Problem 1:

Please describe what happens with memory mapping in below situations:

1) write to a shared object which has been shared by multiple progresses;
   Nothing happen and other progresses will see the changes.
2) write to a private copy-on-write object which has been mapped to multiple progresses;
   The write firstly triggers a protection fault. The fault handler then creates a new copy of the page in physical memory, updates the page table entry to point to the new copy, and then restores write permissions to the page. When the fault handler returns, the CPU reexecutes the write, which now proceeds normally on the newly created page.
3) fork() is called in a progress;
   When the fork function is called by the current process, the kernel creates exact copies of the current process's mm_struct, area structs, and page tables. It flags each page in both processes as read-only, and flags each area struct in both processes as private copy-on-write. When the fork returns in the new process, the new process now has an exact copy of the virtual memory as it existed when the fork was called.
4) exec() is called in a progress.
   Delete the existing area structs in the user portion of the current process's virtual address. Create new area structs for the text, data, bss, and stack areas of the new program. All of these new areas are private copy-on-write. If there were any shared objects, then these objects are dynamically linked into the program, and then mapped into the shared region of the user's virtual address space. The last thing that execve does is to set the program counter in the current process's context to point to the entry point in the text area.

Problem 2:

Given an input file hello.txt that consists of the string "Hello, world!\n", write a C program that uses ***mmap*** to change the contents of hello.txt to "Jello, world!\n".

```
#include "csapp.h"
/*
* mmapwrite - uses mmap to modify a disk file
*/
void mmapwrite(int fd, int len)
{
      char *bufp;
      bufp = Mmap(NULL, len, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
      bufp[0] = 'J';
}
/* mmapwrite driver */
int main(int argc, char **argv)
{
      int fd;
      struct stat stat;
```

```
    /* check for required command line argument */
    if (argc != 2) {
        printf("usage: %s <filename>\n", argv[0]);
        exit(0);
    }
    /* open the input file and get its size */
    fd = Open(argv[1], O_RDWR, 0);
    fstat(fd, &stat);
    mmapwrite(fd, stat.st_size);
    exit(0);
}
```

意思对即可。

Problem 3:

Write a C program ***mmapcopy.c*** that uses ***mmap*** to copy an arbitrary-sized disk file to ***stdout***. The name of the input file should be passed as a command line argument.

```
#include "csapp.h"
/*
* mmapcopy - uses mmap to copy file fd to stdout
*/
void mmapcopy(int fd, int size)
{
    char *bufp; /* Pointer to memory mapped VM area */
    bufp = Mmap(NULL, size, PROT_READ, MAP_PRIVATE, fd, 0);
    Write(1, bufp, size);
    return;
}

/* mmapcopy driver */
int main(int argc, char **argv)
{
    struct stat stat;
    int fd;

    /* Check for required command line argument */
    if (argc != 2) {
        printf("usage: %s <filename>\n", argv[0]);
        exit(0);
    }

    /* Copy the input argument to stdout */
    fd = Open(argv[1], O_RDONLY, 0);
    fstat(fd, &stat);
```

```
        mmapcopy(fd, stat.st_size);
        exit(0);
}
```
意思对即可。