

Operational Systems – Scheduler ver02

מערכות הפעלה

סיכום נושא תהליכים

COVAL Systems saul@coval.net - http://www.coval.net

מקורות מידע

- ספר הקורס (ראשי) למצגת זו: Silberschatz and Galvin, Operating Systems Concepts (5th ed.) Wiley

המצגת מורכבת מעבודות של סטודנטים. ייתכן שחלקם נלקחו מתוך אתרים פרטיים. אני רק ריכזתי את עבודות הסטודנטים ללא דרישות לזכויות כל שהם.

מקור מידע מאינטרנט

<http://webcourse.technion.ac.il/236364>

COVAL Systems 6 December 2004 עמוד 3

מהי מערכת הפעלה?

- שכבת תוכנה לניהול והסתרת פרטים של חומרת המחשב.
- מספקת לאפליקציה אבסטרקציה של מכונה וירטואלית ייעודית וחזקה (זיכרון עצום, מעבד ייעודי חזק מאוד...)
- מנהלת את משאבי המערכת ומשתפת אותם בין תהליכים, תוכניות, ומשתמשים.

COVAL Systems 6 December 2004 עמוד 4

תפקיד מערכת ההפעלה

מאפשרת להריץ אפליקציות

- מבטיחה נכונות.
 - גבולות זיכרון
 - עדיפויות
 - מצב יציב
- מספקת נוחיות.
 - הסתרת פרטים
 - תיאום
 - קריאות מערכת-הפעלה
 - מערכת קבצים

The diagram shows a box for 'אפליקציות' (Applications) and a box for 'מערכת הפעלה' (Operating System) connected by lines to a hardware layer. The hardware layer includes 'CPU' and 'memory', with three 'device' boxes connected to the CPU.

COVAL Systems 6 December 2004 עמוד 5

שיתוף משאבים

אפליקציה רוצה את כל המשאבים:

- זמן מעבד
- זיכרון
- קבצים
- אמצעי קלט / פלט
- שעון

מערכת ההפעלה נותנת לכל אפליקציה אשליה של מערכת שלמה משל עצמו.

COVAL Systems 6 December 2004 עמוד 6

התפתחות מערכות הפעלה

- חומרה יקרה ואיטית, כוח-אדם זול
 - Batch jobs, ניצול החומרה 24x7: IBM S/360
- חומרה יקרה ומהירה, כוח-אדם זול
 - Unix: Interactive time-sharing
- חומרה זולה ואיטית, כוח-אדם יקר
 - מחשב אישי לכל משתמש: MS-DOS

COVAL Systems 6 December 2004 עמוד 7

הווה ועתיד

- חומרה זולה מאוד, כוח חישוב רב.
 - ריבוי משימות: Windows NT, OS/2, ...
 - שיתוף משאבים בסיסי: דיסקים, מדפסות, ...
- בחזרה לעבר...
- רשתות מהירות.
 - הרשת היא המחשב: SETI@home, Grid Computing
 - הרשת היא אמצעי אחסון: SAN, Web storage

מבנה המחשב

- התנהגות מערכת ההפעלה מוכתבת (חלקית) על-ידי החומרה שעליה היא רצה
 - סט פקודות, רכיבים מיוחדים
- החומרה יכולה לפשט / לסבך משימות מערכת ההפעלה
 - מחשבים ישנים לא סיפקו תמיכה לזיכרון וירטואלי

מנגנוני חומרה לתמיכה במערכת ההפעלה

- שעון חומרה
- פעולות סנכרון אטומיות
- פסיקות
- קריאות מערכת-הפעלה
- פעולות בקרת קלט / פלט
- הגנת זיכרון
- אופן עבודה מוגן (protected)
- פעולות מוגנות

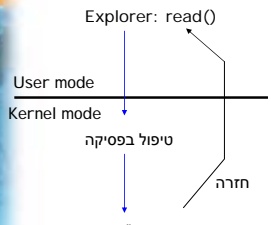
פקודות מוגנות

- חלק מפקודות המכונה מותרות רק למערכת-ההפעלה
 - גישה לרכיבי קלט / פלט (דיסקים, כרטיסי תקשורת).
 - שינוי של מבני הנתונים לגישה לזיכרון (טבלת דפים, TLB).
 - עדכון של סיביות מוד (מצב) מיוחדות (לקביעת עדיפות טיפול בפסיקות).
 - פקודת halt.

עבודה במצב מוגן

- הארכיטקטורה תומכת בשני מצבים לפחות:
 - kernel mode
 - user mode(במעבדי IA32 יש ארבעה מצבים...)
- המצב נשמר באמצעות status bit ברגיסטר מוגן.
 - תכניות משתמש רצות ב-user mode.
 - מערכת ההפעלה רצה ב-kernel mode.
- המעבד מבצע פקודות מוגנות רק ב-kernel mode.

אז איך משתמש ניגש לדיסק?

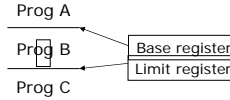
- קריאה לפרוצדורת מערכת-הפעלה (system call)
 - גורמת לפסיקה
 - פרמטר מזהה את קריאת המערכת
 - שומרת את מצב התוכנית הקוראת
 - מוודאת את הפרמטרים (למשל, מצביעי זבל)
 - דרך לחזור לתוכנית הקוראת כאשר מסיימים
- 
- ```
graph TD
 subgraph User_Mode [User mode]
 Explorer[Explorer: read()]
 end
 subgraph Kernel_Mode [Kernel mode]
 Routine[read() kernel routine]
 end
 Explorer --> Routine
 Routine -- חזרה --> Explorer
 Routine -- טיפול בפסיקה --> Routine
```

## דוגמא על Linux על מעבדי Intel

- אתחול:
  - טבלת מזהי פסיקה (Interrupt Descriptor Table) עם מטפלים לכל אחד מסוגי הפסיקות.
  - ווקטור 821 (= 0x80) מתאים ל-system calls.
  - לקוד גרעין יש עדיפות 0, לקוד משתמש עדיפות 3.
- כניסה ב IDT, המתאימה לווקטור 128, מכילה:
  - מצביע לקטע קוד גרעין המטפל ב-system calls.
  - אישורים עבור קוד עם עדיפות 3 ליזום קריאה לפסיקה זו.
- בביצוע system call, תהליך המשתמש:
  - מציב ברגיסטר eax את מספר ה-system call המבוקש.
  - מנין פרמטרים נוספים לפי השרות המבוקש
  - מבצע פקודת "int 0x80" (פסיקה יזומה ע"י תוכנה).

## הגנה על הזיכרון

- מערכת ההפעלה צריכה להגן על תוכניות המשתמשים, זו מפני זו (עם או בלי כוונה רעה).
- מערכת ההפעלה צריכה להגן על עצמה מפני תוכניות המשתמשים.
  - ועל תוכניות המשתמשים מפניה?
- שיטה פשוטה: base register, limit register לכל אחת מהתוכניות.
  - מוגנים בעצמם.
- זיכרון וירטואלי.



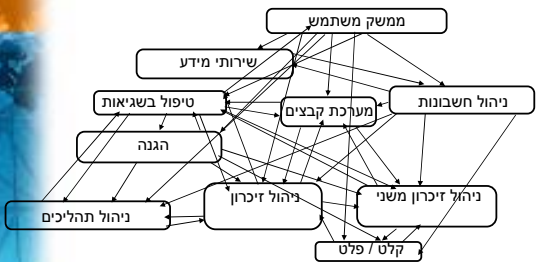
## יום בחיים של מערכת ההפעלה

- מלבד אתחול המערכת, נכנסים לגרעין רק בגלל מאורע.
  - הגרעין מגדיר אופן טיפול בכל מאורע.
    - חלק נקבע על ידי ארכיטקטורת המעבד.
    - מנגנון כפי שראינו.
- פסיקות ו-exceptions:
  - פסיקות נגרמות על-ידי רכיבי חומרה (שעונים, סיום ק/פ)
  - Exceptions מגיעות מהתוכנה (פקודה מפורשת, page fault)

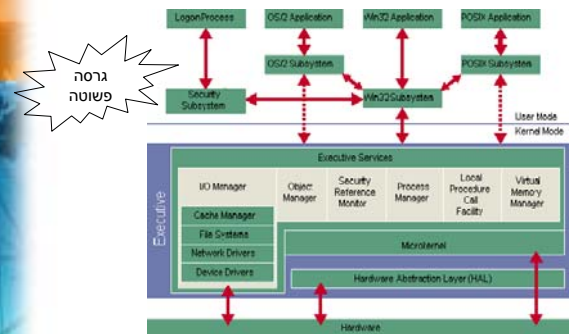
## רכיבי מערכת ההפעלה

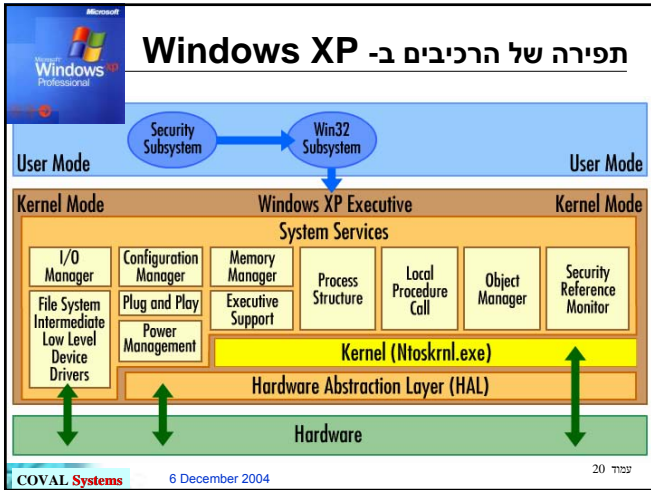
- תהליכים
  - זיכרון
  - קלט / פלט
  - זיכרון משני
  - מערכות קבצים
  - הגנה
  - ניהול חשבונות משתמשים
  - ממשק משתמש (shell)
- זאת ועוד...
  - אתחול
  - גיבוי
  - ...
  - דפדפן?

## תפירה של הרכיבים



## תפירה של הרכיבים ב Windows NT





### ארגון מערכת ההפעלה

בראשית... מונוליתית ✓ תקשורת זולה בין מודולים

- × קשה להבין
- × קשה לשנות או להוסיף רכיבים

מה האלטרנטיבה?

|                    |
|--------------------|
| תוכניות משתמש      |
| גרעין מערכת ההפעלה |
| חומרה              |

COVAL Systems 6 December 2004 עמוד 21

### ארגון מערכת ההפעלה

בראשית... מונוליתית היום... גרעין קטן

|                                                                                                                                                                                                                                |               |                                                       |                                                  |       |           |                                                                                                                          |               |                    |       |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|-------------------------------------------------------|--------------------------------------------------|-------|-----------|--------------------------------------------------------------------------------------------------------------------------|---------------|--------------------|-------|
| <table border="1"> <tr><td>תוכניות משתמש</td></tr> <tr><td>System processes: networking, file system, scheduling</td></tr> <tr><td>Micro-kernel: זיכרון וירטואלי, ניהול המעבד, הגנה</td></tr> <tr><td>חומרה</td></tr> </table> | תוכניות משתמש | System processes: networking, file system, scheduling | Micro-kernel: זיכרון וירטואלי, ניהול המעבד, הגנה | חומרה | User mode | <table border="1"> <tr><td>תוכניות משתמש</td></tr> <tr><td>גרעין מערכת ההפעלה</td></tr> <tr><td>חומרה</td></tr> </table> | תוכניות משתמש | גרעין מערכת ההפעלה | חומרה |
| תוכניות משתמש                                                                                                                                                                                                                  |               |                                                       |                                                  |       |           |                                                                                                                          |               |                    |       |
| System processes: networking, file system, scheduling                                                                                                                                                                          |               |                                                       |                                                  |       |           |                                                                                                                          |               |                    |       |
| Micro-kernel: זיכרון וירטואלי, ניהול המעבד, הגנה                                                                                                                                                                               |               |                                                       |                                                  |       |           |                                                                                                                          |               |                    |       |
| חומרה                                                                                                                                                                                                                          |               |                                                       |                                                  |       |           |                                                                                                                          |               |                    |       |
| תוכניות משתמש                                                                                                                                                                                                                  |               |                                                       |                                                  |       |           |                                                                                                                          |               |                    |       |
| גרעין מערכת ההפעלה                                                                                                                                                                                                             |               |                                                       |                                                  |       |           |                                                                                                                          |               |                    |       |
| חומרה                                                                                                                                                                                                                          |               |                                                       |                                                  |       |           |                                                                                                                          |               |                    |       |

COVAL Systems 6 December 2004 עמוד 22

### ארגון מערכת ההפעלה

- שכבה דקה מספקת שירותי גרעין היום... גרעין קטן
- ✓ אמינות גבוהה יותר
- ✓ קל להרחיב ולשנות
- × ביצועים גרועים (מעבר בין user-mode לבין kernel-mode)

דוגמאות: Mach, OS X, ~Windows NT

|                                                                                                                                                                                                                                |               |                                                       |                                                  |       |           |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|-------------------------------------------------------|--------------------------------------------------|-------|-----------|
| <table border="1"> <tr><td>תוכניות משתמש</td></tr> <tr><td>System processes: networking, file system, scheduling</td></tr> <tr><td>Micro-kernel: זיכרון וירטואלי, ניהול המעבד, הגנה</td></tr> <tr><td>חומרה</td></tr> </table> | תוכניות משתמש | System processes: networking, file system, scheduling | Micro-kernel: זיכרון וירטואלי, ניהול המעבד, הגנה | חומרה | User mode |
| תוכניות משתמש                                                                                                                                                                                                                  |               |                                                       |                                                  |       |           |
| System processes: networking, file system, scheduling                                                                                                                                                                          |               |                                                       |                                                  |       |           |
| Micro-kernel: זיכרון וירטואלי, ניהול המעבד, הגנה                                                                                                                                                                               |               |                                                       |                                                  |       |           |
| חומרה                                                                                                                                                                                                                          |               |                                                       |                                                  |       |           |

COVAL Systems 6 December 2004 עמוד 23

### תהליך

- מהו תהליך?
- מבני הנתונים לניהול תהליכים.
- החלפת הקשר.
- ניהול תהליכים ע"י מערכת ההפעלה.

COVAL Systems 6 December 2004 עמוד 24

### תהליך

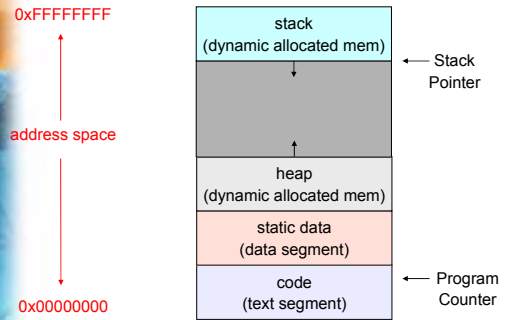
- אבסטרקציה:
  - יחידת הביצוע לארגון "פעילות" המחשב
  - יחידת הזימון ביצוע במעבד ע"י מערכת ההפעלה
  - תוכנית בביצוע (סדרתי = פקודה-אחת-אחר-השנייה) נקרא גם job, task, sequential process

COVAL Systems 6 December 2004 עמוד 25

## מה מאפיין תהליך

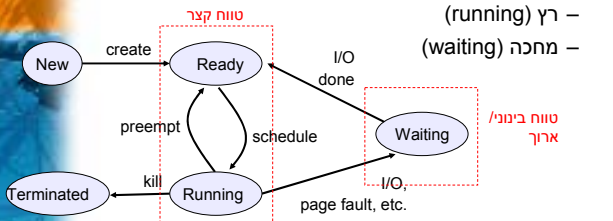
- מרחב כתובות
  - קוד התוכנית
  - נתונים
  - מחסנית זמן-ביצוע
  - program counter
  - רגיסטרים
  - מספר תהליך (process id)
- תהליך לעומת תוכנית:
- תוכנית היא חלק ממצב התהליך
  - תוכנית יכולה לייצר כמה תהליכים

## מרחב הכתובות של התהליך



## מצבי התהליך

- כל תהליך נמצא באחד המצבים הבאים:
  - מוכן (ready)
  - רץ (running)
  - מחכה (waiting)



- מתי מזמנים / מפנים תהליכים?

## מבני הנתונים של תהליך

ב Linux, מכיל +95 שדות!

|                     |
|---------------------|
| Process id (PID)    |
| Execution state     |
| Program counter     |
| Stack pointer       |
| Registers           |
| Memory limits       |
| Scheduling priority |
| Username            |

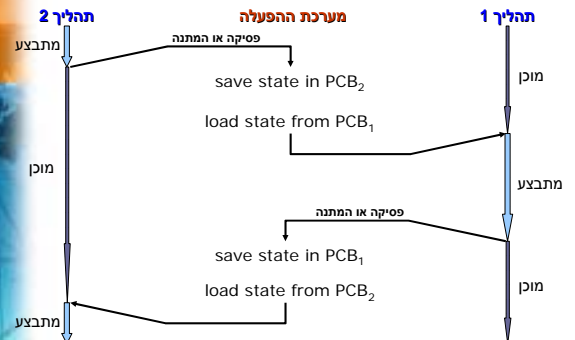
- בכל זמן, הרבה תהליכים פעילים במערכת
- לכל תהליך מצב
- Process control block (PCB) שומר את מצב התהליך כאשר אינו רץ.
- נקרא Process Descriptor ב-Linux
- נשמר כאשר התהליך מפונה, נטען כאשר התהליך מתחיל לרוץ.

## מצב המעבד וה-PCB

- כאשר תהליך רץ, המצב שלו נמצא במעבד:
  - SP, PC, רגיסטרים
  - רגיסטרים לקביעת גבולות זיכרון
- כאשר המעבד מפסיק להריץ תהליך (מעבירו למצב המתנה), שומר את ערכי הרגיסטרים ב-PCB.
- כאשר המעבד מחזיר תהליך למצב ריצה, טוען את ערכי הרגיסטרים מתוך ה-PCB.

**Context Switch:** העברת המעבד מתהליך אחד לשני.

## Context switch





## ה-PCB ותורי המצבים

ה-PCB הוא מבנה נתונים בזיכרון מערכת ההפעלה.

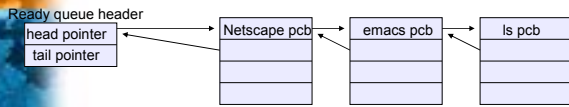
- כאשר התהליך נוצר, מוקצה עבורו PCB (עם ערכי התחלה), ומשורשר לתור המתאים (בדרך-כלל, ready).
- ה-PCB נמצא בתור המתאים למצבו של התהליך.
- כאשר מצב התהליך משתנה, ה-PCB שלו מועבר מתור לתור.

- כאשר התהליך מסתיים, ה-PCB שלו משוחרר.

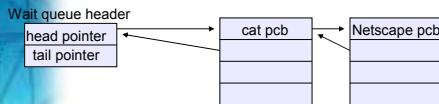
עמוד 33

## תורי מצבים

- מערכת ההפעלה מחזיקה תורים של תהליכים – תור ready (או מבנה נתונים מתוחכם יותר)



- תור waiting, למשל ל-device מסוים



עמוד 32

## יצירת תהליכים ב-UNIX: fork()

- יוצר ומאתחל PCB.
- מיצר מרחב כתובות חדש, ומאתחל אותו עם העתק מלא של מרחב הכתובות של האב.
- מאתחל משאבי גרעין לפי משאבי האב (למשל, קבצים פתוחים).
- שם את ה-PCB בתור המוכנים.
- עכשיו יש שני תהליכים, אשר נמצאים באותה נקודה בביצוע אותה תוכנית.
- שני התהליכים חוזרים מה fork().
- הבן, עם ערך 0
- האב, עם מספר התהליך (pid) של הבן
- למעשה, יש מימוש יעיל יותר ל fork() – נראה בהמשך.

```
int main(int argc,
char **argv)
{
 int child_pid = fork();
 if (child_pid == 0) {
 printf("Son of %d is %d\n",
 getpid(),getpid());
 return 0;
 } else {
 printf("Father of %d is
 %d\n",
 child_pid,getpid());
 return 0;
 }
}
```

עמוד 35

## יצירת תהליך

- תהליך אחד (האב) יכול ליצור תהליך אחר (הבן) – שדה ppid בביצוע ps -al ב-Linux.
- בדרך-כלל, האב מגדיר או מוריש משאבים ותכונות לבניו. – ב-Linux, הבן יורש את שדה user, ועוד.
- האב יכול להמתין לבנו, לסיים, או להמשיך לרוץ במקביל. – רק תהליך אב יכול להמתין לבניו
- ב Windows יש קריאת CreateProcess(...,prog.exe,...) – מייצרת תהליך חדש (ללא קשרי אבות/בנים) אשר מתחיל לבצע את prog.

עמוד 34

## דוגמה UNIX shell:

```
int main(int argc, char **argv)
{
 while (1) {
 char *cmd = get_next_command();
 int child_pid = fork();
 if (child_pid == 0) {
 execv(cmd);
 fprintf(stderr, "exec failed!");
 } else {
 wait(child_pid);
 }
 }
}
```

עמוד 37

## איך מפעילים תוכנית חדשה?

int execv(char \*prog, char \*\*argv)

- עוצר את ביצוע התוכנית הנוכחית.
- טוען את prog לתוך מרחב הכתובות.
- מאתחל את מצב המעבד, וארגומנטים עבור התוכנית החדשה.
- מפנה את המעבד (ה-PCB מועבר לתור המוכנים)

לא יוצר תהליך חדש!

עמוד 36

## זימון תהליכים

- מדיניות בסיסיות: RR, FCFS, SJF
- הערכת זמן ריצה ושימוש בעדיפויות
- ריבוי תורים ודוגמאות

עמוד 38

COVAL Systems 6 December 2004

## זימון תהליכים

- זימון טווח-קצר:
  - בוחר תהליך מתור המוכנים ומריץ אותו ב-CPU.
  - מופעל לעיתים קרובות (אלפיות-שנייה).
  - חייב להיות מהיר.
- זימון טווח-ארוך:
  - בוחר איזה תהליך יובא לתור המוכנים.
  - מופעל לעיתים "רוחות" (שניות, דקות).
  - יכול להיות איטי.
  - תהליכים יכולים להשתחרר יחד מהמתנה בעקבות אירוע (למשל המתנה ל timer) או אחד מקבוצה בכל אירוע (למשל המתנה למשאב שאינו ניתן לשיתוף כמו מדפסת)

עמוד 39

COVAL Systems 6 December 2004

## מדדים להערכת אלגוריתם לזימון תהליכים

עבור זימון טווח-קצר, המדדים העיקריים הם:

- זמן תגובה מינימאלי
- זמן תגובה = זמן המתנה + זמן ביצוע
- תקורה מינימאלית
- אי-אפשר לנצח בשני המדדים (trade-off).

💡

לפעמים מעוניינים במטרות נוספות:

- ניצול מקסימאלי של המעבד
- הספק (throughput) מקסימאלי (מספר התהליכים שמסתיימים בפרק זמן)

עמוד 40

COVAL Systems 6 December 2004

## Processes and Threads

The diagram illustrates the relationship between processes and threads. It shows three overlapping boxes representing processes: Process 1, Process 2, and Process 3. Each process box contains sub-boxes for 'Memory Address Space', 'System Resource', and 'Thread'. Process 1 has three threads, Process 2 has one thread, and Process 3 has two threads. This shows that multiple threads can exist within a single process and share its resources.

עמוד 41

COVAL Systems 6 December 2004

## Multitasking

The diagram compares two multitasking models. In the 'Preemptive' model, a 'Windows System Scheduler' box has arrows pointing to 'Thread 1' (pink) and 'Thread 2' (blue). The timeline shows Thread 1 running, then Thread 2 running, then Thread 1 running again, illustrating how the scheduler can interrupt a thread to run another. In the 'Cooperative' model, 'Application 1' and 'Application 2' have arrows pointing to 'Thread 1' (pink) and 'Thread 2' (blue) respectively. The timeline shows Thread 1 running until it voluntarily yields, then Thread 2 runs, then Thread 1 runs again.

עמוד 42

COVAL Systems 6 December 2004

## Thread Priorities

The diagram shows a vertical scale of priority levels from 0 to 31. At the bottom, level 0 is labeled 'Applications'. At the top, level 31 is labeled 'Critical System Tasks'. A box labeled 'Priority Levels 32' is positioned between levels 20 and 31, indicating the range of priority levels available for user applications.

עמוד 43

COVAL Systems 6 December 2004

## אלגוריתם First-Come, First-Served

- התהליך שהגיע ראשון לתור הממתינים ירוץ ראשון
  - נותן עדיפות לתהליכים חישוביים (CPU bound)
  - ממזער ניצול התקנים
  - לא מספק דרישות שיתוף (time sharing)
- ◀ non-preemptive (ללא הפקעות): תהליך מקבל את המעבד עד לסיומו.
- ◀ מימוש פשוט: תור התהליכים המוכנים הוא FIFO.

## FCFS: דוגמה

זמן ההמתנה של תהליך מרגע הגעתו לתור המוכנים ועד לתחילת ביצועו תלוי בסדר הגעת התהליכים לטווח הקצר למשל, אפשרות אחת (התהליכים מגיעים "ביחד"):



$$\text{זמן המתנה ממוצע} = 17 = (0+24+27)/3$$



$$\text{זמן המתנה ממוצע} = 3 = (0+3+6)/3$$

## אפקט השיירה Convoy

C תהליך עתיר חישובים  
 $I_1, \dots, I_n$  תהליכים עתירי I/O

מה קורה?

- תהליך C תופס את המעבד.
  - תהליכי  $I_1$  מצטברים בתור המוכנים.
  - התקני קלט / פלט מובטלים!



## Round Robin (RR)

- תור מוכנים מעגלי
- המעבד מוקצה לתהליך הראשון בתור
- אם זמן הביצוע של התהליך גדול מקצבת זמן מסוימת, q, התהליך מופסק ומועבר לסוף תור המוכנים.

◀ preemptive

◀ בדרך-כלל, q = 10-100msec.

◀ מימוש באמצעות timer שמייצר פסיקה כל q יחידות-זמן.

## השוואת זמן הסיום של תהליכים

| RR   | FCFS | תהליך |
|------|------|-------|
| 991  | 100  | 1     |
| 992  | 200  | 2     |
|      |      | ⋮     |
|      |      | 10    |
| 1000 | 1000 |       |

- 10 תהליכים
- כל תהליך דורש 100 יחידות-זמן.
- q=1

## אפיונים של Round Robin

- אם קצבת הזמן q קטנה, הזימון "הוגן"
  - כאילו כל תהליך רץ במעבד משלו בקצב  $1/N$  כאשר N הוא מספר התהליכים.
  - זמן תגובה (כמעט) ליניארי בזמן החישוב ו-N.
  - התקורה עלולה להיות גבוהה!

• אם q גדול מאד, RR הופך ל-FCFS

• זמן התגובה (= זמן המתנה + זמן ביצוע) הממוצע תחת RR הוא לכל היותר פעמיים האופטימאלי.



## Selfish Round-Robin

- תהליכים חדשים ממתנים בתור FIFO
- תהליכים וותיקים מוחזקים בתור Round-Robin לביצוע
- כאשר אין תהליך בתור הוותיקים, בוחרים את הראשון בתור החדשים ומצרפים אותו לוותיקים
- תהליכים מזדקנים: בכל יחידת זמן, עדיפות התהליך גדלה, עד שעוברת סף מסוים, והתהליך עובר לתור הוותיקים.

הזדקנות מהירה (או מיידית)  $\Leftarrow$  RR  
 הזדקנות איטית (או אינסופית)  $\Leftarrow$  FCFS

## Shortest Job First (SJF)

נקרא גם Shortest Processing Time First  
 מריצים את התהליך עם זמן ביצוע מינימאלי, עד לסיומו.  
 – כל התהליכים מגיעים יחד.  
 – זמן הביצוע של תהליך ידוע מראש.  
 < Non-preemptive

דוגמא: P1 (6) P2 (8) P3 (7) P4 (3)



זמן המתנה ממוצע =  $(0+3+9+16)/4 = 7$

מינימאלי לכל סדר זימון אפשרי!



## Shortest Remaining Time to Completion First (SRTF)

כאשר מגיע תהליך  $P_i$  שזמן הביצוע הנותר שלו קצר יותר מזמן הביצוע הנותר של התהליך שרץ כרגע  $P_k$  מכניסים את  $P_i$  למעבד, במקום  $P_k$

< preemptive  
 < ממזער את זמן השהייה הממוצע במערכת.

SRTF אופטימאלי כאשר תהליכים לא מגיעים יחד, בניגוד ל-SJF.

## ניבוי זמן הריצה של תהליך

אומדן סטטיסטי של הזמן עד לוויטור על המעבד, על-פי הפעמים הקודמות שהתהליך החזיק במעבד

$\tau_i$  – הערכת זמן הביצוע לסיבוב ה- $i$

$t_i$  – זמן הביצוע בפועל בסיבוב ה- $i$

$$\tau_{i+1} = \alpha \cdot \tau_i + (1 - \alpha) \cdot t_i$$

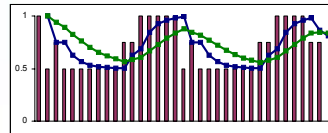
$$0 \leq \alpha \leq 1$$

לדוגמא,

$$t_0 = 1$$

$$\alpha = 1/2$$

$$\alpha = 3/4$$

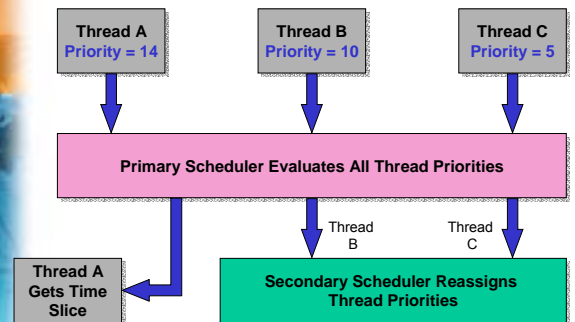


## עדיפויות

- לכל תהליך יש עדיפות התחלתית.
- עדיפות התחלתית גבוהה ניתנת:
  - לתהליכים של משתמשים חשובים (למשל, sysadmin)
  - לתהליכים שסיומם דחוף
  - לתהליכים אינטראקטיביים

תהליכים בעדיפות מינימאלית יכולים לשמש לתרומת זמן מעבד לצרכים לא חשובים (SETI@home)

## Scheduling



## Multilevel Feedback Queues

- קיימים מספר תורים לפי סדר עדיפות.
  - תור גבוה לתהליכים בעלי עדיפות גבוהה יותר.
  - לתורים הנמוכים מוקצה אחוז קטן יותר של זמן מעבד.
  - quantum גדול יותר לתורים נמוכים יותר.
- תהליך מתחיל בתור הגבוה ביותר.
- בסוף ה-quantum, יורד לתור נמוך יותר.
- אפליה מתקנת של תהליכים עתירי ק/פ לעומת תהליכים עתירי-חישוב
- אם תהליך משחרר המעבד לפני סוף ה-quantum (עקב פעולת קלט / פלט), הוא חוזר לתור גבוה יותר.

## זימון לפי עדיפויות

- התהליך עם העדיפות הגבוהה ביותר מקבל את המעבד.
- SJF הוא זימון לפי עדיפויות כאשר העדיפות היא ההופכי של זמן הביצוע.



הרעבה של תהליכים עם עדיפות נמוכה



הזדקנות (זמן שהייה ארוך) של תהליכים גורמת להגדלת העדיפות שלהם (לדוגמא, selfish round-robin).

## חישוב עדיפויות ב-UNIX

עדיפות תהליך בתחילת יחידת הזמן ה- $i$  היא:

$$P_j(i) = Base_j + NICE_j + \frac{CPU_j(i-1)}{2}$$

$$CPU_j(i) = \frac{1}{2}U_j(i) + \frac{1}{2}CPU_j(i-1)$$

$Base_j$  = Base priority of Process  $j$   
 $NICE_j$  = User controllable adjustment factor  
 $CPU_j(i)$  = Average processor utilization by  $j$   
 $U_j(i)$  = Processor utilization of  $j$  in interval  $i$

## זימון ב-UNIX

- זימון לפי עדיפויות
- עדיפות מספרית נמוכה = עדיפות טובה יותר
- חישוב העדיפות מתבסס על דעיכה אקספוננציאלית – תהליך שהשתמש לא מזמן במעבד מקבל עדיפות גבוהה (גרועה)
- ככל שעובר הזמן, עדיפותו של התהליך דועכת (משתפרת)
- לכן תהליכים שיוותרו מראשון על המעבד (עתירי ק / פ) יחזרו אליו מהר יותר מתהליכים שעברו הפקעה (עתירי חישוב).

## זימון ב Linux: רעיון בסיסי

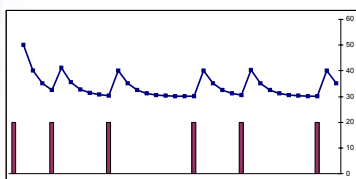
- גרסה נוספת של תורי עדיפויות
- נדון רק בזימון תהליכים רגילים
  - זימון תהליכי זמן-אמת (real time) יתואר בתרגול.
- בדומה ל-Unix, לכל תהליך יש עדיפות המורכבת מערך בסיס קבוע + בונוס דינאמי
- המתכנת יכול לשנות את ערך הבסיס באמצעות קריאת המערכת nice()
- הבונוס הדינאמי גדל כשתהליך חוזר מהמתנה וקטן (עד לערך שלילי) כאשר התהליך נמצא הרבה בטווח הקצר
- תהליך עתיר ק/פ צפוי לקבל עדיפות גבוהה יותר מתהליך עתיר חישוב בעל אותה עדיפות בסיס

## חישוב עדיפויות ב-UNIX

עדיפות תהליך בתחילת יחידת הזמן ה- $i$  היא:

$$P_j(i) = Base_j + NICE_j + \frac{CPU_j(i-1)}{2}$$

$$CPU_j(i) = \frac{1}{2}U_j(i) + \frac{1}{2}CPU_j(i-1)$$



$Base = 40$   
 $NICE = -10$

## זימון ב Linux: תהליכים חישוביים ואינטראקטיביים

- טיפול שונה:
  - תהליכים חישוביים רוצים הרבה זמן מעבד, אבל יכולים להכות
  - תהליכים אינטראקטיביים רוצים מעט זמן מעבד, אבל מייד
- רוצים לתת לתהליכים אינטראקטיביים כמה זמן שנחוץ
  - מחדשים להם את ה time slice לריצה נוספת בתקופה הנוכחית
- מאפיינים תהליך כאינטראקטיבי אם ממתין הרבה זמן מיוזמתו (בטווח הבינוני), כחלק מזמן הריצה הכולל שלו



פרטים נוספים בתרגול!

## זימון ב-Windows NT

- גם כאן, שימוש בתורים לפי עדיפות
  - 32 תורים (dispatcher ready queues)
    - Real time priority (עדיפויות 1-61)
    - Variable priority (עדיפויות 1-15)
    - (עדיפות 0 שמורה למערכת)
- מדיניות Round Robin
  - על התור העדיף ביותר שאינו ריק
- העדיפות יורדת אם נצרך כל ה-quantum
- העדיפות עולה אם תהליך עובר מ-wait ל-ready
  - העדיפות נקבעת על-פי סוג הקלט / פלט
  - ק"פ מהמקלדת מקנה עדיפות גבוהה

## תהליכים-דייט: חוטים

- מוטיבציה
- חוטי משתמש וחוטי מערכת
- תמיכת מערכת ההפעלה
- דוגמאות

## עלות ריבוי תהליכים

- תהליכים דורשים משאבי מערכת רבים
  - מרחב כתובות, גישה לקלט/פלט (file table)...
- זימון תהליכים הינו פעולה כבדה
  - context switch לוקח הרבה זמן.
- תקשורת בין תהליכים עוברת דרך מערכת ההפעלה
  - מערכת ההפעלה שומרת על ה"שערים" בחומות שבנתה בין תהליכים שונים

## טבעי לי מקבילי

- ...אך הרבה בעיות קל יותר לפתור באמצעות ריבוי תהליכים לדוגמא, שרת קבצים:
  - ממתין לקבל בקשה
  - כאשר מקבל בקשה, חייב להפסיק לחכות ולעבור לטפל בבקשה
- פתרון:
  - לבצע fork
  - תהליך הבן מטפל בבקשה שהגיעה
  - האב ממשיך להמתין לבקשה
- לא-יעיל כי מחייב הקצאת מרחב כתובות, PCB...

## שומרי משקל

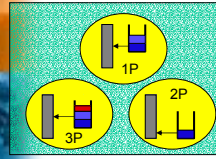
- התהליכים שנוצרים לטיפול בבקשות דומים זה לזה:
  - אותו קוד.
  - אותם משאבים ונתונים.
- אבל לא זהים:
  - מטפלים בבקשות שונות.
  - נמצאים בשלבים שונים במהלך הטיפול.

רעיון!

תהליכים-דייאט (lightweight processes) אשר משתפים מרחב כתובות, הרשאות ומשאבים

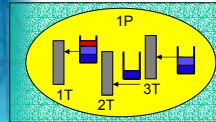
## תהליכים-דייאט = חוטים

- חוט (thread) הינו יחידת ביצוע (במקרה) בתוך תהליך



במערכות הפעלה קלאסיות "חוט" יחיד בכל תהליך

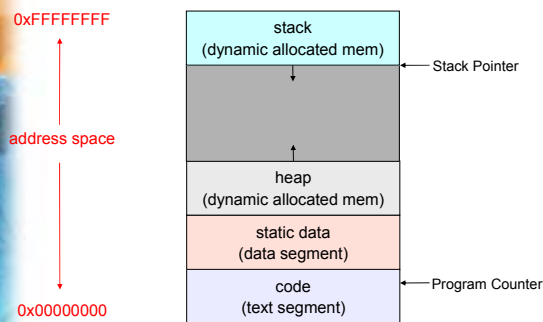
במערכות הפעלה מודרניות תהליך הוא רק מיכל לחוטים



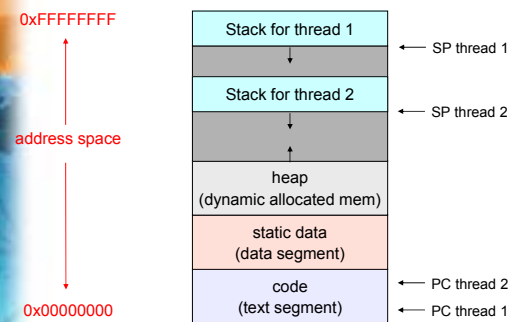
- לכל חוט דרושים המשאבים:

- program counter
- מחסנית
- רגיסטרים
- נמצאים ב Thread Control Block (TCB)

## תזכורת: מרחב הכתובות של תהליך



## מרחב הכתובות של תהליך מרובה-חוטים



## חוטים לעומת תהליכים

ייחודי לתהליך ייחודי לחוט

- Program Counter ✓
- Registers ✓
- Execution Stack ✓
- Address Space ✓
- Open Files ✓
- File position ✓

## שוב, שרת קבצים

- תהליך מקבל בקשות עם הפרמטרים הבאים:
  - סוג: קריאה / כתיבה
  - זיהוי קובץ
  - מיקום בקובץ
  - אורך
  - חוצץ (buffer)
- התהליך מחזיר:
  - מחזורת תווים (במקרה של קריאה)
  - סטאטוס (הצלחה/כישלון)
  - אורך קריאה/כתיבה (בפועל)

## שרת קבצים: מימוש עם חוט יחיד

```
do forever
 get request;
 execute request;
 return results;
end;
```

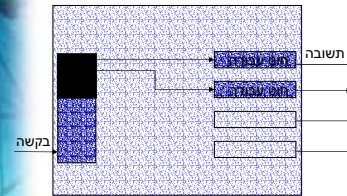
✓ פשטות.

✗ ניצול משאבים לקוי, למשל בהמתנה לקלט/פלט.

ניתן ליעל ע"י קלט / פלט אסינכרוני וטיפול במספר בקשות בו זמנית.

## מימוש באמצעות חוטים

- חוט מנהל
  - מקבל בקשה
  - מייצר חוט עבודה, ומעביר אליו את הבקשה.



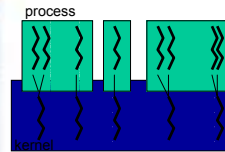
- חוט עבודה
  - מבצע את הבקשה
  - מחזיר תשובה

- ✓ תפוקה גבוהה יותר
- ✗ תכנות מורכב יותר

## יתרונות וחסרונות

- ✓ יצירת חוט יעילה יותר
  - רק יצירת thread control block והקצאת מחסנית
- ✓ ניצול טוב יותר של משאבים
  - למשל, במערכות מרובות מעבדים
- ✓ תקשורת נוחה יותר בין חוטים השייכים לאותו תהליך
  - זיכרון משותף
- ✓ תכנות מובנה יותר
  - ✗ חוסר הגנה בין חוטים באותו תהליך
  - חוט עלול לדרוס את המחסנית של חוט אחר
  - גישה לא מתואמת למשתנים גלובליים

## חוטי משתמש וחוטי מערכת



- חוט משתמש (user threads)
  - מוגדרים ע"י סביבת התכנות
  - לא דורשים קריאות מערכת
  - זימון בשיטות פשוטות
  - אין החלפת הקשר בגרעין
  - מה קורה כאשר חוט נחסם?

- חוט מערכת (kernel threads)
  - מוכרים למערכת ההפעלה
  - נקראים lightweight processes

חוטי משתמש מאפשרים לאפליקציה לחקות ריבוי חוטים גם במערכת הפעלה single-threaded.

## תמיכת מערכת הפעלה בחוטים

- יצירת והריסת חוטים, לדוגמה
 

```
thread_create(char *stack)
thread_exit(...)
thread_join(...)
thread_kill(...)
```
- מבני נתונים פנימיים
  - thread control block
  - Program counter, stack, registers
- מנגנוני סנכרון
  - למשל, תיאום גישה לזיכרון משותף

## דוגמא: חוטי POSIX

- ממשק סטנדרטי (IEEE 1003.1c) ליצירת וניהול חוטים.
  - רק application program interface.
- דומה מאוד לפעולות המתאמות עבור ניהול תהליכים:
 

```
pthread_create
pthread_exit
pthread_join
```
- כולל מנגנונים רבים לתיאום בין חוטים.
  - בקרוב מאוד!
- נתמך בהרבה מערכות "UNIX"
  - בפרט, Pthreads ב Linux.
  - מימושים שונים



## השוואת ביצועים

זמן יצירה / סיום:

|     |                               |            |
|-----|-------------------------------|------------|
| 251 | fork / exit                   | תהליכים    |
| 94  | pthread_create / pthread_join | חוטי גרעין |
| 4.5 | pthread_create / pthread_join | חוטי משתמש |

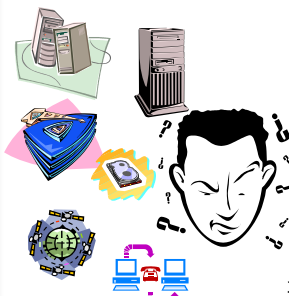
במיקרו שניות, על 700MHz Pentium, עם Linux 2.2.16  
[Steve Gribble, 2001]

## דוגמא: Windows NT

- היחידה הבסיסית הינה תהליך.
- תהליך יכול להכיל כמה חוטים – kernel threads.
- חוט יכול להכיל כמה סיבים (fibers), שהם בעצם user threads
- זימון נעשה ברמת החוטים
- תהליך רק מגדיר את מרחב הזיכרון, ומהווה מיכל לחוטים

## מדוע מערכות הפעלה?

### משאבים קיימים



- **כח חישוב (מעבד)**
  - ביצוע סידרתי של פקודות, מס' משתנה של מעבדים
- **זיכרון**
  - משאב סופי מוגבל
- **קלט/פלט, למשל דיסקים (אחסון נתונים)**
  - מערך גדול מאוד של בתים
- **רשת**
  - תשתיות שונות, עם הבטחות שונות של זמן, אמינות וכד'

## מדוע מערכות הפעלה?

### הפשטת משאבים

- **כח חישוב**
  - לכל משתמש ו/או אפליקציה, מעבד "פרטי"
- **זיכרון**
  - לכל תהליך זיכרון פרטי (גדול יותר מהמשאב הפיסי)
- **דיסקים (אחסון נתונים)**
  - מדריכים, מספר שרירותי של קבצים "אינסופיים"
- **רשת**
  - ערוץ "פרטי" בין שני תהליכים, הבטחת סדר, אמינות

## הפשטת משאבים (המשך) מחיר: ביצועים, ניצולת

- **כח חישוב**
  - זימון תהליכים
- **זיכרון**
  - ניהול זיכרון וירטואלי
- **דיסקים (אחסון נתונים)**
  - ניהול מע' קבצים, זמן גישה, ...
- **רשת**
  - headers, יתירות, המרת פרוטוקולים

## הפשטת משאבים (המשך) ... אבל זה כדאי ...

- **סדר, מודולריות**
- **ניצולת, ניהול יעיל של משאבים**
  - מהי ניצולת המעבד במחשב אישי?
- **הגנה**
  - בין משתמשים, בין תהליכים
- **הסתרת תקלות**
  - סקטור מקולקל בדיסק, איבוד מנות תקשורת
- **הבטחת מידע**
- ...

## מערכות הפעלה הפשטת משאבים כסל שרותים

- **זיכרון**
  - זיכרון וירטואלי, הקצאת זיכרון, איסוף אשפה
- **מע' קבצים**
  - ניהול קבצים, ספריות, יחידות אחסון
- **רשתות תקשורת**
  - תקשורת בין קצוות
  - ניצול משאבים מרוחקים
- **מקביליות**
  - תהליכים וחוסים
- **תיאום**
  - Semaphores, מנעולים
- **קלט/פלט**
  - הפשטה ע"י Device Drivers

## היסטוריה

- **חומרה יקרה ואיטית, כח-אדם זול**
  - Batch jobs, ניצול החומרה :24x7 IBM S/360
- **חומרה יקרה ומהירה, כח-אדם זול**
  - Unix, TSO :Interactive time-sharing
- **חומרה זולה ואיטית, כח-אדם יקר**
  - מחשב אישי לכל משתמש: MS-DOS
- **חומרה זולה מאוד, כח חישוב רב**
  - ריבוי משימות: Windows NT, OS/2
  - שיתוף משאבים בסיסי: דיסקים, מדפסות, ...
- **רשתות מהירות**
  - הרשת היא המחשב: SETI@home, Grid Computing