CS 537: Introduction to Operating Systems
Fall 2024: Midterm Exam #1

This exam is closed book, but you may use 1 sheet of notes.

No calculators may be used.

You have 1 hour and 30 minutes to complete this exam.

Write all of your answers on the accu-scan form with a #2 pencil:

- CANVAS LAST NAME - fill in your last (family) name starting at the leftmost column
- CANVAS FIRST NAME - fill in the first five letters of your first (given) name
- IDENTIFICATION NUMBER - This is your UW Student WisCard ID number
- ABC of SPECIAL CODES - Write your lecture number as a three digit value:
  - 001 – TuTh 9:30-10:45 (Louis)
  - 002 – TuTh 11:00-12:15 (Shivaram)

These exam questions, your sheet of notes, and the scantron must be returned at the end of the exam, but we will not grade anything in this booklet. You will not be getting your note sheet back.

Unless stated (or implied) otherwise, you should make the following assumptions:

- The OS manages a single uniprocessor (single core)
- All memory is byte addressable
- Page table and page directory entries require 4 bytes
- Data is allocated with optimal alignment, starting at the beginning of a page
- Assume leading zeros can be removed from numbers (e.g., 0x06 == 0x6).
- Hex numbers are represented with a proceeding “0x”

The following might help you with some calculations:

- 0x100 = 2^8 = 256
- 2^{10} = 1024
- 2^{12} = 4096
- 2^{10} bytes = 1KB
- 2^{20} bytes = 1MB

This exam has 60 questions. Each question has the same number of points.

Good luck!

Processes and Scheduling

True(A)/False(B)

1. The return value of `execvp()` is 0 on success. - **False. Exec does not return on success**
2. In the cooperative approach to limited direct execution, the scheduler waits for a process to perform a system call or voluntarily give up the CPU. **True**
3. The PCB holds the most recent instruction pointer of a process while it is actively running on the CPU. **False - most recent IP is in the hardware register.**

Multiple Choice

The scheduler of a single-cpu system has three processes and is context-switching between them. When a process performs IO it blocks for 5 time steps and when it executes next it enters the RUN:io_done state. Each process needs to run for 3 time steps to finish. The timeline for the first 9 timesteps and each processes' state is shown below:

```
Time        PID: 0        PID: 1        PID: 2      CPU      IOs
  1         RUN:io         READY         READY        1
  2        BLOCKED       RUN:cpu         READY        1        1
  3        BLOCKED        RUN:io         READY        1        1
  4        BLOCKED       BLOCKED        RUN:io        1        2
  5        BLOCKED       BLOCKED       BLOCKED                 3
  6        BLOCKED       BLOCKED       BLOCKED                 3
  7        RUN:io_done   BLOCKED       BLOCKED        1        2
  8        RUN:io        BLOCKED       BLOCKED        1        2
  9        BLOCKED   RUN:io_done       BLOCKED        1        2
```

4. What is a possible state for the three processes at timestep 10, the next timestep?

    A. PID: 0 - DONE PID: 1 - RUN:cpu PID: 2 - BLOCKED
    B. PID: 0 - BLOCKED PID: 1 - RUN:cpu PID: 2 - BLOCKED
    C. PID: 0 - BLOCKED PID: 1 - RUN:io PID: 2 - RUN:io_done
    D. PID: 0 - BLOCKED PID: 1 - RUN:cpu PID: 2 - READY
    E. PID: 0 - BLOCKED PID: 1 - READY PID: 2 - READY

Option D. PID 0 has to be blocked. PID 1 can be in RUN or READY state, PID 2 has to be in RUN:io_done or READY state. Exactly one process needs to be running. Only Option D matches these requirements.

5. In a system that has three process states – READY, RUNNING, and BLOCKED — if a process is in the BLOCKED state for the current timestep then its previous timestep must have been

    A. Non-existent because it just started and had no previous timestep.
    B. READY
    C. RUNNING
    D. BLOCKED

E. RUNNING or BLOCKED

Option E (only a running process can become blocked or it could have been blocked in the previous timestep)

6. In the same system as the previous question, if a process is in the READY state, what are the possible states that it could be in for the next timestep?
    A. READY
    B. RUNNING
    C. BLOCKED
    D. RUNNING or BLOCKED
    E. READY or RUNNING

Option E (It could remain in the READY state or be selected to RUNNING)

A workload of 5 jobs is being scheduled on a single-cpu system with a timeslice of 1 second. The jobs do not perform any IO, just compute. The jobs' arrival times and lengths are shown in the table below. If the scheduler has a choice the job that arrives earlier is selected. If there still is a choice the job that comes first alphabetically is selected.

Job	**Arrival Time (sec)**	**Job Length (sec)**
A	0	4
B	0	2
C	4	1
D	4	3
E	5	1

7. If the scheduler is running the FIFO algorithm, what is the average turnaround time of this workload?
    A. 25 / 5 seconds
    B. 21 / 5 seconds
    C. 14 / 5 seconds
    D. 11 / 5 seconds
    E. 10 / 5 seconds

Option A. The trace would be AAAABBCDDDE. Turnaround times are A=4-0, B=6-0, C=7-4, D=10-4, E=11-5.

8. If the scheduler is running the SJF algorithm, what is the average response time of this workload?
    A. 21 / 5 seconds
    B. 6 / 5 seconds
    C. 7 / 5 seconds

D. 11 / 5 seconds
E. 10 / 5 seconds

Option E. Trace is BBAAAACEDDD. Response times are A=2-0, B=0-0, C=6-4, D=8-4, E=7-5.

9. If the scheduler is running MLFQ with 3 queues – high, middle, and low – and the timeslice for each queue is 1 second with no boosting, what will the queues look like **after 6 timesteps have completed**? Each queue uses the same tie breaker policy as indicated above.

A.

high	E
middle	D
low	A

B.

high	D, E
middle	A
low	

C.

high	D, E
middle	B, C
low	

D.

high	C, D, E
middle	B
low	

E.

high	C, D
middle	
low	A

Option A. The trace for the first 6 timesteps is ABABCD.

10. If the scheduler now has boosting occurring after every 4 timeslices, what will the queues look like after 6 timesteps have completed?

A.

high	E
middle	D
low	A

B.

high	D, E
middle	A
low	

C.

high	C
middle	D, E
low	A

D.

high	C, D, E
middle	B
low	

E.

high	E

Option B. The trace for the first 6 timesteps is ABABAC.

middle	C, D
low	A, B

11. If your goal is to minimize turnaround time, which non-preemptive algorithm should you choose?
    A. FCFS
    B. FIFO
    C. SJF
    D. STCF
    E. RR

Option C. STCF and RR are preemptive. FIFO and FCFS are the same algorithm.

12. Recall the initial rules for MLFQ are:

    1. If Priority(A) > Priority(B) then A runs
    2. If Priority(A) == Priority(B) then A&B run in RR
    3. Jobs start at top priority
    4. If a job uses its whole time slice then it is demoted

These initial rules had several problems and needed to be changed. What is one problem with these initial rules and the modification that was made to correct for this problem?

A. Long running, cpu-bound jobs might starve. To correct this rule 4 was modified so a job would be demoted when it uses up its time allotment regardless of how many time slices it takes.
B. Programmers could game the system, creating programs that receive more than their fair share of CPU time. To correct this rule 3 was modified to start jobs at a middle priority.
C. Jobs would not be sorted into those that are cpu-bound and those that are io-bound. To correct this a 5th boosting rule was added.
D. Programmers could game the system, creating programs that receive more than their fair share of CPU time. To correct this rule 4 was modified so a job would be demoted when it uses up its time allotment regardless of how many time slices it takes.
E. Long running, cpu-bound jobs might starve. To correct this lower priority queues would have a longer time slice before a process was demoted.

Option D.

Processes and Their Address Spaces

Multiple Choice

Look at the code for the *prog* and *other* programs shown below. Assume all function and system calls are successful, output occurs immediately, and all the necessary includes are present.

prog.c

```c
int
main()
{
    printf("1 ");
    fork();
    printf("2 ");
    int rc = fork();
    if (rc == 0) {
        fork();
        printf("3 ");
        char *myargs[2];
        myargs[0] = strdup("./other");
        myargs[1] = NULL;
        execv(myargs[0], myargs);
        printf("4 ");
    } else {
        printf("5 ");
        int wc = wait(NULL);
        printf("6 ");
    }
    printf("7 ");
}
```

other.c

```c
int A;

int* foo(int *Z)
{
   Z = malloc(sizeof(int));
   *Z = 2;
   return Z;
}

int
main()
{
    A = 10;
    int *B;
    B = malloc(sizeof(int));
    *B = 5;
    int *C = foo(B);
    if (*C == 2)
    {
      printf("A ");
    } else
    {
      printf("B ");
    }
    free(B);
    free(C);
}
```

13. When `prog` is executed how many times will the number 3 be printed?
    A. 1
    B. 2
    C. 3
    D. 4
    E. Something else

Option D

14. When `prog` is executed, which of the following outputs is impossible?
    A. 1 2 3 4 5 6 7 A
    B. 1 2 2 3 3 3 3 A A A A 4 4 4 4 5 5 6 6 7 7
    C. 1 2 2 3 3 3 3 5 5 6 6 A A A A 7 7
    D. 1 2 2 5 5 3 3 3 3 A A A A 4 4 4 4 6 6 7 7
    E. All are impossible outputs

Option E. 4 should never be printed eliminating A, B, & D. 6 cannot be printed before A eliminating C.

15. When the `other` program executes, what area of address space will contain the `Z` variable?
    A. Stack area
    B. Heap area
    C. Code / Static area

Option A as Z is the pointer which is a function argument and hence on the stack

16. When line 5 of the `other` program is executed, how many frames will be on the process' call stack?
    A. 1
    B. 2
    C. 3
    D. 4
    E. D

Option B. Two - one for main and one for foo

Virtual Addressing using Segmentation

True(A)/False(B)

17. One disadvantage of segmentation is external fragmentation. **True**
18. Virtual addresses and physical addresses must be the same size. **False**
19. In kernel mode the MMU checks the VA's offset against the base and bounds for that segment. **False – the MMU only needs to compare the offset with the bound and not with the base**
20. For a segment that grows positively, If a VA's offset is within bounds then the PA is the segment's base + the offset. **True**
21. For a segment that grows negatively, if a VA's offset is out-of-bounds then a page fault will occur. **False - Bound should be offset - max_segment_size, also a segmentation fault would occur not a page fault.**

Multiple Choice

22. A system uses 16-bit virtual addresses with the 2 high order bits specifying the segment and the remaining bits for the offset. The system has 1MB of addressable physical memory. What is the maximum segment size?
    A. 4KB
    B. 8KB
    C. 16KB
    D. 32KB
    E. None of the above

    Option C. 2^14 = 16KB

23. In this same system, a particular VA has an offset of 8KB and the segment for this VA is one that grows negatively. What is the negative offset?
    A. -4KB
    B. -8KB
    C. -16KB
    D. -32KB
    E. None of the above

    Option B. 8-16KB = -8KB

The segment table for a process running in this system is shown below:

Segment (Bits)	**Base**	**Bounds (Size)**	**Grows Positive?**	**Protection Bits (RWX)**
Code (00)	0x3000	1KB	1	101
Heap (01)	0x3800	2KB	1	110
Unused (10)			0	000
Stack (11)	0x8800	4KB	0	110

24. Convert the VA 0x8011 to its PA.
    A. 0x01C11
    B. 0x01811
    C. 0x04411
    D. 0x04FEE
    E. None of the above

Option E. Segmentation Fault trying to use the Unused segment.

25. Convert the VA 0x7011 to its PA.
    A. 0x09011
    B. 0x0FFEE
    C. 0x04011
    D. 0x04FEE

E. None of the above

Option E. Segment is Heap and offset is greater than 2KB. (12305 decimal)

26. What is the largest VA for this process that will not cause any faults?
    A. 0xFFFF
    B. 0x8800
    C. 0x87FF
    D. 0xD800
    E. None of the above

Option A. 0xFFFF will be stack (11) with offset of (11 1111 1111 1111). The negative offset is -1, whose absolute value is smaller than 4KB.

27. What is the largest VA for this process that will not cause any faults and be in the heap segment?
    A. 0x4400
    B. 0x4800
    C. 0x4FFF
    D. 0x4000
    E. None of the above

Option B. heap (01) with offset at 2KB = 0100 10000 0000 0000 = 0x4800.

Paging

True(A)/False(B)

28. A page table entry holds the virtual page number for a given physical page. **False, it holds the PFN.**
29. By marking PTE's invalid, no physical page needs to be allocated for that entry. **True.**
30. If the MMU finds that a PTE is invalid it will allocate a physical page and update the PTE. **False, The MMU would cause a fault.**
31. In a system that uses paging, each process will have its own page table. **True.**
32. For a system that uses 32-bit virtual addresses with a 12-bit offset and has 4-byte page table entries, the page table is 4MB in size. **True, 20 bits for the VPN so 1M pages * 4 = 4MB**

Multiple Choice

33. On a 32-bit machine, how big is the virtual address space for each process?
    A. 4GB
    B. 4KB
    C. 4MB
    D. 1GB
    E. 1KB

Option A, 4GB. 2^32

34. On the same 32-bit machine, if the address space for a process is divided into pages of size 4KB and each page table entry is 8 bytes, how large would a linear page table be for each process?
    A. 2MB
    B. 4MB
    C. 8MB

D. 12MB
E. 16MB

Option C. 8MB. 1M pages * 8bytes = 8MB

35. On the same system, how many pages are there in the virtual address space of a process?
    A. 1M pages
    B. 4K pages
    C. 256K pages
    D. 1G pages
    E. None of the above

Option A. 1M pages

A system uses linear page tables and has the following parameters:

Address Space size: 32KB
Physical Memory size: 64KB
Page size: 4KB

A process that is currently running on the system has the following page table base register and page table as shown below. Note, only 5 bits of every PTE are shown. The full size of a PTE is 1 byte.
PTBR: 0x2004

Page Table

PFN	**Valid**
1111	1
0011	1
0000	0
0000	0
0000	0
0000	0
0000	0
1010	1

The process's instructions and virtual addresses are:

```
0x0000 movl 0x1100, %edi  (load x)
0x0008 addl $0x1, %edi    (add 1)
0x000C movl %edi, 0x1100  (store x)
```

36. What is the size of a virtual address?
    A. 12 bits
    B. 15 bits
    C. 16 bits
    D. 18 bits
    E. None of the above

Option B, 15 bits as address space is 32KB.

37. What is the first physical address that is accessed?
    A. 0x0000
    B. 0x1100
    C. 0x2004
    D. 0xF000
    E. None of the above

Option C. The first VA is 0x0000 to load the instruction, but it must be converted to a PA by going to the proper page of the page table. PTBR + (VPN*size of PTE) = 0x2004.

38. What is the second physical address that is accessed?
    A. 0x1100
    B. 0x2005
    C. 0xF000
    D. 0x3100
    E. None of the above

Option C. PFN + offset = 1111 + (0000 0000 0000) = 0xF000

39. How many total memory accesses are there?
    A. 2
    B. 4
    C. 5
    D. 10
    E. None of the above

Option D. Each VA must be converted to a PA by consulting the page table and then going to memory to access content. 5 VA * 2 = 10 memory accesses

Translation-Lookaside Buffer

True(A)/False(B)

40. The purpose of the TLB is to speed address translation. **TRUE**
41. The valid bit in the TLB means the same as the valid bit in the entries of the page table. **FALSE**
42. In a hardware managed TLB, if the MMU finds that a TLB entry is invalid, it will read the PTE from memory and add it to the TLB. **TRUE**
43. The purpose of the ASID in a TLB entry is to reduce the size of a process' page table. **FALSE - ASID is used to reduce the number of TLB flushes performed on context switch**

Multiple Choice

In C an array is stored in row-major format. For example, an integer array of shape 4x4 with data like the one is shown in Figure 2 in memory as shown in Figure 3.

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

Figure 2: Example 2D integer array

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

Figure 3: Memory representation in row-major format

Assume there is a tiny address space where each page is of 8 bytes and there are 16 such pages. We have a 4x4 array and the element a[0] is located at virtual address 20. (Virtual Page Number=02, offset = 04). Assume size of integer is 4 bytes

Consider the following function:

```
void print_array(int *arr, int cols, int rows){
   for (size_t i=0; i<rows; i++){
      for (size_t j=0; j<cols; j++){
         printf("%d\n", arr[i*cols+j];
      }
    }
}
```

44. Ignoring variables i and j, what will be the first array entry we get a TLB hit for if the TLB can hold 4 entries with LRU replacement policy.
    A. a[0]
    B. a[1]
    C. a[2]

D. a[4]
E. None of the above

Option C. 2 entries can be stored per page. As array starts at offset is 04, only one entry fits in the first page. So arr[0] is alone in the first page and then arr[1], arr[2] in the second page and so on.
Now consider a function where we print the array contents in column order

```
void print_col_array(int *arr, int cols, int rows){
     for(size_t i=0; i<rows; i++){
         for(size_t j=0; j<cols; j++){
                printf("%d\n", arr[j*rows+i]);
        }
}
```

45. What will be the first TLB hit with the same TLB size and replacement policy as before?
    A. a[0]
    B. a[1]
    C. a[2]
    D. a[4]
    E. None of the above

Option C. Accesses are in the index order (with TLB hit indicated with *): 0, 4, 8, 12, 1, 5, 9, 13, 2*, …

46. If the TLB size was increased to 8 entries, the TLB hit rate for the first function print_array would be:
    A. 1/16
    B. 8/16
    C. 12/16
    D. 14/16
    E. None of the above
    The hit rate would be the same as the first case of having 4 entries 7/16: 0, 1, 2*, 3, 4*, 5, 6*, 7, 8*, 9, 10*, 11, 12*, 13, 14*, 15.

Smaller Tables

True(A)/False(B)

47. For the same address space size, having smaller pages means you will have smaller page tables. **False, smaller pages means a smaller offset and more bits for the VPN, thus a larger page table.**
48. By having a multi-level page table a TLB miss becomes even more costly. **True, a miss means multiple accesses to memory in order to find the PTE.**
49. Sparsely used address spaces are poor candidates for multi-level page tables. **False, they are excellent candidates because an entry in the PD can be marked invalid not needing a whole range of PTEs for large sections of the address space.**

Multiple Choice

A system uses a 2-level page table with the following parameters:

Page Sizes are 32 bytes
Virtual Address space is 1024 pages (or 32 KB)
Physical Memory consists of 128 pages
A Virtual Address needs 15 bits (5 for the offset, 10 for the VPN)
A Physical address needs 12 bits (5 for the offset, 7 for the PFN)

The system uses a multi-level page table. The upper five bits of a VA are the index into the page directory to get the PDE. Each PDE is 8 bits. If the PDE is valid, it points to a page of the page table. Each page of the page table holds 32 8-bit PTEs. If the PTE is valid, it holds the desired translation (PFN).

The format of a PTE is:

VALID | PFN6 ... PFN0

The format of a PDE is identical:

VALID | PT6 ... PT0

The PDBR holds the value: 0x54 (decimal: 84) [This means the page directory is held in this page]

The content of physical memory is on the next two pages. The left-most column shows the physical page number in hex and decimal. The top two rows show the offset of the bytes in the pages in both hex and decimal.

```
hex offset    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
dec offset    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

0x00 (  0): 0f 17 16 15 06 1e 18 0b 04 09 01 16 0e 19 1d 14 05 09 05 03 06 0b 04 16 0e 0e 1e 18 19 02 04 12
0x01 (  1): 12 02 08 07 0f 15 19 06 1c 11 18 0e 02 0d 0e 08 02 0a 0d 06 04 0c 09 01 09 13 12 01 00 10 0b 13
0x02 (  2): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f a3 7f 7f 7f 7f 7f 7f dc 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x03 (  3): 10 15 0f 05 17 02 1b 11 18 03 11 1a 17 17 18 13 15 0c 15 07 0f 05 0e 0f 13 09 04 14 07 05 06 1a
0x04 (  4): 15 0c 11 01 11 0b 10 03 16 03 13 0d 10 05 17 00 19 09 16 05 14 1a 01 18 1b 07 00 16 1b 0e 17 16
0x05 (  5): 0c 1a 15 11 0e 18 02 17 07 0b 0b 10 18 17 11 0a 0f 11 16 02 0a 11 16 08 1d 09 17 07 13 0d 0b 19
0x06 (  6): 07 16 16 0c 0b 11 17 13 00 10 16 0c 03 1e 02 03 1e 19 10 18 1a 06 04 15 0e 0e 12 1a 14 1a 13 0c
0x07 (  7): 03 1e 08 0a 08 09 0a 09 02 10 15 07 18 19 15 07 08 05 03 07 1d 1b 1b 19 0c 1d 10 11 1e 04 09 0d
0x08 (  8): 0d 01 0b 0a 06 06 18 09 0f 14 0f 18 11 01 08 1b 0e 16 0c 17 05 0e 0d 0f 12 10 0a 01 07 09 14 1e
0x09 (  9): 0f 10 05 10 0c 03 1c 00 16 0a 0b 18 07 19 02 07 02 14 10 02 08 06 1c 09 0a 02 09 0f 06 1e 18 0c
0x0a ( 10): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0b ( 11): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f f6 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 83 7f 7f 7f 7f 7f
0x0c ( 12): 00 00 19 04 01 1e 0d 02 0a 07 0e 12 17 11 05 08 1a 18 13 12 00 1c 05 12 11 1c 12 15 12 06 0e 18
0x0d ( 13): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0e ( 14): 0e 15 09 18 17 04 12 17 1b 0e 03 17 1c 17 1b 01 15 03 13 15 06 07 1d 06 03 17 0a 0b 0b 03 1b 0f
0x0f ( 15): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10 ( 16): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x11 ( 17): 19 10 15 1e 13 15 05 0d 0d 1c 13 18 06 16 00 0d 1d 01 0b 13 04 03 1e 09 1d 06 01 0c 08 18 00 03
0x12 ( 18): c8 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f f9 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x13 ( 19): 0f 1a 0b 19 0a 05 0a 1e 07 05 06 03 0c 0b 0d 18 1c 1c 02 12 11 15 1e 02 01 07 06 14 0f 02 04 08
0x14 ( 20): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x15 ( 21): 03 09 13 0d 1c 14 04 04 1a 1c 17 17 0c 0c 1c 15 0b 0b 12 1d 1a 03 09 05 13 01 0f 10 16 18 09 00
0x16 ( 22): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x17 ( 23): 7f 7f 7f 7f 85 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f cd 7f 7f 7f 7f 7f 7f 7f 7f
0x18 ( 24): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x19 ( 25): 7f 7f 7f 7f 7f 7f 7f fe 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f db 7f 7f 7f 7f 7f 7f 7f c2
0x1a ( 26): 07 1a 14 1d 02 19 01 03 04 1c 06 19 07 18 08 01 1c 0a 01 09 0e 15 10 10 06 02 0c 17 13 1d 07 0b
0x1b ( 27): 16 08 10 03 18 07 16 0a 07 07 02 09 0e 10 17 14 06 09 17 04 08 18 17 00 01 14 12 09 1a 1e 1a 01
0x1c ( 28): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f b9 7f
0x1d ( 29): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 9f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x1e ( 30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1f ( 31): 19 16 12 19 10 13 07 15 11 11 0b 00 02 08 18 14 11 0e 01 16 01 1b 0a 14 05 12 03 15 02 1a 19 0c
0x20 ( 32): 7f 7f 7f 7f 7f 7f b7 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 9b 7f 7f 80 7f 7f 7f 7f 7f 7f 7f c5
0x21 ( 33): fb 7f 7f 7f 7f e3 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x22 ( 34): 15 10 0c 02 0b 14 00 0c 02 0d 06 10 03 1e 1e 19 1d 05 07 05 17 01 00 13 0c 16 14 0d 06 1a 09 17
0x23 ( 35): 15 13 0f 1e 12 07 0e 0c 1c 0d 11 0b 03 0d 1a 10 03 0e 19 1c 1d 1e 0b 1a 1b 1d 02 11 11 14 10 0d
0x24 ( 36): 11 16 18 0b 0b 02 1d 18 03 0d 03 1d 08 1c 17 1b 13 0c 05 12 04 0e 16 12 1e 00 10 07 03 13 04 0c
0x25 ( 37): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x26 ( 38): 0a 1b 07 0d 16 08 13 14 12 0e 09 17 1a 11 1d 0a 0e 16 09 19 14 03 08 09 04 17 19 04 04 17 1e 14
0x27 ( 39): 7f 7f 7f 7f 7f e0 7f 7f 7f 7f 91 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x28 ( 40): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x29 ( 41): 7f c0 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f ea 7f 7f 7f 7f 7f 7f 7f 7f
0x2a ( 42): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2b ( 43): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f c4 f1 7f 7f 7f 7f 7f 7f 7f 7f b2 7f
0x2c ( 44): 0e 07 19 09 12 0f 1e 05 04 0a 15 11 19 1a 0d 14 0d 0c 03 07 05 01 16 11 00 03 00 0d 0b 03 0d 10
0x2d ( 45): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2e ( 46): 1e 07 11 1d 03 11 03 09 12 04 08 0a 17 0e 1a 03 1b 15 1b 07 06 11 12 07 16 08 0f 11 10 10 08 1b
0x2f ( 47): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x30 ( 48): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x31 ( 49): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x32 ( 50): 1d 04 12 11 17 0a 15 0f 1b 12 1d 04 19 02 1b 19 0a 08 0a 04 0e 07 0b 05 1d 19 13 03 09 13 03 14
0x33 ( 51): 0c 17 1d 0d 06 09 1d 09 07 09 0d 06 0d 09 04 0c 1c 18 1b 18 0a 07 04 01 18 0c 0b 15 0e 1d 14 02
0x34 ( 52): 00 1a 07 0d 0d 0f 06 0e 15 15 07 12 19 15 04 13 0d 0d 04 03 05 03 1e 0a 16 0c 1c 19 0b 06 04 19
0x35 ( 53): 11 08 06 0a 13 0d 1c 1c 0c 1a 02 1c 1a 0f 0e 18 1b 1e 03 0b 00 0c 1a 10 14 0b 03 17 08 08 16 12
0x36 ( 54): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x37 ( 55): 1b 07 04 0f 09 0b 13 0f 03 02 1a 0e 0d 19 10 13 04 0e 15 19 04 07 00 17 09 1d 10 0d 19 17 10 0c
0x38 ( 56): 7f fd 7f c9 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f a4 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x39 ( 57): 01 1b 15 1b 15 19 00 15 06 16 00 1a 10 06 01 0c 1c 0d 00 0f 07 03 02 1c 0b 16 02 10 10 06 05 0f
0x3a ( 58): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 95 81 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 8e 7f 7f
0x3b ( 59): 1e 1b 07 0e 0f 15 1e 0d 1e 13 13 03 19 03 0a 0c 0f 13 11 04 15 03 05 03 09 13 11 1e 09 0b 0b 1d
0x3c ( 60): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x3d ( 61): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f b3 88 7f 7f 9a 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x3e ( 62): 7f 7f 7f 7f b4 7f b5 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f dd 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x3f ( 63): 0c 02 00 14 10 0c 0c 02 0a 08 06 06 01 1e 01 16 16 0d 17 03 06 16 06 0a 12 09 15 0f 00 0d 0a 02
```

```
0x40 ( 64): 03 16 10 08 13 1c 0b 16 1e 0c 03 14 16 02 1a 11 0c 1a 0f 02 1a 0b 1e 19 08 10 16 04 08 06 17 13
0x41 ( 65): 7f f5 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f a6 7f 7f 7f 7f 7f bb 7f 7f 7f 7f 7f 7f 7f 7f
0x42 ( 66): 14 0c 0c 02 00 0d 04 13 1d 19 00 01 0d 02 13 07 18 03 17 0f 09 08 18 10 14 05 0c 07 08 1e 14 11
0x43 ( 67): 7f 7f 7f 7f 7f 7f 7f ae 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
hex offset   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
dec offset   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

0x44 ( 68): 06 09 18 1a 05 01 14 19 1d 0f 0f 17 03 04 14 13 08 03 0d 0e 06 1c 0d 01 14 1a 03 0a 11 0f 1e 02
0x45 ( 69): 01 00 14 18 12 17 10 12 19 01 12 19 11 02 0e 1e 1d 0a 0d 19 18 05 0b 14 0e 13 14 1b 10 19 15 1e
0x46 ( 70): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x47 ( 71): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x48 ( 72): 02 1b 0a 09 14 17 1a 15 10 18 15 17 11 19 08 0d 04 19 1a 1e 19 15 02 07 1a 11 1c 11 14 14 03 0d
0x49 ( 73): 10 06 0b 1d 18 07 1c 03 03 17 15 02 12 19 00 04 04 0c 0a 18 1b 18 11 09 11 1b 19 14 05 12 0b 0d
0x4a ( 74): 7f 7f 7f fc 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 84 7f 7f 7f 7f f8 7f 7f
0x4b ( 75): 10 07 1c 0e 17 19 15 09 17 1d 0f 00 1b 0d 11 11 0e 00 18 01 11 06 17 02 04 1a 11 16 18 19 1e 17
0x4c ( 76): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4d ( 77): 11 00 01 14 13 0d 1b 17 1b 1c 0c 00 1d 1c 15 10 01 0b 17 14 0b 0c 13 16 11 09 1e 17 05 17 1a 13
0x4e ( 78): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4f ( 79): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x50 ( 80): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f e6 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x51 ( 81): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f d3 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f df a2 ac 7f 7f 7f 7f
0x52 ( 82): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x53 ( 83): 07 1c 02 0c 17 18 02 14 14 05 0d 00 1d 00 15 19 19 0b 09 0c 0e 13 15 0e 1b 0e 0c 0d 08 0e 02 17
0x54 ( 84): be a7 e4 9d 7f 7f 92 7f d7 99 a0 e5 a9 9c a1 82 ca ba f2 7f b8 97 bd c1 e8 e2 d0 d1 ab d9 c3 8b
0x55 ( 85): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x56 ( 86): 05 08 08 1a 00 10 03 03 08 19 0e 11 10 13 00 15 0b 10 0d 0f 01 11 07 17 1b 00 08 1b 0a 0b 18 10
0x57 ( 87): 7f 86 7f 87 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f ed 7f 7f 7f 7f 7f 7f 7f 7f d6 7f 7f 7f 7f 89 7f
0x58 ( 88): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x59 ( 89): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f e1 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f cb 7f 7f 7f 7f
0x5a ( 90): 08 0e 03 07 02 08 0f 10 07 1c 01 0a 0e 15 15 0e 15 02 16 1b 1e 10 11 16 16 1a 06 05 1e 18 1c 01
0x5b ( 91): 05 05 16 15 12 10 13 17 14 04 07 1a 0b 14 16 07 0c 0c 0c 07 0d 0f 1c 19 1c 12 18 09 07 1d 1b 19
0x5c ( 92): 1b 1b 1d 04 0b 0b 03 06 08 0e 16 1a 1a 0c 0a 0f 1a 06 13 0d 03 09 0e 0a 19 15 1b 02 19 18 04 1e
0x5d ( 93): 13 00 09 12 0e 06 05 1b 05 18 1a 00 16 11 11 13 14 12 1e 0f 1e 0a 0c 01 14 1e 09 16 17 01 00 1b
0x5e ( 94): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x5f ( 95): 0c 11 0a 09 07 1b 1c 02 07 10 12 02 09 1a 13 02 15 04 19 08 18 17 06 19 10 13 09 1c 0f 1d 02 0d
0x60 ( 96): 0a 16 05 12 1d 1d 04 05 04 0d 0a 05 0c 11 18 06 10 07 10 0b 15 03 14 08 18 04 02 08 0c 16 12 0d
0x61 ( 97): 13 01 01 16 11 0c 0e 10 02 0f 00 06 10 19 0a 0d 0f 17 14 14 04 1c 15 19 15 07 03 10 1b 0d 1b 1d
0x62 ( 98): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f bf 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x63 ( 99): 16 00 16 0d 05 0f 0c 10 0c 15 03 06 01 1e 17 0b 08 1e 09 0d 18 1b 12 19 05 00 03 0b 06 0d 0e 1a
0x64 (100): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f fa 7f 7f 7f 7f 7f 7f
0x65 (101): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 8c 7f 7f ef 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x66 (102): 02 05 07 19 06 1a 16 0e 16 01 1e 14 17 05 11 0d 04 1c 10 12 19 07 04 17 15 12 18 08 1e 05 17 17
0x67 (103): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x68 (104): 7f 7f 7f 7f 7f 7f 7f 7f da 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f ee 7f 7f 7f 7f
0x69 (105): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6a (106): 14 13 0e 17 0c 02 04 18 0e 14 01 18 12 17 1a 14 0d 14 16 0c 03 12 1d 07 08 16 06 02 0b 0d 1a 04
0x6b (107): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6c (108): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6d (109): 07 14 05 11 0a 07 0a 0f 08 0b 0d 0a 19 08 0d 0d 01 18 04 13 1b 01 09 0e 0e 1e 1a 03 0f 0a 01 1d
0x6e (110): 12 02 19 1d 0f 1c 0e 1b 02 16 01 01 1d 02 0a 1d 05 08 01 1e 00 1d 16 0a 1b 19 1a 1a 1c 06 13 0e
0x6f (111): 19 0a 00 02 0a 08 08 0d 07 01 14 18 04 02 16 0c 0b 1c 1b 04 05 0d 11 04 00 02 09 1b 01 15 04 0d
0x70 (112): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x71 (113): 14 17 05 08 14 18 0c 14 1b 0b 18 13 0c 13 06 1e 10 12 0c 04 1c 05 11 1d 05 10 05 1a 15 14 13 03
0x72 (114): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 93 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x73 (115): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x74 (116): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x75 (117): 1c 0d 19 0e 03 1c 1d 18 0c 10 18 1a 1a 08 0b 08 15 04 06 1a 1d 14 0b 17 00 1b 1b 1e 09 17 10 06
0x76 (118): 18 06 19 09 01 11 11 1d 08 0f 10 1c 10 1e 0f 05 0e 0e 04 13 0a 11 1c 11 0f 0e 06 17 12 02 06 0c
0x77 (119): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x78 (120): 1c 15 15 1c 11 06 18 09 07 00 17 0b 05 18 17 08 01 01 02 14 00 13 10 19 05 11 01 13 05 06 1e 19
0x79 (121): 19 07 04 19 07 14 14 06 01 14 19 02 15 0d 14 0f 14 01 06 04 13 0c 0a 15 13 03 12 1e 07 13 11 02
0x7a (122): 04 00 06 01 1a 13 03 1d 00 03 02 0d 1a 11 0f 14 0c 16 04 18 05 0b 03 17 1a 09 10 17 13 14 00 1c
0x7b (123): 12 17 1e 06 1c 1d 02 0a 1a 1c 02 02 11 06 07 1a 0d 1c 1a 1a 09 12 0b 02 16 09 16 08 03 1a 1b 17
0x7c (124): 13 1a 09 1d 1a 17 10 08 0c 0a 04 0f 13 13 04 04 00 14 15 1b 10 14 05 11 03 1a 05 18 14 04 1e 0b
0x7d (125): 12 1c 06 0e 0b 12 0d 0f 04 19 17 07 09 1c 07 10 04 10 07 0b 0b 1a 09 04 09 1e 1e 06 05 1d 05 01
0x7e (126): 16 0a 03 1a 1d 06 0f 0c 16 0f 0a 19 07 04 00 02 0c 1d 0c 1e 1c 19 1d 15 0c 0b 18 03 17 0f 15 1c
0x7f (127): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

50. When a user program attempts to access virtual address **0x6da9**, what will be the first page accessed?
    A. 9
    B. 13
    C. 27
    D. 84
    E. Error or None of the above

Option D. First consult the Page Directory on page 84.

51. What index of the page directory will be accessed first (hexadecimal)?
    A. 0x09
    B. 0xd1
    C. 0xd0
    D. 0x1b
    E. Error or None of the above

Option D. VA 0x6da9 with top 5 bits the PD offset (11011 - 27 - 0x1b)

52. What will be the second page accessed (decimal)?
    A. 13
    B. 80
    C. 81
    D. 84
    E. Error or None of the above

Option C. PDE content 0xd1 (valid 1, pfn 0x51 [decimal 81])

53. What are the contents of the corresponding PTE that will be read (hexadecimal)?
    A. 0x0d
    B. 0xd3
    C. 0x53
    D. 0xa7
    E. Error or None of the above

Option B. Page 81 offset of 13 – contents 0xd3

54. What is the final physical address for this virtual address (hexadecimal)?
    A. 0xa69
    B. 0x0db5
    C. 0xd05
    D. 0xcd9
    E. Error or None of the above

Option A. 0xd3 (valid 1, pfn 0x53 [decimal 83]) – Translates to 0xa69

Beyond Physical Memory

Multiple Choice

55. A system has 3 pages worth of empty physical memory for a process that begins to run and the process accesses its pages in the following order: 1, 1, 5, 2, 5, 3, 1, 4, 4, 3, 5, 3.

    With a LRU replacement policy, what will be the page hitrate?
    A. 8 / 12
    B. 7 / 12
    C. 6 / 12
    D. 5 / 12
    E. None of the Above or Not enough information

Option D. Here is the order pages are accessed with memory content after access in LRU order, underlined items are hits:
1(1), <u>1(1)</u>, 5(5,1), 2(2,5,1), <u>5(5,2,1)</u>, 3(3,5,2), 1(1,3,5), 4(4,1,3), <u>4(4,1,3)</u>, <u>3(3,4,1)</u>, 5(5,3,4), <u>3(3,5,4)</u>

56. With the optimal replacement policy, how many page faults will occur?
    A. 4
    B. 5
    C. 6
    D. 7
    E. None of the above or not enough information

Option B. Underlined items are where a page fault occurred. <u>1</u> 1 <u>5</u> <u>2</u> 5 <u>3(remove 2)</u> 1 <u>4(remove 2)</u> 4 3 5 3.

Projects

Multiple choice

The portion of trap.h and syscall.h files for xv6 are shown in the table below.

trap.h	syscall.h

`// x86 trap and interrupt constants.`
 	
`...`
`#define T_ALIGN         17      // aligment check`
`#define T_MCHK          18      // machine check`
`#define T_SIMDERR       19      // SIMD floating point error`
 	
`// These are arbitrarily chosen, but with care not to overlap`
`// processor defined exceptions or interrupt`
	`// System call numbers`
	`...`
	`#define SYS_read    5`
	`#define SYS_kill    6`
	`#define SYS_exec    7`
	`#define SYS_fstat   8`
	`#define SYS_chdir   9`
	`#define SYS_dup    10`
	`#define SYS_getpid 11`
	`#define SYS_sbrk   12`
	`#define SYS_sleep  13`

```
vectors.
#define T_SYSCALL       64      // system call
#define T_DEFAULT      500      // catchall

#define T_IRQ0          32      // IRQ 0
corresponds to int T_IRQ

...
```

```
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
...
```

57. In order to execute a `getpid` system call, what interrupt should a process execute?
    A. 5
    B. 11
    C. 32
    D. 64 - system call interrupt
    E. None of the above

58. What does the `-Wall` flag for gcc do?
    A. It enables a broad set of warning messages
    B. It enables debugging with gdb
    C. It turns on compile time optimizations
    D. It specifies the executable name
    E. None of the above

Take a look at the following portion of a Makefile:

```
CC = gcc
CFLAGS-common = -std=gnu18 -Wall -Wextra -Werror -pedantic
CFLAGS = $(CFLAGS-common) -O2
CFLAGS-dbg = $(CFLAGS-common) -Og -ggdb
TARGET = wsh
SRC = $(TARGET).c

all: $(TARGET) $(TARGET)-dbg $(TARGET)-asan

$(TARGET): $(SRC) $(TARGET).h
        $(CC) $(CFLAGS) $< -o $@
```

59. What does the `$@` mean?
    A. It is the file name of the target rule
    B. It is the first prerequisite
    C. It is a phony target
    D. It is the list of prerequisites
    E. None of the above

60. In C, when redirecting output from a command, before calling `execv()` on the command, you must first
    A. **Open a file for writing and get its file descriptor, fd, then call `dup2(fd, STDOUT_FILENO)`**

B. Open a file for writing and get its file descriptor, fd, then call
   `dup(fd, STDOUT_FILENO)`
C. Open a file for reading and get its file descriptor, fd, then call
   `dup2(fd, STDIN_FILENO)`
D. Open a file for reading and gets its file descriptor, fd, then call
   `dup(fd, STDIN_FILENO)`
E. None of the above
   THIS PAGE INTENTIONALLY LEFT BLANK