1、 进程是一个动态的概念，可以通过它使用系统的部分来概括它，比如内存、寄存器。
2、 Fork 用来创建子进程，exec 用来执行新的应用，wait 父进程用来管理子进程，shell 可以用来修改当前的环境，进程可以用 fork 与 exec 组合来使一个进程创建不同功能的进程，shell 可以在 fork 与 exec 之间更改环境。比起 window 一个 API 实现新进程的创建，linux 通过几个小的 api 实现新进程的创建可以实现更加灵活的操作。
3、
第二题
代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
    int rc = fork();

    int fd = open("./test.txt", O_RDWR);

    if (rc < 0) {
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) {
        printf("child fd: %d\n", fd);

        char buffer1[] = "child: hello, ostep!\n";
        write(fd, buffer1, sizeof(buffer1));
    } else {
        printf("parent fd: %d\n", fd);

        char buffer2[] = "parent: hello, ostep!\n";
        write(fd, buffer2, sizeof(buffer2));
    }
    return 0;
}
```

结果如下：

```
parent fd: 3
child fd: 3
```

test.txt 文件也成功被修改了，但是写入的是子进程的内容

```
> cat test.txt
child: hello, ostep!
```

第四题
代码如下：

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h>
#include<string.h>
#include<sys/wait.h>
int main(int argc,char *argv[]){
	int rc=fork();
	if(rc<0){
		printf("error");
		exit(1);
	}
	else if(rc==0){///child
		printf("This is child\n");
		execl("/bin/ls","ls","-l",NULL);

	}
	else{//father
		waitpid(rc,NULL,0);
		char* v[]={"ls","-l",NULL};
		printf("This is father\n");
		execve("/bin/ls",v,NULL);
		execv("/bin/ls",v);
		execvp("/bin/ls",v);
	}
return 0;
}
```

函数的用法参数列表可以在 RTFM 中找。所有的 exec 变体都可以运行程序/bin/ls。execve()是基础的系统调用，其他的变种都是在这个基础上包装的库函数。因为要应对各种不同的需求，所以才衍生出了这么多的形式。比如“l”表示参数以列表的形式表示，“v”表示参数以数组的形式表示，“p”表示在 PATH 中搜索执行文件，“e”表示可附加环境参数。

第六题
代码如下：

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

```
int main(int argc,char *argv[]){
        int rc=fork();
        if(rc<0){
                printf("error");
                exit(1);
        }
        else if(rc==0){///child
            ///    int res=waitpid(NULL);
                printf("child %d \n",(int)getpid());
        }
        else{//father
                int res= waitpid(-1,NULL,0);
                printf("father %d %d\n",(int)getpid(),res);
        }
return 0;
}
```

当我们需要等待特定的子进程时，由 pid 的参数决定等待的子进程，waitpid(pid_t, int* status,int options)，通常后两者取 NULL,0，pid_t 取要等待结束的 pid。

第八题

代码如下：

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

int main(int argc, char *argv[]) {
    int p[2];

    if (pipe(p) < 0) exit(1);

    int child1 = fork();
    if (child1 < 0) {
        fprintf(stderr, "fork child1 failed\n");
        exit(1);
    } else if (child1 == 0) {
        dup2(p[1], STDOUT_FILENO);
        printf("child1 pid: %d\n", (int) getpid());
        return 0;
    }

    int child2 = fork();

    if (child2 < 0) {
        fprintf(stderr, "fork child2 failed\n");
```

```
            exit(1);
        } else if (child2 == 0) {
            dup2(p[0], STDIN_FILENO);
            printf("child2 pid: %d\n", (int) getpid());
            char buffer[100];
            read(p[0], buffer, 100);
            printf("child2 pipe out: \t%s", buffer);
            return 0;
        }

        printf("parent pid: %d\n", (int) getpid());

        return 0;
}
```