Started on	Thursday, 18 March 2021, 2:46:01 PM
State	Finished
Completed on	Thursday, 18 March 2021, 3:51:01 PM
Time taken	1 hour 5 mins
Grade	**15.79** out of 20.00 (**79**%)

Question **1**
Correct
Mark 1.00 out of 1.00

Select the correct statements about sched() and scheduler() in xv6 code

- [x] a. sched() and scheduler() are co-routines ✓
- [x] b. scheduler() switches to the selected process's context ✓
- [x] c. After call to swtch() in scheduler(), the control moves to code in sched() ✓
- [x] d. When either sched() or scheduler() is called, it results in a context switch ✓
- [x] e. After call to swtch() in sched(), the control moves to code in scheduler() ✓
- [x] f. Each call to sched() or scheduler() involves change of one stack inside swtch() ✓
- [x] g. When either sched() or scheduler() is called, it does not return immediately to caller ✓
- [x] h. sched() switches to the scheduler's context ✓

Your answer is correct.

The correct answers are: sched() and scheduler() are co-routines, When either sched() or scheduler() is called, it does not return immediately to caller, When either sched() or scheduler() is called, it results in a context switch, sched() switches to the scheduler's context, scheduler() switches to the selected process's context, After call to swtch() in scheduler(), the control moves to code in sched(), After call to swtch() in sched(), the control moves to code in scheduler(), Each call to sched() or scheduler() involves change of one stack inside swtch()

Question **2**

Partially correct

Mark 0.19 out of 0.25

Select the compiler's view of the process's address space, for each of the following MMU schemes:
(Assume that each scheme,e.g. paging/segmentation/etc is effectively utilised)

Relocation + Limit	one continuous chunk	✓
Segmentation	many continuous chunks of variable size	✓
Segmentation, then paging	Many continuous chunks each of page size	✗
Paging	one continuous chunk	✓

Your answer is partially correct.

You have correctly selected 3.
The correct answer is: Relocation + Limit → one continuous chunk, Segmentation → many continuous chunks of variable size, Segmentation, then paging → many continuous chunks of variable size, Paging → one continuous chunk

Question **3**

Correct

Mark 0.50 out of 0.50

Suppose a processor supports base(relocation register) + limit scheme of MMU.

Assuming this, mark the statements as True/False

True	False		
○ ✗	● ✓	The compiler generates machine code assuming appropriately sized semgments for code, data and stack.	✓
● ✓	○ ✗	The OS may terminate the process while handling the interrupt of memory violation	✓
○ ✗	● ✓	The hardware may terminate the process while handling the interrupt of memory violation	✓
● ✓	○ ✗	The compiler generates machine code assuming continuous memory address space for process, and calculating appropriate sizes for code, and data;	✓
○ ✗	● ✓	The process sets up it's own relocation and limit registers when the process is scheduled	✓
● ✓	○ ✗	The hardware detects any memory access beyond the limit value and raises an interrupt	✓
○ ✗	● ✓	The OS detects any memory access beyond the limit value and raises an interrupt	✓
● ✓	○ ✗	The OS sets up the relocation and limit registers when the process is scheduled	✓

The compiler generates machine code assuming appropriately sized semgments for code, data and stack.: False
The OS may terminate the process while handling the interrupt of memory violation: True
The hardware may terminate the process while handling the interrupt of memory violation: False
The compiler generates machine code assuming continuous memory address space for process, and calculating appropriate sizes for code, and data;: True
The process sets up it's own relocation and limit registers when the process is scheduled: False
The hardware detects any memory access beyond the limit value and raises an interrupt: True
The OS detects any memory access beyond the limit value and raises an interrupt: False
The OS sets up the relocation and limit registers when the process is scheduled: True

Question **4**

Correct

Mark 0.50 out of 0.50

Assuming a 8- KB page size, what is the page numbers for the address 1115565 reference in decimal :

(give answer also in decimal)

Answer: 136 ✓

The correct answer is: 136

Question **5**

Correct

Mark 0.50 out of 0.50

Map the functionality/use with function/variable in xv6 code.

Setup kernel part of a page table, mapping kernel code, data, read-only data, I/O space, devices	setupkvm() ✓
Return address of page table entry in a given page directory, for a given virtual address; creates page table if necessary	walkpgdir() ✓
return a free page, if available; 0, otherwise	kalloc() ✓
Array listing the kernel memory mappings, to be used by setupkvm()	kmap[] ✓
Create page table entries for a given range of virtual and physical addresses; including page directory entries if needed	mappages() ✓
Setup kernel part of a page table, and switch to that page table	kvmalloc() ✓

Your answer is correct.

The correct answer is: Setup kernel part of a page table, mapping kernel code, data, read-only data, I/O space, devices → setupkvm(), Return address of page table entry in a given page directory, for a given virtual address; creates page table if necessary → walkpgdir(), return a free page, if available; 0, otherwise → kalloc(), Array listing the kernel memory mappings, to be used by setupkvm() → kmap[], Create page table entries for a given range of virtual and physical addresses; including page directory entries if needed → mappages(), Setup kernel part of a page table, and switch to that page table → kvmalloc()

Question **6**

Correct

Mark 1.50 out of 1.50

Consider the reference string
6 4 2 0 1 2 6 9 2 0 5

If the number of page frames is 3, then total number of page faults (including initial), using FIFO replacement is:

Answer: 10 ✓

#6# 6,4# 6,4,2 #0,4,2# 0,1,2 #0,1,6 #9,1,6# 9,2,6# 9,2,0 #5,2,0

The correct answer is: 10

Question **7**

Correct

Mark 0.50 out of 0.50

In xv6, The struct context is given as

```
struct context {
  uint edi;
  uint esi;
  uint ebx;
  uint ebp;
  uint eip;
};
```

Select all the reasons that explain why only these 5 registers are included in the struct context.

- ☐ a. xv6 tries to minimize the size of context to save memory space
- ☐ b. esp is not saved in context, because it's not part of the context
- [✓ c. The segment registers are same across all contexts, hence they need not be saved] ✓
- [✓ d. eax, ecx, edx are caller save, hence no need to save] ✓
- [✓ e. esp is not saved in context, because context{} is on stack and it's address is always argument to swtch()] ✓

Your answer is correct.

The correct answers are: The segment registers are same across all contexts, hence they need not be saved, eax, ecx, edx are caller save, hence no need to save, esp is not saved in context, because context{} is on stack and it's address is always argument to swtch()

Question **8**

Correct

Mark 0.50 out of 0.50

Suppose the memory access time is 190ns and TLB hit ratio is 0.7, then effective memory access time is (in nanoseconds);

Answer: 247 ✓

The correct answer is: 247.00

Question **9**

Correct

Mark 0.50 out of 0.50

Consider the following list of free chunks, in continuous memory management:

10k, 25k, 12k, 7k, 9k, 13k

Suppose there is a request for chunk of size 15k, then the free chunk selected under each of the following schemes will be

Best fit:

25k

✓

First fit:

25k

✓

Worst fit:

25k

✓

Question **10**

Correct

Mark 0.25 out of 0.25

Select the state that is not possible after the given state, for a process:

New: Running ✓

Ready : Waiting ✓

Running: : None of these ✓

Waiting: Running ✓

Question **11**

Correct

Mark 1.00 out of 1.00

Given that the memory access time is 100 ns, probability of a page fault is 0.9 and page fault handling time is 8 ms,
The effective memory access time in nanoseconds is:

Answer: 7200010 ✓

The correct answer is: 7200010.00

Question **12**

Partially correct

Mark 0.83 out of 1.50

Arrange the following events in order, in page fault handling:

Page tables are updated for the process	8	✓
OS makes available an empty frame	7	✗
Process is kept in wait state	5	✗
OS schedules a disk read for the page (from backing store)	4	✗
Disk interrupt wakes up the process	6	✗
Restart the instruction that caused the page fault	9	✓
A hardware interrupt is issued	2	✓
The reference bit is found to be invalid by MMU	1	✓
Operating system decides that the page was not in memory	3	✓

Your answer is partially correct.

You have correctly selected 5.
The correct answer is: Page tables are updated for the process → 8, OS makes available an empty frame → 4, Process is kept in wait state → 6, OS schedules a disk read for the page (from backing store) → 5, Disk interrupt wakes up the process → 7, Restart the instruction that caused the page fault → 9, A hardware interrupt is issued → 2, The reference bit is found to be invalid by MMU → 1, Operating system decides that the page was not in memory → 3

Question **13**

Correct

Mark 0.50 out of 0.50

For each function/code-point, select the status of segmentation setup in xv6

after seginit() in main()	gdt setup with 5 entries (0 to 4) on one processor	✓
entry.S	gdt setup with 3 entries, at start32 symbol of bootasm.S	✓
bootasm.S	gdt setup with 3 entries, at start32 symbol of bootasm.S	✓
after startothers() in main()	gdt setup with 5 entries (0 to 4) on all processors	✓
bootmain()	gdt setup with 3 entries, at start32 symbol of bootasm.S	✓
kvmalloc() in main()	gdt setup with 3 entries, at start32 symbol of bootasm.S	✓

Your answer is correct.

The correct answer is: after seginit() in main() → gdt setup with 5 entries (0 to 4) on one processor, entry.S → gdt setup with 3 entries, at start32 symbol of bootasm.S, bootasm.S → gdt setup with 3 entries, at start32 symbol of bootasm.S, after startothers() in main() → gdt setup with 5 entries (0 to 4) on all processors, bootmain() → gdt setup with 3 entries, at start32 symbol of bootasm.S, kvmalloc() in main() → gdt setup with 3 entries, at start32 symbol of bootasm.S

Question **14**

Partially correct

Mark 0.91 out of 1.00

W.r.t. Memory management in xv6,

xv6 uses physical memory upto 224 MB onlyMark statements True or False

True	False		
● ✓	○ ✗	The process's address space gets mapped on frames, obtained from ~2MB:224MB range	✓
● ✓	○ ✗	xv6 uses physical memory upto 224 MB only	✓
● ✓	○ ✗	The switchkvm() call in scheduler() changes CR3 to use page directory kpgdir	✓
● ✓	○ ✗	The stack allocated in entry.S is used as stack for scheduler's context for first processor	✓
○ ✗	● ✓	The switchkvm() call in scheduler() changes CR3 to use page directory of new process	✓
● ✓	○ ✗	PHYSTOP can be increased to some extent, simply by editing memlayout.h	✓
○ ✗	● ✓	The switchkvm() call in scheduler() is invoked after control comes to it from swtch() scheduler(), thus demanding execution in new process's context	✓
● ✓	○ ✗	The kernel code and data take up less than 2 MB space	✓
● ✓	○ ✗	The switchkvm() call in scheduler() is invoked after control comes to it from sched(), thus demanding execution in kernel's context	✓
● ✓	○ ✗	The free page-frame are created out of nearly 222 MB	✓
● ✗	○ ✓	The kernel's page table given by kpgdir variable is used as stack for scheduler's context	✗

The process's address space gets mapped on frames, obtained from ~2MB:224MB range: True
xv6 uses physical memory upto 224 MB only: True
The switchkvm() call in scheduler() changes CR3 to use page directory kpgdir: True
The stack allocated in entry.S is used as stack for scheduler's context for first processor: True
The switchkvm() call in scheduler() changes CR3 to use page directory of new process: False
PHYSTOP can be increased to some extent, simply by editing memlayout.h: True
The switchkvm() call in scheduler() is invoked after control comes to it from swtch() scheduler(), thus demanding execution in new process's context: False
The kernel code and data take up less than 2 MB space: True
The switchkvm() call in scheduler() is invoked after control comes to it from sched(), thus demanding execution in kernel's context: True
The free page-frame are created out of nearly 222 MB: True
The kernel's page table given by kpgdir variable is used as stack for scheduler's context: False

Question **15**

Partially correct

Mark 1.11 out of 2.50

Order events in xv6 timer interrupt code

(Transition from process P1 to P2's code.)

P1 is marked as RUNNABLE	18	✗
change to context of the scheduler, scheduler's stack in use now	10	✗
P2 will return from sched() in yield()	13	✗
P2's trap() will return to alltraps	15	✗
yield() is called	8	✓
Trapframe is built on kernel stack of P1	6	✓
Timer interrupt occurs	4	✗
alltraps() will call iret	16	✗
Process P1 is executing	1	✓
Process P2 is executing	17	✗
change to context of P2, P2's kernel stack in use now	12	✗
jump to alltraps	5	✓
P2 is selected and marked RUNNING	12	✓
P2's yield() will return in trap()	14	✗
trap() is called	7	✓
jump in vector.S	4	✓
Change of stack from user stack to kernel stack of P1	3	✓
sched() is called,	9	✗

Your answer is partially correct.

You have correctly selected 8.

The correct answer is: P1 is marked as RUNNABLE → 9, change to context of the scheduler, scheduler's stack in use now → 11, P2 will return from sched() in yield() → 15, P2's trap() will return to alltraps → 17, yield() is called → 8, Trapframe is built on kernel stack of P1 → 6, Timer interrupt occurs → 2, alltraps() will call iret → 18, Process P1 is executing → 1, Process P2 is executing → 14, change to context of P2, P2's kernel stack in use now → 13, jump to alltraps → 5, P2 is selected and marked RUNNING → 12, P2's yield() will return in trap() → 16, trap() is called → 7, jump in vector.S → 4, Change of stack from user stack to kernel stack of P1 → 3, sched() is called, → 10

Question **16**

Partially correct

Mark 0.86 out of 1.00

Compare paging with demand paging and select the correct statements.

Select one or more:

- [x] a. Both demand paging and paging support shared memory pages. ✓
- [x] b. The meaning of valid-invalid bit in page table is different in paging and demand-paging. ✓
- [x] c. Paging requires some hardware support in CPU ✓
- [x] d. Demand paging always increases effective memory access time. ✓
- [ ] e. TLB hit ration has zero impact in effective memory access time in demand paging.
- [ ] f. Demand paging requires additional hardware support, compared to paging.
- [ ] g. With paging, it's possible to have user programs bigger than physical memory.
- [x] h. Calculations of number of bits for page number and offset are same in paging and demand paging. ✓
- [ ] i. Paging requires NO hardware support in CPU
- [x] j. With demand paging, it's possible to have user programs bigger than physical memory. ✓

Your answer is partially correct.

You have correctly selected 6.

The correct answers are: Demand paging requires additional hardware support, compared to paging., Both demand paging and paging support shared memory pages., With demand paging, it's possible to have user programs bigger than physical memory., Demand paging always increases effective memory access time., Paging requires some hardware support in CPU, Calculations of number of bits for page number and offset are same in paging and demand paging., The meaning of valid-invalid bit in page table is different in paging and demand-paging.

Question **17**

Correct

Mark 2.00 out of 2.00

Given below is shared memory code with two processes sharing a memory segment.
The first process sends a user input string to second process. The second capitalizes the string. Then the first process prints the capitalized version.
Fill in the blanks to complete the code.

// First process

```
#define SHMSZ    27

int main()
{
    char c;
    int shmid;
    key_t key;
    char *shm, *s, string[128];
    key = 5679;
    if ((shmid =
[shmget]
✓  (key, SHMSZ, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        exit(1);
    }
    if ((shm =
[shmat]
✓  (shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }
    s = shm;
    *s = '$';
    scanf("%s", string);
    strcpy(s + 1, string);
    *s = '
[@]
✓  '; //note the quotes
    while(*s != '
[$]
✓  ')
        sleep(1);
    printf("%s\n", s + 1);
    exit(0);
}
```

//Second process

```
#define SHMSZ    27
int main()
{
    int shmid;
    key_t key;
    char *shm, *s;
    int i;
    char string[128];
    key =
[5679]
✓ ;
    if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {
        perror("shmget");
        exit(1);
    }
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }
    s =
shm
✓ ;
    while(*s != '@')
        sleep(1);
    for(i = 0; i < strlen(s + 1); i++)
        s[i + 1] =  toupper(s[i + 1]);
    *s = '$';
    exit(0);
}
```

Question **18**

Correct

Mark 0.25 out of 0.25

The complete range of virtual addresses (after main() in main.c is over), from which the free pages used by kalloc() and kfree() is derived,are:

- a. end, P2V(4MB + PHYSTOP)
- [✓ b. end, P2V(PHYSTOP)] ✓
- c. P2V(end), PHYSTOP
- d. end, PHYSTOP
- e. end, (4MB + PHYSTOP)
- f. P2V(end), P2V(PHYSTOP)
- g. end, 4MB

Your answer is correct.

The correct answer is: end, P2V(PHYSTOP)

Question **19**

Partially correct

Mark 0.38 out of 0.50

Consider the image given below, which explains how paging works.

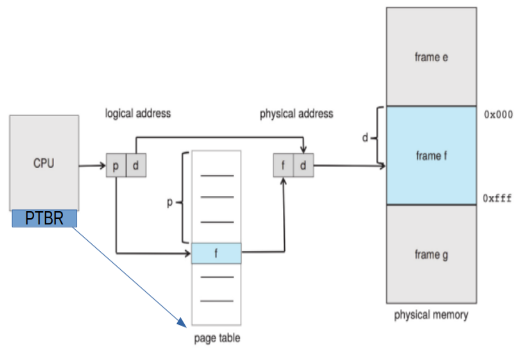

Figure 9.8 Paging hardware.

Mention whether each statement is True or False, with respect to this image.

True	False		
● ✓	○ ✗	Maximum Size of page table is determined by number of bits used for page number	✓
○ ✗	● ✓	The locating of the page table using PTBR also involves paging translation	✓
● ✓	○ ✗	The PTBR is present in the CPU as a register	✓
○ ✓	● ✗	The physical address may not be of the same size (in bits) as the logical address	✗
● ✓	○ ✗	The page table is indexed using page number	✓
● ✗	○ ✓	Size of page table is always determined by the size of RAM	✗
○ ✗	● ✓	The page table is indexed using frame number	✓
● ✓	○ ✗	The page table is itself present in Physical memory	✓

Maximum Size of page table is determined by number of bits used for page number: True
The locating of the page table using PTBR also involves paging translation: False
The PTBR is present in the CPU as a register: True
The physical address may not be of the same size (in bits) as the logical address: True
The page table is indexed using page number: True
Size of page table is always determined by the size of RAM: False
The page table is indexed using frame number: False
The page table is itself present in Physical memory: True

Question **20**

Partially correct

Mark 0.36 out of 0.50

After virtual memory is implemented

(select T/F for each of the following)One Program's size can be larger than physical memory size

True	False		
● ✗	○ ✓	Virtual addresses are available	✗
● ✓	○ ✗	Cumulative size of all programs can be larger than physical memory size	✓
● ✓	○ ✗	One Program's size can be larger than physical memory size	✓
● ✓	○ ✗	Code need not be completely in memory	✓
○ ✗	● ✓	Virtual access to memory is granted	✓
● ✓	○ ✗	Logical address space could be larger than physical address space	✓
○ ✓	● ✗	Relatively less I/O may be possible during process execution	✗

Virtual addresses are available: False
Cumulative size of all programs can be larger than physical memory size: True
One Program's size can be larger than physical memory size: True
Code need not be completely in memory: True
Virtual access to memory is granted: False
Logical address space could be larger than physical address space: True
Relatively less I/O may be possible during process execution: True

Question **21**

Correct

Mark 0.25 out of 0.25

The data structure used in kalloc() and kfree() in xv6 is

- ○ a. Doubly linked circular list
- ○ b. Singly linked circular list
- [✓ c. Singly linked NULL terminated list] ✓
- ○ d. Double linked NULL terminated list

Your answer is correct.

The correct answer is: Singly linked NULL terminated list

Question **22**

Incorrect

Mark 0.00 out of 1.00

Select all the correct statements about MMU and it's functionality

Select one or more:

- ☐ a. MMU is inside the processor
- ☑ b. MMU is a separate chip outside the processor ✗
- ☑ c. Logical to physical address translations in MMU are done with specific machine instructions ✗
- ☑ d. The Operating system sets up relevant CPU registers to enable proper MMU translations ✓
- ☐ e. Illegal memory access is detected by operating system
- ☑ f. The operating system interacts with MMU for every single address translation ✗
- ☐ g. Logical to physical address translations in MMU are done in hardware, automatically
- ☑ h. Illegal memory access is detected in hardware by MMU and a trap is raised ✓

Your answer is incorrect.

The correct answers are: MMU is inside the processor, Logical to physical address translations in MMU are done in hardware, automatically, The Operating system sets up relevant CPU registers to enable proper MMU translations, Illegal memory access is detected in hardware by MMU and a trap is raised

Question **23**

Partially correct

Mark 0.33 out of 0.50

Match the pair

Hashed page table	Linear search on collsion done by OS (e.g. SPARC Solaris) typically	✓
Hierarchical Paging	More memory access time per hierarchy	✓
Inverted Page table	Linear/Parallel search using frame number in page table	✗

Your answer is partially correct.

You have correctly selected 2.

The correct answer is: Hashed page table → Linear search on collsion done by OS (e.g. SPARC Solaris) typically, Hierarchical Paging → More memory access time per hierarchy, Inverted Page table → Linear/Parallel search using page number in page table

Question **24**

Correct

Mark 0.50 out of 0.50

Suppose a kernel uses a buddy allocator. The smallest chunk that can be allocated is of size 32 bytes. One bit is used to track each such chunk, where 1 means allocated and 0 means free. The chunk looks like this as of now:

10011010

Now, there is a request for a chunk of 50 bytes.

After this allocation, the bitmap, indicating the status of the buddy allocator will be

Answer: 11111010 ✓

The correct answer is: 11111010

Question **25**

Partially correct

Mark 0.57 out of 1.00

Mark True, the actions done as part of code of swtch() in swtch.S, in xv6

True	False		
● ✓	○ ✗	Save old callee saved registers on kernel stack of old context	✓
○ ✗	● ✓	Jump to code in new context	✓
● ✓	○ ✗	Switch from one stack (old) to another(new)	✓
● ✗	○ ✓	Save old callee saved registers on user stack of old context	✗
● ✗	○ ✓	Restore new callee saved registers from user stack of new context	✗
● ✗	○ ✓	Switch from old process context to new process context	✗
● ✓	○ ✗	Restore new callee saved registers from kernel stack of new context	✓

Save old callee saved registers on kernel stack of old context: True
Jump to code in new context: False
Switch from one stack (old) to another(new): True
Save old callee saved registers on user stack of old context: False
Restore new callee saved registers from user stack of new context: False
Switch from old process context to new process context: False
Restore new callee saved registers from kernel stack of new context: True

◄ inode-code-similarities-plagiarism

Jump to...

syscall-add-xv6 ►