Shell, Basics, pipes and redirection

תרגיל 1

מה זה Shell?

ה- Shell הינו תוכנית מאפשרת למשתמש עבודה ב- Live mode מול מערכת הפעלה. כמו הרצת תוכנית, ניהול על הספריות שעל המחשב, יצירת קבצים והצגת פרטים עליהם, וכו'.

בעבר עבודה מתוך Shell היתה הדרך העיקרית לביצוע משימות אלו. בעשור האחרון התפתחו בממשק רבה ממשקים גראפיים לביצוע אותן הפעולות. למשל, X windows במערכות linux, unix. אך השימוש ב- Shell עדיין נפוץ מאוד ולעתים יש נוח יותר. למשל, כשמתחברים למחשב מרוחק שקו התקשורת אינו איטי.

ישנם סוגים שונים של Shell. אותנו נעבוד עם Shell שנקרא bash.
ניתן לפתוח מספר חלונות console במקביל.

ספריות וקבצים:

כדי להציג את שמות הקבצים בספרייה הנוכחית:

1. נקיש ls -l, הפרמטר "-l" יראה רשימה (list) עם נתונים על כל הקובץ.

2. על המסך תופיע רשימת הקבצים והספריות ב- Home directory:

```
$ ls -l
```

```
total 4
drwx------   3 artzi   math1      4096 Jan 26 18:02 Desktop
```

מייצג ספרייה directory.
מייצג את הרשאות שניתנו לקובץ
מייצג קובץ רגיל.

- יש אם לא הכנסת הראשונה שלך, לעתיד, סביר להניח שיהיו ב- Home directory שלך מספר קבצים שנוצרים באופן אוטומטי לכל משתמש חדש.

המשך ...

אין צורך, אבל כדאי להבין אותן.

התווים הבאים אחריו (בדוגמא שלנו rwx------) מייצגים את ההרשאות שניתנו לקובץ, נרחיב עליהם מאוחר יותר.

בטור השני מופיע מספר (בדוגמא שלנו 3) שמסמן את מספר ההצבעות שיש אל הקובץ (בשלב זה לא נדבר על הנושא הזה).

לכל קובץ ב- linux יש שם משתמש שהוא הבעלים של הקובץ, ושם קבוצת משתמשים שהיא הקבוצה של הקובץ. בד"כ מדובר בשם המשתמש שיצר את הקובץ, ובקבוצת המשתמשים שלו. השיוך הזה משמעותי בעיקר לצרכי הרשאות גישה לקובץ.

בטור השלישי והרביעי מופיעים שם המשתמש שהוא הבעלים של הקובץ ושם הקבוצה של הקובץ (בדוגמה שלנו artzi ו- math1).

בטור החמישי יופיע גודל הקובץ בבתים (בדוגמא שלנו 4096).

בטור השישי יופיע תאריך השנוי האחרון של הקובץ (Jan 26 18:02).

ובטור השישי מופע שם הקובץ (Desktop).

בקרות בסיסיות:

שם הפקודה	מה עושה הפקודה
ls -l	מציגה את תוכן הספרייה הנוכחית
pwd	מציגה את שם הספרייה הנוכחית
cd dirname	נכנסת לספרייה dirname
cd ..	עולה לספריית האב של הספרייה הנוכחית
cd	חוזרת לספרייה הבית של המשתמש
mkdir dirname make	יוצרת ספרייה חדשה בשם dirname
rmdir dirname remove	מוחקת את הספרייה ששמה dirname (הספרייה צריכה להיות ריקה לפני המחיקה).

cp src dst	מעתיקה את הקובץ src לקובץ חדש בשם dst.
rm filename	מוחקת את הקובץ filename
mv src dst	משנה את שם הקובץ src לשם dst
cat filename	מציג את תוכן הקובץ filename
man command	מציג תיעוד על הפקודה command (מה היא עושה, פרמטרים, וכד'). ליציאה ממסך העזרה יש להקיש q.

הערה: כל הפקודות ב- shell הן case-sensitive.

```
sara@sara-VirtualBox:~$ ls
Desktop    Downloads  Pictures  snap       Videos
Documents  Music      Public    Templates
sara@sara-VirtualBox:~$ mkdir Food
sara@sara-VirtualBox:~$ ls
Desktop    Downloads  Music     Public  Templates
Documents  Food       Pictures  snap    Videos
sara@sara-VirtualBox:~$ cd Food
sara@sara-VirtualBox:~/Food$ touch fruit vegetable
sara@sara-VirtualBox:~/Food$ ls
fruit  vegetable
sara@sara-VirtualBox:~/Food$
```

ls → open the current folder

mkdir food → make new folder named Food

cd food → open the folder named food

touch fruit vege.... → make new file named fruit vege....

```
Desktop    Downloads  Pictures  snap       Videos
Documents  Music      Public    Templates
sara@sara-VirtualBox:~$ mkdir Food
sara@sara-VirtualBox:~$ ls
Desktop    Downloads  Music     Public  Templates
Documents  Food       Pictures  snap    Videos
sara@sara-VirtualBox:~$ cd Food
sara@sara-VirtualBox:~/Food$ touch fruit vegetable
sara@sara-VirtualBox:~/Food$ ls
fruit  vegetable
sara@sara-VirtualBox:~/Food$ cd~
Command 'cd~' not found, did you mean:
  command 'cdp' from deb irpas (0.10-9)
  command 'cdi' from deb cdo (2.0.4-1)
  command 'cdw' from deb cdw (0.8.1-2)
  command 'cde' from deb cde (0.1+git9-g551e54d-1.2)
  command 'cd5' from deb cd5 (0.1-4)
  command 'cdb' from deb tinycdb (0.78build3)
  command 'cdo' from deb cdo (2.0.4-1)
Try: sudo apt install <deb name>
sara@sara-VirtualBox:~/Food$ cd..
cd..: command not found
sara@sara-VirtualBox:~/Food$ cd ..
sara@sara-VirtualBox:~$
```

cd~ } לא עובד
cd.. } linux לא

cd .. → back to the main computer folder

↓

בדרך רווח.
מנידן.

```
Command 'cd~' not found, did you mean:
  command 'cdp' from deb irpas (0.10-9)
  command 'cdi' from deb cdo (2.0.4-1)
  command 'cdw' from deb cdw (0.8.1-2)
  command 'cde' from deb cde (0.1+git9-g551e54d-1.2)
  command 'cd5' from deb cd5 (0.1-4)
  command 'cdb' from deb tinycdb (0.78build3)
  command 'cdo' from deb cdo (2.0.4-1)
Try: sudo apt install <deb name>
sara@sara-VirtualBox:~/Food$ cd..
cd..: command not found
sara@sara-VirtualBox:~/Food$ cd ..
sara@sara-VirtualBox:~$ cd Food
sara@sara-VirtualBox:~/Food$ rm fruit
sara@sara-VirtualBox:~/Food$ rm vegetable
sara@sara-VirtualBox:~/Food$ ls
sara@sara-VirtualBox:~/Food$ rmdir Food
rmdir: failed to remove 'Food': No such file or directory
sara@sara-VirtualBox:~/Food$ cd ..
sara@sara-VirtualBox:~$ rmdir Food
sara@sara-VirtualBox:~$ ls
Desktop    Downloads  Pictures  snap       Videos
Documents  Music      Public    Templates
sara@sara-VirtualBox:~$
```

cd food → open the food folder

rm fruit → remove file named fruit

```
sara@sara-VirtualBox:~$ mkdir newD
sara@sara-VirtualBox:~$ ls
Desktop    Downloads  newD      Public  Templates
Documents  Music      Pictures  snap    Videos
sara@sara-VirtualBox:~$ cd newD
sara@sara-VirtualBox:~/newD$ touch pandora
sara@sara-VirtualBox:~/newD$ ls
pandora
sara@sara-VirtualBox:~/newD$  cp pandora pandora_rule
sara@sara-VirtualBox:~/newD$ ls
pandora  pandora_rule
sara@sara-VirtualBox:~/newD$ mv pandora original
sara@sara-VirtualBox:~/newD$ ls
original  pandora_rule
sara@sara-VirtualBox:~/newD$ cd original
bash: cd: original: Not a directory
sara@sara-VirtualBox:~/newD$ cat original
sara@sara-VirtualBox:~/newD$ ./original.txt
bash: ./original.txt: No such file or directory
sara@sara-VirtualBox:~/newD$ ./original
bash: ./original: Permission denied
sara@sara-VirtualBox:~/newD$ hghhd
hghhd: command not found
```

touch pandora → create a file named pandora in the NewD Folder.

copy pandora file to file pandora_rule

move

original לא מכיל תוכן מ"ג"ב

↓

File

```
sara@sara-VirtualBox:~/newD$ cat output
sara@sara-VirtualBox:~/newD$ echo "out is very important" <
output
out is very important
sara@sara-VirtualBox:~/newD$ cat output
sara@sara-VirtualBox:~/newD$
```

echo → מדפיס ישיר על המסך

* echo " ..... " | head
* echo " ..... " | more

בקובץ

* data | echo בקובץ שמייבית את תאריך על המסך.

create File example →

כתוב בקובץ את ab cd

מ"ג"ב את תוכן שלהן.

```
touch example.txt
ls
echo "a b c d" < example.txt > pandora
cat example.txt
cat pandora
```

↓

```
example.txt
a b c d
a b c d
```

תרגיל 2

Pipes:

כשמריצים תוכנית, הפלט שלה מופיע על המסך. ה- Shell מאפשר לשנות את ההתנהגות הזו, ולהפנות את הפלט למקום אחר (כקלט לתוכנית אחרת, למשל).

כדי להעביר פלט תוכנית כקלט לתוכנית אחרת, נכתוב prog1 | prog2 (כש- prog1 היא התוכנית שרצה ראשונה, ו- prog2 היא התוכנית שרצה שנייה, ומקבלת כקלט את פלט התוכנית הראשונה).

לדוגמא, התוכנית head מחזירה את 10 השורות הראשונות שהיא קיבלה בקלט. ולכן הפקודה ls –l | head תחזיר את 10 השורות הראשונות של הפלט של ls –l.

Redirection:

ניתן גם להפנות את הפלט של תוכנית לתוך קובץ. כדי להעביר פלט תוכנית לתוך קובץ (במקום למסך), נכתוב prog > filename (כאשר prog הוא שם התוכנית, ו- filename הוא שם הקובץ בו נרצה לשמור את הנתונים).

לדוגמא, כדי לשמור את הפלט של ls –l בקובץ output, נרשום ls –l > output.

Nano filename, מאפשר לנו לכתוב ישירות לתוך הקובץ. כלומר אותה לנו תיבת טקסט כמו בכתיבה לתוך הקובץ.

תיעוד:

את המדריך למשתמש של כל הפקודות שהצגנו (וכל פקודה אחרת במערכת) ניתן להציג באמצעות הפקודה man. למשל, man ls יציג תיעוד מפורט של הפקודה ls .

* ls > filename ← מעבירה את כל מה שיש בקובץ הנוכחי שרץ לבפל'ס באמצעות ls לתוך filename.

* head -n 5 filename

- head ← נותנת לנו לראות את התחלת הקובץ.
- -n ← אומר לו מאיפה שורה להתחיל
- 5 ← מספר שורות שרצים לקרות
- filename ← שם הקובץ שלה תתבצע הפקודה.

* tail -n 2 filename

נותן לנו את הסוף של הקובץ

אותו הדבר

* tail -n +3 filename

לדפדפ מ שורה 3 והלאה בקובץ.

* tr לפרוץ פרמטרים בקובץ

לפורמט.

1) tr '[:lower:]' '[:upper:]' < filename > result ← קובץ חדש שיש בה את תוצאת הרצת

תחליף כאן את כל האותיות קטנות לאותיות גדולות שיש בקובץ filename ותשמור אותן בקובץ result.

2) tr -d 'A' < result > result2

מחיקה — את הדבר שאנחנו רוצים למחוק.

תמחוק את 'A' שבקובץ result ותשמור את התוצאה ב - result2.

לא להשתמש ב- NANO

31.צור קובץ בעל תוכן ארוך בשם longfile (ללא כתיבה ידנית לקובץ).
א. הדפס על המסך את 5 השורות הראשונות של הקובץ.
ב. הדפס על המסך את 5 השורות האחרונות של הקובץ.
ג. הדפס על המסך את כל הקובץ החל משורה 5.

תזכורת:

man command	מציג תיעוד על הפקודה command (מה היא עושה, פרמטרים, וכד'). ליציאה ממסך העזרה יש להקיש q.

```
localhost:~# ls > longfile
localhost:~# cat longfile
bench.py
hello.c
hello.js
longfile
readme.txt
localhost:~#
```

במקום לבדוק אותן למסך ע"ם ">" מכניס אותן לקובץ הדומה שנוצר longfile

* אם כבר קיים קובץ בשם longfile אזי > מוחק אותו ומושך את קובץ החדש.

```
localhost:~# touch file1 file2 file3
localhost:~# ls
bench.py    file2       hello.c     longfile
file1       file3       hello.js    readme.txt
localhost:~# pwd
/root
localhost:~# head -n 5 longfile
bench.py
hello.c
hello.js
longfile
readme.txt
localhost:~# head -n 2 longfile
bench.py
hello.c
localhost:~#
```

(ב) 5

(א) 2

```
localhost:~# tail -n +3 longfile
hello.js
longfile
readme.txt
localhost:~#
```

(ג)

32.שנה את הקובץ longfile כך שיהיה כתוב כולו באותיות גדולות בלבד CAPITALS.

```
localhost:~# tr '[:lower:]' '[:upper:]'< longfile > result
localhost:~# cat result
BENCH.PY
HELLO.C
HELLO.JS
LONGFILE
README.TXT
```

33.מחק כל האותיות A מן הקובץ. בדוק שהן אכן נמחקו!! (איך תבדוק?)

```
README.TXT
localhost:~# tr -d 'A' < result > result2
localhost:~# cat result2
BENCH.PY
HELLO.C
HELLO.JS
LONGFILE
REDME.TXT
```

34. מהו התוכן של a1,a2,a3,a4 לאחר ביצוע הפקודות הבאות? נסה והסבר!

```
$ date > a1
$ pwd > a2
$ ln -s a1 a4
$ who >> a2
$ ln a2 a3
$ pwd >> a3
$ who > a1
```

מכ"ל את התאריך הנוכחי ומכניס אותו ל- a1.

מכ"ל את Road של הקובץ ומכניס אותו ל- a2.

עושה לנו קישור (סומלת על ל-) בין a1 ל- a4.

who - מראה איך משתמש משתמש ברשיון בטרמינל.

>> - עושה "append", כלומר לא פורס את הקובץ רק מוסיף לקובץ.

קישור קשיח

```
localhost:~# echo "append" >> result2
localhost:~# cat result2
BENCH.PY
HELLO.C
HELLO.JS
LONGFILE
REDME.TXT
append
```

פוינטאו

37. piping ו redirection:

א. כתוב פקודה השומרת את רשימת שמות התיקיות תחת תיקיית המשתמש הראשית אל תוך קובץ הנקרא folders.list.

ב. מיין הקובץ folders.list שיצרת? (רמז: היזכר בפקודה sort).

ג. כיצד תבצע א+ב ללא שימוש בקובץ? כלומר, למיין רשימת התיקיות, ולהדפיסה על המסך (ללא יצירת אף קובץ)?

ד. כיצד תשמור הפלט של ג' בתוך קובץ folders.list1? השתמש ב redirection!

ה. כיצד תשמור הפלט של ג' בתוך קובץ מבלי לדרוס תוכן הקובץ? כלומר להוסיפו בסוף הקובץ.

ו. כיצד תוכל להשתמש בפקודה echo על מנת להדפיס לתוך קובץ?

א

```
localhost:~# ls
a1          a4          file1       food        longfile    result2
a2          bench.py    file2       hello.c     readme.txt
a3          file        file3       hello.js    result
localhost:~# ls > folders.list
localhost:~# cat folders.list
a1
a2
a3
a4
bench.py
file
file1
file2
file3
folders.list
food
hello.c
hello.js
longfile
readme.txt
result
result2
localhost:~#
```

ב

```
localhost:~# sort folders.list
a1
a2
a3
a4
bench.py
file
file1
file2
file3
folders.list
food
hello.c
hello.js
longfile
readme.txt
result
result2
```

ג

```
localhost:~# ls | sort | echo

localhost:~# ls | sort | cat
a1
a2
a3
a4
bench.py
file
file1
file2
file3
folders.list
food
hello.c
hello.js
longfile
readme.txt
result
result2
```

ד

עושינו את זה באמצעים "|" "|"

בפעם אחת ← פורס הקובץ

בפעמים ← לא פורס הקובץ

ה

```
localhost:~# ls | sort | cat > final
localhost:~# cat final
a1
a2
a3
a4
bench.py
file
file1
file2
file3
final
folders.list
food
hello.c
hello.js
longfile
readme.txt
result
result2
```

```
localhost:~# history
```

39. היסטוריה של הפקודות:
   א. הדפס על המסך את רשימת הפקודות האחרונות שהשתמשת בהן.
   ב. הפעל שוב את הפקודה האחרונה (ללא שימוש בחצים)
   ג. הפעל את הפקודה החמישית.
   ד. הפעל פקודה שמתחילה ב ec.
   ה. בדוק כמה מלים מכילה רשימת הפקודות האחרונות שלך (ה-history). השתמש בpiping.

LinuX Cat -i command

```
#!/bin/bash

echo "this is the first parameter: $1"

name=50
 echo "name is : $name"

if [ "$1" -gt "$2" ]; then
echo " the first parameter is greater than the second"
 else
 echo "the first parameter is smaller than the second"
fi


for item in "$@"; do
echo "Item: $item"
done
```

```
localhost:~# chmod +x a.sh
```

```
localhost:~# ./a.sh 15 20
```

תרגול 3
16/4/2024

תהליכים Process

תהליכים: הם אבן יסוד של מערכת הפעלה.

מערכת הפעלה חייבת לנהל תהליכים: 1) הקצאות משאבים.
2) ניהול מצב התהליך הנוכחי.
3) איחסון תהליכים במבני נתונים.
4) סינכרון בין תהליכים.
5) שיתוף מידע בין תהליכים.

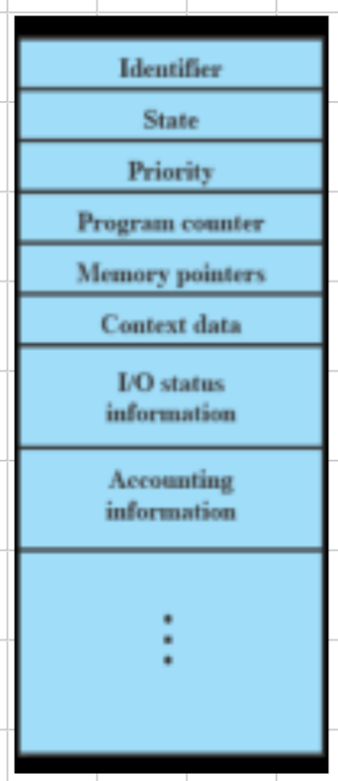
מופע/הרצה של תוכנית/אפליקציה = תהליך.

מימוש תהליכים

- מאחורי הקופסא השחורה שנקראת תהליך:
  - קוד לביצוע
  - סט נתונים/מידע לעיבוד
  - מצב התהליך ברגע נתון

- מ"ה מחזיקה עבור כל תהליך מידע קריטי המאפשר ניהול תהליכים במקביל. למשל:
  - מזהה
  - מצב נוכחי
  - עדיפות
  - Program counter
  - ועוד...

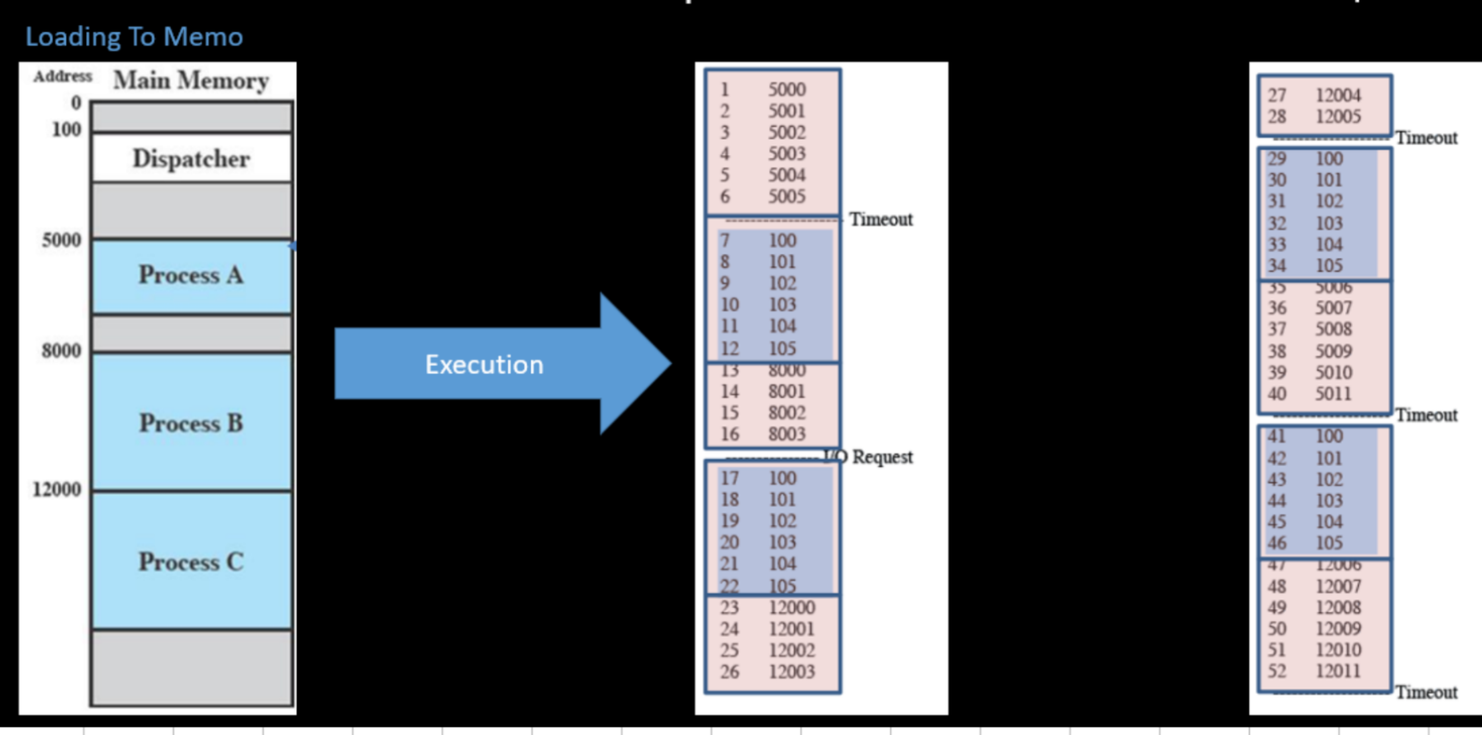
- המידע עבור תהליך מאוחסן ב Process Control Block
- מזכיר מעט את מערכת הקבצים...



- מערכת ההפעלה מאפשרת הרצה של יותר מתוכנית אחת במקביל
- האם באמת במקביל? לא
- מי קובע כמה זמן מעבד לכל תהליך? איך נקבע? מ"כ
  במפלך בקורס נפגש.
- Time Slice – כמות זמן המעבד/ריצה שמ"ה מקצה לתהליך

מימוש תהליכים

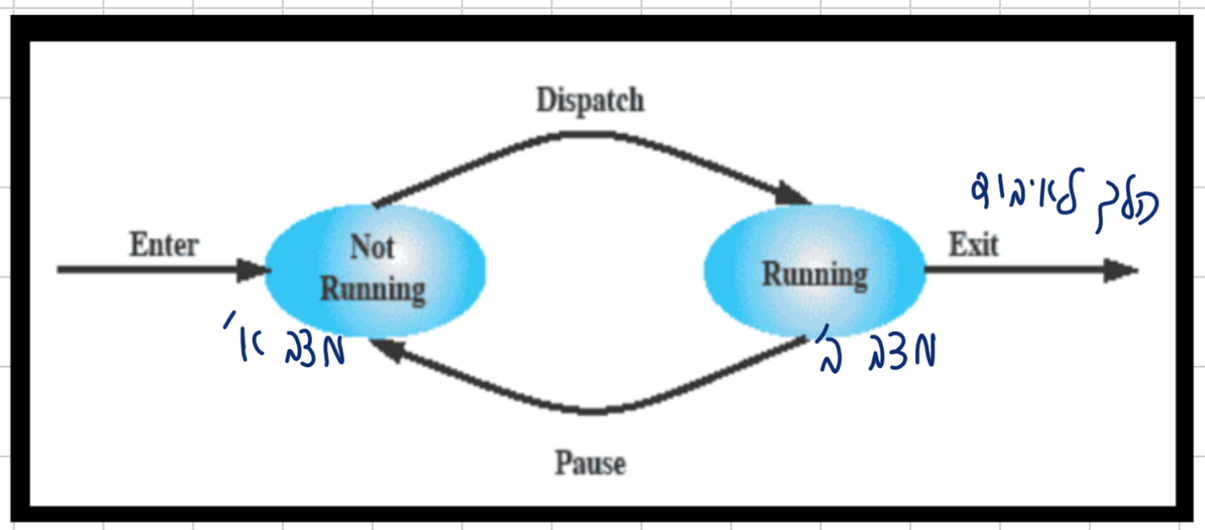
Dispatcher – תוכנית המאפשרת context switch בין תהליכים



ניהול תהליכים

Two - State Model

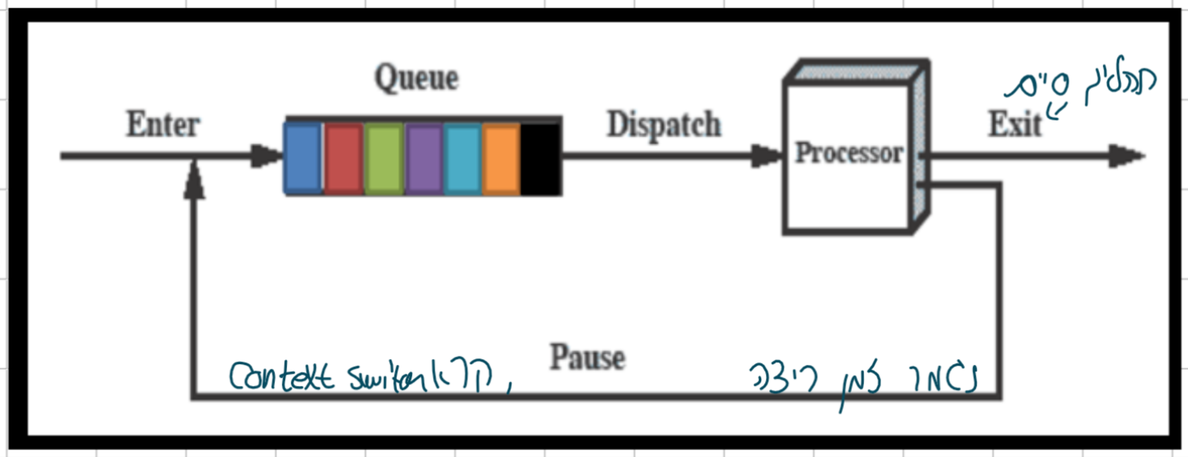
שני מצבים בסיסיים בלבד



חסר לנו כאן, מצבי ביניים, יעילות IO למשל נחשב לא"י הרבה יותר מבחינה ולא נרצה לבזבז זמן מעבד בעמן המתנה ל- IO.

ניתן לממש באמצעות תור.

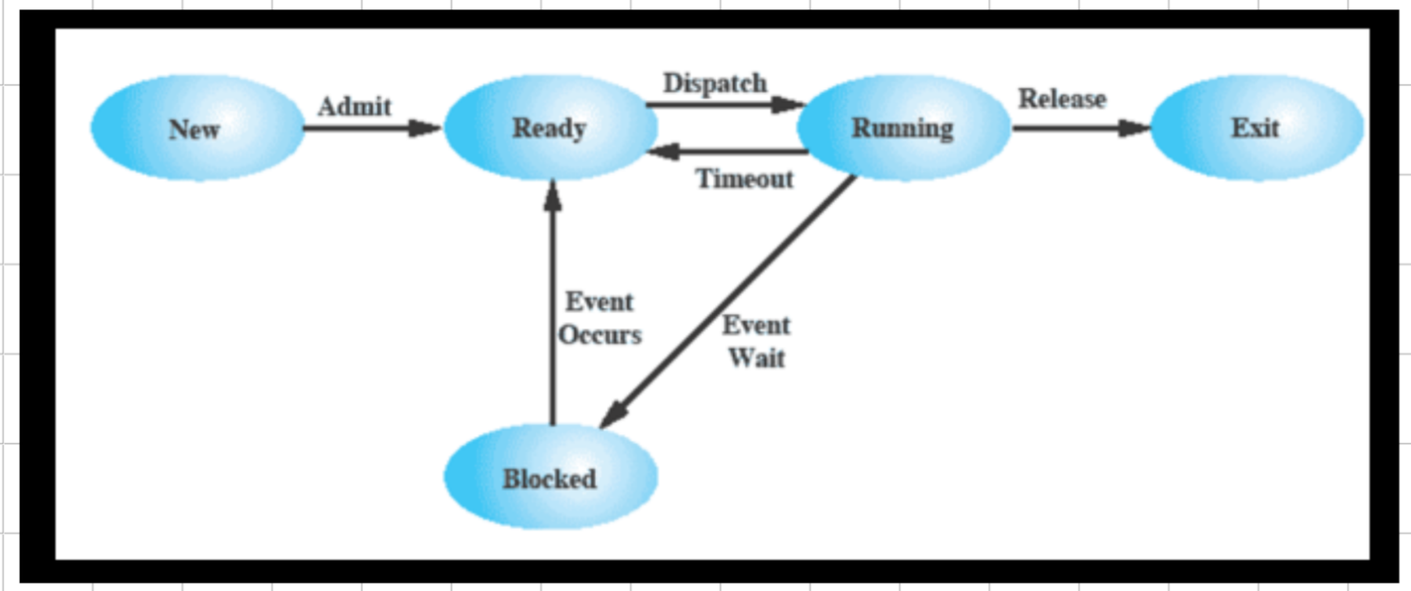
יתכן יותר מתור Ready אחד, או- תור עדיפויות (לפי תהליך במערכת יש עדיפות).



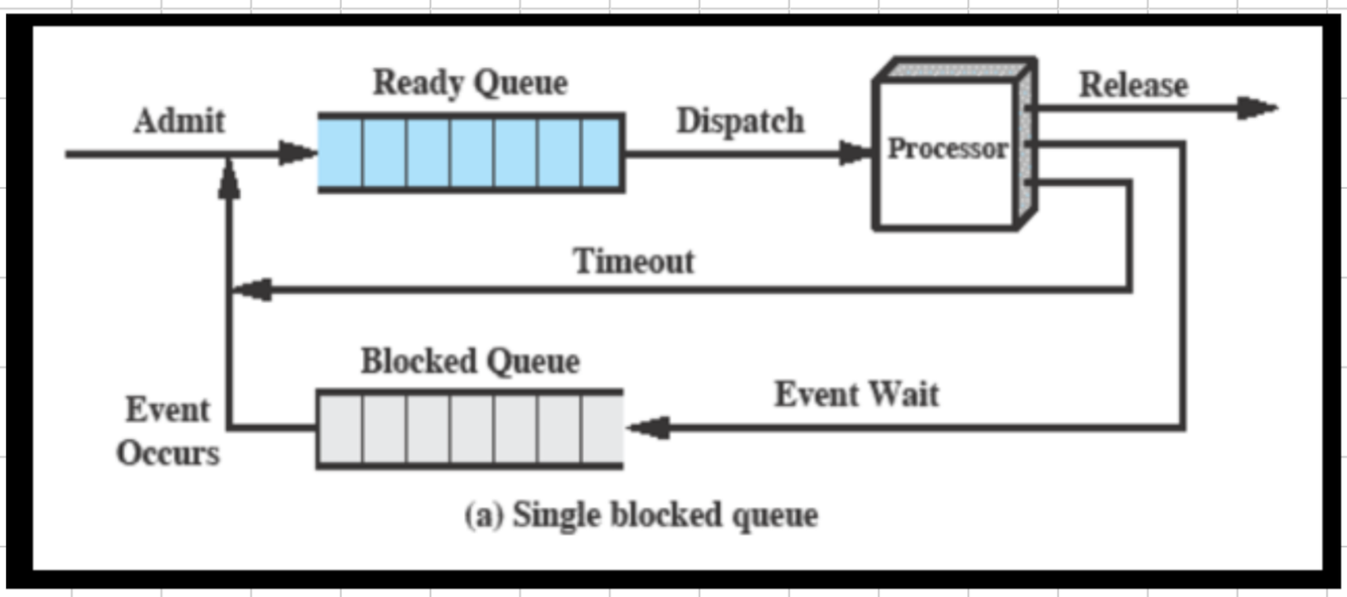
Five - State model

מודל משודרג עם מצבי ביניים:

New → Ready → Running → Blocked → Exit/Terminated

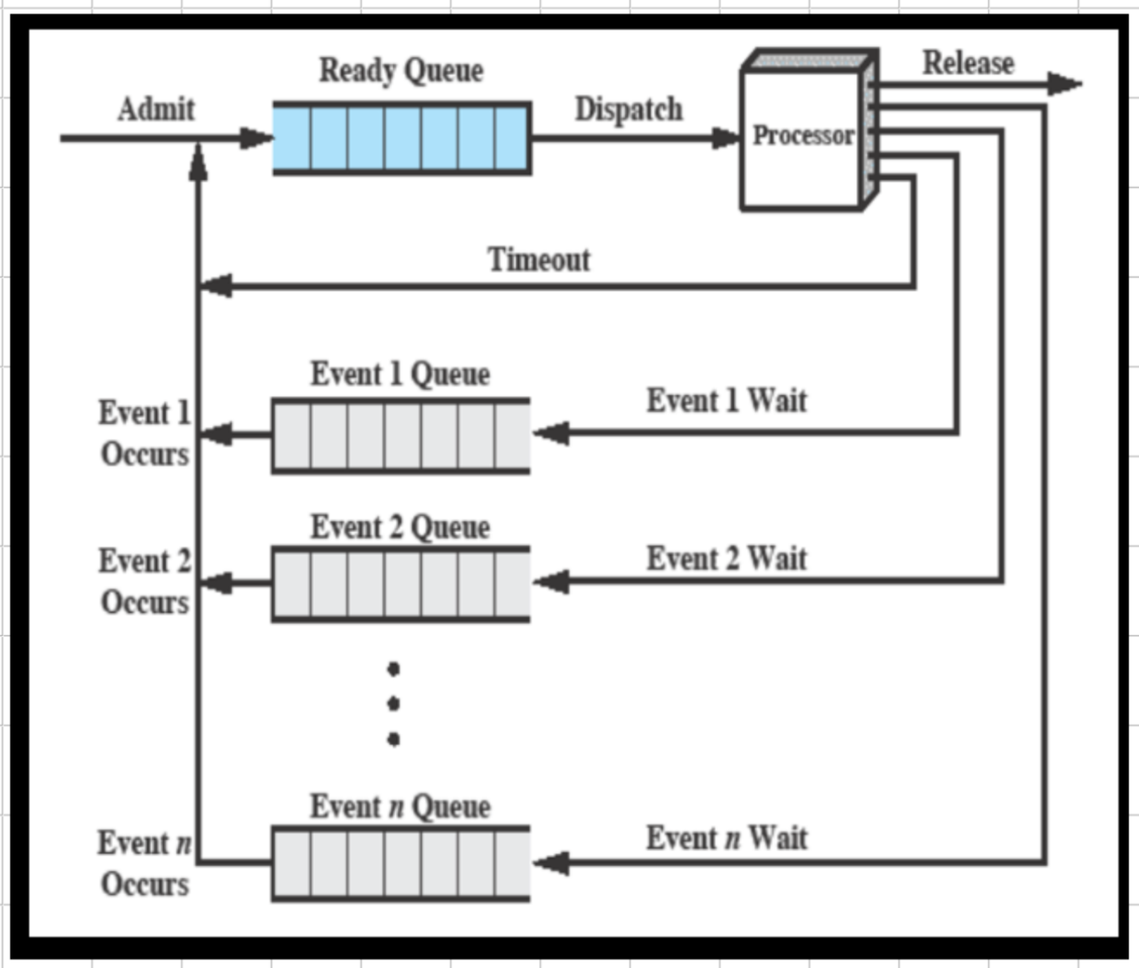


תהליך נוצר → Admit
ח"י אותם עדיין לא בועלם/לא עקה מכרון.



(a) Single blocked queue

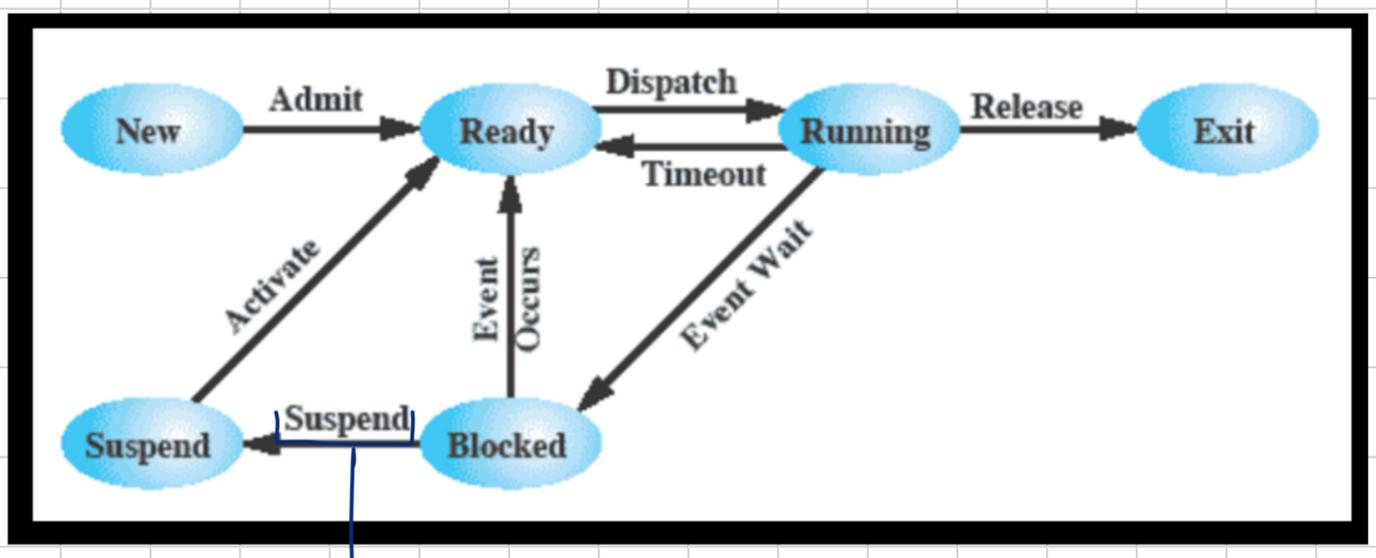
Multi-State model



אין מספר כמה רובה.
אבל לוקח מקום מהזיכרון overhead

לכל event יש תור משלו.

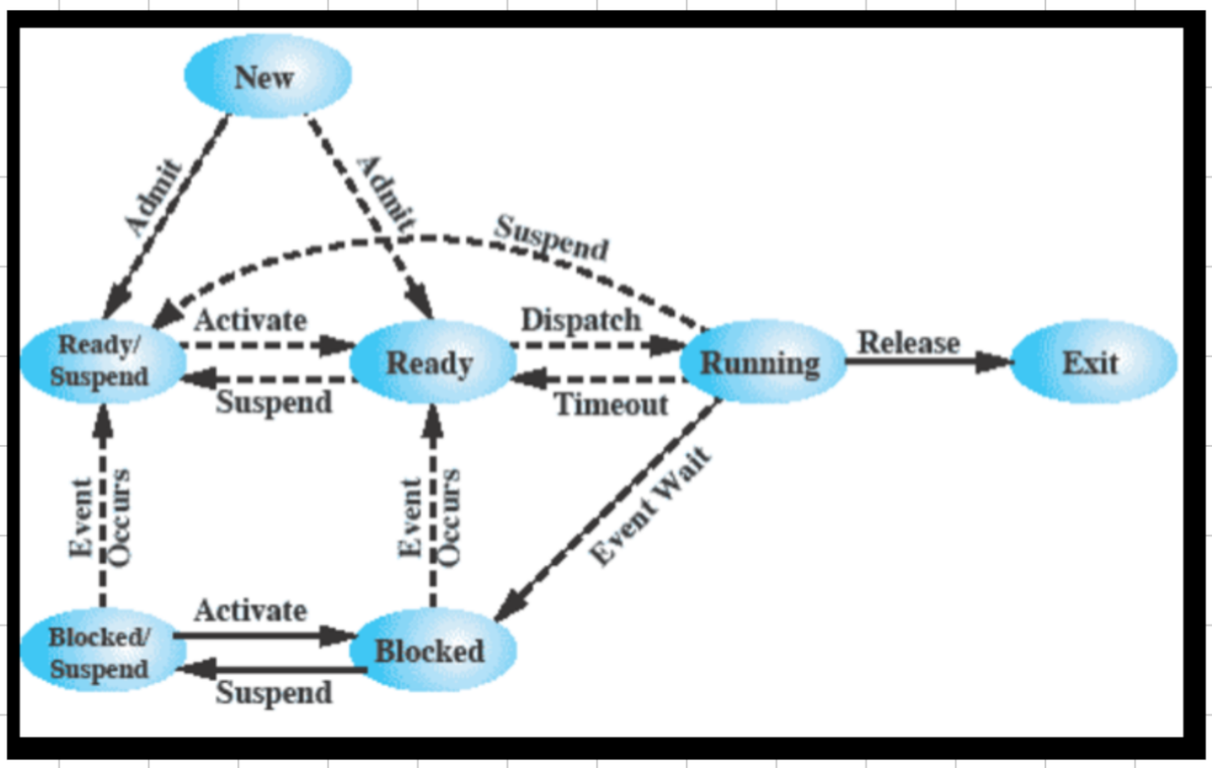
One Suspend-State model



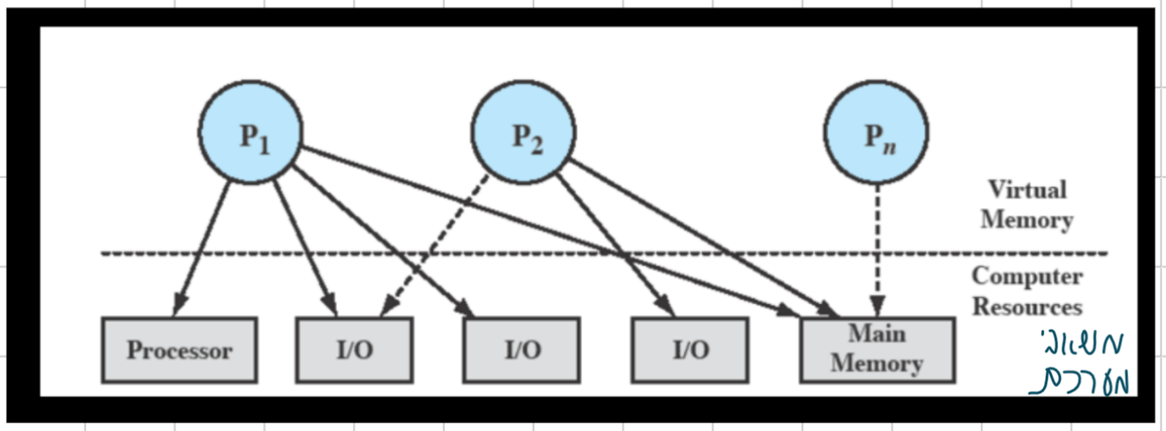
* פעולת תהליך לזיכרון המשני (Disk)
* מאפשר פינוי מקום בזיכרון הראשי (RAM) לטובת תהליכים נוספים.

Two Suspend-State Model

כמו שיתכן יותר מתור אחד לכל מצב.
גם מסלולים יכולים לתקוע על עצמם.
למשל, יש לנו כאן שתי מסלולי Admit.



משאב: מערכת

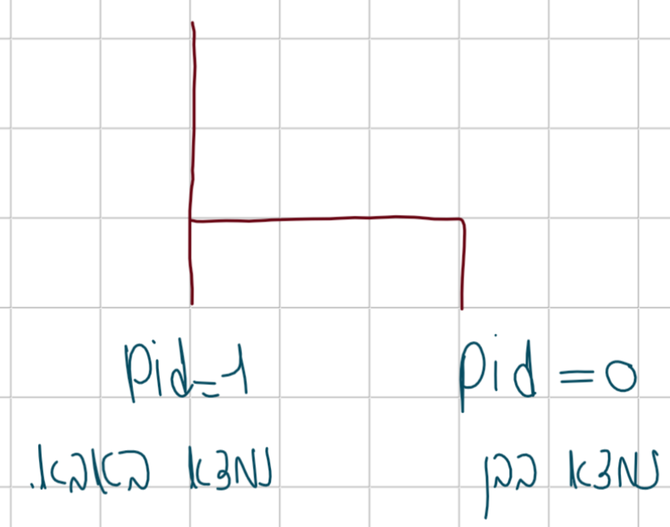


* כל תהליך נפרס למשאב מערכת שונה במהלך חייו
* תהליך יכול לבקש יותר ממשאב אחד בו זמנית.
* המיפוי אופולי מבה המערכת בזמן מסוים (מבחינת משאבים ותהליכים) מאוחסן ומנוהל ע"י מ"ה (עשיו עדיות מאוחסן במבנה מלבה).
* מ"ה חייבת לשמור על בירורות בזמן פרדה מקבילית.

Processes API

- abort() – parent can abort child process. Cause abnormal process termination
  - Child process uses too many resources
  - Child task is no longer relevant
- exit() - causes normal process termination and the value of *status & 0377* is returned to the parent
- waitpid(child_process_id, ...) – block function, wait for child process to complete
  - Similar method from threads API?
    - join()
- exec("/path/to/a/program", argv) – execute program
- fork() - creates a new *child* process by duplicating the calling process. The new process, referred to as the *child*, is an exact duplicate of the calling process (except for some variables).

- fork() return values:
  - Val < 0 //fork failed. Common cause for that is not enough memory left for new process
  - Val = 0 //we are in the child process
  - Val > 0 //success. value represents the child PID



- fork() workflow demonstration (on whiteboard)
- fork() execution demo (on whiteboard)

- Why fork()?
  - Your shell uses fork to run the programs you invoke from the command line.
  - Web servers like apache use fork to create multiple server processes, each of which handles requests in its own address space. If one dies or leaks memory, others are unaffected, so it functions as a mechanism for fault tolerance.
  - Google Chrome uses fork to handle each page within a separate process. This will prevent client-side code on one page from bringing your whole browser down.

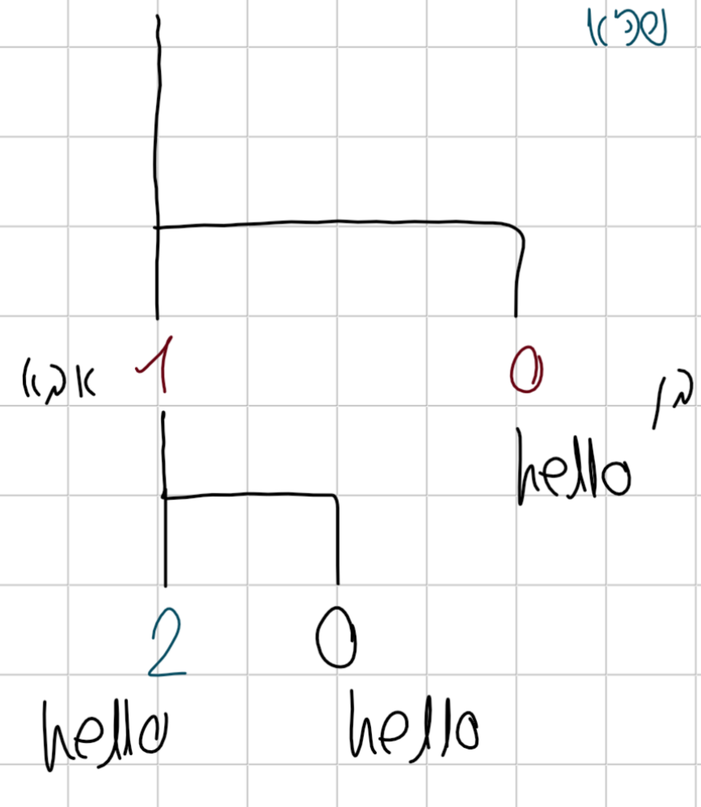
דוגמא 1:

```c
void main()
{
    int pid = fork();
    if (pid > 0)
    {
        fork();
        printf("hello");
    }
    printf("hello");
}
```

א. בהינתן התוכנית הבאה, כמה פעמים יודפס "hello" ?
ב. האם ניתן לדעת מהו סדר ההדפסה?

5 פעמים יודפס hello.

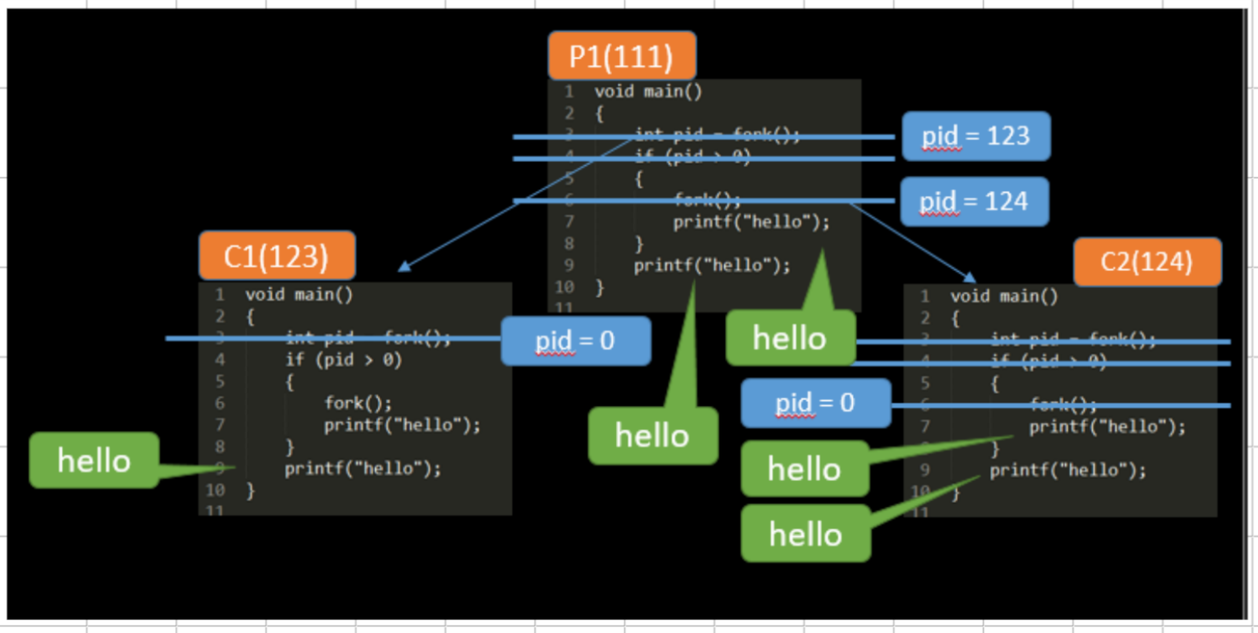
לא, תלוי במתרכת ההפעלה וזמן הריצה שכיו נתונים לתהליך.



hello ← שנמצא בבלוק if.

hello ← שנמצא בבלוק אחשו.

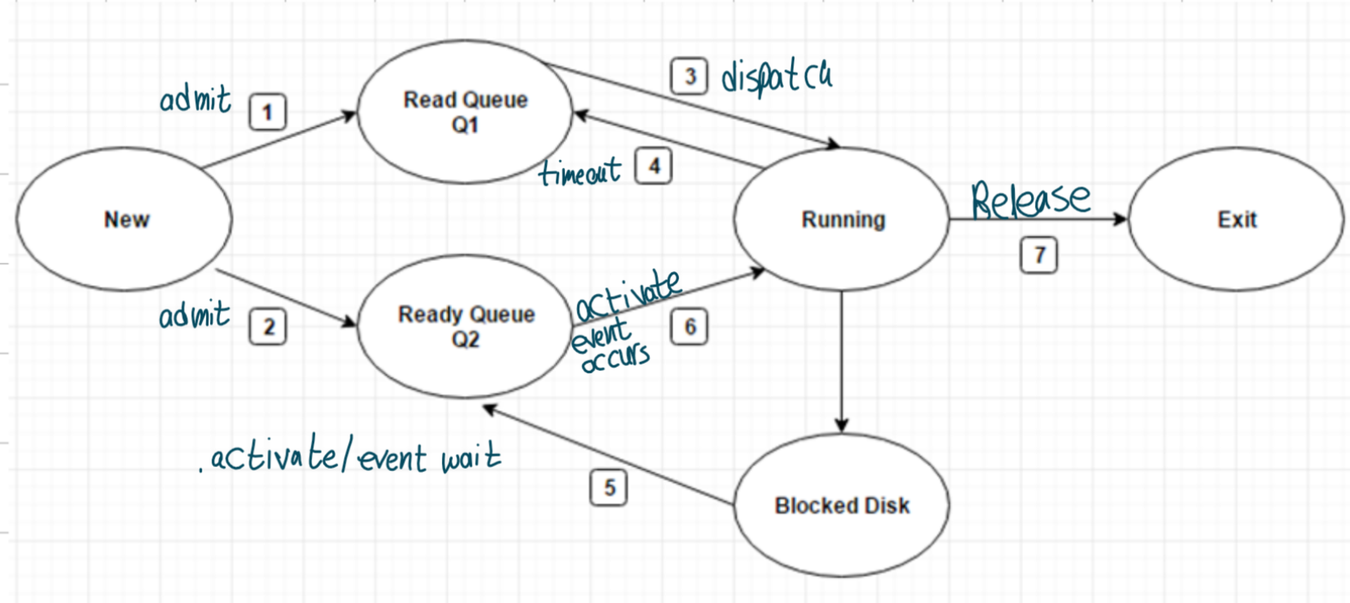
א. 5 פעמים
ב. לא, סדר הביצוע של התהליכים תלוי במ"ה, ואינו ידוע מראש (כמו תהליכונים)



תרגיל 2:

נתונה מ"ה המנהלת את זמן ה CPU (TIME SLICE) באמצעות דיאגרמת המצבים הבאה:

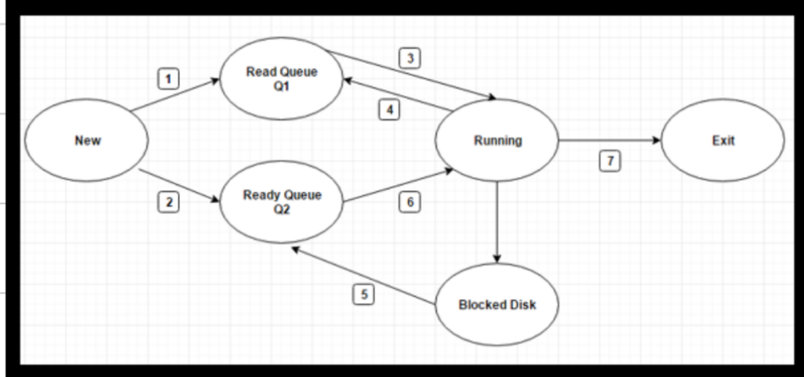
1. ציין עבור כל מספר את שם המעבר בין מצבים



2. הסבר מה ההבדל בין תהליך CPU BOUND לבין תהליך I/O

אלו הפורשים יותר עיבוד הם תהליכים שפורשו זמן מעבד, ותהליכי קלט/פלט יבלו יותר זמן בלהכנת קלט ופלט, בחזרה פורשים מעבד.

המשך...



P1 =

Disk				V		V	
CPU	V	V	V		V		V

P2 =

Disk			V				
CPU	V	V		V	V	V	V

P3 =

Disk		V		
CPU	V		V	V

3. נתון כי זמן ה TIME SLICE הוא קבוע לכל תהליך, ושווה ל 2T (2 יחידות זמן) בלי קשר לעדיפות התהליך. למערכת נכנסים שני תהליכים P1, P2 בזמן T=1 והם בעלי עדיפות Q1. למערכת נכנס תהליך P3 בזמן T=3 בעדיפות Q2.

למעלה מוזכרות תוכניות עבודה עבור כל תהליך כאשר כל משבצת מהווה יחידת זמן אחת (1T). בהנתן תוכניות אלו מלא את הטבלה הבאה מ T=1 ועד לזמן בו אחרון התהליכים סיים את עבודתו:

Time(T)	Ready Q1	Ready Q2	Running	Disk

חוקי ה"משחק"

החוקים הבאים קבועים לתרגילים מסוג זה, אלא אם כן צויין אחרת במפורש בגוף התרגיל:

1. בכל נקודת זמן תהליך יכול להמצא במצב אחד בלבד
2. תהליך מסיים ריצה כאשר קיבל את כל הזמן הדרוש לו לפי תוכנית העבודה שלו
3. במידה וקיימים יותר מתור ready אחד, התהליך הבא שירוץ הוא תמיד הראשון בתור עם העדיפות הגבוהה יותר
4. $Q_q - Process\ Priority, \quad P_n - Process\ ID$
5. ניתן להניח שמעברים בין מצבים שונים הם מידיים ולא עולים ביחידת זמן

פיתרון תרגיל 2

ניתן לוודא: סכימת כל זמן המעבד שתהליכים דרשו בטבלאות עבודה צריך להיות שווה למספר השורות בטבלה (בהנחה שהמעבד תמיד תפוס ע"י תהליך כלשהו)

Time(T)	Ready Q1	Ready Q2	Running	Disk
1	P2		P1	
2	P2		P1	
3	P1	P3	P2	
4	P1	P3	P2	
5	P2	P3	P1	
6			P3	P2, P1
7		P2	P1	P3
8		P3	P2	P1
9		P1, P3	P2	
10	P2	P1	P3	
11	P2	P1	P3	
12		P1	P2	
13		P1	P2	
14			P1	

P1 =

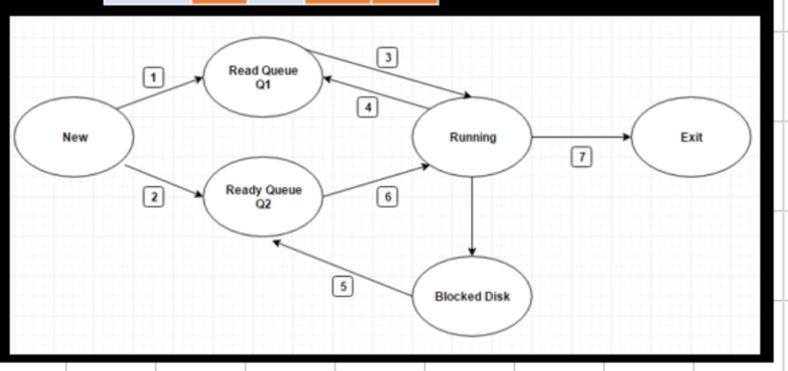
Disk				V		V	
CPU	V	V	V		V		V

P2 =

Disk			V				
CPU	V	V		V	V	V	V

P3 =

Disk		V		
CPU	V		V	V



תרגול 4
30/4/2024

fork and semaphora

pid - כל תהליכון יש לו מספר מזהה משלו.

getPid() - מחזיר את המספר המזהה של התהליכון.

fork() - מחזיר "0" עבור תהליכון בן. ומספר גדול מ-0 (את מספר הזהות של תהליכון הבן שנוצר) לאבא כפי שתהיה לנו אינדקס מ' בפ מ'.

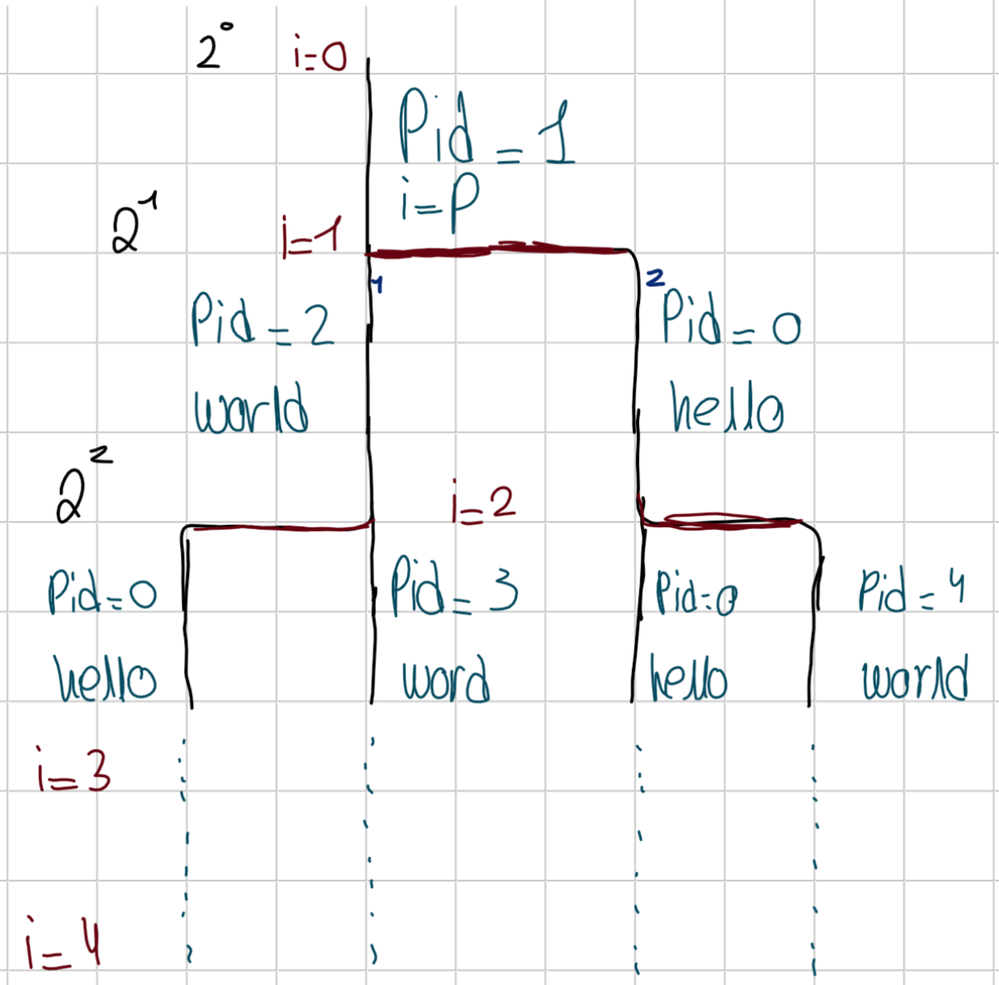
שאלות:

נתון הקוד הבא, המבצע שימוש בפונקציה fork :

```
void main(){
        int i=0;
        while(i<5){
                pid = fork();
                if (pid == 0)
                        printf("hello");
                else
                        printf("world");
                i++;
        }
}
```

א. כמה פעמים תודפס המילה "hello" וכמה פעמים תודפס מילה "world"?
ב. ציירו את היררכיה של התהליכים שנוצרו.

ב)



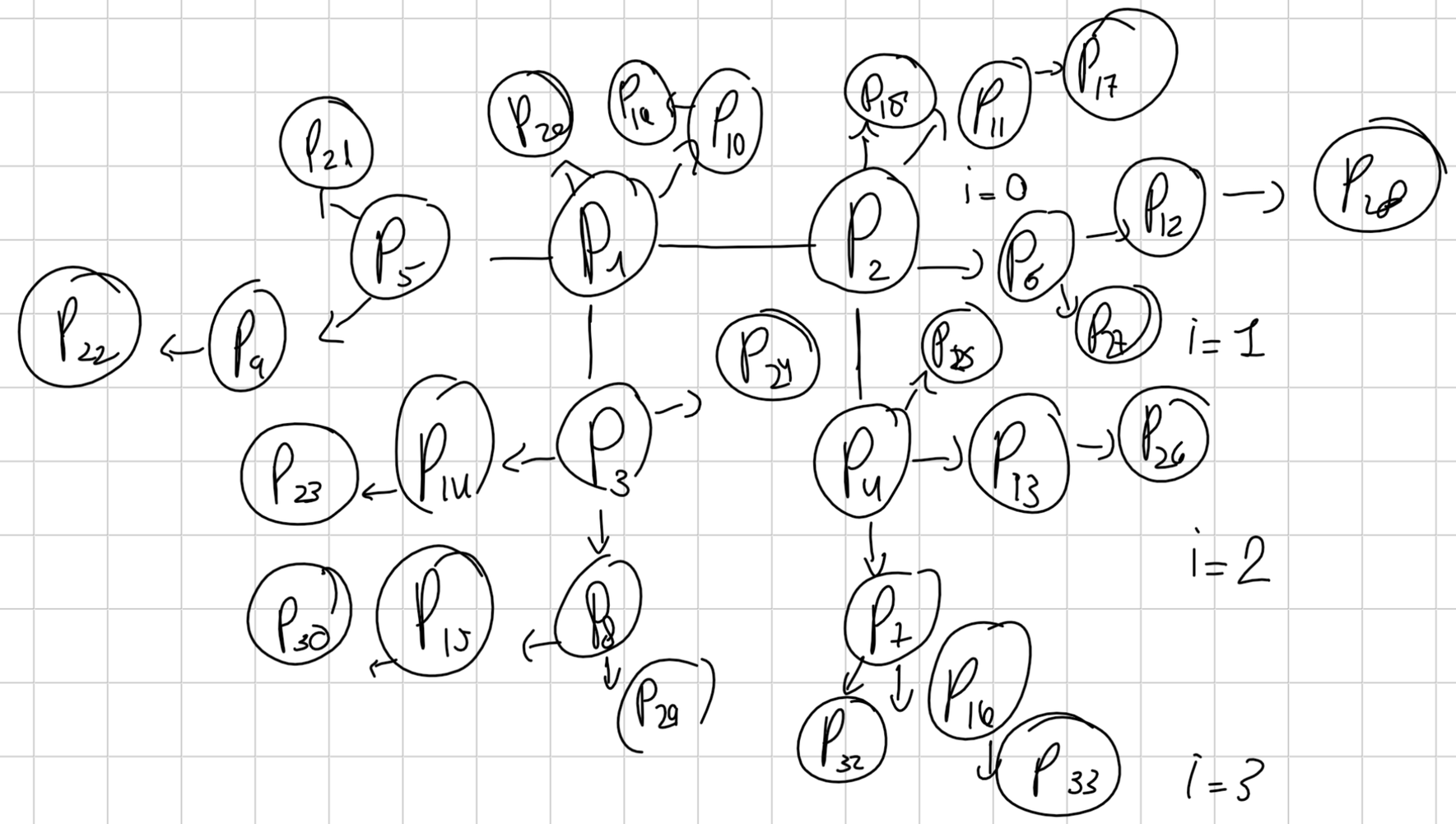
ג)

$$\underbrace{1}_{2^0}+2^1+2^2+2^3+2^4=31$$

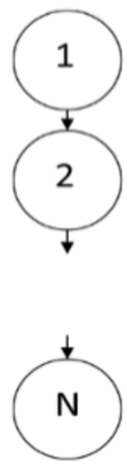
פעמים ל- world ו31 hello.

א. 31 פעמים המילה hello, ו31 פעמים המילה world.
הסבר: נשים לב לעץ ההירכיה, באיטרציה הראשונה יש לנו 2 תהליכונים, כל אחד מדפיס מילה אחרת. באיטרציה השנייה אנו מקבלים 4 תהליכונים, 2 "אבות" ו2 "בנים", כל שניים כאלו מדפיסים מילה hello ו world.
באיטרציה השלישית אנו מקבלים כבר 4 אבות ו4 בנים וכן על זה הדרך. ישנן 5 איטרציות סל הכל אז כדי לספור את כמות הפעמים שכל מילה מודפסת נבצע סכימה (כמובן שכדי לקצר ניתן לעשות סכום סדרה חשבונית).

בתרגיל שלי



ג. כתובו תוכנית (בקובץ pdf) אשר תקבל ממשתמש מספר שלם חיובי N ותיצור שרוך תהליכים:



יש להדפיס pid של כל אחד מהתהליכים שנוצרו.

בתרגיל:

```
int main(){
    int n;
    cin>>n;
    int pid = fork;
    for ( int i=0 ; i<n ; i++){
        if( pid ==0)
            pid = fork;
        else
            Exit();
    }
```

יוצר

מייצר

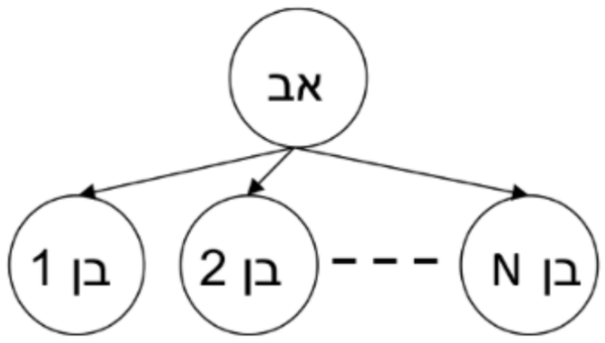
בן

קולטה מהמשתמש n

מייצר רק לבן fork

נשים לב כי נרצה שכל תהליכון בן יוצר בן משלו ואז ימות. הרעיון כאן יהיה שרק הבנים ייצרו תהליכון.

ד. כתוב תוכנית אשר תקבל מהמשתמש מספר שלם חיובי N ותיצור עץ תהליכים בגובה 1 (תהליך אב ו-N בנים שלו):



פתרון:

```
int main(){
    int n;
    cin>>n;
    int pid = fork;
    for (int i=0 ; i<n ; i++){
        if(pid>0)
            pid = fork;
        else
            Exit(1);
    }
```

יוצא

מייצר

בן

קולט מהמשתמש n

מייצר רק לאבא fork

שאלה 2

להלן תבנית של תוכנית עם שני תהליכים ופונקציית main.

main()	p1()	p2()

	{start();
	while (TRUE){
	server_create();
	finish();}
	}
		{ while (TRUE)
		{before();
		client_use();
		cleanup();}}

התניות:

א. קטע start צריך לסיים לרוץ לפני שקטע before ירוץ אפילו פעם אחת.

ב. קטע server_create הוא יצרן וקטע client_use הוא הצרכן של מה שהיצרן יצר.

ג. קטעים before, finish, cleanup הם קטעים קריטיים אחד לשני ואסור שירוצו יחד.

הוסף לפסאודו-קוד הנ"ל פקודות של סמפור כדי לקיים את שלוש ההתניות. השתמש אך ורק בפקודות הבאות: wait(s), signal(s), init_sem(s,n).

```
main {
    init-sem(S1, 0)
    init-sem(S2, 1)
    init-sem(S3, 0)
    init-sem(S4, 1)
}
```

```
P1() {
    start()
    singal(S1)
    while(true)
        server_create()
        singal(S3)
        wait(S4)
        finish()
        singal(S4)
}
```

```
P2() {
    wait(S1)
    while(true)
        wait(S4)
        before()
        singal(S4)
        wait(S3)
        client_use()
        singal(S2)
        wait(S4)
        clean_up()
        singal(S4)
}
```

הסבר: עבור התנאי הראשון ניצור סמפור s1 על מנת שיגביל את S2 מלעבוד- הוא ישוחרר רק לאחר שS1 יסיים.

עבור התנאי השלישי ניצור סמפור S4 כך שישמש כקטע קריטי, מי שרוצה להיכנס לקטע זה חייב לקחת כרטיס קודם, ומי שמסים משחרר כרטיס.

התנאי השני הוא יחסית מורכב יותר- היינו חושבים שמספיק שימוש בסמפור בודד, כך שלפני ייצור ניקח כרטיס נייצר ונחזיר, ולפני צריכה אותו הדבר.

מדוע העניין לא כ"כ פשוט?

חשבו על התרחיש הבא: היצרן ייצר דבר אחד על המדף (בהחה והמדף אינו מלא), הצרכן נכנס לקטע שלו (לקח כרטיס ושחרר כרטיס מאותו הסוג), כעת, מכיוון שהכרטיס שהצרכן דורש הוא מאותו הסוג הוא יכול להיכנס שוב ולקחת שוב- למרות שאין כלום על המדף!

מה זה אומר? בשאלה זו עלינו להפריד את פעולות הצריכה והייצור (גם במחיר של פגיעה קטנה בסדר הקוד). נשים סמפור אחד S2 לפני הייצור – שאותו נאתחל ל1, וסמפור שני S3 לפני הצריכה שאותו נאתחל ל0.- כלומר היצרן תמיד ייצור 1, הצרכן לא יוכל לקחת לפני שיש משהו. לאחר שהיצרן סיים הוא מעיר את הצרכן, ולאחר שהצרכן מסיים הוא מעיר את היצרן. (השפענו במידת מה על זרימת הקוד – למשל כעת מצב בו היצרן ייצר הרבה ואז צרכן ייקח הרבה פחות אפשרית, אבל מנענו יציאה מגבולות המדף.)

שאלה 3:

נתונים 3 תהליכים שרצים במקביל:

main	P1	P2	P3

	{
	B
	E
	}
		{
		A
		C
		}
			{
			D
			F
			}

התניות:

- קטע A חייב להסתיים לפני שמתחילים B, C ו-D.
- קטעים B, C, D הם קטעי קוד קריטי ואסור שיפעלו באותו זמן.
- קטע E תתבצע רק אחרי סיום B, C ו-D.

תעתיקו את התהליכים לקובץ בתוספת מנגנון שמקיים את ההתניות בעזרת סמפורים.

```
main() {
    init_sem(S1, 1)
    init_sem(S2, 0)
    init_sem(S3, 0)
}
```

```
P1() {
    wait(S2)
    B
    Singal(S2)
    wait(S3)
    wait(S3)
    E
}
```

```
P2() {
    wait(S1)
    A
    Singal(S2)
    wait(S2)
    C
    Singal(S2)
    Singal(S3)
}
```

```
P3() {
    wait(S2)
    D
    Singal(S2)
    Singal(S2)
    F
}
```

wait ( ) → P [from the Dutch word proberen, which means "to test"]

signal ( ) → V [from the Dutch word verhogen, which means " to increment"]

Definition of wait ( ):

```
P (Semaphore S) {
    while (S <= 0)
      ; // no operation
    S-- ;
}
```

Definition of signal ( ):

```
V (Semaphore S) {
    S++ ;
}
```

All the modifications to the integer value of the semaphore in the wait () and signal() operations must be executed indivisibly. That is, when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.

Types of Semaphores:

1. Binary Semaphore:

The value of a binary semaphore can range only between 0 and 1. On some systems, binary semaphores are known as mutex locks, as they are locks that provide mutual exclusion.

2. Counting Semaphore:

Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

תרגיל 5
7/5/2024

Semaphore

הגדרה – Semaphore

- המילה מתארת שיטת תקשורת המבוססת על שימוש בדגלים לאיתות
- בהקשר שלנו – תהליכים יאותתו זה לזה האם ניתן/אסור להכנס לק"ק
- הסמפור עצמו הוא משתנה שלם, אותו מאתחלים בערך כלשהו, ומוגדרות עליו שתי פעולות **אטומיות:** (מתאפשר ע"י פונקציית test & set)

המתנה

```
wait(sem)
{
    while (sem <= 0)
        {/*wait - no op*/}
    sem--;
}
```

שחרור

```
signal(sem)
{
    sem++;
}
```

- הסמפור הוא מושג כללי, שבין היתר מאפשר הגנה על ק"ק.(מאפשר גם סנכרון)

Semaphore בינארי

- ערכו תמיד יהיה 0 או 1
- מתאים לשמירה על Mutex, מפני שרק תהליך אחד יורשה להיות בק"ק בכל זמן נתון
- תבנית:

בסמפור בינארי, נאתחל ל - 1

```
int sem = 1;

while(true)
{
    wait(sem) ;
    /*CS*/
    signal(sem);
    /*remainder section*/
}
```

אם ק"ק פנוי, כנס וסגור הדלת לאחרים

לאחר ביצוע ק"ק, פתח הדלת לאחרים

ראינו בשקף הקודם...

מהי "פתיחת" / "סגירת" דלת?

בפרק כלל אנחנו לא נכתוב כך ולכן

Semaphore s1 = new Semaphore(3, true);

נשתמש במחלקה שיש ב-Java

כמות כרטיסים

יש כניסות

Semaphore כללי

- סמפור לא משמש רק למשימות mutex – עשוי לשמש גם לסנכרון
- באמצעות משחק עם המספר בו נאתחל את הסמפור, נוכל לקבוע כמה תהליכים יורשו להכנס לקטע מסויים
- דוגמא: נניח שנרצה לכפות שתהליך P2 יוכל לבצע קטע קוד כלשהו רק אחרי ש P1 הכין לו נתונים. כלומר, P1 יהיה הראשון לבצע קטע קוד כלשהו, ואז P2 יהיה השני לבצע קטע קוד כלשהו. נוכל לבצע זאת באופן הבא:

אתחול

```
int sem = 0;
```

P1

```
/*preparation code*/
signal(sem);
```

P2

```
wait(sem);
/*some code*/
```

- **הסמפורים שתוארו אינם חסינים בפני הרעבה**
- כדי לפתור את בעיית ההרעבה בסמפור, על הממתינים להכנס לתור

תרגיל כיתה

נתון הקוד הבא:

אתחול
```
int sem1 = 1;
int sem2 = 0;
```

P1
```
wait(sem1);
/*code A*/
signal(sem2);
```

P2
```
wait(sem2);
/*code B*/
signal(sem2);
```

P3
```
wait(sem2);
/*code C*/
signal(sem1);
```

מהם סדרי ביצוע קטעי הקוד A, B, C האפשריים?

$(A\,B^{1,*}\,C)^{1,*}$

חייבת להתרחש לפחות בפעם אחת.

P_1 - תמיד עובר הראשון כי הוא משתמש ב- sem1 שיש לו כרטיס.

P_2 ו- P_3 לא יעבדו בהתחלה כי אין להן עדיין כרטיס.

אחרי ש- P_1 יסיים הוא ישחרר הכרטיס ל sem2. לכן sem2=1

* שחרור כרטיס שגם סמפור אחר עלול לגרור ל deadlock.

אחרי כך P_2 או P_3 ס"ל כמן מעבד יקחו את הכרטיס.

תרגיל כיתה

נתון הקוד הבא:

אתחול
```
int S = 1;
int Q = 1;
```

P1
```
wait(S);
wait(Q);
/*move1*/
signal(S);
signal(Q);
```

P2
```
wait(Q);
wait(S);
/*move2*/
signal(Q);
signal(S);
```

האם הקוד בטוח?

לא, יש לנו כאן deadlock.

P_1, לוקח כמן מעבד ומבצע את wait(S) ואחר כך P_2, לקח כמן מעבד ומבצע את wait(Q).

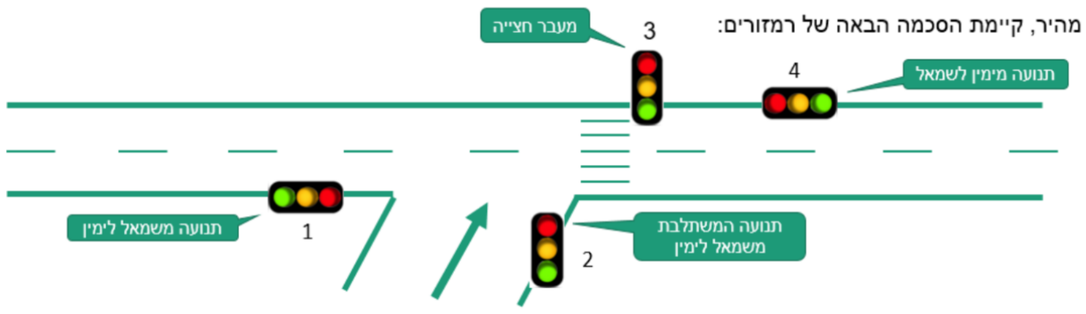
אבל עשיו יש לו שתי משאלות תלויות.

כמן מעבד חוזר ל P_1 בעזרת המתן בקוף אבל Q תפוס. אותו דבר על P_2 עבור S.

ואז מכאן נוצר לנו deadlock כל אחד מקה לשחרר את המשאל שנתפס אצלו.

תרגיל כיתה

בכביש מהיר, קיימת הסכמה הבאה של רמזורים:



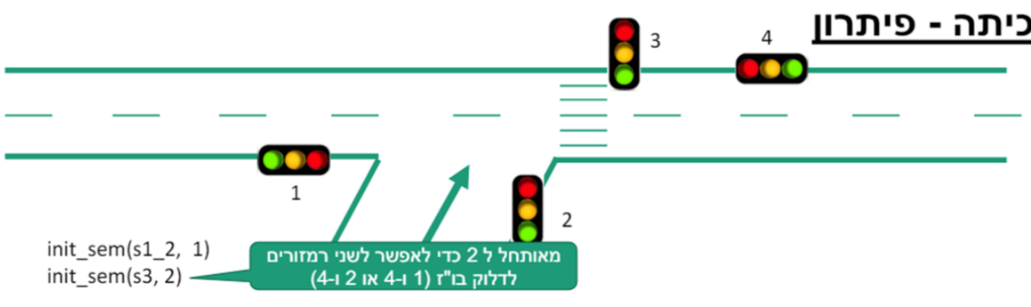
כל הרמזורים שואפים להיות ירוקים למשך 3 שניות, אך תחת התנאים הבאים:

1. רמזור 3 רשאי להיות ירוק רק כשכל האחרים אדומים
2. רמזור 1 רשאי להיות ירוק רק כש – 2 ו-3 אדומים
3. רמזור 4 רשאי להיות ירוק רק כש – 3 אדום
4. רמזור 2 רשאי להיות ירוק רק כש – 3 אדום ו- 1 אדומים

א. כתוב קוד עבור כל אחד מהרמזורים/תהליכים, כך שתתקיים החוקיות הנ"ל. השתמש בסמפורים השתמש במתודה ()goGreen3 לקבוע לרמזור צבע ירוק למשך 3 שניות (בסופן הוא יוחזר לאדום)

ב. נוסף חוק שרק רמזור אחד יכול להיות ירוק בכל רגע נתון. שנה את הקוד כך שיתמוך בחוק זה

תרגיל כיתה - פיתרון




```
//1
while (true){
    wait(s1_2)
    wait(s3)
    goGreen()
    signal(s3)
    signal(s1_2)
}
```

```
//2
while (true){
    wait(s1_2)
    wait(s3)
    goGreen()
    signal(s3)
    signal(s1_2)
}
```

```
//3
while (true){
    wait(s3)
    wait(s3)
    goGreen()
    signal(s3)
    signal(s3)
}
```

```
//4
while (true){
    wait(s3)
    goGreen()
    signal(s3)
}
```

ראשית נמיר במקום "ירוק ואדום", אם המילה "עובדת".
מבפרישרת מובן כי 4 לא יכול לעבור עם 3 ו-3 לא יכול לעבור עם 4, לכן קודם נשים את אותו הסמפור עם שניהם (S3).
נשים לב כי ישנה אפשרות ש-2 רמזורים יעבדו יחדו 1 ו-4 או 1-2. לכן נרצה לשנות את המסמור ממקומם ולאתחל אותו עם שני כרטיסים, בעצם מתן הרשאה ל-2 תהליכים לרוץ יחדו לאחר שינוי כפי נצטרך "לתקן" את ההמלה של 3, שיבקש 2 כרטיסים, כך שאף אחר לא יעבוד איתו.
כעת נבחר על שתי ההמלות של 1-2. ונשים לב ש-1 יכול לעבוד אך לא כש-2 זו עובדים. משמש אותנו בדרכים סמפור תפוש הכניסה ל-1 ו-2 בעצם S1_2. למה לא עוד אחד עבור S3? מכיוון שכבר S3 חוסם אותו שם. ולכן מספיק לשים את S1_2. (מכיוון שיש 2 כרטיסים ל S3 צריך סמפור נוסף)
מספיק לחסום את 4 רק עם S3 בלבד.
וההמלה של 2 היא כמו של 1 ולכן אותו סימון סמפורים יעבוד שם.

ב) אנחנו רק נוכל סמפור אחד

```
init_sem(s, 1)
```

```
//1
while (true){
    wait(s)
    goGreen()
    signal(s)
}
```

```
//2
while (true){
    wait(s)
    goGreen()
    signal(s)
}
```

```
//3
while (true){
    wait(s)
    goGreen()
    signal(s)
}
```

```
//4
while (true){
    wait(s)
    goGreen()
    signal(s)
}
```

שאלה 5 (20 נקודות)

נתון הפתרון הבא לבעיית היצרנים-צרכנים כאשר הפעם ניתן להכניס או להוציא פריט בגודל k יחידות. החוצץ (buffer) המשותף הוא בגודל N ו- $1 \le k \le N$.

Producer	Consumer
{	{
For i=1 to k	**Down**(full)
Down(empty)	Remove item size k
Insert item size k	For i=1 to k
Up(full)	**Up**(empty)
}	}

המשתנים empty ו-full הם סמאפורים המאותחלים : full=0, empty=N.

א. (5 נק') האם הפתרון מבטיח שלא ייכנס לחוצץ תוכן הגדול מ-N? נמקו.

ב. (5 נק') האם ייתכן שהצרכן יוציא פריט לפני שהיצרן הכניס? כלומר שהחוצץ ריק? נמקו.

ג. (10 נק') האם מובטח שלא יהיה Deadlock? נמקו ואם יש Deadlock הראו דוגמה.

א. לא כי ע"פ הנתונים $k \le N$. תמיד.
ב. לא ייתכן שצרכן יוציא פריט לפני שהיצרן הכניס.
כן, יכול להיות שחוצץ ריק.
כאשר הproducer עשה A מוכרים ו-consumer לקח אותן כולן.

ג. לא, כי כל אחד מעדכן את המשתנה של השני, כי אין טריק שתופס הגורם של producer ואת consumer.

א. חייב להתחיל מיצרן. מה שקורה כאן הוא שביצרן – אנחנו לוקחים K כרטיסים ואז מכניסים איבר בגודל K ואז משחררים כרטיס אחד של לקיחה. הצרכן צריך רק כרטיס אחד של לקיחה
ולאחר מכן מוציא איבר בגודל k יחידות.
הפתרון המוצע תקין שכן לאחר הכנסה של איבר בגודל k לא נוכל להכניס עוד אחד אלא אם כן ישנם מספיק כרטיסים לכך. (אם יש סימן שK2 קטן מ N) ולכן לא נוכל להכניס יותר מN.
ב. לא ייתכן שהצרכן יוציא לפני היצרן אך כן ייתכן שהבאפר יהיה ריק אם נכניס איבר בגודל K ואז ניקח אותו.
ג.כדי שיווצר כאן מצב של דדלוק- היצרן צריך לחכות לצרכן ולהפך. במקרה שלנו, ישנם שני איברים שמאותתים לכל אחד מהם, והם מתעדכנים ישירות לאחר הפעולה. אף אחד מהתהליכים לא מחזיק משאב ומחכה למשאב נוסף. – לכן אין סיבה שיהיה פה דדלוק.

שאלה 4 (20 נקודות)

קבוצת ילדים רוצה לצאת לגלידה. בשולחן בגלידריה יש מקום רק לשלושה ילדים. כלומר, ילד שיגיע לאחר שהתיישבו שלושה, לא יוכל להצטרף.

בקבוצה ישנו ילד בישן, יוסי, שלא מוכן להגיע לפני שחברו הטוב, דוד, יגיע. אולם הוא רוצה להגיע מייד לאחר דוד. אורי מאוד עסוק היום בהכנת שיעורי בית, ולכן רוצה להגיע שלישי (כלומר להיות השלישי שיושב ליד השולחן).

כתוב אלגוריתם, המשתמש בשלושה סמפורים, המקיים את האמור לעיל, עבור כל אחד מהילדים: דוד, יוסי, אורי וכל ילד אחר בכיתה.

דוד
יוסי
אורי

semaphore t=3
semaphore u=0
semaphore y=0

David	Yosi	Uri	Any other child:
t.wait	y.wait	u.wait	t.wait
sit	t.wait	u.wait	sit
y.singal	sit	t.wait	u.singal
u.singal	u.singal	sit	

שאלה 2 (20 נקודות)

במכרה פחם עובדים בצורה הבאה. ברגע שמגיעה רכבת, העובדים מתחילים להביא עגלות של פחם כדי למלא אותה. יש צורך ב- n עגלות בדיוק בכדי למלא רכבת אחת. אחרי שהעובדים הביאו n עגלות, הרכבת עוזבת והעובדים מחכים עד שהרכבת הבאה מגיעה. (שימו לב שהעבודה במכרה היא עבודה סיזיפית במיוחד – היא לא מסתיימת לעולם...).

ממשו בפסאודו-קוד תהליכים של (א) רכבת ו-(ב) של עובד, תוך שימוש **בסמפורים בינאריים** כדי לסנכרן ביניהם.

```
Binary semphore t=0
Binary semaphore w=0

Train:
    while (true){
        t.wait
        Fill train and go
        w.signal
    }

worker:
    while (true){
        for (i=1 to n)
        bring agala
        t.signal
        w.wait
    }
```

* יש לשים לב שבלולאה משלישה על מה שבבמשך הדורה שלה. כלומר ממלאים את הרכבת הראשונה בעגלות, ורק אז מאותתים לרכבת לצאת לעובדים לנוח לרכבת לבמיץ שוב.

תרגול 6

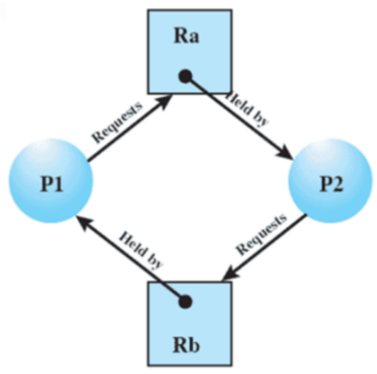
21/5/2024

Deadlock

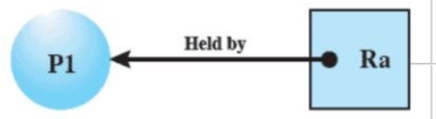
אומר שאין Deadlock.

Resource Allocation Graph

- גרף המאפשר לזהות האם המערכת בטוחה או לא (האם קיים סיכוי ל **Deadlock**)
- הגרף ממחיש ויזואלית את מצב המערכת מבחינת תהליכים ומשאבים
- כאשר **כל** התהליכים והמשאבים בגרף חסומים בגלל מעגל המתנה כלשהו – המערכת אינה בטוחה



(a) Resouce is requested



(b) Resource is held

תרגיל כיתה 1

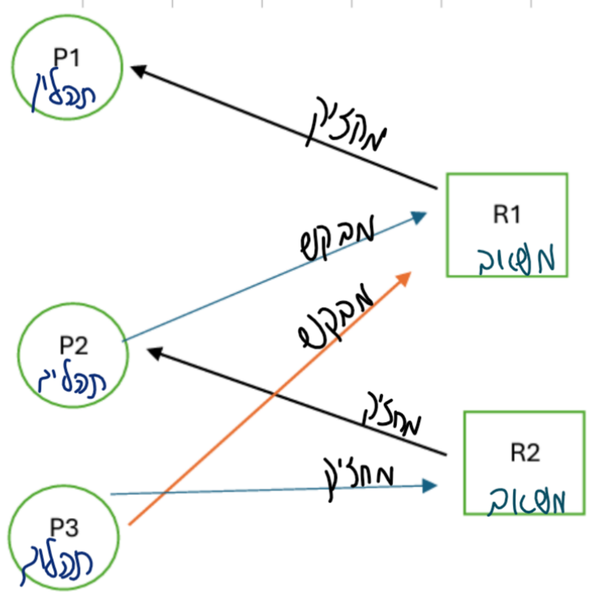
- נתונה המערכת הבאה:
  - כמויות משאבים במערכת:
    - R1 – 1
    - R2 - 1
  - תמונת המצב ברגע נתון:
    - תהליך P1 מחזיק במשאב R1
    - תהליך P2 מחזיק במשאב R2
    - תהליך P2 מבקש את משאב R1
    - תהליך P3 מבקש את משאב R1
    - תהליך P3 מבקש את משאב R2

א. צייר את תמונת המצב של המערכת באמצעות Resource Allocation Graph

ב. בהנתן תמונת המצב הנ"ל, האם המצב הנוכחי בטוח ? הסבר מדוע

ג. נתון גם שתהליך P1 מבקש את R2. האם לאחר בקשה זו המצב בטוח ? הסבר מדוע

1. א.



ב. * P_1 בינתיים כי אין תהליך אחר שמו שיכים לו בפלוק.

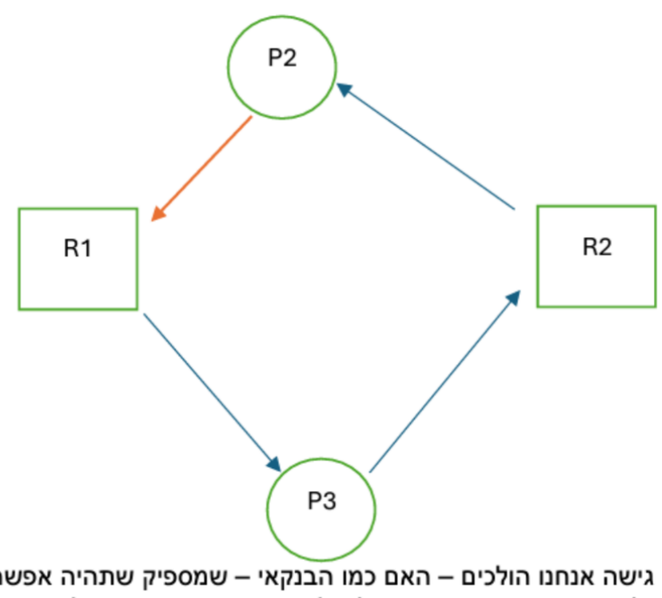
P_3 תופס R_1 ומבקש את R_2

P_2 תופס R_2 ומבקש את R_1.

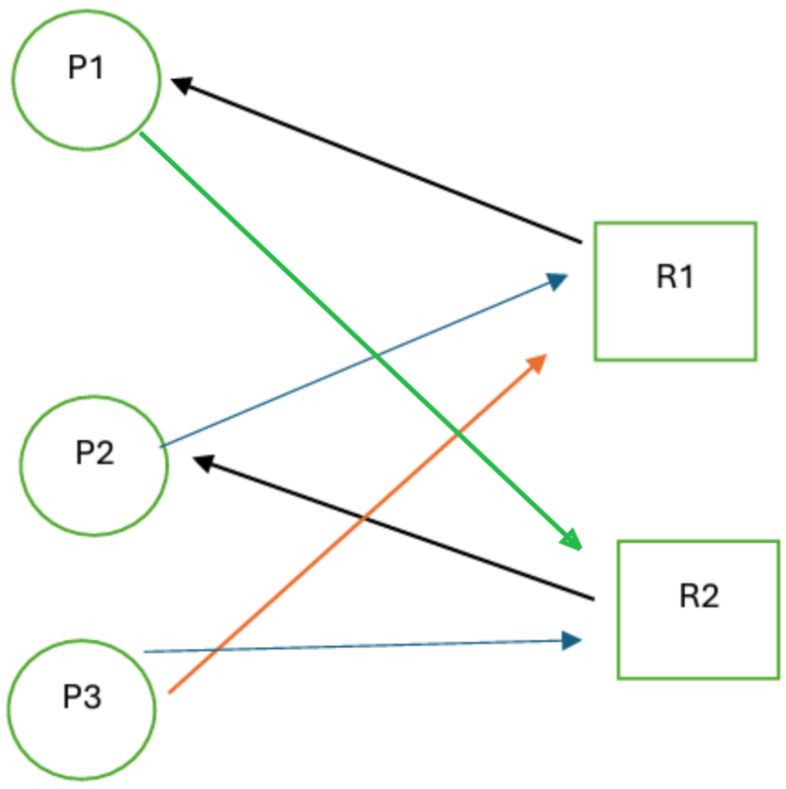
יש לנו כאן Deadlock מסוג.

ב. על פניו – נראה שאין לנו כאן אפשרות "להסתבך". אך נשקול את המצב הבא:

P1 סיים את פעולתו (הוא היחיד שיכול לסיים). נניח שp3 תפס את המשאב R1 – איזה מצב נוצר לנו כרגע? P2 תופס את r2 ומחכה לr1 וP3 תופס את r1 ומחכה לr2. – זהו מצב של דדלוק! אף אחד לא יתקדם.



- תלוי לפי איזה גישה אנחנו הולכים – האם כמו הבנקאי – שמספיק שתהיה אפשרות אחת תקינה או לפי זה שיש היתכנות אחת לדדלוק- כי ישנה אפשרות שלא יהיה פה דדלוק- אם P2 יקבל את r1 קודם.



ב.

P_3 תופס R_1 ומבקש את R_2

P_2 תופס R_2 ומבקש את R_1.

P_1 מבקש את R_2 אבל R_2 תפוס ע"י P_2 ו- P_2 כבר לשחרר את R_2 כי אז צריכה לקבל את R_1 שכבר תפוס ע"י P_3 שכבר גם מחכה ל- R_2 כדי לסיים את עבודתו.

ג. בעיקרון לאחר בקשה זו בוודאות יהיה דדלוק מכיוון שאף תהליך לא יסיים את ריצתו. P1 יחכה לp2 לעד וP2 יחכה לp1 לעד.

תרגיל כיתה 2

- נתונה המערכת הבאה:
  - כמויות משאבים במערכת:
    - 2 – A
    - 2 – B
  - תמונת המצב ברגע נתון:
  - תהליך P1 מחזיק את משאב B
  - תהליך P1 מבקש את משאב A
  - תהליך P2 מחזיק את משאב A
  - תהליך P3 מבקש את משאב B
  - תהליך P3 מחזיק את משאב A
  - תהליך P4 מחזיק את משאב B

מחזיק

Allocation (משאבים)

	A	B
P1	0	1
P2	1	0
P3	1	0
P4	0	1

מבקש

Need (כמות המשאבים)

	A	B
P1	1	0
P2	0	0
P3	0	1
P4	0	0

Resources

A	B
2	2

Available

A	B
0	0

תיאור הנתונים באמצעות טבלאות

אם עשינו את כל האיטרציות ושום תהליך לא מתשחרר אז יכול לנו Deadlock.

P_2, P_4 לא מבקשות כלום לכן יכולים לברוח לשחרר את המשאבים שמחזיקים אותן.

בהנתן תמונת המצב הנ"ל, האם המערכת במצב בטוח? הסבר באמצעות Resource Allocation Graph

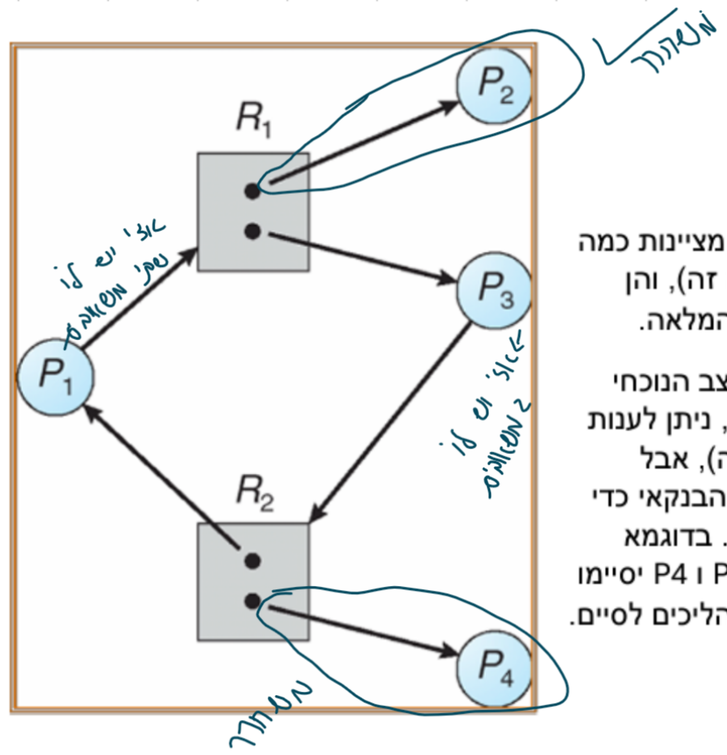
שאלה 2:

1 – הוא במשמעות של A

2 – הוא במשמעות של B.

שימו לב לנקודות בתוך המשאבים – הנקודות מציינות כמה יחידות יש מכל משאב (למשל 2 מ R1 במקרה זה), והן חשובות בציור הגרף שלכם כחלק מהתשובה המלאה.

במידה ובשאלה שואלים אתכם האם האם המצב הנוכחי בטוח לאחר שהתבקשתם לצייר גרף משאבים, ניתן לענות לפי הגרף (אלא אם התבקשתם אחרת בשאלה), אבל מומלץ בכל מקרה לבצע חישוב של אלגוריתם הבנקאי כדי להיות בטוחים (בגרף גדול קל לפספס דברים). בדוגמא למעלה למשל, המערכת בטוחה כי לאחר ש P2 ו P4 יסיימו לרוץ, הם ישחררו משאבים שיאפשרו לכל התהליכים לסיים.



* אנחנו כאן מניחים שכאן כתהליך משחרר את כל המשאב שבאו מחזיק כי אז כרגע כרושונך כאשר עיקר זמן מסתיים ולא מחזיק אותו לאינסוף.

תרגיל כיתה 3

נתונה המערכת הבאה, בה 4 תהליכים המסומנים ב P, ו-4 משאבים A,B,C,D ותמונת המצב הבאה במערכת:

Allocation

	A	B	C	D
P1	0	1	0	0
P2	1	1	0	0
P3	0	0	1	0
P4	0	0	0	1

Need

	A	B	C	D
P1	1	0	0	0
P2	0	0	1	0
P3	0	1	0	0
P4	0	0	0	0

Available

A	B	C	D
0	0	0	0

Resources

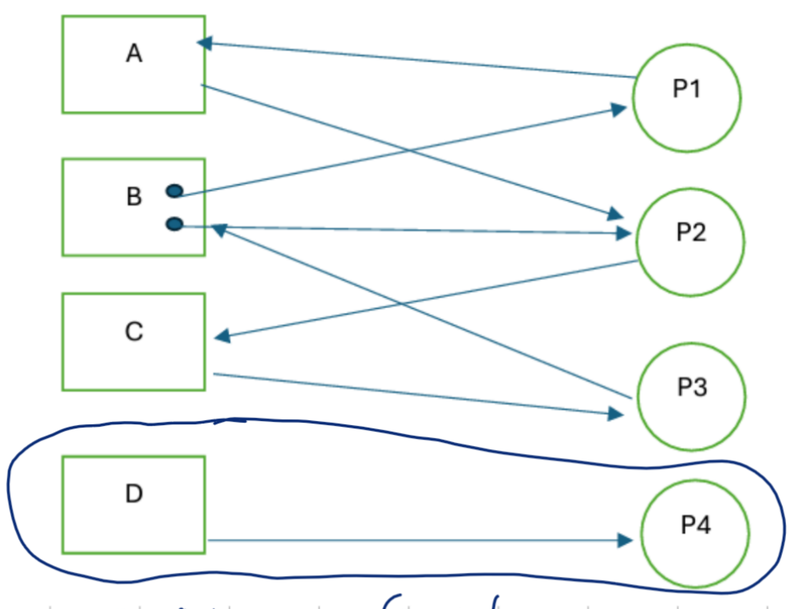
A	B	C	D
1	2	1	1

א. בהנתן תמונת המצב הנ"ל, האם המערכת במצב בטוח? הסבר באמצעות Resource Allocation Graph

ב. נתון כי P3 ויתר על משאב R3 שבו החזיק. האם המערכת במצב בטוח? הסבר באמצעות Resource Allocation Graph

ב. בורסיה החדשה, C אינו תפוס ע"י אף אחד- ולכן נוכל למצוא מערך שבו יש פתרון, למשל – P4 יסיים קודם, לאחר מכן p2 יוכל לסיים. ואז אחד מהבאים יוכל לעשות p1\p3.

א. המערכת אינה במצב בטוח. אמנם P4 יכול לסיים – אך כל השאר לא יזוזו, נתבונן בגרף ונבחין כי p1 לא יוכל להתקדם עד שa ישוחרר. A לא ישוחרר עד שP2 לא יסיים והוא צריך את c. C תפוס ע"י P3 ולא ישוחרר עד שb ישוחרר. bl לא ישוחרר עד p2\p1 לא יסיימו ושניהם לא מתקדמים. לפי הטבלאות:



אין כל כך מעניין.

The Banker's Algorithm

- אלגוריתם של דיאקסטרה (המציא גם את אלגוריתם המאפייה)
- בהנתן תמונת המצב של המערכת, מאפשר לדעת / לחזות האם המערכת בטוחה (האם יתכן Deadlock), ובהתאם לכך לחסום או לאפשר גישה למשאבים.
- כדי להשתמש באלגוריתם זה – חייב לדעת אילו משאבים כל תהליך יצטרך במהלך חייו, ומהו מספר המשאבים במערכת (צריך להיות מספר סופי)

The Banker's Algorithm - Definitions

וקטור המציין כמות מכל משאב במערכת:
$RECOURCES = R = (R_1, R_2, R_3, \dots R_N)$

וקטור המציין כמה יחידות פנויות מכל משאב קיימות במערכת:
$AVAILABLE = (V_1, V_2, V_3, \dots V_N)$

מטריצה המתארת כמה יחידות משאב כל תהליך יצטרך:
$CLAIM = C = [CLAIM\ MATRIX]$

מטריצה המתארת כמה משאבים מוקצים לכל תהליך ברגע נתון:
$ALOOCCATION = A = [ALLOC\ MATRIX]$

מטריצה המתארת כמה משאבים נדרשים לכל תהליך ברגע נתון:
$NEED = [NEED\ MATRIX]$

The Banker's Algorithm - Definitions

- אם הצלחנו לתת לכל התהליכים את כל המשאבים שהם ביקשו – המערכת בטוחה. אחרת – לא
- בדיקה – לאחר הרצת הבדיקה, במידה והמערכת בטוחה אמור להתקיים Resources = Available
- עבור שאלה בסגנון: "נניח שתהליך P1 ביקש משאב: (1,0,2,...) ... ":
  - נוסיף את הוקטור הזה לשורה של P1 בטבלת Alloc
  - נפחית מ Avail
  - נמצא את מטריצת Need לפי הנתונים החדשים
  - ננסה להקצות משאבים בהתאם לבקשות התהליכים
- אם משאב לא מבקש כלום במטריצת Need – הוא מיד מסיים לרוץ ומחזיר הקצאות

תרגיל כיתה 4

נתונה המערכת הבאה:

Allocation

	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

Claim / Max Demand

	A	B	C
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Resources

A	B	C
10	5	7

א. האם המצב הנוכחי הוא מצב בטוח? נמק

ב. נתון כי במצב הנוכחי P1 ביקש את המשאבים (1, 0, 2). הראה שהמערכת בטוחה (לפי אלגו' הבנקאי)

ג. הנח כי במצב החדש לאחר סעיף ב', P4 מבקש את המשאבים (3, 3, 0). האם המערכת עדיין בטוחה? פרט את החישוב

ד. הנח כי במצב החדש לאחר סעיף ב', P0 מבקש את המשאבים (0, 2, 0). האם המערכת עדיין בטוחה? פרט את החישוב

ב. בורסיה החדשה, C אינו תפוס ע"י אף אחד- ולכן נוכל למצוא מערך שבו יש פתרון, למשל – P4 יסיים קודם, לאחר מכן p2 יוכל לסיים. ואז אחד מהבאים יוכל לעשות p1\p3.

שאלה 4:

נתונה המערכת הבאה:

Resources

A	B	C
10	5	7

Claim / Max Demand

	A	B	C
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Allocation

	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

א. האם המצב הנוכחי הוא מצב בטוח? נמק

ב. נתון כי במצב הנוכחי P1 ביקש את המשאבים (1, 0, 2). הראה שהמערכת בטוחה (לפי אלגו' הבנקאי)

ג. הנח כי במצב החדש לאחר סעיף ב', P4 מבקש את המשאבים (3, 3, 0). האם המערכת עדיין בטוחה? פרט את החישוב

ד. הנח כי במצב החדש לאחר סעיף ב', P0 מבקש את המשאבים (0, 2, 0). האם המערכת עדיין בטוחה? פרט את החישוב

וקטור Avail לא נתון לנו, אבל נתון לנו Resources וגם Allocation. מהם נוכל למצוא את וקטור Avail

Resources – Sum of Allocations = avail vector

בשאלות אחרות עשוי להיות נתון Avail ואז צריך למצוא את Resources (חילוץ Resources מהמשוואה הנ"ל)

ולכן נקבל את הוקטור:

AVAIL

A	B	C
3	3	2

נמצא את מטריצת Need:

Claim – Alloc = Need

ונקבל:

	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

Resources

A	B	C
10	5	7

10-7=3 } 5-2=3 } 7-5=2

NEED

	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

עכשיו כשיש בידינו את כל הנתונים, אפשר להתחיל לעבוד:

Aviliable

נבדוק כל אחד מהתהליכים מבקש ב Need, וננסה להקצות לו מתוך ~~Allocation~~. מקצים משאבים רק אם ניתן להקצות את כל המשאבים הדרושים לתהליך. לאחר שמקצים לתהליך, הוא מסיים ומשחרר את כל המשאבים שלו ממטריצת ALLOC, ואז מוסיפים אותם ל AVAIL

*שאלו בכיתה: מדוע משחררים משאבים ממטריצת Alloc ולא מ Claim. הסיבה היא שתהליך יכול לשחרר רק משאב שנמצא <u>בחזקתו בפועל</u>, ולא משאבים שהוא עתיד לבקש, ומשאבים שמוחזקים בפועל אצל התהליך נמצאים במטריצת Alloc. ב Claim לעומת זאת נמצאים המשאבים שהתהליך יצטרך לאורך כל חייו, אבל אין זה אומר שהוא מחזיק את כולם כבר עכשיו.

סדר הקצאות הזיכרון

P0 = Not enough available

P1 = Finished, Released allocation -> new Avail (5, 3, 2)

P2 = Not enough available

P3 = Finished, Released allocation -> new Avail (7, 4, 3)

P4 = Finished, Released allocation -> new Avail (7, 4, 5)

Now we try again the processes that failed to get resources: (loop)

P0 = Finished, Released allocation -> new Avail (7, 5, 5)

P2 = Finished, Released allocation -> new Avail **(10, 5, 7)**

כל התהליכים יכולים לסיים, ולכן ניתן לומר כי המערכת בטוחה, ורצף אפשרי של הקצאות הוא <P1,P3,P4,P0,P2>

שימו לב שמתקיים:

Avail vector = Resources vector

בדיקה זו מאפשרת לנו לוודא שלא טעינו בחישוב

סעיף ב

כאשר תהליך מבקש משאב נוסף, נרצה לבדוק האם הקצאת המשאב לאותו תהליך תשאיר את המערכת בטוחה או לא. לכן אנחנו מנסים "להקצות" את המשאב לאותו תהליך, ואם אנו מסיקים שלאחר ההקצאה המערכת בטוחה, נאמר שההקצאה בטוחה (ולהפך). בגלל שאנחנו מנסים להקצות את המשאב לתהליך, נוסיף את שורת הבקשות של P1 לזו הקיימת ב Alloc matrix, ונחסר אותה מ Avail vector. כעת נבצע את כל החישובים מחדש, כולל מציאת המטריצה Need ומעבר על התהליכים להקצאת משאבים.

Alloc המעודכן:

Alloc

	A	B	C
P0	0	1	0
P1	3	0	2
P2	3	0	2
P3	2	1	1
P4	0	0	2

Avail המעודכן:

(2,3,0)

Need המעודכן:

Need

	A	B	C
P0	7	4	3
P1	0	2	0
P2	6	0	0
P3	0	1	1
P4	4	3	1

לאחר חישוב נגיע למסקנה כי המערכת עדיין בטוחה, ורצף הקצאות אפשרי הוא <P1, P3, P4, P0, P2>

סעיף ג

הכוונה ב"**מצב חדש"** – המצב החדש של מטריצת Allocation לאחר שהוספנו אליה את הדרישה של P1.

אין מספיק משאבים עבור P4 לכן המערכת אינה בטוחה

סעיף ד

אין מספיק משאבים עבור P0 לכן המערכת אינה בטוחה

*שימו לב כי התשובות לסעיפים ב, ג,ד הן תשובות קצרות וחלקיות. בתרגיל הבית או במבחן מצופה מכם לתת הסבר מלא כמו זה שבסעיף א, המתאר איזה תהליכים קיבלו הקצאה ואיזה לא (ולמה), ולבסוף במשפט או שניים לתאר את המסקנה מהחישוב.

תרגול 7

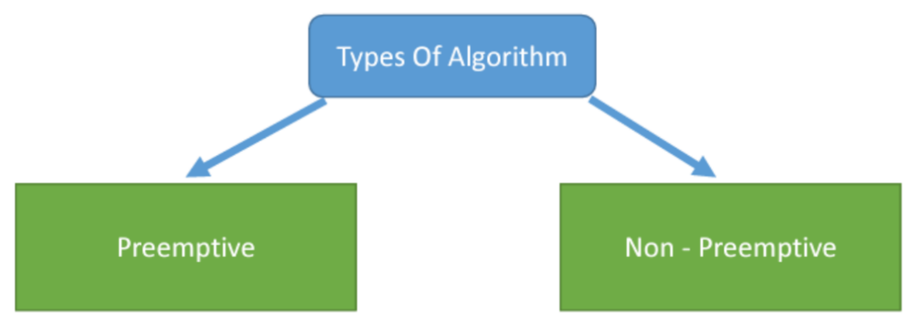
תזמון תהליכים

הקדמה

- המעבד מספק משאב הנקרא **זמן ריצה**
- תהליכים **מתחרים** בניהם על משאב זה
- מערכת ההפעלה צריכה **להקצות** את המשאב לתהליכים השונים
- באילו כללים חשוב שנעמוד בתהליך ההקצאה?
  - זמן תגובה
  - יעילות מעבד (המעבד מוקצה לעבודת תהליכים בפועל, ולא "בסרק")
  - חלוקת הזמן באופן הוגן בין תהליכים (מניעת הרעבה)
  - מתן עדיפות לתהליכים חשובים (Real Time Deadlines)

CPU – Scheduling Algorithm

- Used to **select** among the processes in memory that **are ready to execute**
- **Allocates** the CPU to one of them



תהליכים שיכולים לעצור אותן.
כלומר יכולים לקחת ממנו זמן מעבד.

תהליכים שלא יכולים
לעצור אותן.
לא ניתן לקחת ממנו
זמן מעבד.

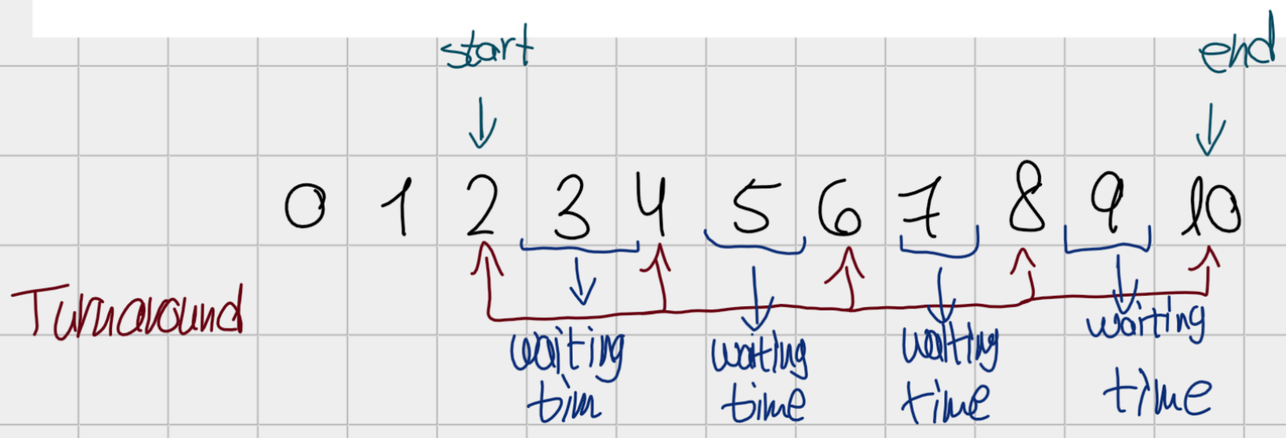
CPU – Scheduling Algorithm

- **CPU Utilization** – Average fraction of time for which processor or CPU is busy
- **Throughput** – no. of processes the system can execute in a period of time

} Maximize

- **Turnaround Time** – interval between submission of the process and completion
- **Response Time** – Amount of time it takes when a request was submitted until the first response is produced
- **Waiting Time** – The period of time a process spends waiting for CPU execution time

} Minimize



FCFS – FIRST COME FIRST SERVED

- FIFO based algorithm
- Non – preemptive
- Disadvantage?
  - Short processes wait for long ones

תרגיל כיתה - FCFS

נתונה טבלת התהליכים הבאה.

PROCESS	BURST TIME כמה זמן לרוץ	ARRIVAL TIME (ms) מתי הוא יגיע
P1	5	0
P2	4	2
P3	3	4
P4	6	6

א. מלאו את טבלת ההרצה בשיטת FCFS

ב. חשבו את זמן ההמתנה הממוצע של תהליך

(א)

PROCESS	EXECUTION TIME	WAITING TIME (ms)
P_1	0-5	0ms
P_2	5+4=9	5-2=3ms
P_3	9+3=12	9-4=5ms
P_4	12+6=18	12-6=6ms

עקרון מי יגיע ראשון

↓

* עובדים בתהליכים Non-preemtive.

Arrival time - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

process - P_1 P_2 P_3 P_4

↓

5-2=3

זמן לסיום P_1 — זמן שבו הגיע

* בדרך שבה תהליך ירקדה לתהליך הנוכחי לסיים (Non-preemtive).

* התהליך יתחיל לרוץ מרגע שבתהליך הישן עבר.

(ב)

$$\frac{\text{זמן המתנה}}{\text{מספר תהליכים}} = \frac{0+3+5+6}{4} = \frac{14}{4} = 3.5ms$$

ROUND ROBIN

- Uses round queue
- Preemptive
- Preemption based on clock (**time slicing**)
- Clock interrupt is generated at **periodic intervals**
- When an interrupt occurs, the currently running process is placed in the **ready queue** and the next ready job is selected
- **Advantages:**
  - No starvation. Waiting time is bounded by $(n-1) * q$
- **Disadvantage?**
  - Selecting proper slice time - **overhead**

מספר תהליכים

ברק זמן שנתן לתהליך לעני שעוקחים ממנו זמן ריצה.

* יכול לעבור בתהליך האמצעי.

* אם בתהליך לא סיים אני בוא חוזר בצורה מעגלית לתור.

ROUND ROBIN – תרגיל כיתה

נתונה טבלת התהליכים הבאה, ו- time slice = 2

PROCESS	BURST TIME	ARRIVAL TIME (ms)
P1	4	0
P2	5	1
P3	6	2
P4	1	4
P5	3	6
P6	2	7

א. מלאו את טבלת ההרצה בשיטת ROUND ROBIN

ב. חשבו את זמן ההמתנה הכולל של תהליך P2

Queue: $P_1\ P_2\ P_3\ P_4\ P_5\ P_6$

תור מעגלי.

* צריך לשים עם התהליך שנתק אז זמן מתנה ולך בצורה מעגלית לתור.

PROCESS	EXECUTION TIME(ms)
P_1	0-2 ms
P_2	2-4 ms
P_1	4-6 ms
P_3	6-8 ms
P_2	8-10 ms
P_4	10-12 ms
P_5	12-13 ms
P_6	13-15 ms
P_3	15-17 ms
P_2	17-19 ms
P_5	19-21 ms
P_3	21-23 ms

סיים התור.

זמן שצריך לסיים

4-2 = 2ms

5-2 = 3ms

2-2 = 0ms סיים

6-2 = 4ms

3-2 = 1ms

1-2 = -1ms = 0ms סיים

3-2 = 1ms

2-2 = 0ms סיים

time slice=2

Arrival time -	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
process -	P_1	P_2	P_3		P_4		P_5	P_6																

Queue =

תור מעגלי.

P_1	P_2	P_2	P_1	P_1	P_3	P_3	P_2	P_2	P_4	P_5	P_5	P_6	P_6	P_3	P_3	P_2	P_2	P_5	P_5	P_3	P_3		
P_2	P_1	P_1	P_3	P_3	P_2	P_2	P_4	P_4	P_5	P_6	P_6	P_3	P_3	P_2	P_2	P_5	P_5	P_3	P_3				
	P_3	P_3	P_2	P_2	P_4	P_4	P_5	P_5	P_6	P_3	P_3	P_2	P_2	P_5	P_5	P_3	P_2						
			P_4	P_4	P_5	P_5	P_6	P_6	P_3	P_2	P_2	P_5	P_5										
						P_6	P_3	P_3	P_2														

	PROCESS	EXECUTION TIME(ms)
	P_1	0-2 ms
1 ←	P_2	2-4 ms
	P_1	4-6 ms
	P_3	6-8 ms
8-4=4 ←	P_2	8-10 ms
	P_4	10-12 ms
	P_5	12-13 ms
	P_6	13-15 ms
	P_3	15-17 ms
17-10=7 ←	P_2	17-19 ms
	P_5	19-21 ms
	P_3	21-23 ms

ב) זמן סיום של תהליך - זמן התחלתי שלו

מלוסם הנשייה = זמן שבילו ריכה.

$P_2 = 1 + 4 + 7 = 12$ ms

↓

זמן שבילו הייו
עד שיקבלו.

SHORTEST PROCESS NEXT

- Non-preemptive רץ שהוא מסיים
- Process with shortest expected processing time is selected next
- Short process jumps ahead of longer processes
- If estimated time for process not correct, the operating system may abort it
- **Disadvantage?**
  - Each process must declare estimated execution time
  - Possibility of starvation for longer processes

תרגיל כיתה – SHORTEST PROCESS NEXT

נתונה טבלת התהליכים הבאה

PROCESS	BURST TIME	ARRIVAL TIME (ms)
P1	5 5-0=5	0
→ P2	7	1
→ P3	4	2
→ P4	3	4
P5	6	6

מלאו את טבלת ההרצה בשיטת SHORTEST PROCESS NEXT

PROCESS	EXECUTION TIME
P_1	0-5
P_4	5-8
P_3	8-12
P_5	12-18
P_2	18-25

* בדפקס הכל בסדר אם הבקבלים שהביא אם יש לו burst time יותר קצר אז לוקחים אותו.

SHORTEST REMAINING TIME

- Preemptive version of shortest process next policy
- Must estimate processing time and choose the shortest
- **Disadvantage?**
  - Possibility of starvation for longer processes

* אותו דבר של קודם אבל כאן הוא preemptive לכן יש כאן יותר content switch
* ויש אם המאבק היא כמעט יותר קצר אזי לוקחים אותו ומבטלים את הנורמ' לרעב.

תרגיל כיתה – SHORTEST REMAINING TIME

נתונה טבלת התהליכים הבאה

PROCESS	BURST TIME	ARRIVAL TIME (ms)
P1	1	8
P2	1	5
P3	7	2
P4	3	4
P5	8	2
P6	2	4
P7	5	3

מלאו את טבלת ההרצה בשיטת SHORTEST REMAINING TIME

PROCESS	EXECUTION TIME
P_3	2-3
P_7	3-4
P_6	4-5 + 5-6
P_2	6-7
P_4	7-8
P_1	8-9
P_4	9-11
P_7	11-15
P_3	15-21
P_5	21-29

$P_7 = 5-1 = 4ms$

$P_3 = 7 - 1 = 6ms$

זמן שנשאר לו

זמן שעבר כבר לו.

זמן כניסת של תהליכים נמשיכים גם בזמן

Arrival time - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ..... 21 29

process - $6 = P_3$ $P_7^{=4}$ $P_6^{=2}$ $P_2^{=1}$ P_6 P_2 $P_1^{=0}$ P_4 P_7 P_3 P_5

$8 = P_5$ P_3 $P_4^{=3}$

P_3 $P_7^{=4}$

$P_3^{=6}$

$P_5^{=8}$

$P_4^{=2}$

$P_7^{=4}$

$P_3^{=6}$

$P_5^{=8}$

} אם שיכים ע"פ הטבלה הנ מצא כאן כי הוא מעדיף ע"פ זמנים קצרים ביותר.

$1 = 1$

$P_6 = P_2$

אותו זמן לכן יין context switch

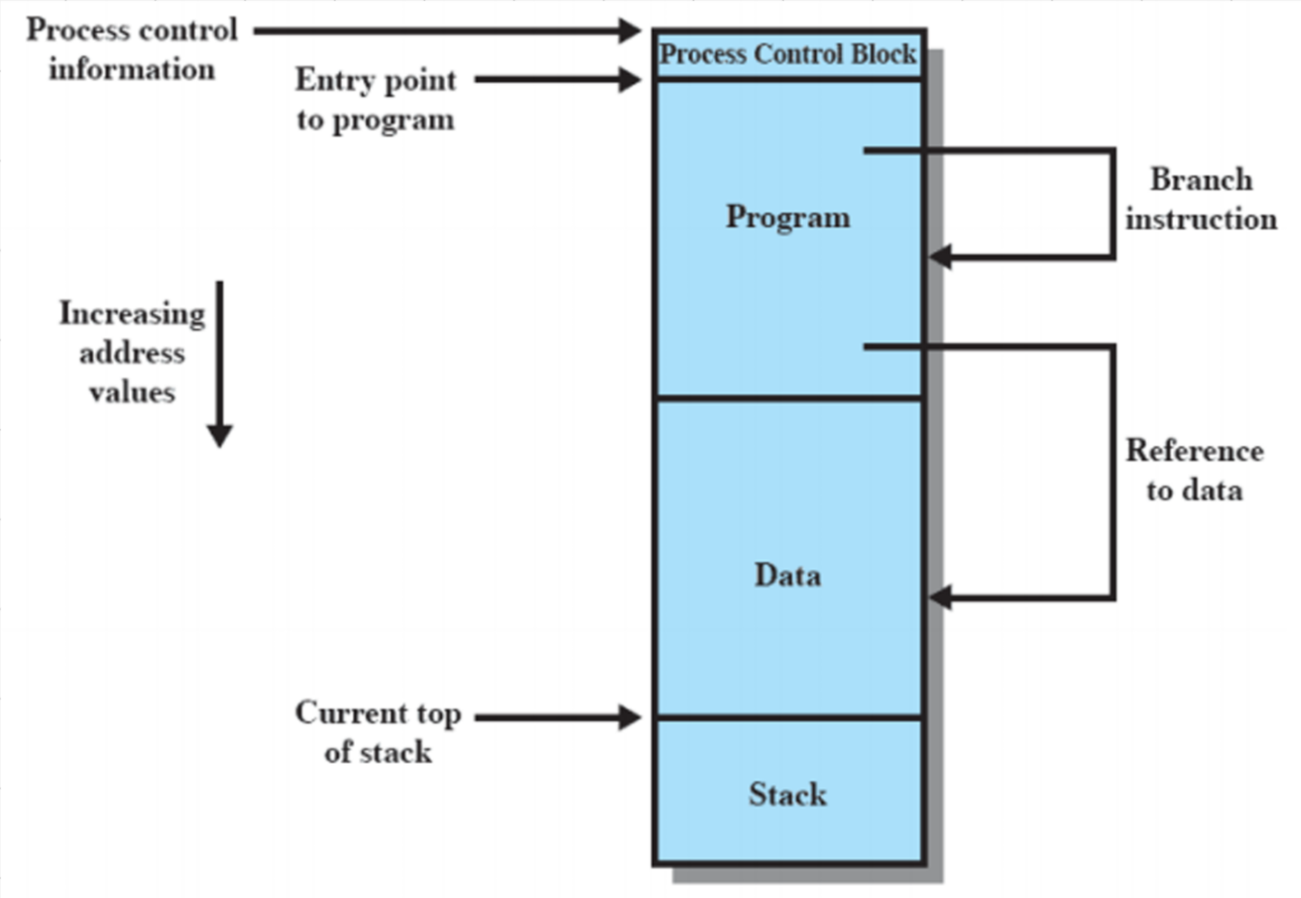
ע"פ מי שיש לו זמן יותר קצר.

סיום "X"

תרגול 7

ניהול זכרון

תהליך בזכרון:



מונחים:

1) Frame: Fixed length block of main memory.

2) Page: Fixed length block of data in secondary memory.

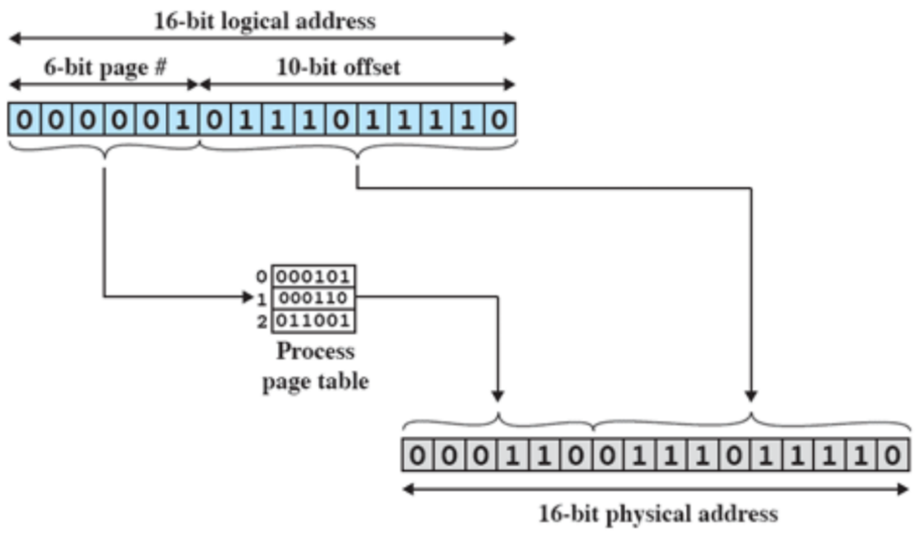
3) Segment: Variable length block of data that resides in secondary memory.

Paging

דפדוף

- Partition memory into small equal fixed-size chunks and divide each process into the same size chunks
- The chunks of a process are called ***pages***
- The chunks of memory are called ***frames***

- Operating system maintains a page table for each process
  - Contains the frame location for each page in the process
  - Memory address consist of a page number and offset within the page



4bit לכיוון, 4bit למספר הדף. ← 8 bits

תרגיל כיתה 1

נתונה מערכת בשיטת דפדוף (paging) שבה כל כתובת מורכבת מ 8 ביטים. גודל כל דף 2^4 בתים.

א. כמה דפים לכל היותר יש לתהליך במערכת?

א. מאחר כל דף כולו 2^4 4bit לכתובת מספר הדף.
אז יש לנו $2^4 = 16$. יכול מספר דפים

ס"ה חוקן 2^n כך ש- n כולו מספר הבתים לכתובת הבתים.

ב. נניח שקיימים התהליכים הבאים, עם דרישות הזיכרון הבאות:

A	5
B	3
C	4
D	2
E	5

מלא את טבלאות הדפים המתאימות כפי שהן נראות לאחר סדרת הפעולות הבאות:

- A נכנס לזכרון
- B נכנס לזכרון
- C נכנס לזכרון
- A יוצא מהזכרון
- D נכנס לזכרון
- A נכנס לזכרון
- B יוצא מהזכרון
- E נכנס לזכרון

#p	#f
0	2
1	3
2	4
3	12
4	13
A	

#p	#f
0	אין
1	אין
2	אין
B	

#p	#f
0	8
1	9
2	10
3	11
C	

#p	#f
0	0
1	1
D	

#p	#f
0	5
1	6
2	7
3	14
4	15
E	

רק נשארים המסגרות הנקיות.

סוגה של frame ←

#f	A - IN	B-IN	C-IN	A-OUT	D-IN	A-IN	B-OUT	E-IN
0	A0	A0	A0		D0	D0	D0	D0
1	A1	A1	A1		D1	D1	D1	D1
2	A2	A2	A2			A0	A0	A0
3	A3	A3	A3			A1	A1	A1
4	A4	A4	A4			A2	A2	A2
5		B0	B0	B0	B0	B0		E0
6		B1	B1	B1	B1	B1		E1
7		B2	B2	B2	B2	B2		E2
8			C0	C0	C0	C0	C0	C0
9			C1	C1	C1	C1	C1	C1
10			C2	C2	C2	C2	C2	C2
11			C3	C3	C3	C3	C3	C3
12						A3	A3	A3
13						A4	A4	A4
14								E3
15								E4

ג. בהתאם לטבלאות שקיבלת בסעיף הקודם, תרגם את הכתובות הוירטואליות הבאות לכתובות פיזיות. הנח כי המסגרות מתחילות בכתובת 00000000 ונמצאות באופן רציף בזכרון. אם לא ניתן לעשות זאת, הסבר מדוע.

1. כתובת וירטואלית במרחב הכתובות של תהליך A: 00110100

0 0 1 1 | 0 1 0 0

היסט שאנחנו לא נגעים בה.

* מחפשים באיזה frame הוא נמצא כאשר הכתובת "3" אחר כך, הופכים את יצוג ה-frame מ-Dec ול Bin.

#p	#f
0	2
1	3
2	4
3	12
4	13
A	

→ 12 = 1100

כתובת חפשה

1 1 0 0 0 1 0 0

2. כתובת וירטואלית במרחב הכתובות של תהליך B: 00000100

0 0 0 0 | 0 1 0 0

אין תהליך B ב-Ram לכן אין לנו page=0 ב-frame.

היסט

3. כתובת וירטואלית במרחב הכתובות של תהליך D: 00100100

2 = 0 0 1 0 | 0 1 0 0

היסט

אין לנו page בכתובת ב frame מספר 2.

#p	#f
0	0
1	1
D	

4. כתובת וירטואלית במרחב הכתובות של תהליך E: 00100101

2 = 0 0 1 0 | 0 1 0 1

היסט

#p	#f
0	5
1	6
2	7
3	14
4	15
E	

7 = 0111

כתובת חפשה

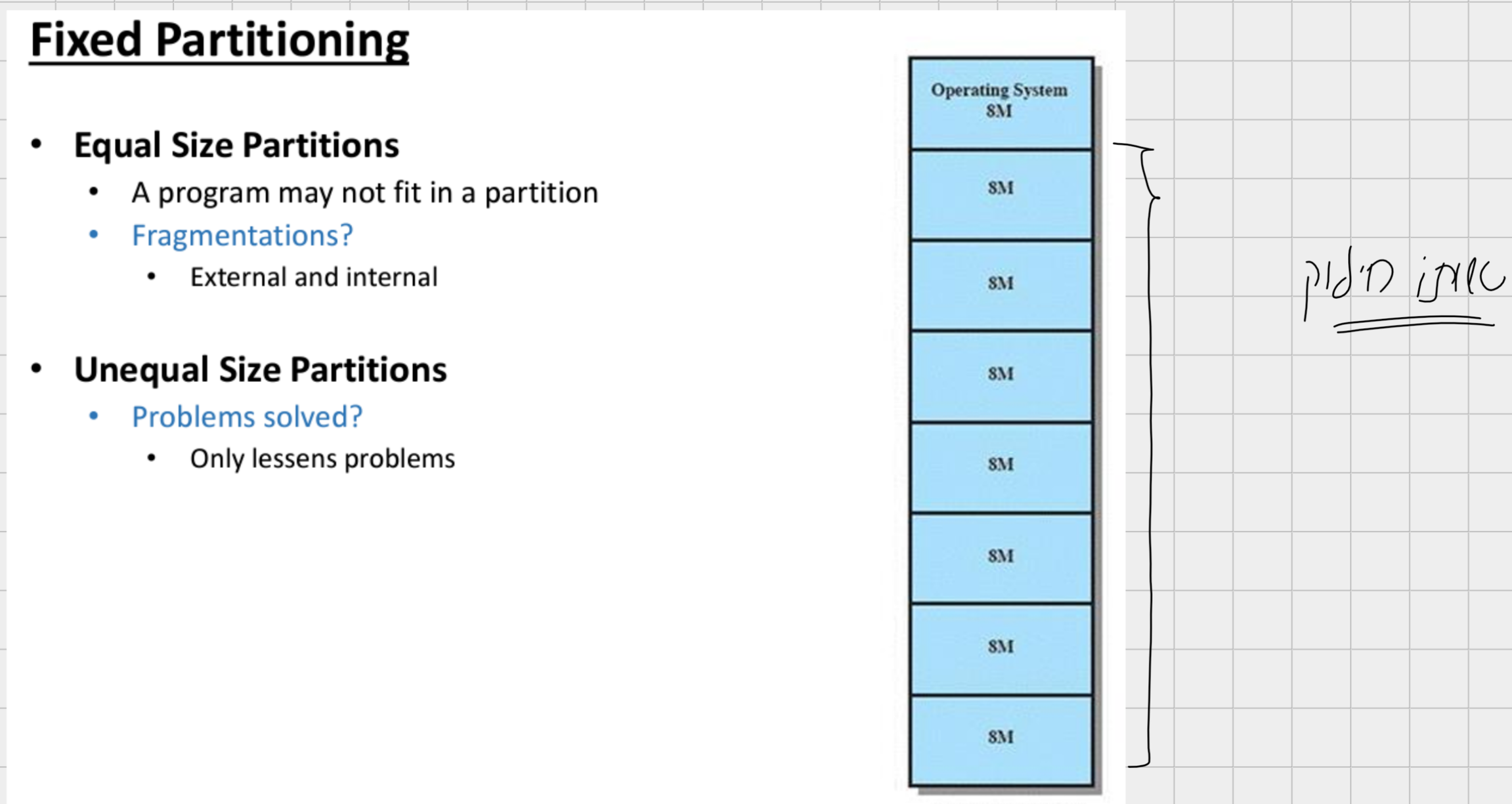
0 1 1 1 0 1 0 1

מושגים מפתח:

1) Fragmentation: Loss of memory because it can't be used
2) Internal: Fragmentation.
3) External: Fragmentation.

סוגי חילוק:

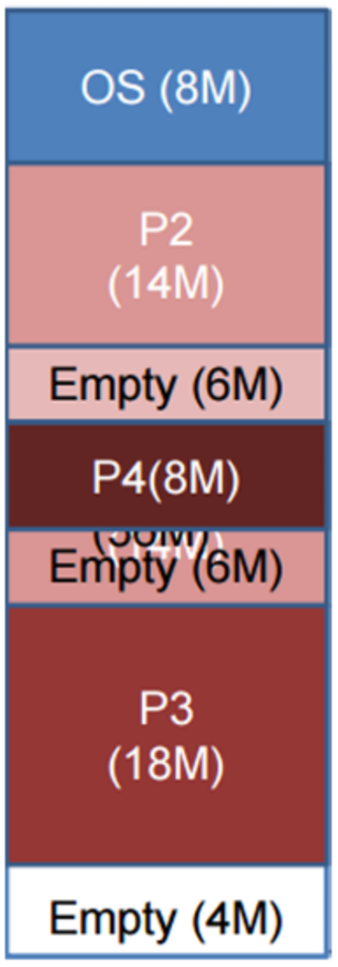
1) Fixed Partitioning - חילוק קבוע כלומר כל הכתובות והפלחים באותו חילוק



2) Dynamic Partitioning - מילוק לפי הצורך שם התהליך, לעומת התהליך נופס מ-RAM את מה שהוא צריך.

Dynamic Partitioning

- Blocks change their size dynamically
- Each process get exactly the size it asked for
- Fragmentations?
  - External only
  - Can be solved by **compaction** קומפקט
    - Expensive (consuming time & CPU)



How operating system decide which free block to allocate to a process?
OR:
How to satisfy a request of size n from a List of free holes

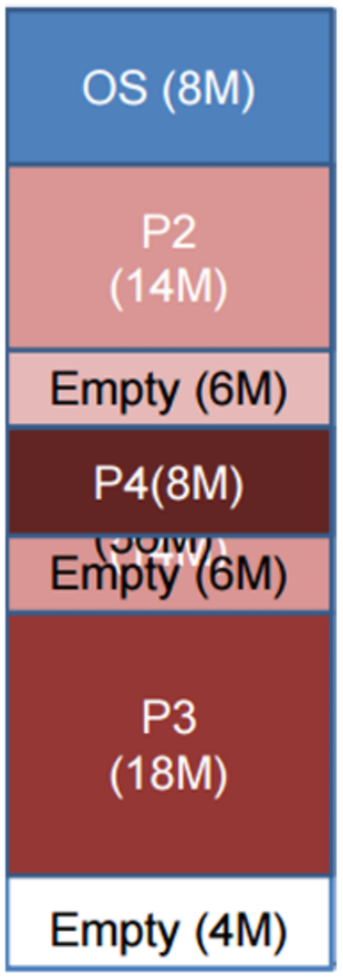
Placement Algorithm

- Used to decide which free block to allocate to a process
- **Goal:** to reduce usage of compaction
- **Possible algorithms:**
  - Best Fit
  - First Fit
  - Next Fit
  - Worst Fit
  - Buddy System

Placement Algorithm

- **First Fit –**
  - Allocate the first hole that is big enough. Searching start at the beginning of the set of holes
- **Next Fit –**
  - Allocate the first hole that is big enough. Searching start at the location where the previous next-fit search ended
- **Best Fit –**
  - Allocate the smallest hole that is big enough. We must search the entire list
- **Worst Fit –**
  - Allocate the largest hole that is big enough. We must search the entire list

מופקים את כל הרשימה



כי מ"ה Dynamic אנו יכולים לבדוק שוב מקום שכבר כתבנו בו.

תרגיל כיתה 2

נתונה תמונת הזיכרון בצד שמאל, ובנוסף נתון שההקצאה האחרונה שהתבצעה במערכת היא עבור תהליך P4.

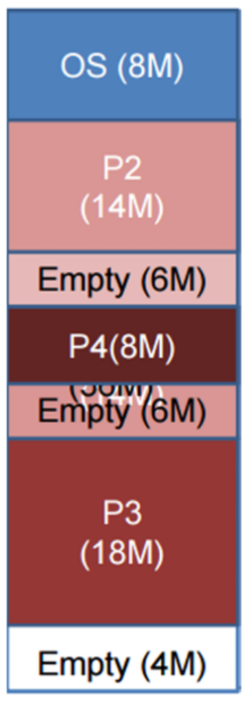
התהליכים הבאים רוצים להכנס למערכת:

זמן התחלה (שעות)	משך ריצה (שעות)	גודל זיכרון	תהליך
T=0	4	3M	P1
T=1	3	3M	P5
T=2	2	4M	P6
T=3	1	4M	P7

עבור כל אחת מהשיטות שלמדנו:

א. צייר את הזיכרון בכל אחד מהשלבים. במידה ותהליך נדחה, ציין זאת

ב. כמה פרגמנטציה פנימית וחיצונית נוצרו בזמן T=3 ?



א)

* משתמשים ב- First fit

← P_1(3M) - נשאר (3M)
עוקף אותו את P_5(3M)
אבל נשאר (0M)

P_6(4M) - נשאר (2M)
מקום לא מספיק
כדי להכניס את P_7
אבן מחפשים מקום אחר

P_7(4M) - נשאר (0M)

* נשאר לנו רק 2M שלא השתמשנו בה.

* משתמשים ב- Next fit

* מתחילים מ- P_4 ע"פ הנתון
(בהקצאה האחרונה שהתבצעה במערכת היא עבור תהליך P_4).

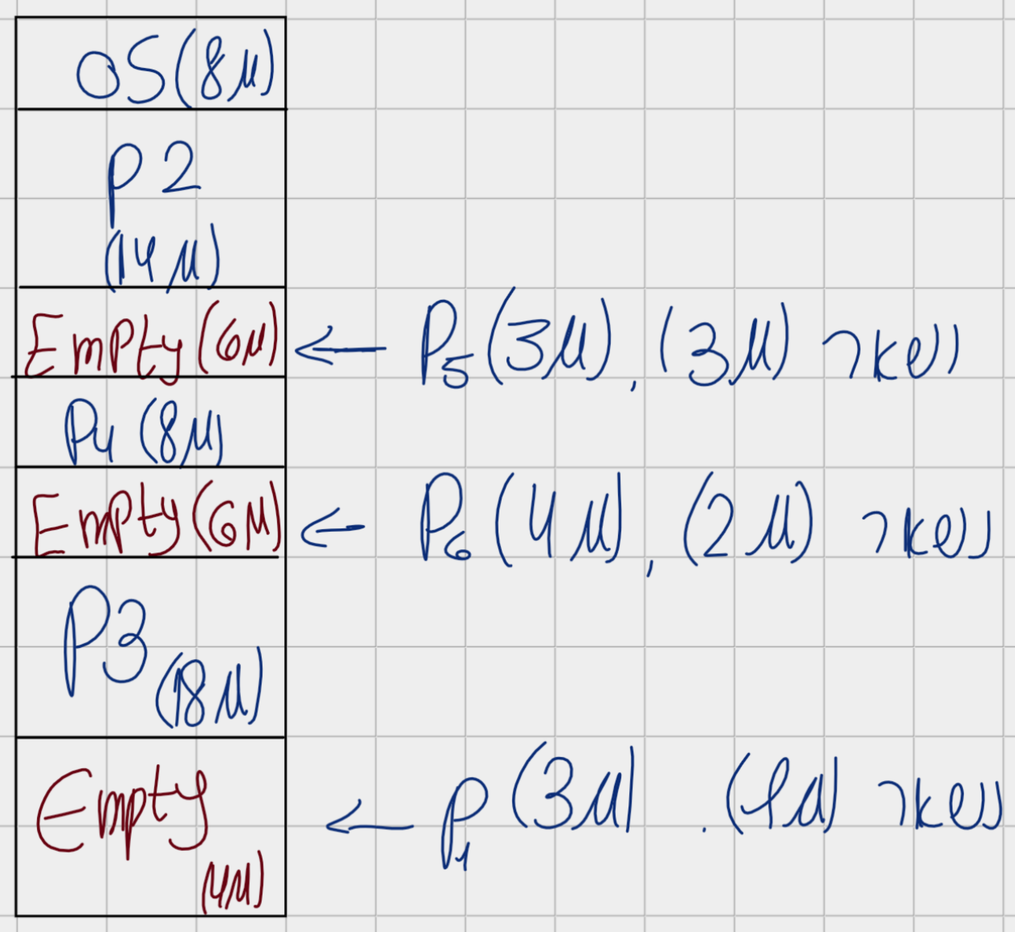
P_7(4M) , נשאר (2M) ← הסוף

P_1(3M) , P_5(3M) → נשאר (0M) (מתישים בפעם הראשונה מכיון)

P_6(4M) , נשאר (0M)

חוצרים עוד פעם להתחלה
אחרי שהגענו לסוף.

* נשתמש ב- Best Fit:



* לגבי P7 אין לו מקום לבכנס אותו.
יש כאן בסה"כ מקום בכמות של (6M) שנאבדו.

תרגיל

נתון זיכרון בגודל 1M.

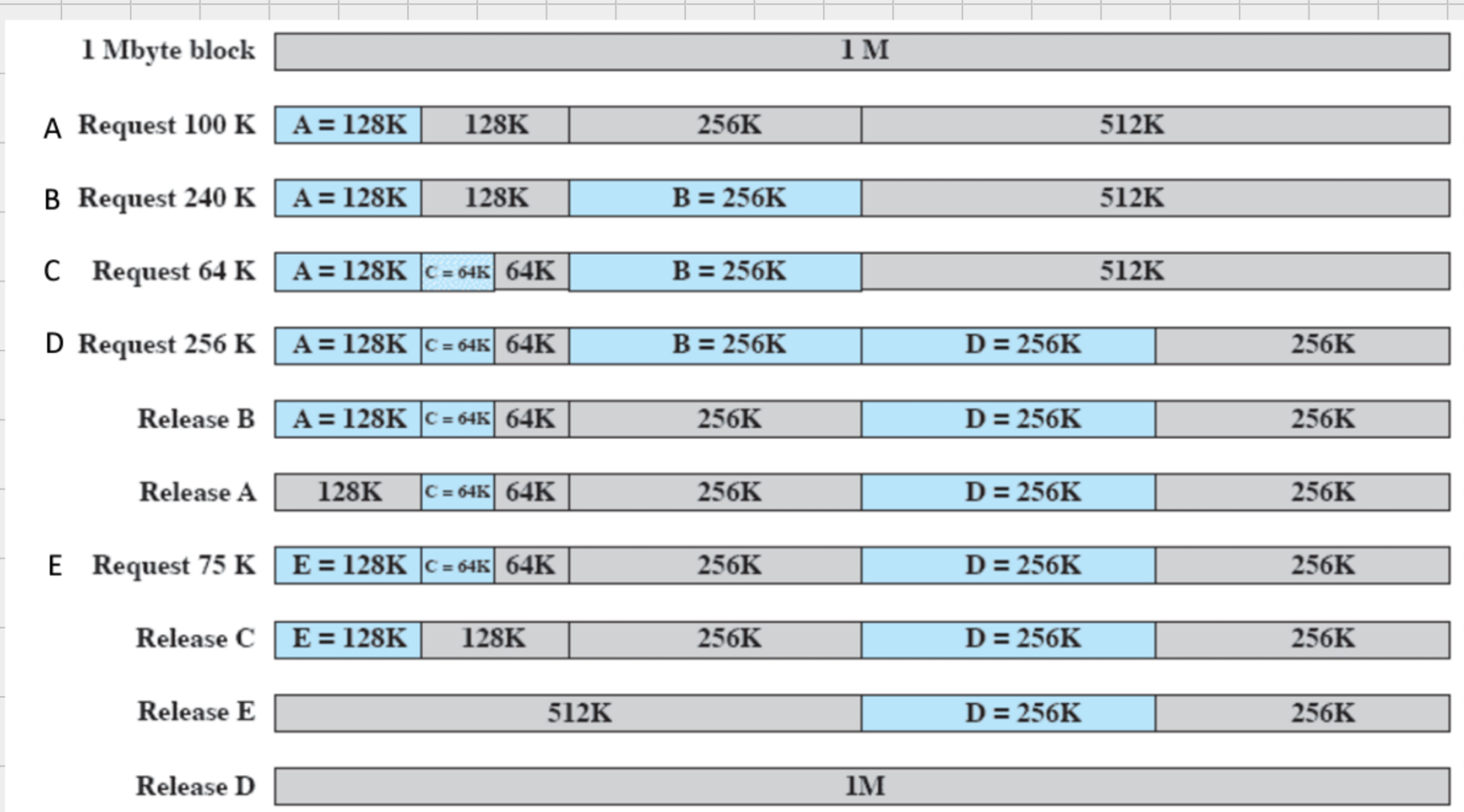
התהליכים הבאים רוצים להכנס למערכת:

זמן התחלה (שעות)	משך ריצה (שעות)	גודל זיכרון (K)	תהליך
T=0	5	100	A
T=1	3	240	B
T=2	5	64	C
T=3	6	256	D
T=6	2	75	E

א. צייר את הזיכרון בכל אחד מהשלבים בשיטת buddy system.

ב. נניח כי תהליך בגודל 600K מנסה להכנס למערכת מיד לאחר שתהליך A נכנס אליה בזמן T=0. כמה פרגמנטציה פנימית וחיצונית יהיו במערכת באותו רגע? הסבר

ג. כמה פרגמנטציה פנימית יש ב T=1 לאחר שתהליך B נכנס למערכת?



ما هو نظام الأصدقاء؟
نظام الأصدقاء هو أسلوب تخصيص الذاكرة المستخدمة في نظام تشغيل الكمبيوتر لتخصيص الذاكرة وإدارتها بكفاءة. تقوم هذه التقنية على تقسيم الذاكرة إلى كتل ذات حجم ثابت، وعندما تطلب إحدى العمليات الذاكرة، يقوم النظام بالعثور على أصغر كتلة متاحة يمكن أن تستوعب حجم الذاكرة المطلوبة.

نظام الأصدقاء هو طريقة للتحكم في الذاكرة تُستخدم في أنظمة العمل لتخصيص الذاكرة ديناميكيًا. يتم تطبيقه بشكل عام في الأنظمة التي يتم فيها تخصيص الذاكرة وإلغاء تخصيصها بشكل متكرر، بما في ذلك في بيئات أو هياكل متعددة المهام ذات متطلبات ذاكرة مختلفة. في نظام الأصدقاء، يتم تقسيم الذاكرة إلى كتل ذات طول ثابت، بانتظام بقدرات (على سبيل المثال، 1 كيلو بايت، 2 كيلو بايت، 4 كيلو بايت، وما إلى ذلك). عند تقديم طلب لتخصيص الذاكرة، يبحث الجهاز عن الكتلة ذات الحجم الصحيح. إذا تم تحديد الكتلة المناسبة، فسيتم تخصيصها بطريقة الاستعلام. ومع ذلك، إذا كان الحجم المطلوب لا يشكل أي كتلة موجودة بدقة، يقوم الجهاز بتخصيص كتلة أكبر وبعد ذلك يقسمها إلى كتل أصغر حتى يتم استلام كتلة ذات حجم دقيق.

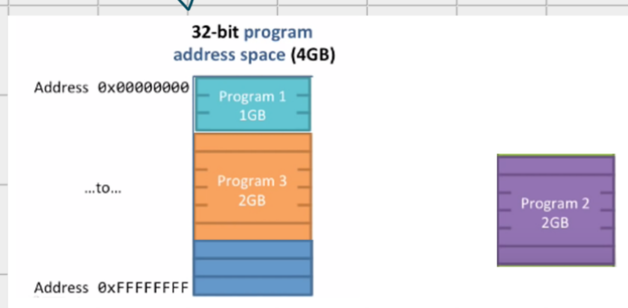
خوارزمية تقنية تخصيص ذاكرة نظام الأصدقاء
فيما يلي الخطوات المتبعة في تقنية تخصيص ذاكرة نظام الأصدقاء

تتضمن الخطوة الأولى تقسيم الذاكرة إلى كتل ذات حجم ثابت لها قوة 2 في الحجم (مثل 2، 4، 8، 16، 32، 64، 128، إلخ)*.
يتم تمييز كل كتلة بحجمها وتعريفها الفريد *.
في البداية، تكون جميع كتل الذاكرة مجانية ومرتبطة معًا في بنية شجرة ثنائية، حيث تمثل كل عقدة كتلة وتمثل أوراق الشجرة أصغر الكتل المتاحة*.
عندما تطلب إحدى العمليات الذاكرة، يعثر النظام على أصغر كتلة متاحة يمكنها استيعاب الحجم المطلوب. إذا كانت الكتلة أكبر من الحجم المطلوب، يقوم النظام بتقسيم الكتلة إلى كتلتين متساويتين * في الحجم.
يحدد النظام إحدى كتل الأصدقاء على أنها مخصصة ويضيفها إلى جدول تخصيص الذاكرة للعملية، بينما يتم إرجاع كتلة الأصدقاء الأخرى إلى تجمع الذاكرة الحرة وربطها مرة أخرى ببنية الشجرة الثنائية*.
عندما تقوم إحدى العمليات بتحرير الذاكرة، يقوم النظام بوضع علامة على الكتلة المقابلة على أنها مجانية ويبحث عن الكتلة الصديقة الخاصة بها. إذا كانت الكتلة الصديقة مجانية أيضًا، يقوم النظام * بدمج الكتلتين في كتلة أكبر ويربطها مرة أخرى ببنية الشجرة الثنائية.

כתרון וירטואלי

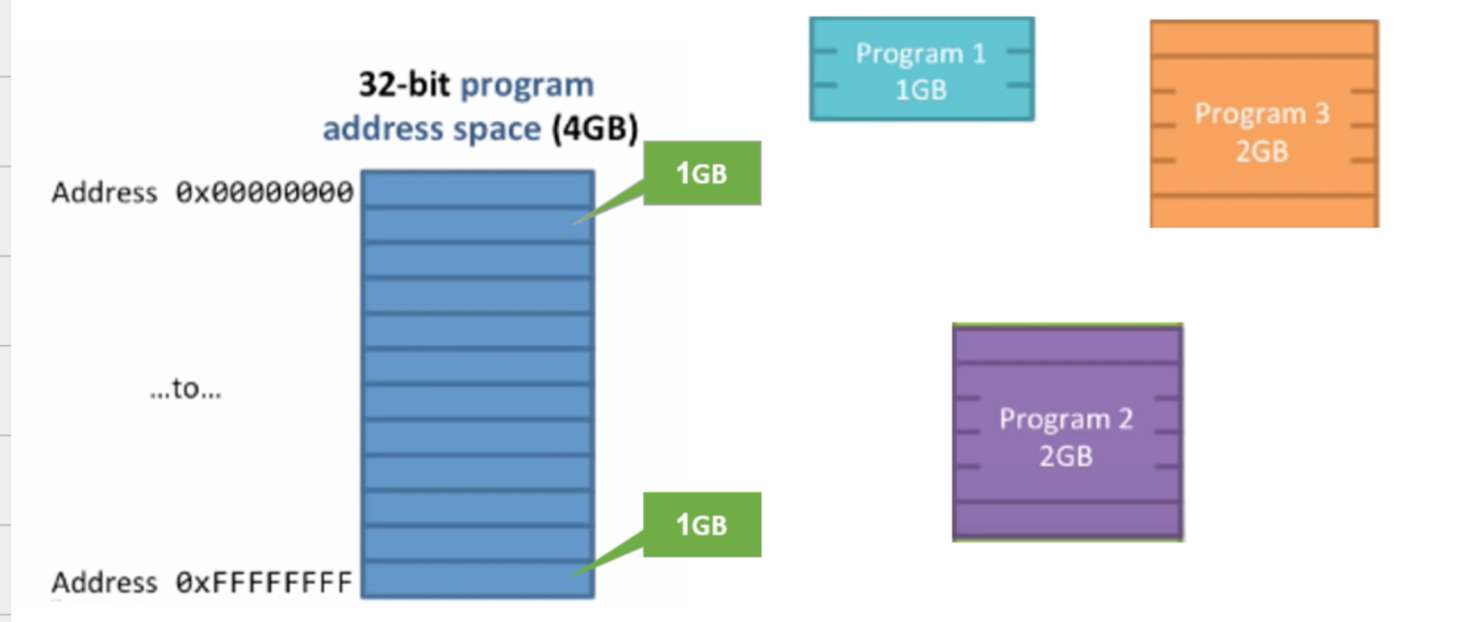
תרגיל 8

יש לנו 2 programs ו- 1 program
וריסם 3. ואנחנו רוצים להכניס אותן
לזכרון. אם נכנסו ס"ל הסדר



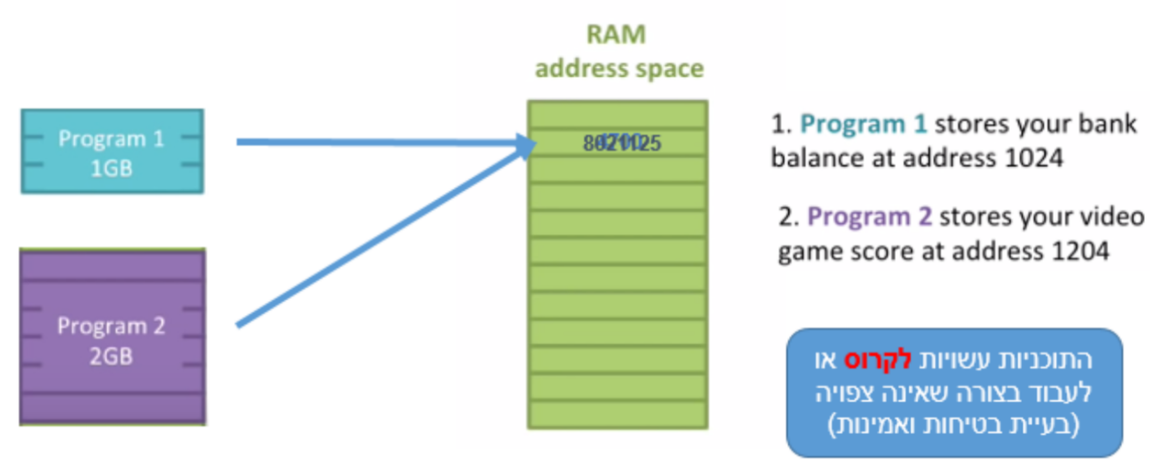
תכף לנו הסדר שאין מקום program 2
אם אם program 3 יבא סדין יש הסדר

בעיה 1: חורים במרחב הזיכרון (פרגמנטציה)

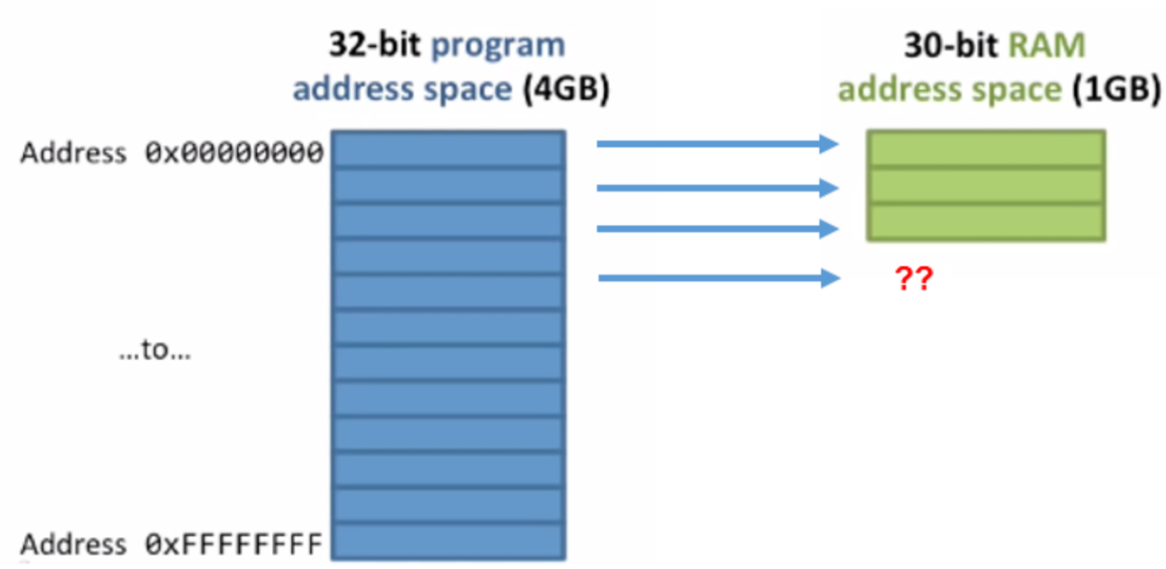


בעיה 2: שמירה על בטיחות תוכנה

- כל תוכנית יכולה לגשת למרחב כתובות בגודל 32-bit
- מה קורה כאשר כמה תוכניות מנסות לכתוב לאותה הכתובת?
  - נניח ששתי התוכניות מעוניינות לכתוב לכתובת 1024



בעיה 3: זיכרון קטן ממרחב הכתובות



פיתרון כללי לבעיות

- עיקר הבעיה: מרחב כתובות משותף לכל התהליכים
- מה יקרה אם נקצה לכל תהליך **מרחב כתובות וירטואלי משלו** ?
  - נמפה מרחב כתובות של כל תהליך למרחב כתובות נפרד בזיכרון

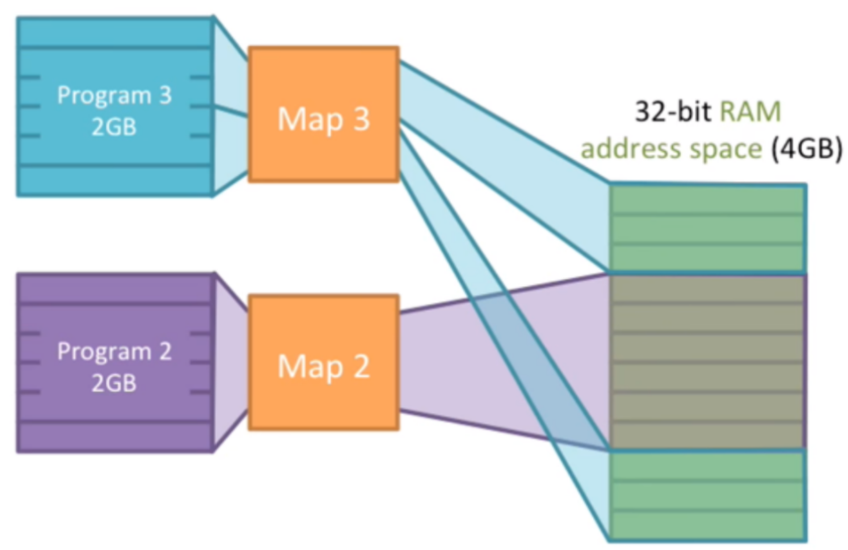
- מרחב כתובות של תהליך – זיכרון וירטואלי
- מרחב כתובות – זיכרון פיזי
- מיפוי – תרגום כתובת וירטואלית לפיזית

- שיטה זו מעניקה לנו גמישות בניהול מרחב הזיכרון
- כעת נוכל לפתור בעיות בטיחות, פרגמנטציה וגודל זיכרון (ואפילו להזיז חלק מהתהליך לזיכרון המשני)

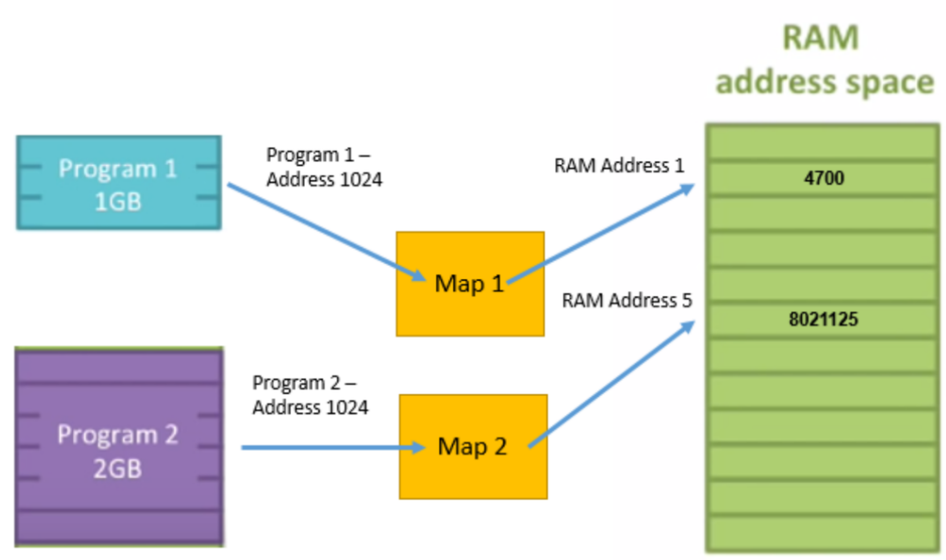
הרעיון: לוקחים כל תהליך ומחלקים אותו ל- page.
כמה עמודים שנמצא נצטרך? לזיכרון אך הראש שיש לנו עמוד שלא נמצא עד כה, אנו נעשה page fault ונביא אותו מהזיכרון הקשיח.

בלגן שהכתובת היא לוגית - אבל אפשר לבנות את הפרוסס והוא לא יחיב להיות רציף בזכרון

פיתרון 1: חורים במרחב הזיכרון (פרגמנטציה)



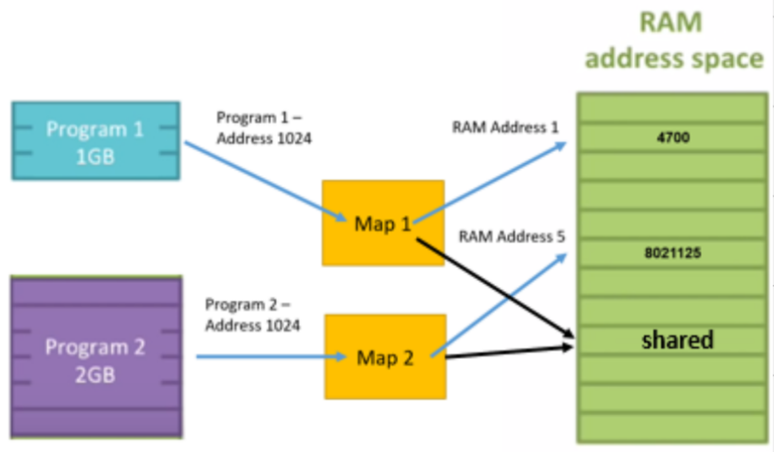
פיתרון 2: שמירה על בטיחות תוכנה



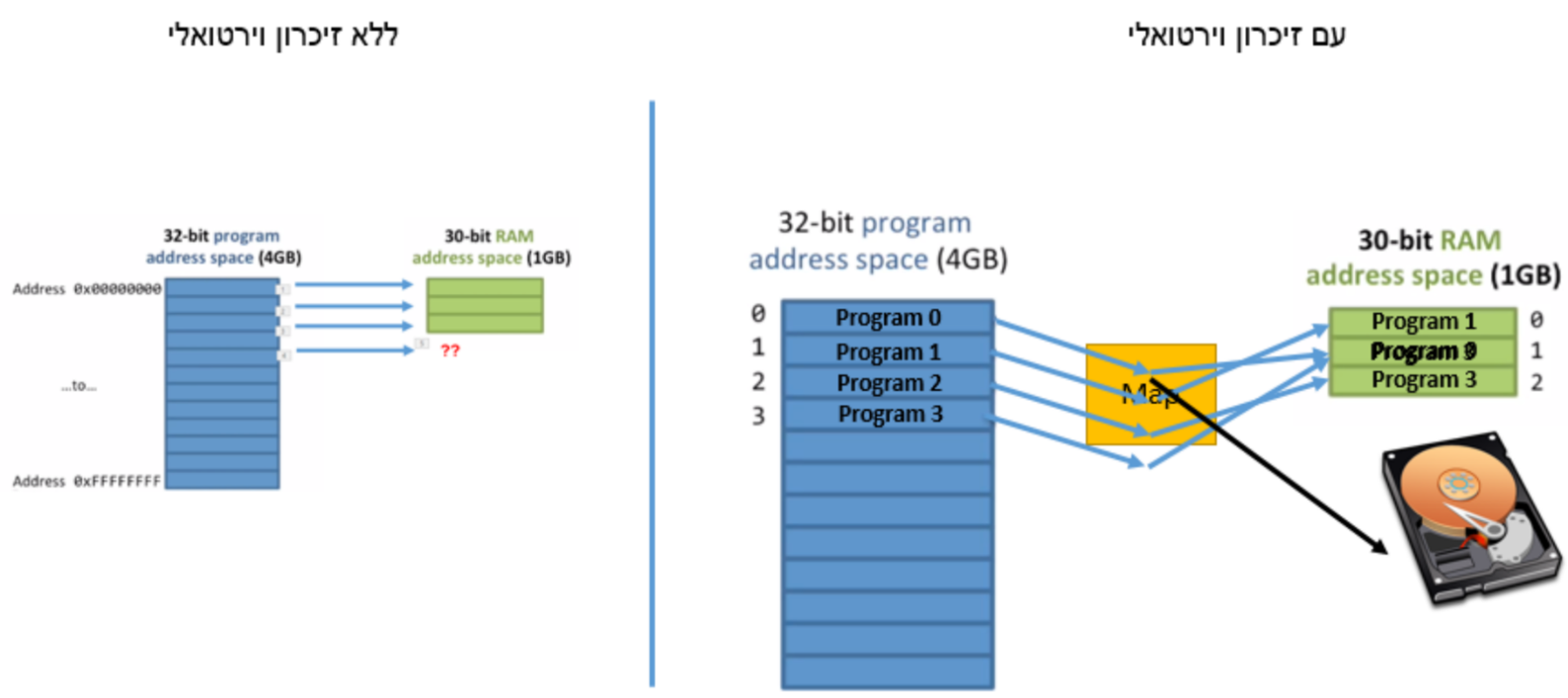
התרגום יודע לא לתת לשני פרוסס את אותו מרחב כתובות פיזי **כי הוא זה שמתרגם אותם**

פיתרון 2: שמירה על בטיחות תוכנה

- אלו חסרונות עשויים להיות להפרדת מרחבי תהליכים?
  - הפרדה מונעת שיתוף בין תהליכים (חיוני ונפוץ מאוד במ"ה)
- פיתרון?
  - שימוש באותו מיפוי כדי לאפשר שיתוף



פיתרון 3: זיכרון קטן ממרחב הכתובות



מה קורה כאשר הזכרון קטן ממרחב הכתובות ? נשתמש בHD... זה בהחלט ה VIRTUAL MEMORY שאנחנו מדברים עליו

פיתרון 3: זיכרון קטן ממרחב הכתובות

- מה יקרה לביצועים כאשר נאחסן ונשלוף תהליכים מהזיכרון המשני?
  - הביצועים יפחתו בצורה משמעותית
  - דיסקים קשיחים איטיים פי 1000x יותר מאשר זכרונות RAM
  - בכל פעם שנגמר הזיכרון הראשי וחייב לאחסן תהליך בזיכרון הראשי, "נשלם" על כך בביצועים ירודים
  - זו הסיבה שהגדלת נפח הזיכרון הראשי משפרת ביצועים (פחות גישות לזיכרון משני)
  - ארכיטקטורה חייבת לתמוך בכמות הזיכרון הראשי (למשל: 32 ביט = עד 4GB זיכרון ראשי)

כל פעם שנרצה מידע שבעצם נמצא בHD – אנחנו נצטרך לשלם על זה בזמן. (כי זה איטי)

Page Faults

- CPU need to use some page, but it's not in RAM (but in HD) — ~ 1 cycles
- Page Table says the page is on disk — ~ 100 cycles
- Hardware (CPU) generates a **page fault exception** — ~ 10,000 cycles
- The hardware jumps to the OS page fault handler to clean up
  - The OS chooses a page to evict from RAM and write to disk
  - If the page is **dirty**, it needs to be written back to disk first — ~ 40,000,000 cycles
  - The OS then reads the page from disk and puts it in RAM — ~ 40,000,000 cycles
  - The OS then changes the Page Table to map the new page — ~ 1,000 cycles
- The OS jumps back to the instruction that caused the page fault — ~ 10,000 cycles
  - (This time it won't cause a page fault since the page has been loaded)
- How long does this take?
  - An amazingly, incredibly, painfully long time

In the time in takes to handle one page fault you could execute **80 million cycles** on a modern CPU

Dirty page = Modified page… we cannot just erase it without saving it

Minimize Page Faults

- Page removed should be the page least likely to be referenced in the near future
- Which page is replaced?
- Most policies predict the future behavior on the basis of past behavior
- OPTIMAL
- FIFO – first in first out
- LRU – least recently used
- CLOCK

אלגוריתמי ניהול זיכרון(כשמדובר בשיטת הPaging ולא בבלוקים של זיכרון), נקרא גם page fault, החלפה והכנסה של דפים לפריימים(לזיכרון):
מסומן עם ביט valid invalid – אם הוא מסומן ב V הדף נמצא בזיכרון, אם הוא מסומן ב I הדף לא בזיכרון(אם צריך דף ב I יש page fault).
הפריימים חייבים להיות בגודל שווה לדפים.

תרגיל כיתה

נתון זיכרון בעל 4 מסגרות (frames). הנח שהזיכרון ריק בהתחלה. עבור סדרת הפניות הבאות (קרא משמאל לימין– כלומר, 1 ראשון) :

1, 3, 5, 2, 4, 3, 1, 2, 5, 2, 5, 1, 4

מלא את הטבלאות הבאות בהתאם לדפים הנמצאים בזיכרון, וסמן את הבקשות עבורן יש page faults עבור השיטות הבאות:

OPTIMAL

FIFO

LRU

CLOCK

FIFO - אם צריך לבצע מקום בקש חפש לבכנס והזיכרון מלא, נוציא את הבקש הראשון שנכנס (כמו תור). קל למימוש אבל לא יעיל

פיתרון FIFO

מחליפים את הדיסק ל-HD
דיסק hard

first in first out

Request	1	3	5	2	4	3	1	2	5	2	5	1	4
[redacted]	1	1	1	1	4	4	4	4	4	4	4	4	4
[redacted]		3	3	3	3	3	1	1	1	1	1	1	1
[redacted]			5	5	5	5	5	5	5	5	5	5	5
[redacted]				2	2	2	2	2	2	2	2	2	2
Page Fault	✓	✓	✓	✓	✓		✓						

אין fault page כי אין צורך לשלוף את 3 כי נמצא כבר.
מחלפים עם 3 כי זה האחרון שנכנס.

את כל השאר לא עושים fault page כי כבר נמצאים

Optimal – מפנים את הדף שלא נשתמש בו בזמן הקרוב, צריך לנבא את העתיד לכן הוא תיאורטי אבל מוציאים מי שנצטרך הכי מאוחר.

פיתרון OPTIMAL

cant page fault = 6

Request	1	3	5	2	4	3	1	2	5	2	5	1	4
	1	1	1	1	1	1	1	1	1	1	1	1	1
		3	3	3	3	3	3	3	5	5	5	5	5
			5	5	4	4	4	4	4	4	4	4	4
				2	2	2	2	2	2	2	2	2	2
Page Fault	✓	✓	✓	✓	✓				✓				

כאן נעיף את 5 כי הוא הכי מאוחר נשתמש בה.

נעיף את 3 כי לא נשתמש בה עכשיו

LRU – least recently used , מוציאים את מי שהשתמשנו בו בפעם הכי רחוקה, מי שהכי רחוק בטבלה שניגשנו אליו.

פיתרון LRU

Request	1	3	5	2	4	3	1	2	5	2	5	1	4
	1	1	1	1 הכי רחוק	4	4	4	4 הכי רחוק	5	5	5	5	5
		3	3	3	3	3	3	3	3	3	3	3 הכי רחוק	4
			5	5	5	5 הכי רחוק	1	1	1	1	1	1	1
				2	2	2	2	2	2	2	2	2	2
Page Fault	✓	✓	✓	✓	✓		✓		✓				✓

את 5 כבר השתמשנו בה לאחר הכי הרבה זמן.

את 3 כבר אחרון שבו השתמשנו בה לא מעניין אותו.

* = ל

-> = poniter

Clock – משתמשים בביט נוסף שנקרא use bit. כשדף נכנס לזיכרון בפעם הראשונה ה use bit מקבל ערך של 1. כשצריך להחליף דף עוברים לפי הסדר ממיקום נוכחי(עם מצביע) ומחליפים כל 1 ב0. הדף הראשון שערכו הוא 0 מוציאים ומכניסים במקומו את החדש עם ערך של 1, ושם יהיה הפוינטר לפעם הבאה(כלומר הוא הראשון שה1 שלו יתחלף ל0). אם לא מכניסים משהו חדש לא משנים את המיקום של הפוינטר(את הביט מדמים בטבלה עם הכוכבית). אם צריך להכניס משהו שכבר נמצא הפוינטר נשאר איפה שהוא ולמה שהיינו צריכים להכניס נחליף את ערך הביט ל1.

פיתרון CLOCK

יש לנו כאן מחצורות.

מחצורות

Request	1	3	5	2	4	3	1	2	5	2	5	1	4
[redacted]	1*	1*	1*	->1*	4*	4*	4*	*4	4	4	4	4	4*
[redacted]	->	3*	3*	3*	->3	->3*	3	3	*5	*5	*5	*5	*5
[redacted]		->	5*	5*	5	5	1*	*1	->1*	->1*	->1*	->1*	->1*
[redacted]			->	2*	2	2	->2	->*2	2	*2	*2	*2	2*
Page Fault	V	V	V	V	V		V		V				

מחליפים

מחליפים

מחליפים

יורדים למעלה

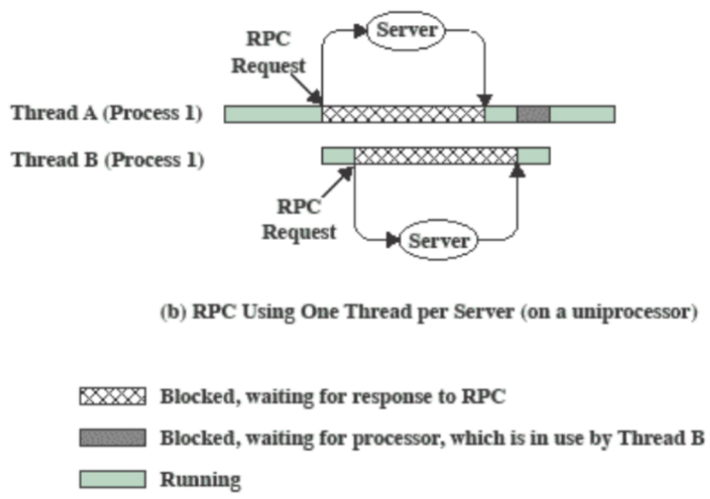
נחשבים page Fault אוטומטי.

כאן אנחנו יורדנו את כל הכוכביות וחוצרים מחצורת כדי לחפש מספר בלי כוכבית שכבר היה בראשון "ל".

5 אין לה כוכבית לכן מחליפים עם "ל".

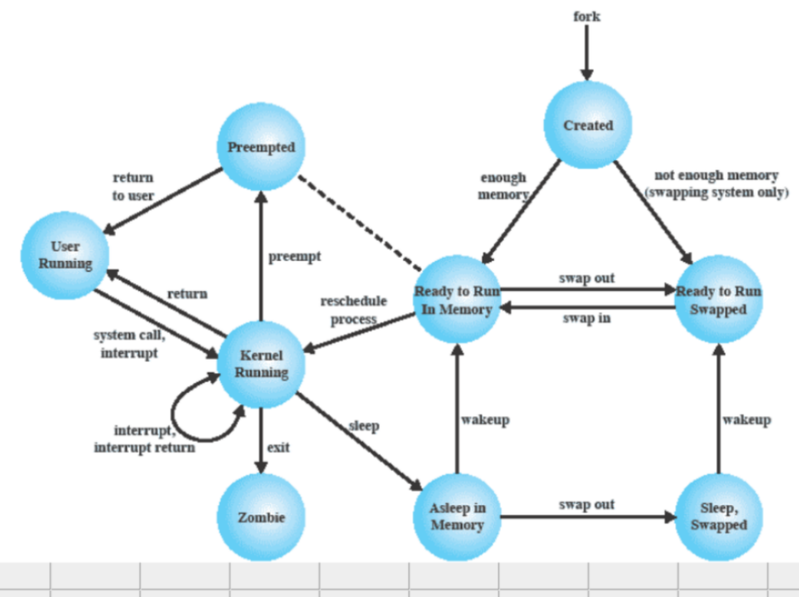
אם כבר נמצא במערכת אני רק תוסיף לו כוכבית.

RPC Using One Thread per Server

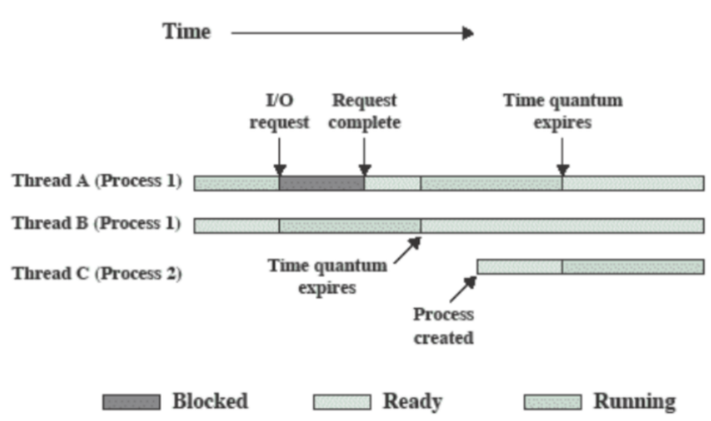


(b) RPC Using One Thread per Server (on a uniprocessor)

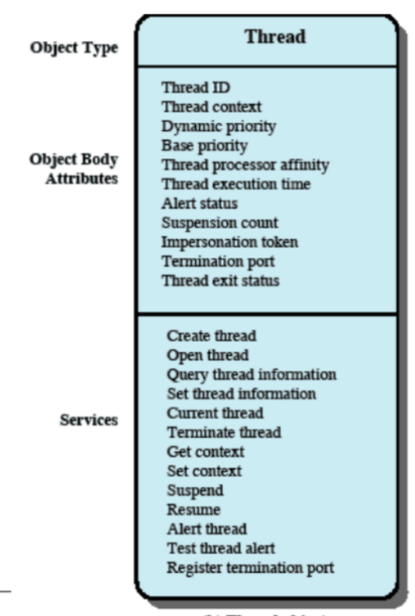
UNIX Process State Transition Diagram



Multithreading on a Uniprocessor



Windows Thread Object



(b) Thread object

Thread States

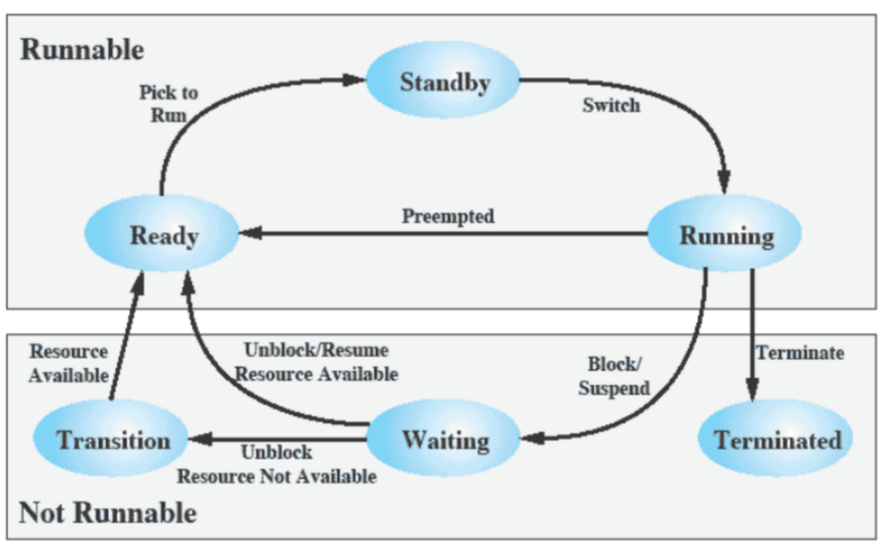


Figure 4.14 Windows Thread States

Linux Process/Thread Model

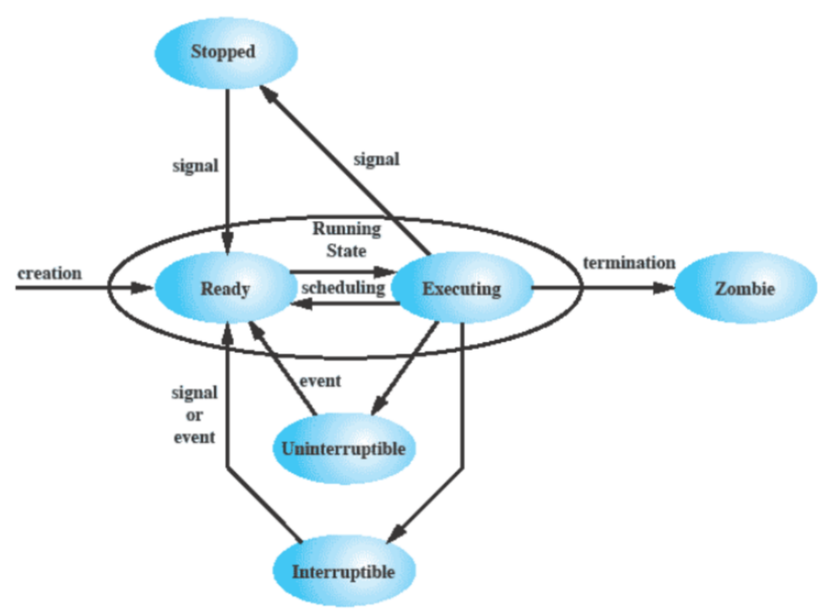


Figure 4.18 Linux Process/Thread Model

Multi-threading in C

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define  threads_num 5
int global_N  = 0;  // Shared variable

// Thread function to generate output
void *printMessage(void *threadid) {
    global_N++;
    long tid;
    tid = (long)threadid;
    printf("Hello, World! It's me, thread #%ld! the value of global_N is %d\n", tid, global_N);
    pthread_exit(NULL); // Properly exit the thread
}

int main() {
    pthread_t threads[threads_num]; // Array to hold two thread IDs
    int rc;
    long t;
    for(t = 0; t < threads_num; t++) {
        printf("In main: creating thread %ld\n", t);
        rc = pthread_create(&threads[t], NULL, printMessage, (void *)t);
        if (rc) {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }

    // Wait for threads to complete
    for(t = 0; t < threads_num; t++) {
        pthread_join(threads[t], NULL);
    }

    printf("In main: All threads completed successfully\n");
    pthread_exit(NULL); // Properly exit the main thread
}
```

Fork vs. Clone vs. POSIX

Feature	fork System Call	clone System Call	POSIX Threads (pthread)
Level	Kernel	Kernel	User-Space
Functionality	Creates a new child process	Creates a new process (can be thread-like)	Creates a new thread
Memory Sharing	Inherits parent's memory (copy-on-write)	Customizable	Shares parent's memory
Resource Management	Inherits parent's resources	Customizable	Shares parent's resources
Synchronization Support	Limited	Limited	Rich synchronization features
Use Case	Creating new processes	Flexible process/thread creation	Creating threads within a process

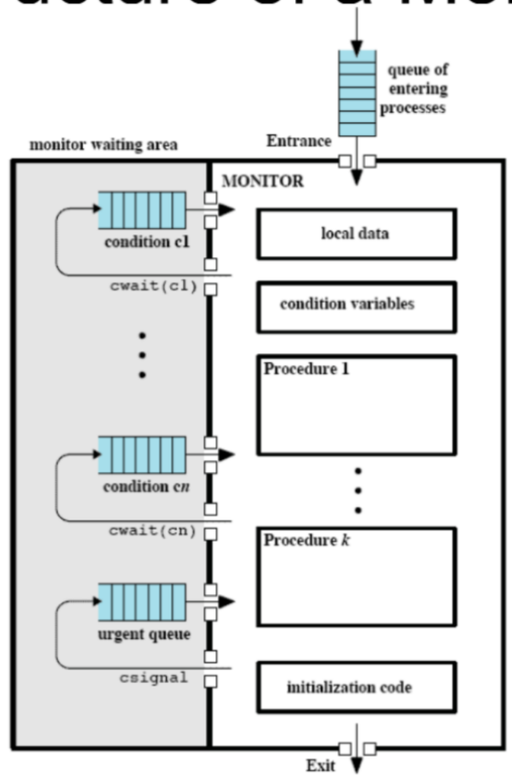
Fork vs. Clone vs. POSIX

- **fork**: This system call creates a new child process that is a nearly identical copy of the parent process. The child process inherits a copy of the parent's memory space (copy-on-write), open file descriptors, and other resources. fork is commonly used for creating new processes to execute programs or perform tasks independently.
- **clone**: This system call offers more flexibility than fork by allowing customization of various aspects of the new process creation, including memory sharing, resource inheritance, and behavior. While it can be used to create processes similar to fork, it can also be configured to create thread-like entities within the same address space. However, using clone for thread creation in user-space applications is not recommended due to the lack of high-level features for thread management.
- **POSIX Threads (pthread):** This library provides a user-space interface for creating and managing threads within a single process. Threads share the same memory space and resources as the parent process. Pthreads offer a rich set of features for thread synchronization (mutexes, semaphores) and communication, making them the preferred choice for creating and managing threads in user-space applications.

POSIX Threads (pthreads) on Linux,

- **In POSIX Threads (pthreads) on Linux,** the threads within a process are scheduled by the Linux kernel. The process that creates the threads doesn't directly manage their scheduling.
- **Kernel-Level Control:** The Linux kernel is responsible for managing all processes and threads running on the system. It has a scheduling algorithm that determines which process or thread gets access to the CPU for execution.
- **Thread Abstraction:** Pthreads provide a user-space interface for creating and managing threads. The creation process involves making system calls to the kernel, which then sets up the necessary data structures and allocates resources for the new thread.
- **Fair Scheduling:** The Linux kernel scheduler aims to provide fair CPU allocation among all runnable processes and threads. It considers factors like thread priority, waiting time, and I/O wait to make scheduling decisions.

Structure of a Monitor



Bounded Buffer Solution Using Monitor

```c
/* program producerconsumer */
monitor boundedbuffer;
char buffer [N];                                    /* space for N items */
int nextin, nextout;                                  /* buffer pointers */
int count;                                     /* number of items in buffer */
cond notfull, notempty;          /* condition variables for synchronization */

void append (char x)
{
    if (count == N) cwait(notfull);      /* buffer is full; avoid overflow */
    buffer[nextin] = x;
    nextin = (nextin + 1) % N;
    count++;
    /* one more item in buffer */
    csignal(notempty);                       /* resume any waiting consumer */
}
void take (char x)
{
    if (count == 0) cwait(notempty);   /* buffer is empty; avoid underflow */
    x = buffer[nextout];
    nextout = (nextout + 1) % N;
    count--;                                  /* one fewer item in buffer */
    csignal(notfull);                       /* resume any waiting producer */
}
{                                                          /* monitor body */
    nextin = 0; nextout = 0; count = 0;          /* buffer initially empty */
}
```

Solution Using Monitor

```c
void producer()
{
    char x;
    while (true) {
    produce(x);
    append(x);
    }
}
void consumer()
{
    char x;
    while (true) {
      take(x);
      consume(x);
    }
}
void main()
{
    parbegin (producer, consumer);
}
```

Bounded Buffer Monitor

```c
void append (char x)
{
    while(count == N) cwait(notfull);   /* buffer is full; avoid overflow */
    buffer[nextin] = x;
    nextin = (nextin + 1) % N;
    count++;                                 /* one more item in buffer */
    cnotify(notempty);                     /* notify any waiting consumer */
}

void take (char x)
{
    while(count == 0) cwait(notempty); /* buffer is empty; avoid underflow */
    x = buffer[nextout];
    nextout = (nextout + 1) % N;
    count--;                                 /* one fewer item in buffer */
    cnotify(notfull);                      /* notify any waiting producer */
}
```

Figure 5.17 Bounded Buffer Monitor Code for Mesa Monitor