Question **1**

Partially correct

Mark 1.40 out of 2.00

Flag question

Match the elements of C program to their place in memory

Global variables	Data	✔
Code of main()	Code	✔
#define MACROS	Code	✖
Global Static variables	Data	✔
Function code	Stack	✖
Local Static variables	Data	✔
Arguments	Stack	✔
Local Variables	Stack	✔
Malloced Memory	Heap	✔
#include files	Code	✖

The correct answer is: Global variables → Data, Code of main() → Code, #define MACROS → No Memory needed, Global Static variables → Data, Function code → Code, Local Static variables → Data, Arguments → Stack, Local Variables → Stack, Malloced Memory → Heap, #include files → No memory needed

Question **2**

Partially correct

Mark 0.71 out of 1.00

Flag question

Mark the statements as True/False w.r.t. the basic concepts of memory management.

True	False		
◉✖	○☑	The kernel refers to the page table for converting each virtual address to physical address.	✖
◉✖	○☑	When a process is executing, each virtual address is converted into physical address by the kernel directly.	✖
◉☑	○✖	The kernel ensures that the MMU is setup before scheduling a process and then the CPU/MMU ensures that the address translation takes place.	✔
○✖	◉☑	The compiler generates the address references for code/data/stack/heap in the executable file as per the memory management schema chosen by the compiler itself, and then the kernel ensures that program is executed with this schema.	✔

		then generates the machine code file.	
◉☑	○✖	When a process is executing, each virtual address is converted into physical address by the CPU hardware directly.	✔
◉☑	○✖	The compiler generates address references for code/data/stack/heap in the executable file, depending on the MM architecture provided by CPU and kernel.	✔

The kernel refers to the page table for converting each virtual address to physical address.: False
When a process is executing, each virtual address is converted into physical address by the kernel directly.: False
The kernel ensures that the MMU is setup before scheduling a process and then the CPU/MMU ensures that the address translation takes place.: True
The compiler generates the address references for code/data/stack/heap in the executable file as per the memory management schema chosen by the compiler itself, and then the kernel ensures that program is executed with this schema.: False
The compiler interacts with the kernel continuously while compiling a program and obtains the correct set of memory addresses for code/stack/heap/data and then generates the machine code file.: False
When a process is executing, each virtual address is converted into physical address by the CPU hardware directly.: True
The compiler generates address references for code/data/stack/heap in the executable file, depending on the MM architecture provided by CPU and kernel.: True

Question **3**

Correct

Mark 1.00 out of 1.00

Flag question

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 **4**

Correct

Mark 1.00 out of 1.00

Flag question

Select the order in which the various stages of a compiler execute.

Loading: does not exist ✔

Linking: 4 ✔

Pre-processing: 1 ✔

Syntatical Analysis: 2 ✔

Intermediate code generation: 3 ✔

The correct answer is: Loading → does not exist, Linking → 4, Pre-processing → 1, Syntatical Analysis → 2, Intermediate code generation → 3

Question 5

Partially correct

Mark 0.80 out of 1.00

Remove flag

Select all the correct statements about zombie processes

Select one or more:

- [ ] a. A process becomes zombie when it's parent finishes
- [ ] b. A zombie process remains zombie forever, as there is no way to clean it up
- [x] c. A process becomes zombie when it finishes, and remains zombie until parent calls wait() on it ✔
- [x] d. A zombie process occupies space in OS data structures ✔
- [ ] e. If the parent of a process finishes, before the process itself, then after finishing the process is typically attached to 'init' as parent
- [ ] f. Zombie processes are harmless even if OS is up for long time
- [x] g. A process can become zombie if it finishes, but the parent has finished before it ✔
- [x] h. init() typically keeps calling wait() for zombie processes to get cleaned up ✔

The correct answers are: A process becomes zombie when it finishes, and remains zombie until parent calls wait() on it, A process can become zombie if it finishes, but the parent has finished before it, A zombie process occupies space in OS data structures, If the parent of a process finishes, before the process itself, then after finishing the process is typically attached to 'init' as parent, init() typically keeps calling wait() for zombie processes to get cleaned up

Question 6

Correct

Mark 1.00 out of 1.00

Flag question

A process blocks itself means

- ( ) a. The application code calls the scheduler
- ( ) b. The kernel code of system call calls scheduler
- ( ) c. The kernel code of an interrupt handler, moves the process to a waiting queue and calls scheduler
- (•) d. The kernel code of system call, called by the process, moves the process to a waiting queue and calls scheduler ✔

The correct answer is: The kernel code of system call, called by the process, moves the process to a waiting queue and calls scheduler

Question 7

Correct

Mark 1.00 out of 1.00

Remove flag

which of the following is not a difference between real mode and protected mode

- ( ) a. in real mode the addressable memory is less than in protected mode
- ( ) b. in real mode the segment is multiplied by 16, in protected mode segment is used as index in GDT
- ( ) c. processor starts in real mode
- ( ) d. in real mode general purpose registers are 16 bit, in protected mode they are 32 bit
- (•) e. in real mode the addressable memory is more than in protected mode ✔

The correct answer is: in real mode the addressable memory is more than in protected mode

Question 8

Correct

Mark 1.00 out of 1.00

Flag question

Which of the following are NOT a part of job of a typical compiler?

- [ ] a. Check the program for syntactical errors
- [ ] b. Convert high level langauge code to machine code
- [x] c. Suggest alternative pieces of code that can be written ✔
- [ ] d. Invoke the linker to link the function calls with their code, extern globals with their declaration
- [x] e. Check the program for logical errors ✔
- [ ] f. Process the # directives in a C program

The correct answers are: Check the program for logical errors, Suggest alternative pieces of code that can be written

Question 9

Correct

Mark 1.00 out of 1.00

Flag question

Predict the output of the program given here.

Assume that there is no mixing of printf output on screen if two of them run concurrently.

In the answer replace a new line by a single space.

For example::

```
good
output
```

should be written as `good output`

```
--


int main() {
  int pid;
  printf("hi\n");
  pid = fork();
  if(pid == 0) {
    exit(0);
  }
  printf("bye\n");
  fork();
  printf("ok\n");
}
```

```
  printf("hi\n");
  pid = fork();
  if(pid == 0) {
    exit(0);
  }
  printf("bye\n");
  fork();
  printf("ok\n");
}
```

Answer: hi bye ok ok ✔

The correct answer is: hi bye ok ok

Question 10

Select the correct statements about paging (not demand paging) mechanism

Question 10

Partially correct

Mark 0.50 out of 1.00

Remove flag

Select the correct statements about paging (not demand paging) mechanism

Select one or more:

- [x] a. The PTBR is loaded by the OS ✔
- [x] b. OS creates the page table for every process ✔
- [x] c. Page table is accessed by the MMU as part of execution of an instruction ✔
- [x] d. An invalid entry on a page means, either it was illegal memory reference or the page was not present in memory. ✖
- [ ] e. User process can update it's own page table entries
- [ ] f. User process can update it's own PTBR
- [ ] g. An invalid entry on a page means, it was an illegal memory reference
- [ ] h. Page table is accessed by the OS as part of execuation of an instruction

The correct answers are: OS creates the page table for every process, The PTBR is loaded by the OS, Page table is accessed by the MMU as part of execution of an instruction, An invalid entry on a page means, it was an illegal memory reference

Question **11**

Correct

Mark 1.00 out of 1.00

Flag question

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)

Paging	one continuous chunk ✔
Relocation + Limit	one continuous chunk ✔
Segmentation	many continuous chunks of variable size ✔
Segmentation, then paging	many continuous chunks of variable size ✔

The correct answer is: Paging → one continuous chunk, Relocation + Limit → one continuous chunk, Segmentation → many continuous chunks of variable size, Segmentation, then paging → many continuous chunks of variable size

Question **12**

Correct

Mark 2.00 out of 2.00

Flag question

Consider the two programs given below to implement the command (ignore the fact that error checks are not done on return values of functions)

$ ls . /tmp/asdfksdf >/tmp/ddd 2>&1

Program 1

```
int main(int argc, char *argv[]) {
    int fd, n, i;
    char buf[128];

    fd = open("/tmp/ddd", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    close(1);
    dup(fd);
    close(2);
    dup(fd);
    execl("/bin/ls", "/bin/ls", ".", "/tmp/asldjfaldfs", NULL);
}
```

Program 2

```
int main(int argc, char *argv[]) {
    int fd, n, i;
    char buf[128];

    close(1);
    fd = open("/tmp/ddd", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    close(2);
```

```
    execl("/bin/ls", "/bin/ls", ".", "/tmp/asldjfaldfs", NULL);
}
```

Select all the correct statements about the programs

Select one or more:

- [ ] a. Both program 1 and 2 are incorrect
- [ ] b. Program 2 makes sure that there is one file offset used for '2' and '1'
- [ ] c. Program 2 ensures 2>&1 and does not ensure > /tmp/ddd
- [ ] d. Program 1 is correct for > /tmp/ddd but not for 2>&1
- [ ] e. Program 1 ensures 2>&1 and does not ensure > /tmp/ddd
- [ ] f. Program 2 does 1>&2
- [ ] g. Program 2 is correct for > /tmp/ddd but not for 2>&1
- [x] h. Program 1 makes sure that there is one file offset used for '2' and '1' ✔
- [ ] i. Both programs are correct
- [x] j. Only Program 1 is correct ✔
- [ ] k. Program 1 does 1>&2
- [ ] l. Only Program 2 is correct

The correct answers are: Only Program 1 is correct, Program 1 makes sure that there is one file offset used for '2' and '1'

Question **13**

Partially correct

Mark 0.75 out of 1.00

Flag question

Select the sequence of events that are NOT possible, assuming an interruptible kernel code

Select one or more:

- [x] a. ✔
  P1 running
  P1 makes sytem call
  Scheduler
  P2 running
  P2 makes sytem call and blocks
  Scheduler
  P1 running again
- [x] b. P1 running ✔
  P1 makes sytem call and blocks
  Scheduler
  P2 running
  P2 makes sytem call and blocks
  Scheduler
  P1 running again
- [ ] c. P1 running
  P1 makes sytem call and blocks
  Scheduler
  P2 running
  P2 makes sytem call and blocks
  Scheduler
  P3 running

keyboard hardware interrupt
keyboard interrupt handler running
interrupt handler returns
P1 running
P1 makes sytem call
system call returns
P1 running
timer interrupt
scheduler
P2 running

The correct answers are: P1 running
P1 makes sytem call and blocks
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again,
P1 running
P1 makes sytem call
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again

Question **14**
Correct
Mark 1.00 out of 1.00
Flag question

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

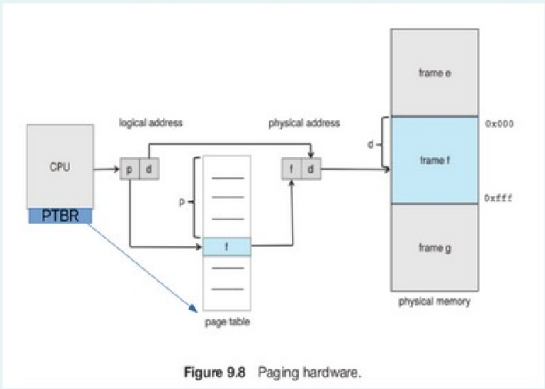

Figure 9.8 Paging hardware.

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

True	False		
○ ✗	◉ ☑	The locating of the page table using PTBR also involves paging translation	✔
○ ✗	◉ ☑	Size of page table is always determined by the size of RAM	✔

◉☑	○✖	The page table is indexed using page number	✔
◉☑	○✖	The physical address may not be of the same size (in bits) as the logical address	✔
○✖	◉☑	The page table is indexed using frame number	✔
◉☑	○✖	Maximum Size of page table is determined by number of bits used for page number	✔
◉☑	○✖	The PTBR is present in the CPU as a register	✔

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

Question **15**

Correct

Mark 1.00 out of 1.00

Flag question

Consider the following code and MAP the file to which each fd points at the end of the code. Assume that files/folders exist when needed with proper permissions and open() calls work.

```
int main(int argc, char *argv[]) {
   int fd1, fd2 = 1, fd3 = 1, fd4 = 1;

   fd1 = open("/tmp/1", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);
   fd2 = open("/tmp/2", O_RDDONLY);
   fd3 = open("/tmp/3", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);
   close(0);
   close(1);
   dup(fd2);
   dup(fd3);
   close(fd3);
   dup2(fd2, fd4);
   printf("%d %d %d %d\n", fd1, fd2, fd3, fd4);
   return 0;
}
```

0 /tmp/2 ✔

1 /tmp/3 ✔

fd2 /tmp/2 ✔

2 stderr ✔

```
    close(1);
    dup(fd2);
    dup(fd3);
    close(fd3);
    dup2(fd2, fd4);
    printf("%d %d %d %d\n", fd1, fd2, fd3, fd4);
    return 0;
}
```

0	/tmp/2 ✔
1	/tmp/3 ✔
fd2	/tmp/2 ✔
2	stderr ✔
fd3	closed ✔
fd1	/tmp/1 ✔
fd4	/tmp/2 ✔

The correct answer is: 0 → /tmp/2, 1 → /tmp/3, fd2 → /tmp/2, 2 → stderr, fd3 → closed, fd1 → /tmp/1, fd4 → /tmp/2

Question **16**

Correct

Mark 1.00 out of 1.00

Remove flag

Select all the correct statements about MMU and it's functionality (on a non-demand paged system)

Select one or more:

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

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 17

Partially correct

Mark 1.43 out of 2.00

Remove flag

Order the events that occur on a timer interrupt:

Execute the code of the new process	7 ✔
Jump to a code pointed by IDT	1 ✘
Save the context of the currently running process	3 ✔
Change to kernel stack of currently running process	2 ✘
Select another process for execution	5 ✔
Jump to scheduler code	4 ✔
Set the context of the new process	6 ✔

The correct answer is: Execute the code of the new process → 7, Jump to a code pointed by IDT → 2, Save the context of the currently running process → 3, Change to kernel stack of currently running process → 1, Select another process for execution → 5, Jump to scheduler code → 4, Set the context of the new process → 6