CSCI 540 Homework Assignment 4

Name: Rama Krishna Kamma

CWID: 50321021

These are short questions (5% for each).

Questions on Chapter 6:

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

Answer:

The benefit of virtual memory over overlays is that programs may be broken up into various portions that individually fit in the memory. It was up to the programmer to divide the program into overlays, choose where each overlay should be kept in secondary memory. Plan for the transit of overlays between main memory and secondary memory, as well as managing the complete overlay process without assistance from the computer.

Virtual memory is a technique for carrying without the programmer's knowledge, the overlay procedure is carried out automatically. It has the benefit of relieving the programmer of a great deal of tedious accounting.

2. **What device translates the virtual address to physical address?**

Answer:

The MMU is in charge of translating physical memory addresses from virtual memory locations. This address is divided into three parts by the MMU. Line 4, Node 36, and Offset 8 in decimal form make up the three parts. The MMU asks the line's home node 36 if its line 4 is cached and, if yes, where, using a request message sent via the interconnection network. It notices that node 36, not node 20, is where the memory word being addressed is originating from, as was previously assumed.

3. **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:

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

Physical RAM

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

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

Answer:

From the given that RAM is 32 bytes,
4 pages make up the number of separated pages.
The Page Size is therefore 32/4 = 8 Bytes.
Page offset is log2(page size in bytes) = $\log_2 8 = 3$ bits

(i). for virtual memory address 0010 100

page offset page number

Page number therefore equals

$= 0*2^0+1*2+0*2^2+0*2^3$

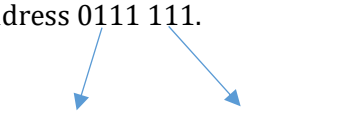
$= 2$

ABCDEFGH (from the physical ram page data) is taken from page table 2 -> 11 of the given information.

Offset = $0*2^0+0*2^1+1*2^2$
= 4 -> 5th position from the data on the page, which is 'E'

Therefore, when we visit address 0010100, we see the data E.

(ii) For virtual memory address 0111 111.



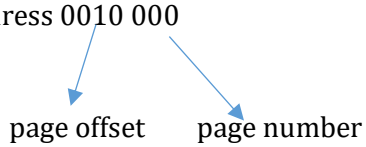
Page number = $1*2^0+1*2+1*2^2+0*2^3$
= 7

(From the actual ram page data) From the supplied page table 7 -> 01 -> QRSTUVWX

Offset = $1 * 2+1 * 2+1 * 2^2$
= 7 -> 8th position from the data on the page = 'X'

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

(iii) For virtual memory address 0010 000



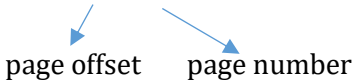
Page number = $0*2^0+1*2+0*2^2+0*2^3$
= 2

ABCDEFGH (from the physical ram page data) is taken from page table 2 -> 11 of the given information.

Offset = $0*2^0+0*2^1+0*2^2$
= 0 -> 1 from the data on the page = 'A'

Consequently, the information we will see when accessing the address 0010000 is A.

(iv) In relation to virtual memory location 0000 100



Page number = $0*2^0+0*2+0*2^2+0*2^3$
= 0

0 through 10 and IJKLMNOP (from the physical ram page data) are taken from the supplied page table.

Offset = $0*2^0+0*2^1+1*2^2$
= 4 -> 5 from the data on the page = 'M'

Consequently, the information we will see when accessing the address 0000100 is M.

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

Answer:

When a program tries to access information or code that is in its address space but is not now present in the system's random-access memory (RAM), a page fault occurs. Before updating the page table with the new physical memory address, an operating system must first read the necessary repeating the problematic instruction, page from the disk. The operating system attempts to make the required page accessible at the address when handling a page fault in physical memory or, in the event of an unauthorized memory access, terminates the program.

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

Answer:

The page frame whose contents have not been used for the longest time is replaced by a new one in the LRU page replacement process. The most recently used items are retained by LRU.
The oldest (longest resident) page is replaced First-In, First-Out (FIFO), a page replacement method, treats the page as frames as a circular list. FIFO only stores items that have just been added.

6. **What is internal fragmentation? How to alleviate the problem?**

Answer:

Internal fragmentation is defined as the difference between memory that has been allotted and requested but has not yet been allocated to applications.

Memory pools and dynamic memory allocation help reduce internal fragmentation. Allocating only the necessary amount of memory is made easier by dynamic memory.

7. **What is external fragmentation? How to alleviate the problem?**

Answer:

When allotted memory is interspersed between free memory blocks that have been fragmented into smaller chunks, external fragmentation results. Even though we still have all the space needed by a process accessible in this situation, we are unable to place the process in memory because the space is not contiguous. The term for this is external fragmentation.

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

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

Answer:

The entire collection of instructions available to application programmers is a collection of instructions at the OSM level. It also compiles a new guidelines that the operating system must follow introduces in addition to almost all of the ISA level instructions.

One area where the two levels differ significantly is in input/output. The differences are due to three factors: security, secrecy, and integrity. Second, because it is so time-consuming and difficult to implement I/0 at the ISA level, reasonable, most rational programmers don't want to handle it themselves.
OSM file storage level the usage of an abstraction called a file is one method of arranging the virtual I/0. A series of bytes written to an I/0 device makes up a file. When a disk or other storage device serves as the I/O device, the file can be read back later. It can't be repeated if the device (for example, a printer) is not a storage device.

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

Answer:

It is an electronic file with an index that enables quick random access to any record specified by the file key. The record must be uniquely identified by the key. If there are multiple indexes in a computer file, the additional one is known as an alternate index. The system creates and maintains the indexes together with the file.

Indexing of the file: The benefit of a file index is that it makes it easy for everyone to locate needed objects and put them back in the appropriate file where they can be quickly found again. Do not automatically assume that you do not need a file index if you are working alone or the only person who has access to your filing system.

10. What do free list and bit map do?

Answer:

Free List: A free list is a type of data structure that employs a dynamic method of memory allocation. It works by creating a linked list out of unallocated memory spaces and utilizing the first word of each as a pointer to the next.

Bit Map: A 32-bit word may store 32 Boolean values because the only circumstance in which a Boolean value is often represented by 1 bit is when there is an entire array of them. A bit map is a type of data structure that is used in numerous situations.
A bit map can be used, for instance, to maintain track of the free blocks on a disk. The bit map has n bits if the disk has n blocks.

Questions on Chapter 7:

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

Answer:

Assembler:
A computer application called Assembler converts straightforward assembly code into machine language.

- A computer executes the assembly language code and instructions.

- The assembler will give access to, control over, and management of a computer's hardware architecture and components to software and application developers.
- Assemblers sometimes serve as interpreters and are also known as compilers in assembly language.

Compiler:

A compiler is a program that converts source code from high-level programming languages, like Java, into machine code for specific computer architectures, like the Intel Pentium architecture. The produced machine code can then be run numerous times, each time against a different set of data.

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

Answer:

Assembly Language: In some circumstances, the term "asm" is used to refer to the computer programming language's assembler or assembly language. Any low-level programming language might have a strong connection between its instructions and the architecture's machine instructions. For this reason, assembly language relies on instructions from machine code. The Assembly language, often known as the symbolic machine, is created specifically for one particular computer architecture.

There are 4 components in an assembly language.
1. A label field: They are used to give memory addresses symbolic names.
2. An operation (opcode) field: If a machine instruction's symbolic representation is what the statement is, then this field either contains a symbolic abbreviation for the opcode or a directive to the assembler itself.
3. An operand field is used to define the addresses and registers that the machine instruction will utilize as operands.
4. The comments box gives programmers a spot to include useful descriptions of the operation of the software for the advantage of future programmers who might use or alter the application.

13. What are the two major reasons why people still learn assembly language?

Answer:

- Compilers must either perform the assembly process themselves or create output that is used by an assembler. Therefore, knowledge of assembly language is necessary in order to comprehend how compilers operate.
- Because learning assembly language makes the actual machine visible. The only way for computer architecture students to truly understand a machine at the architectural level is to write some assembly code.

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

Answer:

- To start, a skilled assembly language programmer who works incredibly hard can occasionally write code that is produced far more quickly and efficiently than by a high-level language coder. Speed and size are important factors for some applications. Numerous embedded applications come within this category, including the inner loops of real-time performance-critical applications, device drivers, string manipulation libraries, smart card or RFID card code, and BIOS routines.

- Some methods necessitate full hardware access, which is typically not achievable in high-level languages. For instance, this category includes device controllers in numerous embedded real-time systems as well as low-level interrupt and trap handlers in OS systems.

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

Answer:

A template representing a pattern of 0 or more assembly language statements that may be shared by several applications is known as an assembly language macro. A macro language is used to define macros in order to achieve this.

S.No	Macro Call	Procedure Call
1	During assembly the macro call is done.	During program execution the procedure call is done.
2	Each time a call is made, the body is placed into the object program.	Not every instance when the call is performed results in the body being added to the object program.
3	The Macro call instruction is not inserted into the object program and there is no execution.	The procedure call instruction is added to the object program, which will then execute it.
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	The body will only appear once in the method call of the object program.

16. Why does the assembly process need two passes?

Answer:

The fundamental purpose of a two-pass system, which is used by the majority of assemblers, is to solve the issue of forwarding references to variables or subroutines that are not yet present in the parsed source code that contains forward references.
The assembler performs a source scan in passes 1 and 2, identifying the size and addresses of all the data and instructions and producing the binary object code. Some assemblers have been built to use a 1.5 pass method, which scans the source just once but simply assumes that any forward references are the largest size required to hold any native machine data type. During the first pass of the assembler, the unknown quantity is momentarily filled in as 0, and the forward

reference is added to a "fix-up list." Following pass 1, the '.5' pass iterates through the fix-up list and updates the output machine code with the values of all resolved forward references. Although this may lead to less-than-ideal opcode construction, it permits a very quick assembly phase.

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

Answer:

Linker: A compiler or an assembler can output one or more object files, which the linker and link editor of the computer program can accept. They can be combined into a single executable file, library file, or object file of any other type.

Working of the Linker: The Linker is a system software that aids in linking program object modules into a single object file. It is capable of linking processes. Link editors and linkers are similar terms. This procedure involves maintaining and compiling data and code into a single file.

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

Answer:

The benefits of dynamic linking are as follows:

- Providing that the function arguments, calling practices, and return values remain constant, the applications that use the functions in a DLL do not require recompilation or relinking. The program must be relinked when the functions in statically linked object code change.
- The DLL is loaded by multiple processes at the same base address in physical memory, but only one copy of the DLL is shared by all of them. The system memory is conserved, and swapping is decreased.
- The after-market assistance offered by a DLL. For example, it might be possible to modify a display driver DLL to support a display that wasn't supported by the application when it was first distributed.
- If a program calls a DLL function using the same calling convention as the DLL function, it can be called from programs written in various programming languages. The manner in which the arguments are provided in registers, the order in which they must be pushed onto the stack, and whether or not the calling function is responsible for clearing the stack are all determined by the calling convention (such as C, Pascal, or standard call). Consult the documentation that came with your compiler for further details.

Working of DLL in Windows System:

All Windows operating system versions will support and largely rely on dynamic linking in the DLL Windows System. It utilizes a unique file type known as a Dynamic Link Library (DLL). The dynamic link library may or may not contain both procedures and data. They are frequently used to enable the sharing of library methods or data between two or more processes. There are many DLLs with the.dll extension, but there are also DLLs with them. There are two extensions in use: dry (for driver libraries) and.fon (for front libraries). A library comprising many the ability for many processes to simultaneously use memory-loaded methods most common sort of DLL.