3. Allow more processes than fit in memory to run concurrently.
2. A process to execute even if all of the process is not in memory.
1. A process to be larger than physical memory, and

   The illusion of an infinite virtual memory enables

   much slower, cache or set of registers

   2. analogous to the way in which main memory is a much larger, but
   main memory

   1. treat disk (or other backing store) as a much larger, but much slower

   OS illusions:

   assumed it was all in memory.

   Up to now, the virtual address space of a process fit in memory, and we

   Today: Demand Pagged Virtual Memory

Segmented Paging: combine the best features of paging and segmentation

   • Each program consists of a number of segments
   • User view of programs
   • Segmentation

Last Class
Demand Paging Virtual Memory: Example

Key Ideas: Locality—the working set size of a process must fit in memory, and must stay there (90/10 rule).

- Else the effective memory access time will approach that of the disk memory.
- For efficiency reasons, memory access must reference pages that are in the valid bit of the page table.
- Once a page is brought from disk into memory, the OS updates the page using a valid bit of the page table (memory map) indicates if the page is on disk or memory.

Demand Paging uses a memory as a cache for the disk.
Difficult to get right due to branches in code.

- Errors may result in removing useless pages.
- If the OS is wrong = page fault
- Allows more overlap of CPU and I/O if the OS guesses correctly.

Pre-paging: OS guesses in advance which pages the process will need and pre-loads them into memory

- page-fault: interrupt that occurs when an instruction references a page that is not in memory
- Process must give up the CPU while the page is being loaded
- May remove a page from memory to make room for the new page

Demand paging: OS loads a page the first time it is referenced.

When to load a page?

When it is through with a page.

Request paging: process tells an OS before it needs a page, and then

- Difficult to do and is error-prone
- Allows virtual address space to be larger than physical address space

Overlays: application programmer indicates when to load and remove

When the process starts: he virtual address space must no larger than

When to load a page?
Page table must be more sophisticated so that it knows where to find a

- Swap space
- Physical memory
- The file system

At any given time, a page of virtual memory might exist in one or more of:

What happens when a page is removed from memory?

Swap Space

Implementation of Demand Paging

7. Continues fetching process (why not continue current process?)
6. Updates the page table entry
5. GETS interrupt that page is loaded in memory
4. Context switches to another process while I/O is being done
3. Starts loading new page into memory from disk
2. Invalidates the old page in the page table
1. Selects a page to replace (page replacement algorithm)

If the OS checks that the address is valid, if so, it

If the page is not in memory, trap to the OS on first reference

J: In memory (either on disk or bogus address)
V: Valid bit in page table indicates if page is in memory

A copy of the entire program must be stored on disk (why?)
All of this is still functionally transparent to the user.

1. If a page is not in memory, OS picks a TLB entry to replace and fills it in as follows
   a. invalidates TLB entry
   b. performs page fault operations as described earlier
   c. fills a TLB entry to replace and then fills it in the new

2. If the TLB hit rate is very high, use software to load the TLB on a TLB miss.

For some implementations, the hardware loads the TLB on a TLB miss.

**Updating the TLB**

What value must \( p \) have?

- If we want the effective access time to be only 10% slower than memory access time,
  \[ \text{Effective access time} = (d - 1) \times 200'000 + \frac{200'000 \times (d - 1)}{1 + 0.3d} \times \text{page fault time} \]

Let \( d \) be the probability of a page fault (0 < \( d \leq 1 \)).

- Spatial locality: If a process accesses an item in memory, it will tend to reference an adjacent item soon.
- Temporal locality: If a process accesses an item in memory, it will tend to reference the same item again soon.
- Fortunately, processes typically exhibit locality of reference.

**Performance of Demand Paging**
Transfer

of the source and destination are in memory before starting the block.

Solution: check that all pages between the starting and ending addresses

be undone.

Block transfer instructions where the source and destination overlap can't

Transparent Page Faults

Solution: unwind side effects

- Register 10 - mov a, (r10); moves a into the address contained in Register 10 and increments

What about instructions with side-effects (CISC)?

- Need hardware support to save

How does the OS transparently restart a failing instruction?

Transparent Page Faults
Number of page faults?

<table>
<thead>
<tr>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Frame 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
</tbody>
</table>

FIFO: First-In-First-Out

Reference stream: A B C A B A D A B C B
Virtual Pages: A B C D
3 page frames

Example: FIFO

LRU: Least Recently Used. A prioritization of MIN that works well if the accessed farthest in the future (probably optimal [belady66]).

MIN: (a,k) Opt. Look into the future and throw out the page that will be accessed farthest in the future (probably optimal [belady66]).

FIFO: First-In, First-Out. Throw out the oldest page.

Random: Amazingly, this algorithm works pretty well.

For a page fault, we need to choose a page to evict.

Page Replacement Algorithms
Number of page faults?

<table>
<thead>
<tr>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Frame 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>A B C</td>
<td>A B C</td>
</tr>
</tbody>
</table>

The longest time

LRU: Least Recently Used. Throw out the page that has not been used in

Example: LRU

Number of page faults?

<table>
<thead>
<tr>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Frame 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>A B C</td>
<td>A B C</td>
</tr>
</tbody>
</table>

Farthest in the future.

MIN: Look into the future and throw out the page that will be accessed

Example: MIN
### FIFO

With FIFO, the contents of memory can be completely different with a different number of page frames.

<table>
<thead>
<tr>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Frame 3</th>
<th>Frame 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

**Does adding memory always reduce the number of page faults?**

**Example: LRU**

When will LRU perform badly?
and improve performance

A good page replacement algorithm can reduce the number of page faults.

switch occurs.

- Processes can share memory more efficiently, reducing the costs when a context switch occurs.
- Processes can start faster because they only need to load a few pages (for code and data).
- Processes can run without being fully loaded into memory.
- Virtual address space can be larger than physical address space.

Benefits of Demand Paging:

**Summary**

**Why?**

With LRU, increasing the number of frames always decreases the number of page faults.

**Adding Memory with LRU**