בזיכרון	הערות
	PAGE	Begins	Ends	Task	File PAGE	Begin @	last End @	Work time	Prot. Bit	N.U.R-ל	State	Notes
	0	100000	10FFFF									
	1	110000	11FFFF									
	2	120000	12FFFF									
	3	130000	13FFFF									
	4	140000	14FFFF									
	5	150000	15FFFF									
	6	160000	16FFFF									
	7	170000	17FFFF									
	8	180000	18FFFF									
	9	190000	19FFFF									
	A	1A0000	1AFFFF									
	B	1B0000	1BFFFF									
	C	1C0000	1CFFFF									
	D	1D0000	1DFFFF									
	E	1E0000	1EFFFF									
	F	1F0000	1FFFFF									
	10	200000	20FFFF									
	11	210000	21FFFF									
	12	220000	22FFFF									
	13	230000	23FFFF									
	14	240000	24FFFF									
	15	250000	25FFFF									
	.....											