Suppose that an application program contains a variable `a`, of type `char *`, which is a pointer to an array of characters. The program can then refer to the ith element of the array as `a[i]`. Each character occupies one byte, and C arrays are contiguous in the application's virtual memory.

Suppose that the system uses 32-bit virtual and physical addresses and paged virtual memory, with a page size of 4KB (2^{12} bytes). The *valid* entries in the process's page table are shown in the following chart. Assume that the entries for any pages not listed in the chart are invalid.

Page #	**Frame #**
0x00010	0x00032
0x00011	0x00033
0x00012	0x00010
0x00040	0x00021
0x00041	0x00022

The following table lists some possible values for the variables `a` and `i`. In each row, indicate what the *physical* address of `a[i]` will be, assuming the values of `a` and `i` indicated in that row, and the page table described above. If the virtual address of `a[i]` cannot be translated, write `"exception"`.

`a`	`i`	**physical address of `pa[i]`**
0x000100F0	0x100	*0x000321F0*
0x00012A00	0x12	*0x00010A12*
0x0001305D	0x2	*exception*
0x00040EF0	0x110	*0x00022000*
0x00041F00	0x100	*exception*

You are designing a paged virtual memory system on a system with 32-bit virtual addresses and 48-bit physical addresses. Each page is 4KB (2^{12} bytes), and each page table entry is 64 bits (8 bytes, 2^3 bytes).

a. (2 marks)

In a single-level paged system, how many bits of a virtual memory address would refer to the page number and how many to the offset? How many bits of a physical address would refer to the frame number and how many to the offset?

20, 12, 36, 12

b. (2 marks)

How many bytes would a single-level page table require?

2^{23}

c. (2 marks)

If a page table entry contains a frame number, a valid bit, a writeable bit, and a single bit for tracking page useage, how many bits per page table entry are unused?

$64 - 36 - 3 = 25$

d. (2 marks)

How many page table entries fit onto a single page?

$2^9 = 512$

(b) (15 marks) Given the following values for limit and relocation registers, fill in the table. Indicate **exception** in the physical address column if an exception would occur.

Segment 0 relocation: 0x7800 0000, limit: 0x1000

Segment 1 relocation: 0x1000 0000 0000 0000, limit: 0x4000 0000

Segment 2 relocation: 0x0, limit: 0x500

Segment 3 relocation: 0x1000, limit: 0x1500

Virtual Address	Segment	Offset	Physical Address
0xF00D F00D F00D F00D			
0xF000 0000 0000 000D			
0x1000			
0x1501			
0xA000 0000 0000 ACE0			

ONE point each entry.

0xF00D F00D F00D F00D - 3 - 0x300D FOOD FOOD FOOD - exception

0xF000 0000 0000 000D - 3 - 0x3000 0000 0000 000D - exception

0x1000 - 0 - 0x1000 - exception

0x1501 - 0 - 0x1501 - exception

0xA000 0000 0000 ACE0 - 2 - 0x2000 0000 0000 ACE0 - exception

6. (8 marks)

A system uses segmented address space for its implementation of virtual memory. Suppose a process initially uses 48KB of memory for its heap. The process then runs low on heap space and requests 16 KB more space. Assume you have access to a procedure sbrk and that sbrk(16*1024) will request that the heap's space be increase by 16 KB by finding a new location in RAM for the heap segment. If successful it returns the new address, otherwise it return NULL.

In roughly 4–6 steps, describe the process that would be required to increase the process's address space.

- Check if there is enough space to accommodate the new, larger heap (1 mark) or return ENOMEM (1 mark).
- Allocate the new heap (1 mark)
- Copy the contents of the old heap to the new one. (2 marks)
- Update the proc's relocation and limit values for the heap in the kernel (1 mark) and on the MMU (1 mark).
- Delete the old heap (1 mark)

À system uses 24-bit virtual addresses, 24-bit physical addresses and memory segmentation. There are four segments and two bits are used for the segment number. The relocation and limit for each of the segments are as follows.

Segment Number	Limit Register	Relocation Register
0x0	0x2 0000	0x40 0000
0x1	0xC 0000	0x70 0000
0x2	0x9 0000	0x50 0000
0x3	0xA 0000	0x60 0000

Translate the following addresses from virtual to physical. Clearly indicate what segment each address belongs to.

a. (2 marks) `0x01 D0D4`

Segment Number:

Address Translation:

segment 0: 0x41 D0D4

b. (2 marks) `0x22 CA10`

Segment Number:

Address Translation:

segment 0: segmentation violation

c. (2 marks) `0x33 47B8`

Segment Number:

Address Translation:

segment 0: segmentation violation

d. (2 marks) `0x1C F008`

Segment Number:

Address Translation:

segment 0: segmentation violation

Consider a physical memory with 3-page frames; using each replacement policy, determine for which memory reference a page fault can happen.

For LRU:

Frame	Reference																
	1	2	1	3	2	1	4	3	1	1	2	4	1	5	6	2	1
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2
1		2	2	2	2	2	2	3	3	3	3	4	4	4	6	6	6
2				3	3	3	4	4	4	4	2	2	2	5	5	5	1
Fault?	Y	Y	N	Y	N	N	Y	Y	N	N	Y	Y	N	Y	Y	Y	Y

The total number of faults is 11.

For clock:

Frame	Reference																
	1	2	1	3	2	1	4	3	1	1	2	4	1	5	6	2	1
0	1	1	1	1	1	1	4	4	4	4	4	4	4	5	5	5	5
1		2	2	2	2	2	2	2	1	1	1	1	1	1	6	6	6
2				3	3	3	3	3	3	3	2	2	2	2	2	2	1
Fault?	Y	Y	N	Y	N	N	Y	N	Y	N	Y	N	N	Y	Y	N	Y

The total number of faults is 9.

What is the maximum file size supported by a file system with 16 direct blocks, single, double, and triple indirection? The block size is 512 bytes. Disk block numbers can be stored in 4 bytes.

block size = 512

number of block numbers in an indirection block

= block size / 4

= 128

number of blocks for file data in that file object

= 16 + 128 + 128^2 + 128^3

Maximum file size:

(direct + single indirect + double indirect + triple indirect) * (blocksize)

= (16 + 128+ (128)^2 + (128)^3) * (512)

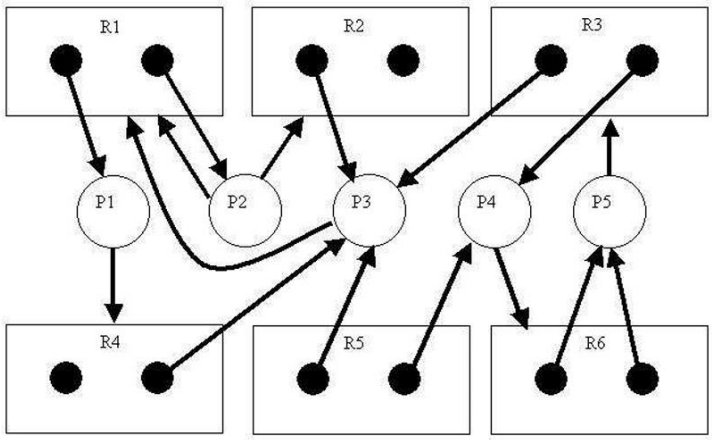
= 1 082 204 160 bytes

Consider the following information about resources in a system:

- There are six classes of allocatable resource labelled R1 through R6
- there are five processes labelled P1 through P5
- there are two instances of each resource
- there are some resource instances already allocated to processes, as follows:
  - one instance of R1 held by P1, another held by P2
  - one instance of R2 held by P3
  - one instance of R3 held by P3, another held by P4
  - one instance of R4 held by P3
  - one instance of R5 held by P3, another held by P4
  - two instances of R6 held by P5
- some processes have requested additional resources, as follows:
  - P1 wants one instance of R4
  - P2 wants one instance of R1 and one instance of R2
  - P3 wants one instance of R1
  - P4 wants one instance of R6
  - P5 wants one instance of R3

a. (5 marks)

Draw the resource allocation graph for this system. Use the style of diagram from the lecture notes.



b. (2 marks)

What is the state (runnable, waiting) of each process? For each process that is waiting, indicate what it is waiting for.

- *P1 is runnable and can acquire R4*
- *P2 is waiting for R1 and can acquire R2*
- *P3 is waiting for R1*
- *P4 is waiting for R6*
- *P5 is waiting for R3*

c. (4 marks)

Is this system deadlocked? If so, state which processes are involved. If not, give an execution sequence that eventually ends, showing resource acquisition and release at each step.

There is no deadlock. A possible sequence:

- *P1 acquires R4, runs to completion and releases R1 and R4*
- *P2 acquires R1 and R2, runs to completion and releases R1 (two instances) and R2*
- *P3 acquires R1, runs to completion and releases R1, R2, R3, R4 and R5*
- *P5 acquires R3, runs to completion and releases R3 and R6 (two instances)*
- *P4 acquires R6, runs to completion and releases R3, R5 and R6*

Suppose a computer has 10,000 bytes of physical memory with a page size of 1000 bytes. The operating system has just begun a program which uses 20,000 bytes of virtual memory. None of the program is currently in memory, and all frames are empty. Assuming the following page reference sequence, which pages are in which frames at the end of the sequence? (Show your working).

0, 1, 2, 3, 4, 1, 0, 5, 9, 11, 15, 9, 10, 15, 16, 2, 3, 11, 10, 6, 8, 17, 19, 6, 3