Memory Paging

Paging Reentrants
Never overwrites the instruction sequence.

Application identifiers …
"is this application already in memory?"
If so, I can run it as a separate process with a different user data space, but the same code.

Page Table Size
Extremely large.
Solution: hierarchical paging.
Chapters, pages

Segmentation
User views programs as segments
- Code
- Data
- Heap
- Stack

Virtual Memory

Executing a process that is not completely in memory

Let's not require a process to be completely in memory. Put part of it in memory and part of it on disk.

Motivation
- All instructions in memory is restrictive
- All instructions in memory is wasteful – you don't always use every subroutine
- All data in memory is wasteful
- Program limit is the physical memory size

Benefits
Application is not limited to the physical memory size. Memory space is not wasted for things like subroutines that are never run.
Degree of multiprogramming goes up.
Only the pages that are used are brought into memory. Therefore, the likelihood of having a free page increases and there is less process swapping.

Programmer's View
Unlimited virtual memory
Rooted at logical address 0

Process memory image is as before, it's just now extremely large.

The logical address bits can be greater than what is actually available in physical memory.

Hardware View

Process pages are stored on disk
Pages swap into memory on demand
Disk performance is critical.

Demand Paging

Pager loads a process page on demand. Thus, those that are not used are not loaded.
The initial load requires a guess of pages.
Page table validity bits provide control

Page Fault ISR

Check page address for validity
Finds free frame in frame table
Schedule disk I/O to read frame
Update validity bits

Performance

$p = probability\ of\ page\ fault$

$effective\ time = (1 - p) \cdot t_{mem} + p \cdot t_{fault}$

Desire p to be small … very small.
Swap space used to decrease fault time.

Page Replacement

When no free frames are available, a victim page must be selected for swapped to disk. Tables must be updated.

Modify bits on read/write pages help to limit the I/O. If no modifications have been made, no swap-out must occur. Instead, you can just bring the new page in. All code is read-only, so this technique can be used.

First In First Out (FIFO)

The first frame that was ever allocated is the one that is chosen for swapping.

Least Recently Used (LRU)

Some variation is commonly implemented. Attempts to collect statistics about data locality. A measure is taken about page usage as memory is accessed. The ones you have recently used are probably going to be used again in the near future.

Thrashing can occur when pages are being continually replaced. A program requires so much memory that none is available. Every time it accesses a page, one has to be chosen for movement

out so the one just accessed can be moved into memory. Guess what? Right away, it accesses a different page. When memory is full, this swapping is constantly occurring.

The key is to minimize thrashing.