Study Set for Lecture 09: Main Memory

Due Oct 26 at 11:59pm **Points** 10 **Questions** 18
Available Oct 20 at 12pm - Dec 8 at 8pm about 2 months **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect09_Main_Memory.pdf**.

Then answer the questions from this study set and submit them to gain access to the further part of the course.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
KEPT	Attempt 4	18 minutes	10 out of 10 *
LATEST	Attempt 4	18 minutes	10 out of 10 *
	Attempt 3	25 minutes	10 out of 10 *
	Attempt 2	4,342 minutes	10 out of 10
	Attempt 1	less than 1 minute	0 out of 10 *

* Some questions not yet graded

ⓘ Correct answers are hidden.

Score for this attempt: **10** out of 10 *
Submitted Dec 7 at 8:27pm
This attempt took 18 minutes.

Question 1 Not yet graded / 0 pts

When can the binding of instructions and data to memory addresses occur?

For each binding type, explain the details of the binding process and the implications.

Your Answer:

Binding of Instructions and Data to Memory:

Address (of both, instructions and data) from the program on the disk must be translated into an address in the memory

The binding can happen at three different stages :

● Compile time:
- if memory location is known a priori, then the compiler can generate absolute code
- code requires recompilation if the starting location changes
● Load time:
- if memory location is not known at compile time, code must be relocatable
-- binding is done when loading program
-- program must stay in memory
● Execution time:
- binding delayed until run time
-- the process must be able to be moved during its execution from one memory location to another.

Question 2

Not yet graded / 0 pts

What are the challenges of load time and execution time binding?

What are the mechanisms that support relocatable code?
Describe the process of loading code into any part of memory.

Your Answer:

Load time blinding: the challenges are that the program must remain in memory and that the code must relocatable. For execution time binding, the challenge is that the process must be portable from one memory segment to another

The mechanisms that supportable relocatable code are "base" and "limit" registers. The base gives the beginning of the space for the process. The limit is the scope of the base (base + limit). the code is located into memory within one of these scopes via one of the allocation approaches.

Question 3

Not yet graded / 0 pts

You have 10 programs that use a certain library. Explain the process of linking the programs with

- a static version of the library.
- a dynamic version of the library.

Your Answer:

Static

Becomes a part of the program after linking

The code within the library is cloned to every program that uses it

Dynamic

Small piece of code called the stub is used to locate the correct memory-resident library routine

The stub then switches itself with the address of the routine and then executes the routine

The OS then checks if the routine is in the process’ memory address

A static library becomes part of the program after linking. It is cloned to every program that uses it. Static libraries use a ".a" extension. For dynamic version, a stub is used to locate the appropriate memory resident library routine. The stub replaces itself with the address of the routine, and executes the routine

.

Static libraries, while reusable in multiple programs, are locked into a program at compile time. **Dynamic,** or **shared libraries** on the other hand, exist as separate files outside of the executable file.

The downside of using a static library is that it's code is locked into the final executable file and cannot be modified without a re-compile. In contrast, a dynamic library can be modified without a need to re-compile.

What does this mean in practical terms? Well, imagine you're a developer who has issued an application to thousands of users. When you want to make a few updates to the app, would you rather have to re-issue the entire program, or would you rather just issue updates in the form of modified libraries? The answer depends on the downsides your application can afford. If you have a lot of files, multiple copies of a static library means an increase in the executable file's size. If, however, the benefits of execution time outweigh the need to save space, the static library is the way to go.

Question 4

Not yet graded / 0 pts

Assume that the hit ratio in the MMU that performs paging with an associative memory (translation look-aside buffer (TLB)) is 90%. The access to main memory is 50 nanoseconds, and the lookup time for the TLB is 10 nanoseconds.

Compute what is the effective access time of this system.
Show the result and explain how did you obtain it.

Your Answer:

EAT = Effective Access Time

TLB = Translation Look-aside Buffer

EAT = memory access time + (lookup time * hit ratio) + (memory access time + lookup time) (1- hit ratio)

EAT = (50 nanoseconds) + (10 nanoseconds*.9) + (50 nanoseconds + 10 nanoseconds) (1-.9)

EAT = (50 nanoseconds) + (9 nanoseconds) + (60 nanoseconds) (.1)

EAT = (59 nanoseconds) + (6 nanoseconds)

EAT = 65 nanoseconds

ALSO

Effective memory access time(EMAT) : TLB is used to reduce effective memory access time as it is a high speed associative cache.
EMAT = h*(c+m) + (1-h)*(c+2m)
where, h = hit ratio of TLB
m = Memory access time
c = TLB access time

h = 0.9;

m = 50 ns;

c = 10;

EMAT = 0.9 * (10 + 50) + (1 - 0.9) * (10 + 2(50)) = 65 ns

Question 5

Not yet graded / 0 pts

You have a system with 1 MB RAM. The system uses 4 kB pages. Your system loads 5 programs with the following lengths:

```
167852 bytes
209376 bytes
32866 bytes
254871 bytes
128527 bytes
```

Calculate the scope of internal fragmentation after all programs are loaded into the memory.
Show the result and explain how did you compute it.

Your Answer:

Page size is 4 * 1024 = 4096 bytes/page

167852/4096 = 41 frames with 0.1 left
209376/4096 = 52 frames with 0.9 left
32866/4096 = 9 frames with 0.98 left
254871/4096 = 63 frames with 0.8 left
128527/4096 = 32 frames with 0.6 left

Total scope of fragmentation is (0.1+0.9+0.98+0.8+0.6) (4096) = 3.38*4096 = 13844.48

ALSO

1MB = 1000000 bytes

4KB = 4000 bytes

frameNum = RAM / frameSize -> framNum = 250, each is 4KB frames = pi / 4000

wastedSpace = pi - 4000(frames - 1)

fragmenta!on = ∑ wastedSpacei

Programs = (p1, p2, ..., p5)

p1 = 167852 bytes, 42 frames, wasted space 3852 bytes

p2 = 209376 bytes, 53 frames, wasted space 1376 bytes

p3 = 32866 bytes, 9 frames, wasted space 866 bytes p4 = 254871 bytes, 64 frames, wasted space 2871 bytes

p5 = 128527 bytes, 33 frames, wasted space 527 bytes

fragmenta!on = 3852 + 1376 +866 + 2871 + 527 = 9492 bytes

Question 6

Not yet graded / 0 pts

Consider the following requests (Rn) for memory allocation (A) and deallocation (D) from a memory pool of size 20:

```
R1 A 6
R2 A 3
R3 A 5
R4 A 2
R1 D
R5 A 4
R4 D
R6 A 1
R7 A 2
```

Compute explicitly (i.e., show all steps) the external fragmentation assuming the **best-fit** allocation policy.

Use the following notation:

```
[xxxxxxxxxxxxxxxxxxx]
[111111xxxxxxxxxxxxx]
[111111222xxxxxxxxxx]
```

which shows the initial state and then the two following allocations.

Your Answer:

Best-Fit

Allocatte the smallest hole that is big enough; must search entire list, unless ordered by size

- Produces the smallest leftover hole

[xxxxxxxxxxxxxxxxxxx]
[111111xxxxxxxxxxxxx]

[111111222xxxxxxxxxxx]
[11111122233333xxxxxx]
[1111112223333344xxxx]
[xxxxxx2223333344xxxx]
[xxxxxx22233333445555]
[xxxxxx22233333xx5555]
[xxxxxx222333336x5555]
[77xxxx222333336x5555]

Question 7

Not yet graded / 0 pts

Consider the following requests (Rn) for memory allocation (A) and deallocation (D) from a memory pool of size 20:

```
R1 A 6
R2 A 3
R3 A 5
R4 A 2
R1 D
R5 A 4
R4 D
R6 A 1
R7 A 2
```

Compute explicitly (i.e., show all steps) the external fragmentation assuming the **worst-fit** allocation policy.

Use the following notation:

```
[xxxxxxxxxxxxxxxxxxxx]
[111111xxxxxxxxxxxxxx]
[111111222xxxxxxxxxx]
```

which shows the initial state and then two following allocations.

Your Answer:

Worst Fit

Allocate the largest hole; must also search entire list

- produces the largest leftover hole

[xxxxxxxxxxxxxxxxxxxx]
[111111xxxxxxxxxxxxxx]
[111111222xxxxxxxxxxx]
[11111122233333xxxxxx]
[1111112223333344xxxx]
[xxxxxx2223333344xxxx]
[5555xx2223333344xxxx]
[5555xx22233333xxxxxx]
[5555xx222333336xxxxx]
[5555xx22233333677xxx]

Question 8

Not yet graded / 0 pts

Consider the following requests (Rn) for memory allocation (A) and deallocation (D) from a memory pool of size 20:

```
R1 A 6
R2 A 3
R3 A 5
R4 A 2
R1 D
R5 A 4
R4 D
R6 A 1
R7 A 2
```

Compute explicitly (i.e., show all steps) the external fragmentation assuming the **first-fit** allocation policy.

Use the following notation:

```
[xxxxxxxxxxxxxxxxxxxx]
[111111xxxxxxxxxxxxxx]
[111111222xxxxxxxxxxx]
```

which shows the initial state and then the two following allocations.

Your Answer:

First fit

Allocate the first hole that is big enough

[xxxxxxxxxxxxxxxxxxxx]

[111111xxxxxxxxxxxxxx]

[111111222xxxxxxxxxxx]

[11111122233333xxxxxx]

[1111112223333344xxxx]

[xxxxxx2223333344xxxx]

[5555xx2223333344xxxx]

[5555xx22233333xxxxxx]

[55556x22233333xxxxxx]

[55556x2223333377xxxx]

Question 9

Not yet graded / 0 pts

What is the difference between a physical address and a logical address? How is the logical address mapped to its physical counterpart? What is the lifetime of such binding?

Your Answer:

1. The basic difference between Logical and physical address is that Logical address is generated by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.
2. Logical Address Space is the set of all logical addresses generated by CPU for a program whereas the set of all physical address mapped to corresponding logical addresses is called Physical Address Space.
3. The logical address does not exist physically in the memory whereas physical address is a location in the memory that can be accessed physically.
4. Identical logical addresses are generated by Compile-time and Load time address binding methods whereas they differs from each other in

run-time address binding method. Please refer **this (https://www.geeksforgeeks.org/memory-management-mapping-virtual-address-physical-addresses/)** for details.

5. The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).

The logical address is mapped to its physical counterpart by the memory management unit (MMU), which is a hardware device that maps logical to a physical address. The binding lives as long as the physical address still holds the corresponding data.

Question 10

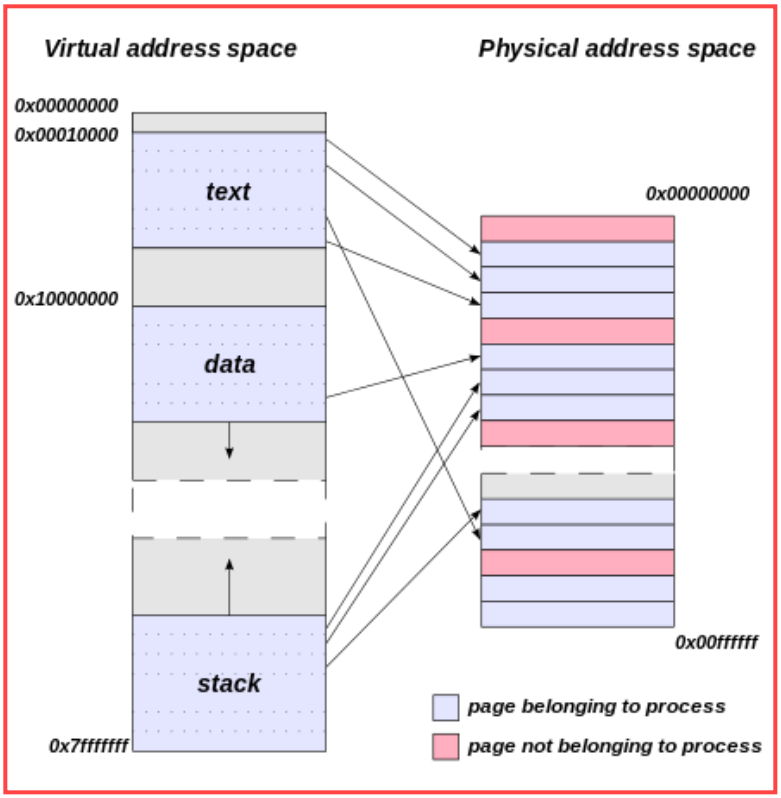
Not yet graded / 0 pts

Explain how paging alleviates the problems inherent to contiguous allocation of memory to programs.
What is a page table?
What is the difference between a page and a frame?
What is a free frame list?

Your Answer:

It reduces fragmentaion from copying whole code of process, paging allows for the program to be kept in logical memory. The logical "page" is the same size as the physical "frame". When needed, the process is allocated an available frame.

A page table is a table used to translate a logical address to a physical address in a set of frames.



A page is a block of logical memory, where a frame is a block of physical memory.

A free frame list is a list of frame positions that aren't allocated to a process and are free to be allocated to by a new program.

Question 11

Not yet graded / 0 pts

What would be the maximum size of a page table for a system with 48-bit addressing assuming a 1 kb frame size and assuming that 4 bytes are needed for every entry.

Describe each of the strategies to keep the size of the page table in check?

Your Answer:

2^48 = number of addresses

2^10 = frame size

(2^48) / (2^10) = 2^38 pages

2^4*2^38 = 2^42bytes per page = 2^12GB?

2^38 pages * 2^2 bytes / page = 2^40 bytes?

Hierarchal Pages

Slide 23

Break up the logical address space into multiple page tables

In other words, divide the page table into its own pages

Only some of the pages in the page table need to be in memory, which means it takes up less space

Hashed Page Tables

Slide 26

Common in 32-bit and up systems

Virtual page number is hashed into a page table

This page table contains a chain of elements hashing to the same location

Virtual page numbers are compared with the elements in order to find a match

If a match is found, the physical frame that belongs to the page number is extracted

Inverted Page Tables

Slide 27

One table for all processes

One entry for each real page of memory

Each entry consists of the virtual address of the page stored in that real memory location(p) as well as the info about who owns the page (pid)

The base is the offset of the entry in the entry table

Decreases space but increases the time it takes to search the table when a page is referenced

Question 12

Not yet graded / 0 pts

Consider a computer system that uses 5-bit addressing scheme with 3 bits for page numbers and two bits for offsets. Let the logical memory of some process be a contiguous sequence of letters from `a` to `z` (26 letters, only lower case).

Show the binary version of the logical address of letter `m` assuming that the following is the content of the page table: `[4, 1, 0, 5, 2, 7, 6, 3]`?

What is the physical address of the letter `m`? Show both the binary and the decimal versions.

Show the result and explain how did you derive it.

Your Answer:

Number of pages = 2^3 = 8

Number of addresses/page = 2^2 = 4

m is the 13th letter in the alphabet and we have 4 addresses per page

Therefore, m is located at the 3rd page (4th entry in the page table) because 13/4 = 3

The binary address is 10100

101 = 5, which is the virtual address space

00 = 0, which is where in the virtual address space m is

Physical address = (page number * page size) + offset = (5 * 4) + 0 =20

OR

we know that physical/logical address comprises of

page no bits	offset bits

since the logical address page number is of 3 bit. so **Page Table** assuming physical address page number is represented by 4-bit, will look like

Logical Address (page no)	**Physical address (page no)**
000	**4 (0100)**
001	1 (0001)
010	0 (0000)
011	5 (0101)
100	2 (0010)
101	7 (0111)
110	9 (1001)
111	11 (1011)

given the offset is of 2 bit which means each of the page contain $2^2 = 4$ letters.

since '**m'** is the 13^{th} alphabet of english and memory stored is contiguous.

so ceil(13/4) = 4 hence **m** will be present in 4^{th} page. i.e. p = 011

13%4 =1 , so '**m'** will be the first value inside 4^{th} page, i.e. offset = 00

logical Adress of m is given by: 01100

for physical adress ;

by page table, fourth entry 011 → 5 and offset = 00

so 6 bit physical address of '**m'** = physical page + number offset

= 0101 00

In decimal physical address = 20

Question 13

Not yet graded / 0 pts

The CPU executing a process generates `(5,0,4)` as a logical address in a system that uses an inverted page table to compute physical addresses. Let the following be the inverted page table: `[(2,1),(5,2),(7,3),(5,0),(6,4),...]`.

Assuming that there are 16 frames in the memory and page size is 8 construct the binary version of the physical address corresponding to this logical address.

Show the result and explain how did you derive it.

Your Answer:

Page number = 3, (5,0) is at page index 3

Page size = 8

4 = 4th spot in the page

24+4 = 28

1 1 1 0 0

index that matches is 3. So, 3 * 8 + 4 = 28.

Question 14

Not yet graded / 0 pts

Let the logical memory of some process be a contiguous sequence of letters from `a` to `z` (26 letters, only lower case).

Consider a computer system that uses 8-bit addressing scheme with six bits for two-level page indexing with the first level taking four bits, the second taking two, and the offsets another two.

Assuming that the following is the content of the outer page table:

```
[7, -, 0, 5, -, -, -, 2, -, 9, -, 11, 13, -, 1, -]
```

and that the content of the third (counting from zero) frame is:

```
[8, 12, 10, 4]
```

and the content of the sixth (again, counting from zero) frame is:

```
[3, 15, 6, 14]
```

show the binary version of the logical address of letter c knowing that the physical address of c is 18. Show the result and explain how did you derive it.

HINT: Draw a diagram showing the address, the two page tables, and the memory frames. Then, follow the links backwards starting with the physical location of the letter c.

Your Answer:

If the physical address of c is 18, then the physical address of a is 16

The page number of a-d is 16 (physical address) / 4 (number of bits/page) = 4

In binary, 18 is 0 0 0 1 0 0 1 0

The last two digits are transferred to the last two digits of the logical address

4 is found at index 3 of the third frame, which forms the 3rd and 4th digits of the logical address

2 is the index of the third frame, which is found at index 7 of the outer page table

The 7 is what makes up the remaining digits of the logical address, giving us an address of 0 1 1 1 1 1 1 0

Question 15

Not yet graded / 0 pts

How does a segmentation memory architecture differ from paging?

How can they both be integrated in a hybrid architecture?
Describe such a hybrid architecture with details on how logical addresses would be translated into their physical equivalents.

Your Answer:

The segmentation memory architecture makes a program a collection of segments, which is a logical unit such as a main program, procedure, function, method, object, local/global variable, stack, ect.

While paging puts the program into chunks of physical memory which can be found using a page and offset put into a page table for translating.

The CPU generates the logical address given to the segmentation unit, which produces linear addresses. The linear address is given to the paging unit which generates physical address in main memory. The paging units form the equivalent of the MMU.

Question 16

Not yet graded / 0 pts

Describe the buddy memory allocation scheme.

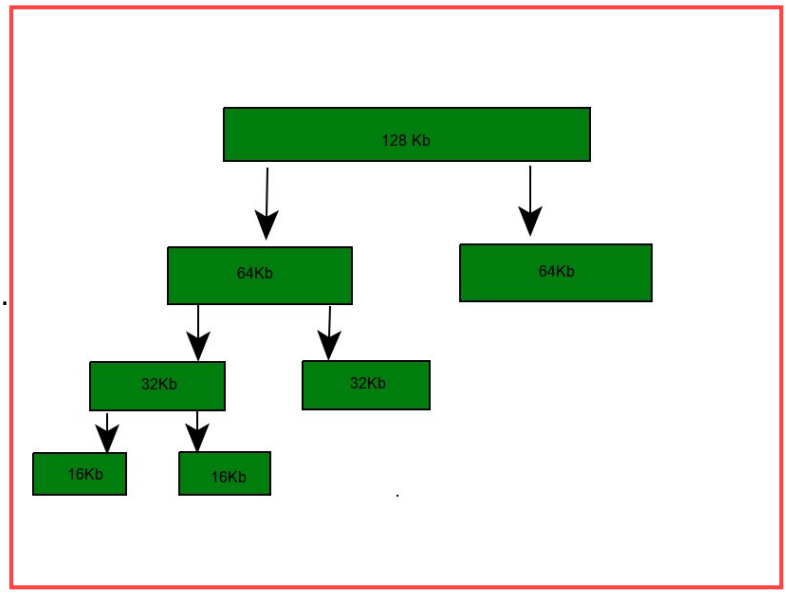
What are the benefits of using the buddy scheme for allocating memory for kernel needs?

Your Answer:

1. Buddy system –

Buddy allocation system is an algorithm in which a larger memory block is divided into small parts to satisfy the request. This algorithm is used to give best fit. The two smaller parts of block are of equal size and called as buddies. In the same manner one of the two buddies will further divide into smaller parts until the request is fulfilled. Benefit of this technique is that the two buddies can combine to form the block of larger size according to the memory request.

Example – If the request of 25Kb is made then block of size 32Kb is allocated.



1. Binary buddy system
2. Fibonacci buddy system
3. Weighted buddy system
4. Tertiary buddy system

Why buddy system?
If the partition size and procees size are different then poor match occurs and may use space inefficiently.
It is easy to implement and efficient then dynamic allocation.

Drawback –
The main drawback in buddy system is internal fragmentation as larger block of memory is acquired then required. For example if a 36 kb request is made then it can only be satisfied by 64 kb segment and reamining memory is wasted.

Question 17

Not yet graded / 0 pts

Describe the slab memory allocation scheme.

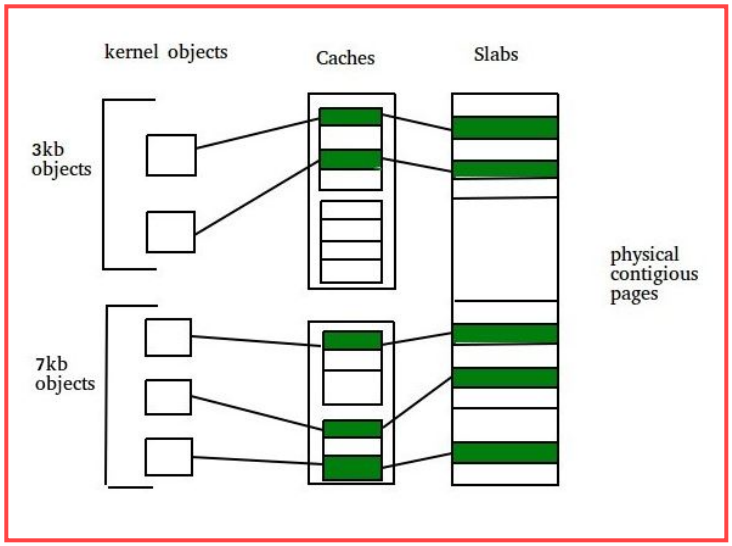
What are the benefits of using the slab scheme for allocating memory for kernel needs?

Your Answer:

2. Slab Allocation –

A second strategy for allocating kernel memory is known as slab allocation. It eliminates fragmentation caused by allocations and deallocations. This method is used to retain allocated memory that contains a data object of a certain type for reuse upon subsequent allocations of objects of the same type. In slab allocation memory chunks suitable to fit data objects of certain type or size are preallocated. Cache does not free the space immediately after use although it keeps track of data which are required frequently so that whenever request is made the data will reach very fast. Two terms required are:

- **Slab –** A slab is made up of one or more physically contiguous pages. The slab is the actual container of data associated with objects of the specific kind of the containing cache.
- **Cache –** Cache represents a small amount of very fast memory. A cache consists of one or more slabs. There is a single cache for each unique kernel data structure.



Each cache is populated with objects that are instantiations of the kernel data structure the cache represents. For example the cache representing semaphores stores instances of semaphore objects, the cache representing process descriptors stores instances of process descriptor objects.

Benefits of slab allocator –

- No memory is wasted due to fragmentation because each unique kernel data structure has an associated cache.
- Memory request can be satisfied quickly.

- The slab allocating scheme is particularly effective for managing when objects are frequently allocated or deallocated. The act of allocating and releasing memory can be a time consuming process. However, objects are created in advance and thus can be quickly allocated from the cache. When the kernel has finished with an object and releases it, it is marked as free and return to its cache, thus making it immediately available for subsequent request from the kernel.

Question 18

10 / 10 pts

I have submitted answers to all questions in this study set.

- (●) True
- ( ) False

Quiz Score: **10** out of 10