Operating Systems (OS) Concepts - IT 308: Lecture 23
Virtual Memory

**Batch:** B.Tech III year

**Instructor:** Rahul Muthu

DA-IICT
The total virtual space of the set of loaded processes can exceed 100% of physical memory by several orders of magnitude.
The total virtual space of the set of loaded processes can exceed 100% of physical memory by several orders of magnitude.

An image of the entire virtual-address space of an active process is maintained on the secondary storage disc.
- The total virtual space of the set of loaded processes can exceed 100% of physical memory by several orders of magnitude.

- An image of the entire virtual-address space of an active process is maintained on the secondary storage disc.

- The operating system provides for movement of parts of the virtual address space back and forth between secondary storage and main memory.
The total virtual space of the set of loaded processes can exceed 100% of physical memory by several orders of magnitude.

An image of the entire virtual-address space of an active process is maintained on the secondary storage disc.

The operating system provides for movement of parts of the virtual address space back and forth between secondary storage and main memory.

The details of the virtual memory management are normally irrelevant to the programmer. Programs can be written assuming a real memory system, and can be adapted to systems with different available memories without reprogramming.
The total virtual space of the set of loaded processes can exceed 100% of physical memory by several orders of magnitude.

An image of the entire virtual-address space of an active process is maintained on the secondary storage disc.

The operating system provides for movement of parts of the virtual address space back and forth between secondary storage and main memory.

The details of the virtual memory management are normally irrelevant to the programmer. Programs can be written assuming a real memory system, and can be adapted to systems with different available memories without reprogramming.

In systems with virtual memory, the speed of processes may be reduced if less physical memory is allocated to them, but the functionality is the same.
External fragmentation can be reduced where segmentation is used by loading the program into arbitrary sized spaces.
- External fragmentation can be reduced where segmentation is used by loading the program into arbitrary sized spaces.
- Since space allocation can be varied, the operating system can speed up the execution of important processes by allocating them more real memory.
External fragmentation can be reduced where segmentation is used by loading the program into arbitrary sized spaces.

Since space allocation can be varied, the operating system can speed up the execution of important processes by allocating them more real memory.

It can also increase the degree of multiprogramming by allocating smaller real memory space to processes.
External fragmentation can be reduced where segmentation is used by loading the program into arbitrary sized spaces.

Since space allocation can be varied, the operating system can speed up the execution of important processes by allocating them more real memory.

It can also increase the degree of multiprogramming by allocating smaller real memory space to processes.

The speed of a program’s execution in a virtual memory system can never exceed the speed of the same program in an otherwise similar system without virtual memory management.
External fragmentation can be reduced where segmentation is used by loading the program into arbitrary sized spaces.

Since space allocation can be varied, the operating system can speed up the execution of important processes by allocating them more real memory.

It can also increase the degree of multiprogramming by allocating smaller real memory space to processes.

The speed of a program’s execution in a virtual memory system can never exceed the speed of the same program in an otherwise similar system without virtual memory management.

Virtual memory provides user convenience and allows execution of large programs (or a set of programs with large address spaces), within smaller real memory. The benefits need to be analysed in terms of the effects of references to missing addresses in real memory during execution.
Virtual memory can be implemented using either segmentation or paging.
• Virtual memory can be implemented using either segmentation or paging.

• Assuming the page model, an address translation should get the logical address of the memory access, and then figure out if the corresponding page is in memory or secondary storage.
- Virtual memory can be implemented using either segmentation or paging.
- Assuming the page model, an address translation should get the logical address of the memory access, and then figure out if the corresponding page is in memory or secondary storage.
- The result of the translation is either a real memory address or a missing item error. In the latter case, the corresponding portion must be loaded into memory before the program execution can continue.
- Virtual memory can be implemented using either segmentation or paging.
- Assuming the page model, an address translation should get the logical address of the memory access, and then figure out if the corresponding page is in memory or secondary storage.
- The result of the translation is either a real memory address or a missing item error. In the latter case, the corresponding portion must be loaded into memory before the program execution can continue.
- The page-map tables (PMTs) can be much larger here, than in the case where no virtual memory is provided. The total number of pages of all active processes may far exceed the total number of pages in the real physical memory.
Virtual memory can be implemented using either segmentation or paging.

Assuming the page model, an address translation should get the logical address of the memory access, and then figure out if the corresponding page is in memory or secondary storage.

The result of the translation is either a real memory address or a missing item error. In the latter case, the corresponding portion must be loaded into memory before the program execution can continue.

The page-map tables (PMTs) can be much larger here, than in the case where no virtual memory is provided. The total number of pages of all active processes may far exceed the total number of pages in the real physical memory.

The PMT carries an extra bit indicating whether a page is loaded or not.
Virtual memory can be implemented using either segmentation or paging.

Assuming the page model, an address translation should get the logical address of the memory access, and then figure out if the corresponding page is in memory or secondary storage.

The result of the translation is either a real memory address or a missing item error. In the latter case, the corresponding portion must be loaded into memory before the program execution can continue.

The page-map tables (PMTs) can be much larger here, than in the case where no virtual memory is provided. The total number of pages of all active processes may far exceed the total number of pages in the real physical memory.

The PMT carries an extra bit indicating whether a page is loaded or not.

There is a separate file-map table (FMT) associated with every active process, which maps to the pages of the image of the full process resident in secondary storage.
In spite of the similarities between paged and segmented memory and virtual memory, special hardware provisions are necessary without which virtual memory cannot be implemented.
Instruction interruptibility

- In spite of the similarities between paged and segmented memory and virtual memory, special hardware provisions are necessary without which virtual memory cannot be implemented.

- Example:
  
  DECREMENT REG[1] AND BRANCH to address $Y$ IF the result is ZERO.
Instruction interruptibility

- In spite of the similarities between paged and segmented memory and virtual memory, special hardware provisions are necessary without which virtual memory cannot be implemented.

Example:

DECREMENT REG[1] AND BRANCH to address $Y$ IF the result is ZERO.

Assume that the operation code and address portion of the instruction are encoded as one word each.
In spite of the similarities between paged and segmented memory and virtual memory, special hardware provisions are necessary without which virtual memory cannot be implemented.

Example:
\[ \text{DECREMENT REG}[1] \text{ AND BRANCH to address } Y \text{ IF the result is ZERO.} \]

Assume that the operation code and address portion of the instruction are encoded as one word each.

These words might be the last word of one virtual page and the last word of the next virtual page of the program respectively. If the second page is not present in memory then a fault occurs in the middle of an instruction.
In spite of the similarities between paged and segmented memory and virtual memory, special hardware provisions are necessary without which virtual memory cannot be implemented.

Example:

DECREMENT REG[1] AND BRANCH to address \( Y \) IF the result is ZERO.

Assume that the operation code and address portion of the instruction are encoded as one word each.

These words might be the last word of one virtual page and the last word of the next virtual page of the program respectively. If the second page is not present in memory then a fault occurs in the middle of an instruction.

If a page fault occurs after the register has been decremented, then an interrupt requires a lot more saving to be done. The alternative is to rollback that part of the instruction.
Partial effects are undone and the entire instruction is reexecuted later.
Partial effects are undone and the entire instruction is reexecuted later.

The instruction is resumed at the exact point of interruption when the missing portion of the address space is brought in.
Partial effects are undone and the entire instruction is reexecuted later.

The instruction is resumed at the exact point of interruption when the missing portion of the address space is brought in.

Memory references are prechecked for page faults before the execution if the instruction.
Management of virtual memory

- Allocation policy is used to decide how much real memory to allocate to each active process.
Management of virtual memory

- **Allocation policy** is used to decide how much real memory to allocate to each active process.
- **Fetch policy** is used to decide which items to bring in to memory from secondary storage and when.
Management of virtual memory

- **Allocation policy** is used to decide how much real memory to allocate to each active process.

- **Fetch policy** is used to decide which items to bring in to memory from secondary storage and when.

- **Replacement policy** is used to decide which pages to evict when there is no free real memory and a page needs to be brought in.
Management of virtual memory

- **Allocation policy** is used to decide how much real memory to allocate to each active process.

- **Fetch policy** is used to decide which items to bring in to memory from secondary storage and when.

- **Replacement policy** is used to decide which pages to evict when there is no free real memory and a page needs to be brought in.

- **Placement policy** determines where to place a page or segment being brought in.
One of the major reasons virtual memory is successful is the fact that in any run of a program large portions of its address space are never accessed. A major example of such unaccessed portions of code is exception handling.
Locality

- One of the major reasons virtual memory is successful is the fact that in any run of a program large portions of its address space are never accessed. A major example of such unaccessed portions of code is exception handling.
- It is not possible to predict the portions of code accessed before the run.
Locality

- One of the major reasons virtual memory is successful is the fact that in any run of a program large portions of its address space are never accessed. A major example of such unaccessed portions of code is exception handling.
- It is not possible to predict the portions of code accessed before the run.
- Studies indicate that programs tend to favour certain subsets of their address space. This phenomenon is called *locality of reference*.
One of the major reasons virtual memory is successful is the fact that in any run of a program large portions of its address space are never accessed. A major example of such unaccessed portions of code is exception handling.

It is not possible to predict the portions of code accessed before the run.

Studies indicate that programs tend to favour certain subsets of their address space. This phenomenon is called \textit{locality of reference}.

\textit{Spatial locality} is the tendency of a program to reference clustered locations in its address space rather than randomly distributed locations.
One of the major reasons virtual memory is successful is the fact that in any run of a program large portions of its address space are never accessed. A major example of such unaccessed portions of code is exception handling.

It is not possible to predict the portions of code accessed before the run.

Studies indicate that programs tend to favour certain subsets of their address space. This phenomenon is called locality of reference.

Spatial locality is the tendency of a program to reference clustered locations in its address space rather than randomly distributed locations.

Temporal locality is the tendency of a program to reference the same location or a cluster of locations repeatedly for a block of time. A common example is for loops.
One of the major reasons virtual memory is successful is the fact that in any run of a program large portions of its address space are never accessed. A major example of such unaccessed portions of code is exception handling.

It is not possible to predict the portions of code accessed before the run.

Studies indicate that programs tend to favour certain subsets of their address space. This phenomenon is called *locality of reference*.

*Spatial locality* is the tendency of a program to reference clustered locations in its address space rather than randomly distributed locations.

*Temporal locality* is the tendency of a program to reference the same location or a cluster of locations repeatedly for a block of time. A common example is for loops.

Locality can be used to great advantage in framing policies for virtual memory environments.
Due to over-allocation of memory in virtual memory systems, a process may reference a missing page and there is no free real memory available to load it.
Due to over-allocation of memory in virtual memory systems, a process may reference a missing page and there is no free real memory available to load it.

The system can suspend the process pending freeing of memory.
Replacement policies

- Due to over-allocation of memory in virtual memory systems, a process may reference a missing page and there is no free real memory available to load it.
- The system can suspend the process pending freeing of memory.
- Alternatively, the system can evict a page from memory to create space to load the referenced missing page.
Replacement policies

- Due to over-allocation of memory in virtual memory systems, a process may reference a missing page and there is no free real memory available to load it.
- The system can suspend the process pending freeing of memory.
- Alternatively, the system can evict a page from memory to create space to load the referenced missing page.
- The second approach is more common, since the fault normally occurs due to the memory management system and not the process.
Replacement policies

- Due to over-allocation of memory in virtual memory systems, a process may reference a missing page and there is no free real memory available to load it.
- The system can suspend the process pending freeing of memory.
- Alternatively, the system can evict a page from memory to create space to load the referenced missing page.
- The second approach is more common, since the fault normally occurs due to the memory management system and not the process.
- Memory reference strings are used to analyse and system use and decide on a replacement policy.
Page replacement policies

- First-in First-out is simple to implement. However it is not ideal.
First-in First-out is simple to implement. However it is not ideal.

Least recently used is good assuming that past behaviour is an indicator of the future.
First-in First-out is simple to implement. However it is not ideal.

Least recently used is good assuming that past behaviour is an indicator of the future.

OPT is an optimal algorithm due to Belady which views the future to make a decision. It is only a theoretical algorithm and can be used only for comparison purposes to judge other algorithms. It is infeasible to implement in practice.
Global and local replacement

The page to be evicted is selected from the set of pages of the process which faults. This is called local replacement.
The page to be evicted is selected from the set of pages of the process which faults. This is called local replacement.

Global replacement policy can select any page in the entire memory to be evicted.