

Page Replacement Algorithms

First-In First Out (FIFO)
Last-Recently Used (LRU)
The Second Change



Simulation scenarios

➤ Problem 36, Chapter 3, textbook

- A computer has four page frames. The time of loading, time of last access, and the R&M bits for each page area as shown below (the times are in clock ticks)

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |

- (a) Which page will NRU replace?
- (b) Which page will FIFO replace?
- (c) Which page will LRU replace?
- (d) Which page will second chance replace?

First-in-First-out (FIFO)

FIFO

- Ideas
 - Replace the page that has been in memory for the longest time


| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |

FIFO

➤ Ideas

- Replace the page that has been in memory for the longest time

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |

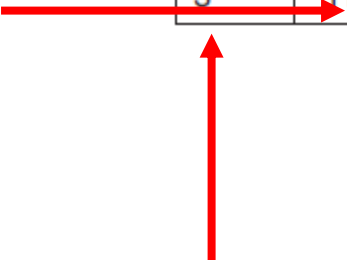


FIFO

➤ Ideas

- Replace the page that has been in memory for the longest time

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |



Page to be removed is 3

The Least Recently Used (LRU)

The Least Recently Used (LRU)

➤ Ideas:

- Keep track of when a page is used
- The page that has been used least recently is evicted

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |

The Least Recently Used (LRU)

➤ Ideas:

- Keep track of when a page is used
- The page that has been used least recently is evicted

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |



Check Last ref. column

The Least Recently Used (LRU)

➤ Ideas:

- Keep track of when a page is used
- The page that has been used least recently is evicted

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |

Check Last ref. column

Page unused for longest time will be removed

The Least Recently Used (LRU)

➤ Ideas:

- Keep track of when a page is used
- The page that has been used least recently is evicted

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |

Check Last ref. column

Page unused for longest time will be removed

Find page with smallest value in the column

The Least Recently Used (LRU)

➤ Ideas:

- Keep track of when a page is used
- The page that has been used least recently is evicted

This page

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |

Check Last ref. column

Page unused for longest time will be removed

Find page with smallest value in the column

The Least Recently Used (LRU)

➤ Ideas:

- Keep track of when a page is used
- The page that has been used least recently is evicted

Page 1 will be removed

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |

Check Last ref. column

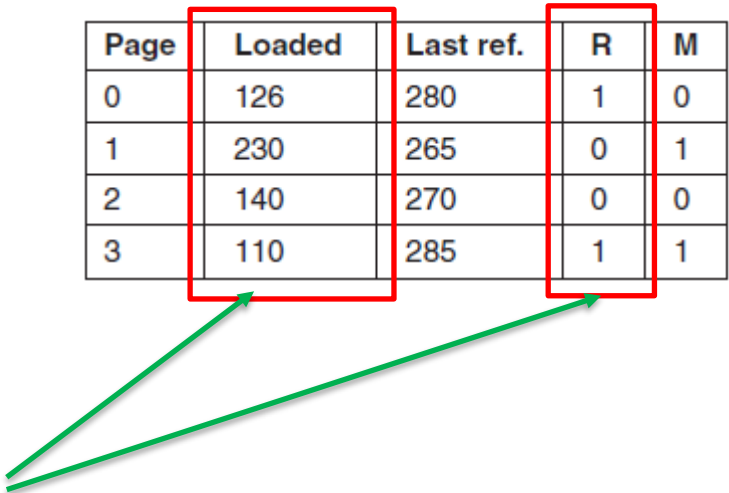
Page unused for longest time will be removed

Find page with smallest value in the column

The Second Chance Algorithm

The Second Chance Algorithm

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |



➤ Idea:

- Modification of FIFO algorithm to avoid the problem of throwing out a heavily used page by inspecting **R** bit of the oldest page.
- If **R** = 0: page is old and unused => remove the page
- If **R** = 1:
 - Bit is clear
 - Page is put onto the end of the list (as a new page)

The Second Chance Algorithm

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |



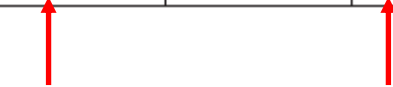
➤ Idea:

- Modification of FIFO algorithm to avoid the problem of throwing out a heavily used page by inspecting **R** bit of the oldest page.
- If **R** = 0: page is old and unused => remove the page
- If **R** = 1:
 - Bit is clear
 - Page is put onto the end of the list (as a new page)

- Check the page which is in the memory for the longest time

The Second Chance Algorithm

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |



➤ Idea:

- Modification of FIFO algorithm to avoid the problem of throwing out a heavily used page by inspecting **R** bit of the oldest page.
- If **R** = 0: page is old and unused => remove the page
- If **R** = 1:
 - Bit is clear
 - Page is put onto the end of the list (as a new page)

- Check the page which is in the memory for the longest time
- Check its R bit

The Second Chance Algorithm

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |

➤ Idea:

- Modification of FIFO algorithm to avoid the problem of throwing out a heavily used page by inspecting **R** bit of the oldest page.
 - If **R** = 0: page is old and unused => remove the page
 - If **R** = 1:
 - Bit is clear
 - Page is put onto the end of the list (as a new page)
- Check the page which is in the memory for the longest time
 - Check its R bit
 - As **R** = 1 => check the page which is in the memory for the second longest time

The Second Chance Algorithm

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |

➤ Idea:

- Modification of FIFO algorithm to avoid the problem of throwing out a heavily used page by inspecting **R** bit of the oldest page.
 - If **R** = 0: page is old and unused => remove the page
 - If **R** = 1:
 - Bit is clear
 - Page is put onto the end of the list (as a new page)
- Check the page which is in the memory for the longest time
 - Check its R bit
 - As **R** = 1 => check the page which is in the memory for the second longest time
 - The page second longest has **R** = 1 so it cannot be removed

The Second Chance Algorithm

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |

➤ Idea:

- Modification of FIFO algorithm to avoid the problem of throwing out a heavily used page by inspecting **R** bit of the oldest page.
 - If **R** = 0: page is old and unused => remove the page
 - If **R** = 1:
 - Bit is clear
 - Page is put onto the end of the list (as a new page)
- Check the page which is in the memory for the longest time
 - Check its R bit
 - As $R = 1$ => check the page which is in the memory for the second longest time
 - The page second longest has $R = 1$ so it cannot be removed
 - Find the next page with longest time

The Second Chance Algorithm

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |

➤ Idea:

- Modification of FIFO algorithm to avoid the problem of throwing out a heavily used page by inspecting **R** bit of the oldest page.
 - If **R** = 0: page is old and unused => remove the page
 - If **R** = 1:
 - Bit is clear
 - Page is put onto the end of the list (as a new page)
- Check the page which is in the memory for the longest time
 - Check its R bit
 - As **R** = 1 => check the page which is in the memory for the second longest time
 - The page second longest has **R** = 1 so it cannot be removed
 - Find the next page with longest time
 - This new page has **R** = 0 => **Select**

The Second Chance Algorithm

| Page | Loaded | Last ref. | R | M |
|------|--------|-----------|---|---|
| 0 | 126 | 280 | 1 | 0 |
| 1 | 230 | 265 | 0 | 1 |
| 2 | 140 | 270 | 0 | 0 |
| 3 | 110 | 285 | 1 | 1 |

Page to be removed is 2

➤ Idea:

- Modification of FIFO algorithm to avoid the problem of throwing out a heavily used page by inspecting **R** bit of the oldest page.
- If **R** = 0: page is old and unused => remove the page
- If **R** = 1:
 - Bit is clear
 - Page is put onto the end of the list (as a new page)
- Check the page which is in the memory for the longest time
- Check its R bit
- As **R** = 1 => check the page which is in the memory for the second longest time
- The page second longest has **R** = 1 so it cannot be removed
- Find the next page with longest time
- This new page has **R** = 0 => **Select**

