Dashboa... / My cours... / Computer Engineering &... / CEIT-even-sem-22-... / OS-even-sem-22-... / Theory : Quizz... / Quiz-1(24 Feb 202...

Started on	Friday, 24 February 2023, 2:44 PM
State	Finished
Completed on	Friday, 24 February 2023, 4:20 PM
Time taken	1 hour 36 mins
Grade	**8.76** out of 10.00 (**87.57**%)

Question **1**

Partially correct

Mark 0.42 out of 0.50

Which of the following instructions should be privileged?

Select one or more:

☑ a. Set value of timer. ✓

☑ b. Access I/O device. ✓

☑ c. Turn off interrupts. ✓

☑ d. Modify entries in device-status table ✓

☐ e. Access a general purpose register

☐ f. Read the clock.

☐ g. Set value of a memory location

☑ h. Switch from user to kernel mode. ✓ This instruction (like INT) is itself privileged - and that is why it not only changes the mode, but also ensures a jump to an ISR (kernel code)

☐ i. Access memory management unit of the processor

Your answer is partially correct.

You have correctly selected 5.
The correct answers are: Set value of timer., Access memory management unit of the processor, Turn off interrupts., Modify entries in device-status table, Access I/O device., Switch from user to kernel mode.

Question **2**

Correct

Mark 0.50 out of 0.50

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)

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

Your answer is correct.

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

Question **3**

Correct

Mark 0.50 out of 0.50

How does the compiler calculate addresses for the different parts of a C program, when paging is used?

Global variables	Immediately after the text	✓
Text	starting with 0	✓
Static variables	Immediately after the text, along with globals	✓
typedef	No memory allocated, as they are not variables, but only conceptual definition of a type	✓
Local variables	An offset with respect to stack pointer (esp)	✓
#define	No memory allocated, they are handled by pre-processor	✓
#include files	No memory allocated for the file, but if it contains variables, then variables may be allocated memory	✓
malloced memory	Heap (handled by the malloc-free library, using OS's system calls)	✓

Your answer is correct.

The correct answer is: Global variables → Immediately after the text, Text → starting with 0, Static variables → Immediately after the text, along with globals, typedef → No memory allocated, as they are not variables, but only conceptual definition of a type, Local variables → An offset with respect to stack pointer (esp), #define → No memory allocated, they are handled by pre-processor, #include files → No memory allocated for the file, but if it contains variables, then variables may be allocated memory, malloced memory → Heap (handled by the malloc-free library, using OS's system calls)

Question **4**

Correct

Mark 0.50 out of 0.50

Map the block allocation scheme with the problem it suffers from

(Match pairs 1-1, match a scheme with the problem that it suffers from relatively the most, compared to others)

Indexed Allocation	Overhead of reading metadata blocks	✓
Linked allocation	Too many seeks	✓
Continuous allocation	need for compaction	✓

Your answer is correct.

The correct answer is: Indexed Allocation → Overhead of reading metadata blocks, Linked allocation → Too many seeks, Continuous allocation → need for compaction

Question **5**

Partially correct

Mark 0.40 out of 0.50

Mark the statements about named and un-named pipes as True or False

True	False		
● ☑	○ ✗	Both types of pipes are an extension of the idea of "message passing".	✓
○ ✗	● ☑	A named pipe has a name decided by the kernel.	✓
● ☑	○ ✗	Un-named pipes are inherited by a child process from parent.	✓
● ✗	○ ☑	The buffers for named-pipe are in process-memory while the buffers for the un-named pipe are in kernel memory.	✗
● ☑	○ ✗	Named pipes can exist beyond the life-time of processes using them.	✓
○ ✗	● ☑	Named pipes can be used for communication between only "related" processes.	✓
○ ✗	● ☑	The pipe() system call can be used to create either a named or un-named pipe.	✓
● ☑	○ ✗	Un-named pipes can be used for communication between only "related" processes, if the common ancestor created it.	✓
○ ☑	● ✗	Named pipe exists as a file	✗
● ☑	○ ✗	Both types of pipes provide FIFO communication.	✓

Both types of pipes are an extension of the idea of "message passing".: True
A named pipe has a name decided by the kernel.: False
Un-named pipes are inherited by a child process from parent.: True
The buffers for named-pipe are in process-memory while the buffers for the un-named pipe are in kernel memory.: False
Named pipes can exist beyond the life-time of processes using them.: True
Named pipes can be used for communication between only "related" processes.: False
The pipe() system call can be used to create either a named or un-named pipe.: False
Un-named pipes can be used for communication between only "related" processes, if the common ancestor created it.: True
Named pipe exists as a file: True
Both types of pipes provide FIFO communication.: True

Question **6**

Incorrect

Mark 0.00 out of 0.50

Doing a lookup on the pathname /a/b/b/c/d for opening the file "d" requires reading [5] ✗ no. of inodes. Assume that there are no hard/soft links on the path.

Write the answer as a number.

The correct answer is: 6

Question **7**

Correct

Mark 0.50 out of 0.50

What is meant by formatting a disk/partition?

- ● a. creating layout of empty directory tree/graph data structure ✓
- ○ b. erasing all data on the disk/partition
- ○ c. storing all the necessary programs on the disk/partition
- ○ d. writing zeroes on all sectors

The correct answer is: creating layout of empty directory tree/graph data structure

Question **8**

Correct

Mark 0.50 out of 0.50

Which of the following parts of a C program do not have any corresponding machine code ?

- □ a. local variable declaration
- ☑ b. #directives ✓
- □ c. pointer dereference
- ☑ d. typedefs ✓
- □ e. function calls
- □ f. expressions
- ☑ g. global variables ✓

Your answer is correct.

The correct answers are: #directives, typedefs, global variables

Question **9**

Partially correct

Mark 0.36 out of 0.50

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

True	False		
○ ✗	● ☑	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.	✓
○ ☑	● ✗	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.	✗
○ ✗	● ☑	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 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.	✓

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
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
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 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

Question **10**

Correct

Mark 0.50 out of 0.50

Mark statements True/False w.r.t. change of states of a process. Note that a statement is true only if the claim and argument both are true.

Reference: The process state diagram (and your understanding of how kernel code works). Note - the diagram does not show zombie state!

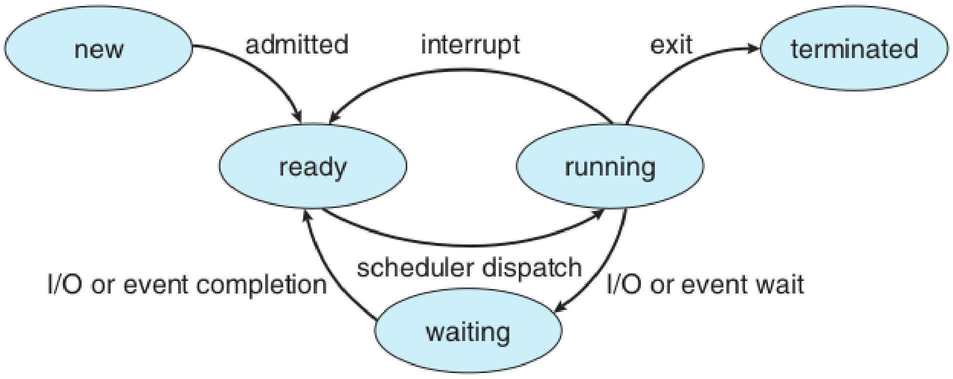

Figure 3.2 Diagram of process state.

True	**False**		
● ☑	○ ✗	Every forked process has to go through ZOMBIE state, at least for a small duration.	✓
● ☑	○ ✗	A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet	✓
○ ✗	● ☑	A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first	✓
● ☑	○ ✗	Only a process in READY state is considered by scheduler	✓
○ ✗	● ☑	A process in READY state can not go to WAITING state because the resource on which it will WAIT will not be in use when process is in READY state.	✓

Every forked process has to go through ZOMBIE state, at least for a small duration.: True
A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet: True
A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first: False
Only a process in READY state is considered by scheduler: True
A process in READY state can not go to WAITING state because the resource on which it will WAIT will not be in use when process is in READY state.: False

Question **11**

Correct

Mark 0.50 out of 0.50

Map each signal with it's meaning

SIGCHLD	Child Stopped or Terminated	✓
SIGALRM	Timer Signal from alarm()	✓
SIGUSR1	User Defined Signal	✓
SIGSEGV	Invalid Memory Reference	✓
SIGPIPE	Broken Pipe	✓

The correct answer is: SIGCHLD → Child Stopped or Terminated, SIGALRM → Timer Signal from alarm(), SIGUSR1 → User Defined Signal, SIGSEGV → Invalid Memory Reference, SIGPIPE → Broken Pipe

Question **12**

Correct

Mark 0.50 out of 0.50

You must have seen the error message "Segmentation fault, core dumped" very often.

With respect to this error message, mark the statements as True/False.

True	False			
● ☑	○ ✗	On Linux, the process was sent a SIGSEGV signal and the default handler for the signal is "Term", so the process is terminated.	✓	
○ ✗	● ☑	The term "core" refers to the core code of the kernel.	✓	core means memory, all memory for the process.
● ☑	○ ✗	The process has definitely performed illegal memory access.	✓	
● ☑	○ ✗	The image of the process is stored in a file called "core", if the ulimit allows so.	✓	see ulimit -a
○ ✗	● ☑	The illegal memory access was detected by the kernel and the process was punished by kernel.	✓	"detection" is done by CPU, not kernel.
○ ✗	● ☑	On Linux, the message is printed only because the memory management scheme is segmentation	✓	No, it's just a term used, even if paging is used for memory management.
● ☑	○ ✗	The core file can be analysed later using a debugger, to determine what went wrong.	✓	use gdb ./core ./executable-filename

On Linux, the process was sent a SIGSEGV signal and the default handler for the signal is "Term", so the process is terminated.: True
The term "core" refers to the core code of the kernel.: False
The process has definitely performed illegal memory access.: True
The image of the process is stored in a file called "core", if the ulimit allows so.: True
The illegal memory access was detected by the kernel and the process was punished by kernel.: False
On Linux, the message is printed only because the memory management scheme is segmentation: False
The core file can be analysed later using a debugger, to determine what went wrong.: True

Question **13**

Partially correct

Mark 0.33 out of 0.50

Mark the statements about device drivers by marking as True or False.

True	False		
● ✗	○ ✓	Different devices of the same type (e.g. 2 IDE hard disks) must need different device drivers.	✗
● ✓	○ ✗	It's possible that a particular hardware has multiple device drivers available for it.	✓
○ ✓	● ✗	Device driver is part of OS code	✗
● ✓	○ ✗	Device driver is an intermediary between the hardware controller and OS	✓
● ✓	○ ✗	Writing a device driver mandatorily demands reading the technical documentation about the hardware.	✓
○ ✗	● ✓	Device driver is part of hardware	✓

Different devices of the same type (e.g. 2 IDE hard disks) must need different device drivers.: False
It's possible that a particular hardware has multiple device drivers available for it.: True
Device driver is part of OS code: True
Device driver is an intermediary between the hardware controller and OS: True
Writing a device driver mandatorily demands reading the technical documentation about the hardware.: True
Device driver is part of hardware: False

Question **14**

Correct

Mark 0.50 out of 0.50

How does the distinction between kernel mode and user mode function as a rudimentary form of protection (security) ?

Select one:

○ a. It prohibits invocation of kernel code completely, if a user program is running

○ b. It disallows hardware interrupts when a process is running

● c. It prohibits a user mode process from running privileged instructions ✓

○ d. It prohibits one process from accessing other process's memory

Your answer is correct.

The correct answer is: It prohibits a user mode process from running privileged instructions

Question **15**

Correct

Mark 0.50 out of 0.50

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

Select all the correct statements about the programs

Select one or more:

☐ a. Program 1 is correct for > /tmp/ddd but not for 2>&1

☐ b. Program 1 ensures 2>&1 and does not ensure > /tmp/ddd

☑ c. Program 1 makes sure that there is one file offset used for '2' and '1' ✓

☐ d. Both programs are correct

☐ e. Program 1 does 1>&2

☐ f. Program 2 ensures 2>&1 and does not ensure > /tmp/ddd

☐ g. Program 2 makes sure that there is one file offset used for '2' and '1'

☐ h. Both program 1 and 2 are incorrect

☑ i. Only Program 1 is correct ✓

☐ j. Program 2 is correct for > /tmp/ddd but not for 2>&1

☐ k. Program 2 does 1>&2

☐ l. Only Program 2 is correct

Your answer 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 **16**

Correct

Mark 0.50 out of 0.50

Predict the output of the program given here.

Assume that all the path names for the programs are correct. For example "/usr/bin/echo" will actually run echo command.

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`

```
--

main() {
    int i;
    i = fork();
    if(i == 0)
        execl("/usr/bin/echo", "/usr/bin/echo", "hi", 0);
    else
        wait(0);
    fork();
    execl("/usr/bin/echo", "/usr/bin/echo", "one", 0);
}
```

Answer: hi one one ✓

The correct answer is: hi one one

Question **17**

Correct

Mark 0.50 out of 0.50

Following code claims to implement the command

/bin/ls -l | /usr/bin/head -3 | /usr/bin/tail -1

Fill in the blanks to make the code work.

Note: Do not include space in writing any option. x[1][2] should be written without any space, and so is the case with [1] or [2]. Pay attention to exact syntax and do not write any extra character like ';' or = etc.

```
int main(int argc, char *argv[]) {
  int pid1, pid2;
  int pfd[
2
✓ ][2];

  pipe(
pfd[0]
✓ );
  pid1 =
fork()
✓ ;
  if(pid1 != 0) {
    close(pfd[0]
[0]
✓ );
    close(
1
✓ );
    dup(
pfd[0][1]
✓ );
    execl("/bin/ls", "/bin/ls", "
-l
✓ ", NULL);
  }
  pipe(
pfd[1]
✓ );

pid2
✓  = fork();
  if(pid2 == 0) {
    close(
pfd[0][1])
✓ ;
    close(0);
    dup(
pfd[0][0]
✓ );
      close(pfd[1]
[0]
✓ );
      close(
1
✓ );
      dup(
pfd[1][1]
✓ );
      execl("/usr/bin/head", "/usr/bin/head", "
-3
✓ ", NULL);
   } else {
      close(pfd
[1][1]
✓ );
      close(
0
✓ );
      dup(
pfd[1][0]
✓ );
      close(pfd
[0][0]
✓ );
      execl("/usr/bin/tail", "/usr/bin/tail", "
-1
✓ ", NULL);
  }
}
```

Question **18**

Correct

Mark 0.50 out of 0.50

Match the elements of C program to their place in memory

Function code	Code	✓
Local Static variables	Data	✓
#define MACROS	No Memory needed	✓
Malloced Memory	Heap	✓
Global Static variables	Data	✓
Code of main()	Code	✓
Arguments	Stack	✓
Local Variables	Stack	✓
Global variables	Data	✓
#include files	No memory needed	✓

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

Question **19**

Correct

Mark 0.50 out of 0.50

Select Yes if the mentioned element should be a part of PCB

Select No otherwise.

Yes	No		
● ✓	○ ✗	Process context	✓
● ✓	○ ✗	List of opened files	✓
● ✓	○ ✗	EIP at the time of context switch	✓
● ✓	○ ✗	Process state	✓
● ✓	○ ✗	Memory management information about that process	✓
○ ✗	● ✓	PID of Init	✓
● ✓	○ ✗	PID	✓
○ ✗	● ✓	Pointer to IDT	✓
● ✓	○ ✗	Pointer to the parent process	✓
○ ✗	● ✓	Function pointers to all system calls	✓

Process context: Yes
List of opened files: Yes
EIP at the time of context switch: Yes
Process state: Yes
Memory management information about that process: Yes
PID of Init: No
PID: Yes
Pointer to IDT: No
Pointer to the parent process: Yes
Function pointers to all system calls: No

Question **20**

Partially correct

Mark 0.25 out of 0.50

Select all the blocks that may need to be written back to disk (if updated, of-course), as "Yes", when an operation of deleting a file is carried out on ext2 file system.

An option has to be correct entirely to be marked "Yes"

Data blocks of the file	No	✓
Block bitmap(s) for all the blocks of the file	Yes	✓
Possibly one block bitmap corresponding to the parent directory	Yes	✓
Superblock	No	✗
One or multiple data blocks of the parent directory	Yes	✗
One or more data bitmap blocks for the parent directory	Yes	✗

Your answer is partially correct.

only one data block of parent directory. multiple blocks not possible. an entry is always contained within one single block

You have correctly selected 3.
The correct answer is: Data blocks of the file → No, Block bitmap(s) for all the blocks of the file → Yes, Possibly one block bitmap corresponding to the parent directory → Yes, Superblock → Yes, One or multiple data blocks of the parent directory → No, One or more data bitmap blocks for the parent directory → No

◄ Quiz-1 Preparation questions

Jump to...

Quiz - 2 (17 March 2023) ►