Questions:

1)

Write a program that calls fork(). Before calling fork(), have the main process access a variable (e.g., x) and set its value to something (e.g., 100). What value is the variable in the child process? What happens to the variable when both the child and parent change the value of x?

Before the fork, we set variable x's value to 100, after fork the value in child process is still 100.
Both parent and child process won't effect the variable x. For example, before the fork we set variable x = 100. Then we fork, we increment by 1 in child process, we decrement by 1 in parent process. In child process we will get 100 + 1 =101. In parent process we got 100-1=99. The variable x is copy during the fork, and each parent and child process hold it's own copy.

3)

Write a program that opens a file (with the open() system call) and then calls fork() to create a new process. Can both the child and parent access the file descriptor returned by open()? What happens when they are writing to the file concurrently, i.e., at the same time?

Yes, both parent and child process can access the file descriptor returned by open().

Bothe the parent and child process can write into the file. In this particular example, the parent process write first, then the child process. The writing sequence depend on the operation system schedule.

3)

Write another program using fork(). The child process should print "hello"; the parent process should print "goodbye". You should try to ensure that the child process always prints first; can you do this without calling wait() in the parent?

Yes, we can use pipe to let child process execute first. We put a one line in parent code block to read from the pipe. One of the properties for pipe is if one of the processes is trying to read it before something write into the pipe, it will hold until something write into it. Thus, we can ensure child process print hello first, then parent process can print goodbye.

4)

Write a program that calls fork() and then calls some form of exec() to run the program /bin/ls. See if you can try all of the variants of exec(), including (on Linux) execl(), execle(), execlp(), execv(), execvp(), and execvpe(). Why do you think there are so many variants of the same basic call?

All of them is used to execute the a file.

System Call	Notes:
execl()	a list of arguments,
execle()	a list of arguments also specify the process environment
execlp()	a list of arguments, the first argument is name of, it will check PATH variable to find the file to execute.
execv()	Second argument is vector strings
execvp()	Second argument is vector strings, the first argument is name of, it will check PATH variable to find the file to execute.
execvpe()	Second argument is vector strings, the first argument is name of, it will check PATH variable to find the file to execute, the last argument to specify process environment.

With those variants of the same basic call, the exec() can handle different situation to execute file. Make the code more robust.

5)

Now write a program that uses wait() to wait for the child process to finish in the parent. What does wait() return? What happens if you use wait() in the child?

If wait success, it will return the child process id who finished the execution. If wait fail, for example no child process to wait or child process does not change the status. Then wait return -1. If we use wait() in the child, since child process didn't fork any process. Thus, wait() will no effect in child process, and it will return -1 on failure. Because wait() always wait for child process no other way around.

6)

Write a slight modification of the previous program, this time using waitpid() instead of wait(). When would waitpid() be useful?

When there is multiple child processes, and we want parent process wait for a specific child process. Then we can use waipid(). The wait() system call only wait any process who terminate first.

7)

Write a program that creates a child process, and then in the child closes standard output (STDOUT FILENO). What happens if the child calls printf() to print some output after closing the descriptor?

Nothing will print out since stand output is closed.

8)

Write a program that creates two children, and connects the standard output of one to the standard input of the other, using the pipe() system call.

Please check the hw8.c