Alan Achtenberg

CSCE 410-500

HW1

DUE: 9/25/2014

**1. Ex. 1**

**• What is the purpose of interrupts?**

It is to stop the processor and direct its attention to another task. It is used to control the flow of the processor so that it can handle events that occur at unknown times.

**• What are the differences between a trap and an interrupt?**

A trap is software generated (ie. An instruction). An interrupt is hardware generated(ie. A pos signal on a wire)

A trap runs synchronously with the clock signal, an interrupt is asynchronous

**• Can traps be generated intentionally by a user program? If so, for what purpose?**

Yes, for making system calls. It allows the processor to switch into kernel mode for the call before switching back to user mode.

**2. Ex. 2**

**• What is the purpose of the command interpreter?**

The purpose is to parse commands from a user or file, and generate the correct system calls associated with the commands.

**• Why is it usually separate from the kernel?**

So that it may be modified or be interchangeable with different interpreters without changing the kernel

**• Would it be possible for the user to develop a new command interpreter using the system-call** **interface provided by the operating system?**

Absolutely yes, because the command interpreters are not a part of the kernel they have to interface thru system calls. Essentially a command interpreter is a user program.

**3. Ex. 3**

**Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in this order), how would**

**each of the algorithms:**

**(a) first-fit**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **100 KB** | **500 KB** | **200 KB** | **300 KB** | **600 KB** |
|  | 212 KB | 112 KB |  | 417 KB |

**(b) best-fit**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **100 KB** | **500 KB** | **200 KB** | **300 KB** | **600 KB** |
|  | 417 KB | 112 KB | 212 KB | 426 KB |

**(c) worst-fit algorithms**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **100 KB** | **500 KB** | **200 KB** | **300 KB** | **600 KB** |
|  | 417 KB |  | 112 KB | 212 KB |

**place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in this order)? Which algorithm makes the most efficient use of memory?**

The best fit algorithm makes the most efficient use of memory, it has the most allocations and least internal fragmentation

**4. Ex. 4**

**Compare the main memory organization schemes of**

**• contiguous-memory allocation**

**• pure segmentation**

**• pure paging**

**with respect to the following issues:**

**(a) external fragmentation**

Contiguous memory allocation- has very bad external fragmentation, when older processes are killed it can leave small holes every where.

Pure segmentation- still has external fragmentation except instead of fragmentation occurring with processes, it occurs when segments are allocated and deallocated, it is just on a smaller scale

Pure paging- does not have external fragmentation due to the page size being the same for all processes

**(b) internal fragmentation**

Contiguous memory allocation- has no internal fragmentation because it is allocated as a complete process contiguously

Pure segmentation- no internal fragmentation because segments are allocated contiguously

Pure paging- it has internal fragmentation, because the process may need a smaller amount of memory than the page can contain, but although the process has internal fragmentation, the amount of fragmentation is limited by the page size

**(c) ability to share code across processes**

Contiguous memory allocation- does not have the ability to share code across processes because it is not split up into accessible segments

Pure segmentation- because the code of a process is divided up into segments , the processes can be share segments. Ie. A user input segment could be shared between two processes with different main function segments, although it can share code, it can only share large chunks of code

Pure paging- with paging, individual pages can be shared with processes, because the pages are small (4KB) it can share much smaller amounts of code between processes making It more efficient with sharing.

**5. Ex. 5**

**Consider the following segment table:**

**Segment # Base Length**

**0 219 600**

**1 2300 14**

**2 90 100**

**3 1327 580**

**4 1952 96**

**What are the physical addresses for each of the following logical addresses?**

**Segment # Offset**

**0 430**

219+430= 649

**1 10**

2300+10= 2310

**2 500**

90+500= out of range, will cause exception/trap

**3 400**

1327+400= 1727

**4 112**

1952+112= out of range, will cause exception/trap

**6. Ex. 6**

**A certain computer provides its users with a virtual-memory space of 232 bytes. The computer has 218 bytes of physical memory. The virtual memory is implemented by paging, and the page size is 4096 bytes. A user process generates the virtual address 11123456 (decimal notation).**

**• Explain how the system establishes the corresponding physical location. You can show it using binary and/or decimal notation.**

It determines the physical page from the first 20 bits by looking it up in the page table, then it uses the last 12 bits to determine which byte to access in the physical page

**• What is the page number and page offset for this virtual address?**

|  |  |
| --- | --- |
| Virtual address(decimal 11123456) | |
| 0001 0001 0001 0010 0011 0100 0101 0110 | |
| Page table index/number (2^20 entries) | Page offset (2^12 entries) |
| 0001 0001 0001 0010 0011 (decimal 69923) | 0100 0101 0110 (decimal 1110) |

**7. Ex. 7**

**Assume we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty page is available or the replaced page is not modified, and 20 milliseconds if the replaced page is modified. Memory access time is 100 nanoseconds. Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds?**

.2 us=(1-P)\*.1us +(.3P)\*8000 us +.7P\*20,000 us

.2= .1-.1\*P +2400P + 14000P //divide by us

.1=16399.9\*P

P is approximately 6.0976E-6 or 6.0976E-4 %

**8. Ex. 8**

**Assume there is an initial 4 KB memory segment where memory is allocated using the Buddy System. Using Figure 9.27, p. 354 as a guide, write which segment sizes are available (you can draw the tree illustrating how the following memory requests are allocated):**

**• request 240 bytes**

**• request 120 bytes**

**• request 60 bytes**

**• request 130 bytes**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **4096** | | | | | | | |
|  | | | | | | | |
| **2048** | | | | | | | **2048** |
|  | | | | | | |
| **1024** | | | | | | **1024** |
|  | | | | | |  |
| **512** | | | | **512** | |
|  | | | |  | |
| **256** | **256** | | | **256** | **256** |
| 240 |  | | | 130 |  |
|  | **128** | **128** | |
|  | 120 |  | |
|  | | **64** | **64** |
|  | | 60 |  |

Available Free segment sizes are 2048, 1024, 256, 128, 64

**Next, write which segment sizes are free and available after the following releases of memory. Which segment is still in use? Perform coalescing whenever possible.**

**• release 240 bytes**

**• release 60 bytes**

**• release 120 bytes**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **4096** | | | | | |
|  | | | | | |
| **2048** | | | | | **2048** |
|  | | | | |
| **1024** | | | | **1024** |
|  | | | |  |
| **512** | | **512** | |
|  | |  | |
|  |  | **256** | **256** |
|  |  | 130 |  |

Available Free segment sizes are 2048, 1024, 512, 256

A segment of 256 bytes is still in use, which in turn disallows allocation of the parent segments 512, 1024, 2048, and 4096