1.

Context switching is where the operating system switches operating system switches from one task to another. For this to work, the state of the current process must be saved. This allows the operating system to use the CPU for multiple tasks. There are many ways of accomplishing this, but they all have different tradeoffs.

Context switching with the compiler ensures that the program will work properly with whatever data it needs. The tradeoff is that it is slower than hardware context switching. The compiler can generally make better decisions, but this requires more overhead.

Using a linker gives you a better understanding what is linked to the external library. This gives a really good understanding of what registers need to be saved. The problem is that this option can be slower because it requires you to save and load all registers in case there is a live value.

Hardware context switches are implemented with a task state segment. The processor can load the information it needs from the task state segment. The problem with it is that it can only save general purpose registers. Also, it saves all general-purpose registers regardless of whether they were used or not. This can cause performance issues. In addition to this, you cannot do any tasks involving floating point. Most operating systems use software level context switching because it can be performed on any CPU.

2.

The value of x is different in the child a parent process. This verifies the copy on write policy.

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

int main()
{
    pid_t pid; // holds a pid
    int x = 0; // variable to test
```

```
    printf("Parent prints address of x: %p\n", (void*)&x); // print t
he address of x
    printf("Value of x: %d\n", x); // print the value of x
    pid = fork(); // create new child process

    if(pid == 0) // child runs this
    {
        printf("Child prints the address of x: %p\n", (void*)&x);
        printf("Value of x: %d\n", x);
        printf("x is now being set to 10.\n");
        x = 10; // change the value of x
        printf("Child prints the address of x: %p\n", (void*)&x);
        printf("Value of x: %d\n", x);
        exit(0); // exit the child process
    }

    else
    {
        wait(0); // child prints first
        printf("Parent prints address of x: %p\n", (void*)&x);
        printf("Value of x: %d\n", x);
        printf("x is now being set to -2.\n");
        x = -2; // change the value of x
        printf("Parent prints address of x: %p\n", (void*)&x);
        printf("Value of x: %d\n", x);
    }
    return 0;
}
```

Output:

```
austinsbrown@DESKTOP-O0AMQ3N:/mnt/c/Users/austi/
Parent prints address of x: 0x7ffd0548e290
Value of x: 0
Child prints the address of x: 0x7ffd0548e290
Value of x: 0
x is now being set to 10.
Child prints the address of x: 0x7ffd0548e290
Value of x: 10
Parent prints address of x: 0x7ffd0548e290
Value of x: 0
x is now being set to -2.
Parent prints address of x: 0x7ffd0548e290
Value of x: -2
```