Assignment 2

1: fork() is an exact copy of the calling process, parent and child. There are 3 fork calls in this program. Therefor, there are 6 fork processes, and hello is printed 7 times.

Problems: Clairity, ambiguity and syncronization are the main problems, along with syntax errors.

Revised code:

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

int main() {
   // Create two child processes
   fork();
   fork();

   // Each process prints "Hello"
   printf("Hello\n");

   return 0;
}
```

2: declares a variable, type pid_t thats used to represents process id's. Again, fork creates a new process with a parent and a child, the parent returns processes id of the child and the child returns 0. The if statement checks the return value of fork() to see if it is > 0, if it is, then it is the parent, if it it 0 it is the child. The parent prints "parent process" and the child makes the child sleep for 5 seconds after "child process is printed. Both eventually return 0 and end.

Parent executes first, then fork() introduces a child. Parent executes if (pid > 0). Child executes else if (pid == 0).

3: This code allocates memory continuously in a loop and doesnt have proper management, which leads to fragmentation. You can always limit allocation by tracking and setting a threshold. Watching memory usage at system and process level is very important, you could also achieve this using memory pools. There are also tactics you could use when failures do appear, like error handling and terminating before the failure.

This programs potential implications ons system recourcers and management include fragmentation, exhaustion, bad performance and others. These all cause very poor performance and utilization.

4: The wait() system call facilitates process synchronization by allowing the parent process to pause and wait for its child process to terminate before it continues. The kernel will tell the parent process that the child has terminated, and it also receives all of the childs info. The parent start again at wait() and executes. This process is crucial for insuring execution with multiple processes.

Creation and termination: The parent creates the child with the fork() process, as a direct copy. Again, the parent returns the childs PID and the child returns 0. The child executes at if (pid == 0), and terminates at exit().