**Assignment-4**

**CSCI 540 Computer Architecture Poojitha Chennuru (CWID -50307728)**

These are short questions (5% for each).

Please make your answer brief. You may draw figures if you need to.

Questions on Chapter 6:

1. What is the advantage of virtual memory comparing to overlays?

**Answer:** The advantage of virtual memory when compared to overlays programs are divided into number of pieces, each of which could fit in the memory. The programmer was responsible for breaking the program into overlays, deciding where in the secondary memory each overlay was to be kept, arranging for the transport of overlays between main memory and secondary memory, and in general managing the whole overlay process without any help from the computer.

A method for performing the overlay process automatically, without the programmer even knowing that it was happening is called virtual memory. It had the advantage of freeing the programmer from a lot of annoying bookkeeping.

1. What device translates the virtual address to physical address?

**Answer:**

The MMU is responsible for the translation between virtual memory addresses and physical memory addresses. The MMU splits this address into the three parts. In decimal, the three parts are node 36, line 4, and offset 8. The MMU sees that the memory word referenced is from node 36, not node 20, so it sends a request message through the interconnection network to the line’s home node, 36, asking whether its line 4 is cached, and if so, where.

1. Let us assume we have such a machine. The physical RAM has 32 bytes, and is evenly divided into 4 pages. The virtual memory has 16 pages.

The content in the page table and physical RAM is shown below:

Physical RAM

|  |  |  |
| --- | --- | --- |
|  | Page Num | Page  data |
| 3 | 11 | ABCDEFGH |
| 2 | 10 | IJKLMNOP |
| 1 | 01 | QRSTUVWX |
| 0 | 00 | YZAEIOU! |

Page table

|  |  |  |
| --- | --- | --- |
| 15 | 1 | 00 |
| 14 | 0 |  |
| 13 | 0 |  |
| 12 | 0 |  |
| 11 | 0 |  |
| 10 | 0 |  |
| 9 | 0 |  |
| 8 | 0 |  |
| 7 | 1 | 01 |
| 6 | 0 |  |
| 5 | 0 |  |
| 4 | 0 |  |
| 3 | 0 |  |
| 2 | 1 | 11 |
| 1 | 0 |  |
| 0 | 1 | 10 |

Let us assume in a page, the offset of the left most byte is 0, and the offset of the right most byte is 7. If we are going to access the following virtual memory addresses, what is the data we will see?

0010 100

0111 111

0010 000

0000 100

**Solution:** Given RAM size = 32 bytes

Number of pages divided into = 4 pages

So, the Page size = 32/4 = 8 bytes

Page offset = log2(page size in bytes)

= log28= 3 bits

1. For virtual memory address 0010 100

Page number page offset

So, page number = 0\*20+1\*2+0\*22+0\*23

= 2

From the given page table 2 🡺 11 🡺 ABCDEFGH (from the physical ram page data)

Offset = 0\*20+0\*21+1\*22

= 4 = 5th position from the page data = ‘E’

So, when we access the address 0010100, the data which we will see is E.

1. For virtual memory address 0111 111

Page number page offset

So, page number = 1\*20+1\*2+1\*22+0\*23

= 7

From the given page table 7 🡺 01 🡺 QRSTUVWX (from the physical ram page data)

Offset = 1\*20+1\*21+1\*22

= 7 🡺 8th position from the page data = ‘X’

So, when we access the address 0111111, the data which we will see is X.

1. For virtual memory address 0010 000

Page number page offset

So, page number = 0\*20+1\*2+0\*22+0\*23

= 2

From the given page table 2 🡺 11 🡺 ABCDEFGH (from the physical ram page data)

Offset = 0\*20+0\*21+0\*22

= 0 🡺 1st position from the page data = ‘A’

So, when we access the address 0010000, the data which we will see is A.

1. For virtual memory address 0000 100

Page number page offset

So, page number = 0\*20+0\*2+0\*22+0\*23

= 0

From the given page table 0 🡺 10 🡺 IJKLMNOP (from the physical ram page data)

Offset = 0\*20+0\*21+1\*22

= 4 🡺 5th position from the page data = ‘M’

So, when we access the address 0000100, the data which we will see is M.

1. What is a page fault? What happens to the physical RAM when there is a page fault?

**Answer :** A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system Random access Memory(RAM) .

After a page fault has occurred, the operating system must read in the required page from the disk, enter its new physical memory location in the page table, and then repeat the instruction that caused the fault. When handling a page fault, the operating system tries to make the required page accessible at the location in physical memory or terminates the program in cases of an illegal memory access.

1. We have 2 page-replacement policies, LRU and FIFO. What is the difference between them?

**Answer:** In LRU page replacement algorithm the page frame whose contents have not been used for the longest time is replaced. LRU keeps the things that were most recently used in memory.

The First-In, First-Out (FIFO) Page Replacement Algorithm is an algorithm in which the page frames are treated as a circular list and the oldest (longest resident) page is replaced. FIFO keeps the things that were most recently added.

1. What is internal fragmentation? How to alleviate the problem?

**Answer:** Internal fragmentation is defined as when memory is allocated to the programs more than needed and the difference between assigned and requested memory is the internal fragmentation.

The internal fragmentation can be alleviated by dynamic memory allocation and memory pools. Dynamic memory helps in allocating only the required amount of memory.

1. What is external fragmentation? How to alleviate the problem?

**Answer:** The external fragmentation arises when free memory is divided into blocks and is interspersed by allocated memory. Here, although we have total space available that is needed by a process still, we are not able to put that process in the memory because that space is not contiguous. This is called external fragmentation.

External fragmentation can be alleviated by increasing the size of the files. Allocating the processes non-contiguously can also reduce the external fragmentation.

1. Why do people implement I/O instructions on the OSM level instead of ISA level?

**Answer:** The OSM-level instruction set is the complete set of instructions available to application programmers. It contains nearly all of the ISA level instructions, as well as the set of new instructions that the operating system adds.

Input/output is one of the areas where the two levels differ considerably. The reasons for this difference are simple: security, confidentiality, integrity. Second, normal, sane programmers do not want to do I/0 at the ISA level themselves because doing so is extremely tedious and complex.

The OSM Level Files One way of organizing the virtual I/0 is to use an abstraction called a file. A file consists of a sequence of bytes written to an I/0 device. If the I/O device is a storage device, such as a disk, the file can be read back later. If the device is not a storage device (e.g., a printer), it cannot be read back.

1. What does a file index do? Why do we need a file index?

**Answer:** It is a computer file which contain index that can allow easy random access to any record given by its file key**.** The key should uniquely identifies the record. In computer file if more than one index is present the other one is called alternate index. The indexes are created with the file and maintained by the system.

**Use of file Index:** The use of file index is it helps every one to find required items and return them to the proper file, so that they can easily find what they are needed for next time. If you are doing work alone or only single person accessing your filing system, don’t assume spontaneously you don’t need a file index.

1. What do free list and bit map do?

**Answer:**

**Free List:** A free list is a data structure in which dynamic memory allocation scheme is used. It operates by connecting unallocated regions of memory together in a linked list, using the first word of each unallocated region as a pointer to the next.

**Bit Map:** The one situation in which a Boolean value is normally represented by 1 bit is when there is an entire array of them, so a 32-bit word can hold 32 Boolean values. Such a data structure is called a bit map and occurs in many contexts.

For example, a bit map can be used to keep track of free blocks on a disk. If the disk has n blocks, then the bit map has n bits.

Questions on Chapter 7:

1. What is an assembler? What is a compiler?

**Answer:**

**Assembler**: Assembler is a computer program in which simple and written in assembly language into machine language**.**

* The code and instructions in assembly language are executed by a computer .
* The assembler will allow software and application developers to access, operate and manage a computers hardware architecture and components.
* In some cases assembler is referred as compiler in assembly language and also provides services to an interpreter.

**Compiler:**

A compiler is a program which can translates a source program that can be written in high- level programming language such asJAVA into machine code for some computer architecture (such as the Intel Pentium architecture). The generated machine code can be later executed many times against different data each time.

1. What is an assembly language? What are the 4 parts of an assembly language statement?

**Answer:**

**Assembly Language**: In the computer programming language the assembler or assembly language in some cases called as ‘asm’ , In any low level programming language that can be strong correspondence in between the architecture machine instructions and instructions in the language. This is the reason the assembly language depends on the machine code instructions.

* Every assembly language is designed for exactly one specific computer architecture.
* The Assembly language can also be called as symbolic machine.

**The 4 parts of an assembly Language are:**

1. A label field: They are used to provide symbolic names for memory addresses
2. An operation (opcode) field: It contains either a symbolic abbreviation for the opcode if the statement is a symbolic representation for a machine instruction or a command to the assembler itself.
3. An operand field: It is used to specify the addresses and registers used as operands by the machine instruction.
4. Comments field: It provides a place for programmers to put helpful explanations of how the program works for the benefit of other programmers who may subsequently use or modify the program.
5. What are the two major reasons why people still learn assembly language?

**Answer:**

* compiler must either produce output used by an assembler or perform the assembly process itself. Thus, understanding assembly language is essential to understanding how compilers work.
* studying assembly language exposes the real machine to view. For students of computer architecture, writing some assembly code is the only way to get a feel for what a machine is really like at the architectural level.

1. What are the two major reasons why people still write programs using assembly language?

**Answer**:

* First, an expert assembly language programmer working extremely hard can sometimes produce code that is much smaller and much faster than a high-level language programmer can. For some applications, speed and size are critical. Many embedded applications, such as the code on a smart card or RFID card, device drivers, string manipulation libraries, BIOS routines, and the inner loops of performance-critical real-time applications fall in this category.
* some procedures need complete access to the hardware, something usually impossible in high-level languages. For example, the low-level interrupt and trap handlers in an operating system and the device controllers in many embedded real-time systems fall into this category.

1. What is a macro in assembly language? What are the differences between a macro call and a procedure call?

**Answer:** An assembly Language macro is a template whose format represents a pattern of 0 or more assembly language statements that might be common to multiple programs. For this purpose, a macro language is used to provide a syntax for defining macros.

**The difference between macro call and procedure call:**

|  |  |  |
| --- | --- | --- |
| S.NO | **Macro Call** | **Procedure Call** |
| 1. | During assembly the macro call is done. | During program execution the procedure call is done. |
| 2. | The body is inserted into the object program every place the call is made. | The body is not inserted into the object program every place the call is made. |
| 3. | The Macro call instruction is not inserted into the object program and there is no execution. | The procedure call instruction is inserted into the object program and there will be an execution. |
| 4. | Macro call doesn’t use return instruction. | Procedure call will use return instruction. |
| 5. | One per micro call will appear in the object program. | One copy of the body will appear in the object program in procedure call. |

1. Why does the assembly process need two passes?

**Answer:** The main reason why most assemblers use a 2- pass system is to address the problem of forwarding references to variables or subroutines that have not yet been encountered when parsing the source code which contain forward references.

Pass 1 of the assembler scans the source, determining the size and address of all data and instructions; then pass 2 scans the source again, outputting the binary object code. Some assemblers have been written to use a 1.5 pass scheme, whereby the source is only scanned once, but any forward references are simply assumed to be of the largest size necessary to hold any native machine data type. The unknown quantity is temporarily filled in as zero during pass 1 of the assembler, and the forward reference is added to a ‘fix-up list’. After pass 1, the ‘.5’ the pass goes through the fix-up list and patches the output machine code with the values of all resolved forward references. This can result in sub-optimal opcode construction but allows for a very fast assembly phase.

1. What is a linker? What does a linker do?

**Answer:**

**Linker**: In the computing the linker and link editor in the computer system program that can takes both one or more object files that can be generated through a compiler or assembler. We can combine them into a single executable file, library file or any other object file.

**Working of Linker:** The Linker can also called as a program in a system that can helps the link a object modules of program into a single object file. It can performs the process linking. Linkers can also called as Link editors. This is a process of maintaining and collecting a piece of code and data into a single file.

1. What is the advantage of dynamic linking? How does a DLL work in Windows system?

**Answer**: The advantages of Dynamic Linking are:

* When the functions in a DLL change, the applications that use them do not need to be recompiled or relinked as long as the function arguments, calling conventions, and return values do not change. In contrast, statically linked object code requires that the application be relinked when the functions change.
* Multiple processes that load the same DLL at the same base address share a single copy of the DLL in physical memory. Doing this saves system memory and reduces swapping.
* A DLL can provide after-market support. For example, a display driver DLL can be modified to support a display that was not available when the application was initially shipped.
* Programs written in different programming languages can call the same DLL function as long as the programs follow the same calling convention that the function uses. The calling convention (such as C, Pascal, or standard call) controls the order in which the calling function must push the arguments onto the stack, whether the function or the calling function is responsible for cleaning up the stack, and whether any arguments are passed in registers. For more information, see the documentation included with your compiler.

**Working of DLL in windows system:**

In DLL windows system all versions of the windows operating system will support dynamic linking and depend heavily on it. It uses a special file format called a Dynamic Link Library (DLL). The dynamic link library can contain both procedures & data or it contain both. They are commonly used to allow two or more processes to share library procedures or data. Many DLLs have extension .dll but other extensions are also in use, including .dry(it for driver libraries) and .fon (for front libraries). The most common form of a DLL is a library consisting of a collection of procedures that can be loaded into memory and accessed by multiple processes at the same time.