Page 8.7

Paging hardware

Page table

Physical memory

Logical address

Physical address

Paging is a memory-management scheme that permits the physical address to be noncontiguous. Paging resolves the considerable problem of finding memory chunks of varying sizes and the breaking since most programs on a process to be noncontiguous. Paging confines the considerable

Another possible solution to the external fragmentation problem is to allocate memory. This scheme can be expressed. Allocate memory; all holes move in the other direction, producing one large hole of memory; all holes move in the other direction, producing one large hole of memory. When compression is possible, we must determine the cost. The simplest compression algorithm is to move all processes forward one end of the process address space of the process to be unallocated. Thus, 8.4

Chapter 6 Main Memory

Page 288
Figure 8.6: Paging model of logical and physical memory

In Figure 8.6, the page size is shown as 16 bytes, which is a common size in many systems. The page size is an important parameter in the design of a memory management system, as it determines the size of the page table entries and the page size alignment requirements.

The page table is used to map logical addresses to physical addresses. Each process has its own page table, which is stored in the process's virtual memory space. The page table contains a list of page frames that are currently in memory, along with the physical addresses of the pages they contain.

The page size and the page table size are important factors in determining the performance of a paging system. A larger page size can reduce the number of page faults, but it can also make memory fragmentation more complex.

The basic method for implementing paging involves breaking down the memory address into its page number, page frame number, and offset. The page number is used to look up the page table entry, which contains the physical address of the page frame that contains the requested page.
Page 85

Figure 8.5 Pagen example for a 32-byte memory with 4-byte pages.

```
<table>
<thead>
<tr>
<th>Logical memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Page table

<table>
<thead>
<tr>
<th>Physical memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
```

Address translation: Logical address 13 maps to physical address 28. Logical address 15 maps to physical address 32. Logical address 14 maps to physical address 33. Logical address 12 maps to physical address 24. Logical address 16 maps to physical address 20. Logical address 11 maps to physical address 15. Logical address 10 maps to physical address 14. Logical address 9 maps to physical address 13.

Page size: 8 bytes

Page frame: 4-byte page size

Page number: 0

Page offset: 4

n

u

\[ n - m \]

u

p

Page table

Page number

Page offset
since the operating system manages physical memory, it must be aware of the allocation details of physical memory—where frames are allocated. Therefore, the CPU needs to be aware of which frames are available, how many total frames there are, and so on. This is

"page framing. The process owns its own address space outside of its page table, and the address inside only those pages that are needed by the process is mapped into a smaller address space. The problem is that pages of data in memory are not aligned in memory, so physical addresses do not necessarily correspond to virtual addresses. The process is required to be aware of the address translation hardware. The logical memory is translated into physical memory by the process, and the actual memory is translated into a smaller space by the operating system. The operating system shows the user the views of memory, and the physical memory is managed by the operating system. The user sees the views of the memory, and the operating system manages the physical memory. This architecture is called the page frame because the memory is divided into page frames."

When a page fault occurs in the system, the page is examined for process needs, and another page is assigned to the process. "If process needs are available, then the page is assigned to the process. This is called the next request algorithm. If process needs are not available, then the page is assigned to the process. This is called the least recently used algorithm."

"When a page fault occurs in the system, the page is examined for process needs, and another page is assigned to the process. "If process needs are available, then the page is assigned to the process. This is called the next request algorithm. If process needs are not available, then the page is assigned to the process. This is called the least recently used algorithm."
the page table so efficiently is a major consideration. The CPU dispatcher...

The page table can be done in several ways. In the simplest case, the page table is implemented as an array of descriptor registers. These registers should be built with very high-speed logic to make the reference. However, the page table values from the shadow registers and during the context switch period, the page table is loaded into the page table.

Each operating system has its own methods for storing page table. Most hardware support.

Figure 8.10 Free frames (a) before allocation and (b) after allocation.

Page 229 Chapter 8 Main Memory
Knee code are wired down.

meaning that they cannot be removed from the TLP's. Furthermore, some TLP's allow entities to be wired down.

special one for replacing replacement policies range from least recently used

zero reference if the TLP is already full, the operating system must

number and frame number in the TLP, of course, we will decide on the

we can use its access protocol (Paging) to also add the page

The TLP is used with page tables in the following way. The TLP contains

unnumbered between 64 and 1/2F.

however, it is extremely desirable to keep the address of a cache hit is

if the page number is provided in the TLP, the page frame number is rounded to the

and a word. When the associative memory is accessed, an

locking hardware and a cache called an associative look-aside buffer (TLB)

important under most circumstances. We might as well treat it as a

The TLP's memory access is slow (327 128 240 320 bytes). For the

which is combined with the page table to produce the actual address. We

are needed to access a file by reading the page table entry for the

can then access the desired page in memory. While the machine can

The problem with the approach is that it is difficult to access a

registers (PITB) points to the page table. Cache line addresses

computer only has as many substantially distinct instructions.

numbered. To prevent this, load the table's register(1) and the page

For these reasons, the use of 16-bit registers is kept in mind because, at

However, also the page table is very large (for example, a million entries)

The use of registers for the page table is satisfied by special

the page table is located in the memory map. Once these lines are kept in

The TLB's page table consists of 8 or 16 entries. The next entry

Accessing them in an example of the page table consists of 8 or 16 entries. The

only the portion of the page table registers are of course, prefetched, so that only

reloads these registers, just as it reloads the other registers. Instructions to load

Page 233
We will further explore the impact of the hit ratio on the TLB in Chapter 9.

For a 95-percent hit ratio, we have

\[ \text{effective access time} = 122 \text{ nanoseconds} \]

\[ = 0.98 \times 120 + 0.02 \times 220 \]

For a 90-percent hit ratio, we have

\[ \text{effective access time} = 140 \text{ nanoseconds} \]

\[ = 0.90 \times 120 + 0.10 \times 220 \]

The effective memory-access time, we write, each case by its probability:

The performance of times that a particular page number is found in the TLB is

Let us also consider the performance of times that a particular page number is found in the TLB.
Suppose, for example, that in a system with a 16-bit address space (16K), we have a program that should use only addresses 0 to 1048575 (CLPS = 16G). We set the bit for each page to allow or disallow access to the page. When this bit is set to '1', the associated page is in the page table.

One additional bit is generally associated to each entry in the page table; a value of 0 means the page is in memory, and a value of 1 means the page is not in memory.
Furthermore, recall that in Chapter 7 we described shared memory as a method of sharing the address space of a task by threads, described in Chapter 4. The sharing of memory among processes on a system is similar to the system’s interface between processes and so on. To elaborate the operating system's role, the operating system must be extended. The read-only nature of shared code should not be compromised. Other heavily used programs can also be shared—compilers, window systems, network libraries, database systems, and so on. Of course, the shared code is non-self-modifying code; it never changes during execution.

The kernel code is non-self-modifying code; it never changes during execution. Therefore, these processes must execute the same code at the same time. Each process has its own copy of registers and data, and each has its own data area. The shared memory has the address space that is shared between the processes. The shared memory is used to share data between processes. The shared memory is used to share code between processes.

8.4.4 Shared Pages
Hierarchical Paging

In this section, we explore some of the most common techniques for structuring the page table.

Figure 8.13 Shaping of code in a paging environment

8.5 Structure of the Page Table
The final 9 bits represent an offset in the desired page. By partitioning the page
section, the next 21 bits represent the logical page number of that section, and
the first 7 high-order bits of the logical address space is divided into two sections, each of which consists of 2^7 pages.

Each section represents a different part of the logical address space of a process.
The VAX architecture also supports a variation of two-level paging. The VAX
page table

<table>
<thead>
<tr>
<th>12</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>id</td>
</tr>
</tbody>
</table>

where id is an index into the outer page table, and id' is the displacement
and a 10-bit page offset. Thus, a logical address is as follows:

and a page of 4 KB. A logical address is divided into a page number

Figure 8.14  A two-level page-table scheme.

Memory

Page Table

Page Table

Page Table
The outer page table is still 2^14 bytes in size.

<table>
<thead>
<tr>
<th>12</th>
<th>10</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>id</td>
<td>id</td>
</tr>
<tr>
<td>and outer page inner page other</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The page table is made up of standard-size pages (214 entries, or 2^14 bytes). A 64-bit address space is used in a three-level paging scheme. Suppose that the outer page table gives us a three-level paging scheme. We can divide the outer page table into various ways. We can divide the outer page into 2^14 bytes. The obvious way to do this is to look like this:

<table>
<thead>
<tr>
<th>12</th>
<th>42</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>id</td>
</tr>
<tr>
<td>and outer page inner page other</td>
<td></td>
</tr>
</tbody>
</table>

The outer page table consists of 2^14 entries, or 2^14 bytes. The obvious way to divide this is to look like this:

The outer page table is made up of standard-size pages (2^14 entries, or 2^14 bytes). A 64-bit address space is used in a three-level paging scheme. Suppose that the outer page table gives us a three-level paging scheme. The obvious way to divide this is to look like this:

The outer page table is made up of standard-size pages (2^14 entries, or 2^14 bytes). A 64-bit address space is used in a three-level paging scheme. Suppose that the outer page table gives us a three-level paging scheme. The obvious way to divide this is to look like this:

The outer page table is made up of standard-size pages (2^14 entries, or 2^14 bytes). A 64-bit address space is used in a three-level paging scheme. Suppose that the outer page table gives us a three-level paging scheme. The obvious way to divide this is to look like this:

The outer page table is made up of standard-size pages (2^14 entries, or 2^14 bytes). A 64-bit address space is used in a three-level paging scheme. Suppose that the outer page table gives us a three-level paging scheme. The obvious way to divide this is to look like this:

The outer page table is made up of standard-size pages (2^14 entries, or 2^14 bytes). A 64-bit address space is used in a three-level paging scheme. Suppose that the outer page table gives us a three-level paging scheme. The obvious way to divide this is to look like this:
A common approach for handling address spaces larger than 32 bits is to use a **hashed page tables** address mapping. A probabilistic number of memory accesses to translate each logical address. For example, the 64-bit Linux kernel supports a multilevel page table scheme. The next step would be a four-level page table scheme, where the second-level page table is also paginated.