Answers on Process Management

When the OS runs the same application twice, is each run treated as the same process? (multiple choice with single correct answer)

A. TRUE
B. FALSE

Answer: B (FALSE)
A process is a program in execution. Two separate executions are thus two different executions/two different processes.

What does the stack pointer point to?
(multiple choice question with SINGLE correct answer)
A. Incorrect. The largest possible address of the stack.
B. Incorrect. The middle portion of the stack.
C. Correct. The smallest possible address of the stack.
The "stack pointer" points to the smallest possible address of the stack. The stack grows downwards such that higher addresses are filled before smaller addresses. We track the bottom of the stack because whenever we make a function call or return from one, we need to either grow or shrink the stack from the location pointed to by the stack pointer.

Where are dynamically allocated variables stored?
(multiple choice question with ONE correct answer)
a. Incorrect. Stack segment
b. Correct. Heap segment
c. Incorrect. Data segment

What is the value of x at the end running the following code? What is the value of y?
(numeric answer question)

```
#include <stdlib.h>
#include <stdio.h>

int sum(int varX, int varY) {
  static int funcCounter = 0;
  funcCounter++;
  return (funcCounter + varX + varY);
}

int main (int argc, char** argv) {
  int x = 0, y = 0;
  x = sum(1,2);
  printf("%d\n",x);
  return 0;
}
```

Answer: 4.
In the function call "sum(1,2)" funcCounter increments to "1" and the return value becomes "1+1+2=4"

Answer: 9.
In the function call “sum(3,4)”, funcCounter increments from “1” to “2” and “2+3+4=9”. Note that funcCounter is a
static variable so it is stored in the Data segment at compile time and only gets defined once,

Which of the following are stored in a process control block (PCB) of a process?
(multiple choice question with multiple correct answers)
A. Stack pointer
B. Program counter
C. Register values
D. Process data files

Answer: A, B, and C.
Explanation: While the PCB of a process stores file descriptors of files in use by the process, the PCB does not store
the files themselves

Each child process can have multiple parent processes.
(multiple choice question with ONE correct answer)
a. Incorrect. True
b. Correct. False
A process is created by another process, known as the parent process, and each process is given a unique PID by the OS. Thus if two processes A and B exist and each create child processes, none of the children could be the same. Each child process has one parent process

Each parent process can have multiple children processes.
(multiple choice question with ONE correct answer)
a. Correct. True
b. Incorrect. False
Parent processes can create multiple children, which each has a unique PID, but all have the same PPID

Which of the following happens after a fork() call is invoked?
(multiple choice question with multiple correct answers)
A . Incorrect. The child process is created and is an identical clone of the parent.
On fork(), the child’s memory is a nearly an exact copy of the parent’s address space, and both have the same PC value. However, the return value of fork() is different, hence they are not identical.
B. Correct. The return value of fork() is set to 0 for the child process. To differentiate the parent and child, the return value of fork() is used. The return value will be 0 if we are in the child process and the OS-assigned PID value within the parent process.
C. Incorrect. The return value of fork() is set to 0 for the parent process. To differentiate the parent and child, the return value of fork() is used. The return value will be 0 if we are in the child process and the OS-assigned PID value within the parent process.
D. Incorrect. The return value of fork() is set to the child's pid for the child process.

To differentiate the parent and child, the return value of fork() is used. The return value will be 0 if we are in the child process and the OS-assigned PID value within the parent process.
E. Correct. The return value of fork() is set to the child's pid for the parent process
To differentiate the parent and child, the return value of fork() is used. The return value will be 0 if we are in the child process and the OS-assigned PID value within the parent process.

Answer: B, E

Which of the following are reset at an exec() system call by a child process?
(multiple choice question with multiple correct answers)
A. Correct. Child process heap
B. Incorrect. Parent process heap
C. Correct. Child process stack

exec() accepts an executable or path to an executable as an argument and replaces the existing executable with that new executable. The child process, as a result of loading a new executable, has a new memory space, meaning both the child heap and child process are reset. The parent process memory is unaffected because after the fork, the child and parent are executing separately

When a parent process dies, what is the child process called?
(multiple choice question with ONE correct answer)
A. Incorrect. Zombie process
For any process that terminates, but does not get waited on, we call this a zombie process. This is irrespective of the parent process' state.
B. Correct. Orphan process By definition, when a parent process terminates (either gracefully or ungracefully), the child process becomes an orphan process

When a child process terminates but the parent does not call wait on the child, what is the child process called?
(multiple choice question with ONE correct answer)
A. Incorrect. Init process
The Init process is the name of the single first process running on your system upon booting it up. For any process that terminates, but does not get waited on, we call this a zombie process.
B. Incorrect. Daemon process
Daemon processes are system-level processes that are started to run OS specific management tasks. The parent process of Daemon processes are init. Since these processes do not terminate as long as the machine is running, this is not the correct answer. Instead, for any process that terminates, but does not get waited on, we call this a zombie process.
C. Correct. Zombie process

For any process that terminates, but does not get waited on, we call this a zombie process

A virtual machine allows us to run multiple operating systems on a single computer. Select the best answer.
(multiple choice question with ONE correct answer)
A. Correct. True
B. Incorrect. False
A virtual machine is a software that can run an operating system within it. Multiple virtual machines can run on a single physical machine

The static global variables that reside in the data segment of a process's memory are generated at compile, not runtime. Select the best answer.
(multiple choice question with ONE correct answer)
A. Correct. True
B. Incorrect. False
This is all known and declared at compile time and put in the data segment. On the flip side, malloc calls dynamically allocate memory at runtime in the heap.

Copy-on-write is a mechanism used to reduce the overhead of forking new processes. Select the best answer.
(multiple choice question with ONE correct answer)
A. Correct. True
B. Incorrect. False
A child's memory space is nearly identical to that of its parent. In a fork() operation, if we blindly create a child that is a replica of the parent, it is a lot of wasted work especially if the child address space is going to be overwritten later on through an exec() call. A lazy copy-on-write mechanism is used instead. If the child process modifies a piece of the memory, then a new copy of it is created.

The "Root" process, init, is created when the OS is booted. Select the best answer.
(multiple choice question with ONE correct answer)
A. Correct. True
B. Incorrect. False
Init is the first process created, at the root of the process hierarchy.

After the exec() call, the child's data segment is preserved and kept similar to that of its parent. Select the best answer.
(multiple choice question with ONE correct answer)
A. Incorrect. True
B. Correct. False
After an exec() call, the child runs an entire new binary, and all its existing memory is reset. The new binary will not understand the old data segment which is replaced with a new binary for the new data segment

Consider the following program:

```
#include <stdlib.h>
#include <stdio.h>
int sum(int varX, int varY) {
  char *cArray = NULL;
```

```
  cArray = (char*) malloc (1024 * sizeof(char));
  static int funcCounter = 0;
  funcCounter++;
  free(cArray);
  return (funcCounter + varX + varY);
}

int main (int argc, char** argv) {
  int x = 0, y = 0;
  x = sum(1,2);
  y = sum(3,4);
  printf("%d %d\n", x, y);
  return 0;
}
```

Where is the cArray stored in memory? Select the best answer.
(multiple choice question with ONE correct answer)
A. Correct. Heap segment
B. Incorrect. Stack segment
C. Incorrect. Data segment
All memory allocated via malloc() calls go into the heap, even if they are called within functions.

Which of the following statements is true if free(cArray) is left out? Select the best answer.
(multiple choice question with ONE correct answer)
A. Incorrect. We will get a segmentation fault when running the program.
B. Incorrect. cArray will be removed from the heap after the sum function call returns.
C. Correct. cArray will be retained on the heap after the sum function call returns.
All malloc() allocated memory goes into the heap even if the malloc()calls are within functions. In this case, not freeing simply results in memory leaks but not segmentation faults. Since cArray is on the heap it will remain there even after the function call ends, if it is not explicitly deallocated.

Where is the variable x stored in memory? Select the best answer.
(multiple choice question with ONE correct answer)
A. Heap segment
B. Stack segment
C. Data segment
Answer: B. Any local variables in main are pushed into the stack if they are not declared as static variables.

If the variable funcCounter in the sum function is declared without the static keyword, what would be the output in
the printf() of the main function? Select the best answer.
(multiple choice question with ONE correct answer)
A. 8
B. 9

C. 7
Solution: A. Calling sum twice would not cause the funcCounter variable's incremented value to persist across function invocations.

Which of the following are part of a process' execution context?
(multiple choice question with multiple correct answers)
A. Correct. Its contents in memory
B. Correct. Its CPU register values
C. Correct. Its open files
D. Incorrect. Files that the process opened many months ago (the computer was shut down and restarted since).

Explanation: A, B, and C are part of the process execution context as presented in the slides. On the other hand, files opened months ago may not be used by the process currently, especially after the computer was rebooted. Even if the files are stored on disk, if they are not used when the application starts, even if they stay on disk, they are not part of that process's current execution context.
Refinement: if we define execution context to exclude memory context, B is the only correct answer.

Which of the following are NOT stored in a PCB?
(multiple choice question with multiple correct answers)
A. Correct. Most current stack pointer value of currently running process
B. Incorrect. Last known stack pointer of a process when it stopped running
C. Correct. Physical memory pages used by process
Explanation: A is not maintained in the PCB at all times. It is kept in the CPU, and only stored in the PCB during the next context switch. Otherwise, the costs of storing in PCB is too high. B is stored in the PCB. The PCB stores a data structure called a page table that tracks where the physical memory pages used by the process
are. But the pages themselves are not in the PCB. Hence C is wrong

Physical memory pages used by process
During context switching from one user level process to another, the kernel transitions from user to kernel mode,
and then back to user mode.
(multiple choice question with one correct answer)
A. Correct. True
The transition to kernel mode from user is required to store the current context of the executing process. The scheduler selects the next process to runs, and then loads in the new context. The OS then transitions back to user mode to run this next process. Incorrect. False

When a blocked process is unblocked, what state does it transition to?
(multiple choice question with SINGLE correct answer)
A. Correct. Ready
B. Incorrect. Running
Explanation: A process will never run immediately after unblocked, since another process

may still be running. It will always transition first to ready state and when chosen by a scheduler to run, transitions to running.

A larger time-slice will result in:
(multiple choice question with MULTIPLE correct answers)
A. Correct. More sluggish behavior for interactive applications
B. Correct. Higher throughput for the OS
Explanation: A is correct because a larger time slice means that if you do a click or keyboard entry, if the application is not currently running, it may be a while before it responds to your interactions given some other process is running. A larger time-slice reduces context switching overhead which increases throughput. Hence B is also true.

When a process calls exec, it can become blocked. [T/F]
Multiple choice question with SINGLE correct answer
A. Correct. True
B. Incorrect. False
An exec call may require reading a binary from disk, which means the process can become blocked.

When a process does a system call to read a file, what states can it end up in right after the call?
(multiple choice question with MULTIPLE correct answers)
A. Correct. Ready
B. Incorrect. Running
C. Correct. Blocked
If the call succeeds right away, the process becomes ready. If the system call needs more time to complete (e.g. the disk is still reading), the process can be blocked. Since a system call requires context switching, the OS won't allow the process to just resume running immediately. Even if the scheduler chooses that process, it will go to ready state first, and then running.

Which of the following are pushed onto the stack during a function call to remember the execution context of a process?
(multiple choice question with MULTIPLE correct answers)
a. Correct. Program counter
b. Correct. Compute registers
c. Incorrect. File descriptor table
Because we are jumping to run another function, we need to remember the execution context of where we left off previously in the main function. The program counter and computer registers are necessary for this, the file descriptor table can be accessed throughout the program.

Is the following statement true or false? Select the best response.
In a function call, there is no need to transition to kernel mode.
A. Correct. True
B. Incorrect. False
Function code is part of the user's application, and is thus accessible within user mode

Is the following statement true or false? Select the best response.
A process can jump to any location in the kernel via a system call.
A. Incorrect. True
B. Correct. False
For security reasons, it is not advisable for user processes to call into any arbitrary functions in the kernel. Otherwise, one process may arbitrarily jump to the kernel and read data that it is not authorized to read or call functions that results in security attacks that require root access. Instead, all system calls go through a limited number of entry points to kernel code, which are available as a limited number of system call functions to the user

Which of the following statements about system calls are true? Select all that apply.
(multiple choice question with MULTIPLE correct answers)
A. Correct. System calls are more expensive in terms of CPU resources compared to function calls.
B. Correct. System calls require context switching but function calls do not.
C. Correct. After a system call, the original system caller may not get to run immediately.
System calls require context switches, because they involve privileged instructions. Functions written by users do not require context switching. Context switching is expensive because the CPU must save the old process state and load in a new process. Because a new process gets a chance to run, the original system caller may need to wait to continue executing after the call is complete.

During a system call, a TRAP instruction occurs and executing the correct system call requires a jump to a specific address in OS address space, as indexed by the system call number.
(multiple choice question with SINGLE correct answer)
A. Correct. True
B. Incorrect. False

Since a system call requires user mode to kernel mode transition, a TRAP has to occur. To avoid the TRAP jumping
into any arbitrary location in the OS, it has to go through a dispatcher that figures out the right function call based on
the system call numbr

When executing a TRAP instruction during a system call, the CPU mode bit changes from supervisor to user.
(multiple choice question with SINGLE correct answer)
A. Incorrect. True
B. Correct. False
It goes from user to supervisor mode

A regular function call requires fewer CPU cycles than a system call (assuming both have the same code).
(multiple choice question with SINGLE correct answer)
A. Correct. True
B. Incorrect. False

A system call requires trapping to kernel, context switching and updating some data structures in the kernel. Hence it requires additional CPU cycles.

Consider the following piece of code where a parent creates a child, a child creates a grandchild, and a grandchild creates a great-grandchild process
Assume that parent, child (child_pid), grandchild (grandchild_pid), and great-grandchild (greatgrandchild_pid) have PIDs 100, 200, 300, and 400, respectively.

```
child_pid = fork();
if (child_pid != 0) {
  /* parent's code*/
} else {
  /* child */
  grandchild_pid = fork();
  if (grandchild_pid != 0) {
    /* child's code */
  } else {
    /* grandchild's code */
    greatgrandchild_pid = fork();
  }
}
```

Enter the value of the child_pid variable in the parent's address space.
(Numeric answer question)
Answer: 200
Explanation: child_pid is equal to the return value of the parent's fork()call which is 200

Enter the value of the greatgrandchild_pid variable in the grandchild's address space.
(Numeric answer question)
Answer: 400
Explanation: greatgrandchild_pid is equal to the return value of the grandparent's fork()call which is 400.

Enter the value of the greatgrandchild_pid variable in the greatgrandchild's address space.
(Numeric answer question)
Answer: 0
Explanation: When the grandchild calls fork(), a new process (greatgrandchild) is created. The return value in the greatgrandchild is 0.

Enter the value of the grandchild_pid variable in the grandchild's address space.
(Numeric answer question)
Answer: 0
Explanation: This has similar reasoning as in question 8. When a fork() call creates a new process, that within the new process, the return value of fork()is 0

Is the following statement true or false?
When a process makes a system call, after the call completes, it may not get to run immediately.
(multiple choice question with ONE correct answer)

A. Correct. True
B. Incorrect. False
Solution: The scheduler may pick another process to run instead of the process that made the call.

Is the following statement true or false?
During a system call, the system call number corresponding to the system call function is pushed onto the stack
initially.
(multiple choice question with ONE correct answer)
A. Correct. True
B. Incorrect. False
Solution: The stack is a convenient place for storing the system call number. This number will be popped off the stack when the kernel takes control, to figure out which system call to invoke.

Is the following statement true or false?
During a system call, the system call function caller's current register values are first stored in the heap and then copied to the PCB.
(multiple choice question with ONE correct answer)
A. Incorrect. True
B. Correct. False
Solution: The current register values are stored in the stack, not heap.

Is the following statement true or false?
System calls are more expensive (i.e. require more instructions to run) than regular function calls.
(multiple choice question with ONE correct answer)
A. Correct. True
B. Incorrect. False
Solution: System calls are more expensive because a context switch is required.