Das… / My … / Computer E… / CEIT-eve… / OS-even… / Theory: ra… / Random Quiz - 3 (processes, memory management, event dri…

Started on	Thursday, 2 February 2023, 9:18 PM
State	Finished
Completed on	Thursday, 2 February 2023, 11:00 PM
Time taken	1 hour 41 mins
Grade	**13.74** out of 20.00 (**68.68**%)

Question **1**

Complete

Mark 0.25 out of 1.00

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

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

Question **2**

Complete

Mark 0.30 out of 1.00

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. Both programs are correct

□ b. Program 1 does 1>&2

□ c. Only Program 1 is correct

□ d. Both program 1 and 2 are incorrect

□ e. Program 2 is correct for > /tmp/ddd but not for 2>&1

□ f. Program 2 does 1>&2

□ g. Only Program 2 is correct

□ h. Program 2 ensures 2>&1 and does not ensure > /tmp/ddd

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

□ j. Program 1 ensures 2>&1 and does not ensure > /tmp/ddd

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

□ l. Program 1 is correct for > /tmp/ddd but not for 2>&1

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 **3**

Complete

Mark 1.00 out of 1.00

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**

Complete

Mark 0.50 out of 1.00

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

☑ a. Process the # directives in a C program

☑ b. Check the program for logical errors

□ c. Convert high level langauge code to machine code

□ d. Check the program for syntactical errors

☑ e. Suggest alternative pieces of code that can be written

☑ f. Invoke the linker to link the function calls with their code, extern globals with their declaration

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

Question **5**

Complete

Mark 0.00 out of 1.00

Select all the correct statements about named pipes and ordinary(unnamed) pipe

Select one or more:

☑ a. a named pipe exists as a file on the file system

☑ b. named pipe exists even if the processes using it do exit()

☑ c. named pipe can be used between any processes

☑ d. named pipes can be used between multiple processes but ordinary pipes can not be used

☑ e. ordinary pipe can only be used between related processes

☑ f. both named and unnamed pipes require some kind of agreed protocol to be effectively used among multiple processes

☑ g. named pipes are more efficient than ordinary pipes

The correct answers are: ordinary pipe can only be used between related processes, named pipe can be used between any processes, a named pipe exists as a file on the file system, named pipe exists even if the processes using it do exit(), both named and unnamed pipes require some kind of agreed protocol to be effectively used among multiple processes

Question **6**

Complete

Mark 1.00 out of 1.00

A process blocks itself means

● a. The kernel code of system call, called by the process, moves the process to a waiting queue and calls scheduler

○ b. The application code calls the 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 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**

Complete

Mark 0.50 out of 1.00

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

Select one or more:

□ a. P1 running
P1 makes system call
timer interrupt
Scheduler
P2 running
timer interrupt
Scheuler
P1 running
P1's system call return

□ b. P1 running
P1 makes system call
system call returns
P1 running
timer interrupt
Scheduler running
P2 running

□ c. P1 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

□ d.
P1 running
P1 makes sytem call
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again

☑ e. P1 running
P1 makes sytem call and blocks
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again

□ f. P1 running
P1 makes sytem call and blocks
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P3 running
Hardware interrupt
Interrupt unblocks P1
Interrupt returns
P3 running

Timer interrupt
Scheduler
P1 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 **8**

Complete

Mark 0.67 out of 1.00

Select the sequence of events that are NOT possible, assuming a non-interruptible kernel code

(Note: non-interruptible kernel code means, if the kernel code is executing, then interrupts will be disabled).

Note: A possible sequence may have some missing steps in between. An impossible sequence will will have n and n+1th steps such that n+1th step can not follow n'th step.

Select one or more:

☐ a. P1 running
P1 makes system call
system call returns
P1 running
timer interrupt
Scheduler running
P2 running

☐ b. P1 running
P1 makes sytem call and blocks
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P3 running
Hardware interrupt
Interrupt unblocks P1
Interrupt returns
P3 running
Timer interrupt
Scheduler
P1 running

☑ c. P1 running
P1 makes system call
timer interrupt
Scheduler
P2 running
timer interrupt
Scheuler
P1 running
P1's system call return

☐ d. P1 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

☐ e.
P1 running
P1 makes sytem call
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again

☑ f. P1 running
P1 makes sytem call and blocks
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again

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 system call
timer interrupt
Scheduler
P2 running
timer interrupt
Scheuler
P1 running
P1's system call return,
P1 running
P1 makes sytem call
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again

Question **9**

Complete

Mark 0.86 out of 1.00

Consider the following code and MAP the file to which each fd points at the end of the code.

```
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;
}
```

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

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

Question **10**

Complete

Mark 0.88 out of 1.00

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

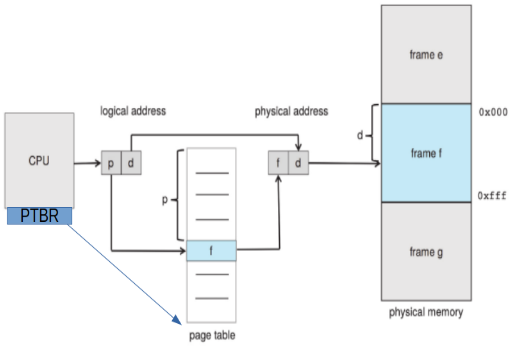

Figure 9.8 Paging hardware.

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

True	False	
●	○	The page table is indexed using page number
○	●	The page table is indexed using frame number
●	○	The physical address may not be of the same size (in bits) as the logical address
●	○	The page table is itself present in Physical memory
●	○	Size of page table is always determined by the size of RAM
●	○	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

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

Question **11**

Complete

Mark 0.57 out of 1.00

Order the events that occur on a timer interrupt:

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

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

Question **12**

Complete

Mark 4.75 out of 5.00

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 **13**

Complete

Mark 1.80 out of 2.00

Match the elements of C program to their place in memory

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

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

Question **14**

Complete

Mark 0.27 out of 1.00

Select all the correct statements about zombie processes

Select one or more:

☑ a. A zombie process occupies space in OS data structures

□ b. A zombie process remains zombie forever, as there is no way to clean it up

□ c. A process becomes zombie when it's parent finishes

□ d. If the parent of a process finishes, before the process itself, then after finishing the process is typically attached to 'init' as parent

□ e. A process can become zombie if it finishes, but the parent has finished before it

☑ f. init() typically keeps calling wait() for zombie processes to get cleaned up

☑ g. Zombie processes are harmless even if OS is up for long time

☑ h. A process becomes zombie when it finishes, and remains zombie until parent calls wait() on it

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 **15**

Complete

Mark 0.40 out of 1.00

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

Pre-processing	1
Linking	3
Syntatical Analysis	2
Loading	4
Intermediate code generation	does not exist

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

◄ Random Quiz - 2: bootloader, system calls, fork-exec, open-read-write, linux-basics, processes

Jump to...

Homework questions: Basics of MM, xv6 booting ►