1. 并发服务器端实现模型和方法：

**a. 多进程服务器：**通过创建多个进程提供服务

**b. 多路复用服务器：**通过捆绑并统一管理IO对象提供服务（select和epoll）

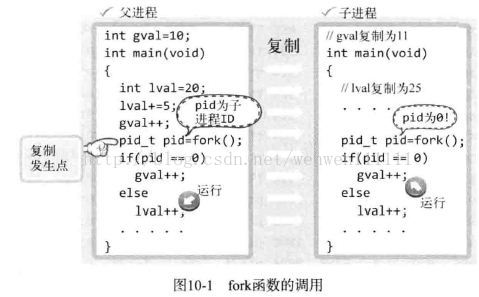
**c. 多线程服务器：**通过生成与客户端等量的线程提供服务

2. 通过fork函数创建进程：复制正在运行的调用fork函数的进程，复制相同的内存空间。两个进程都执行fork函数以后的语句。

1. #include <unistd.h>
3. pid\_t fork(**void**);
5. 成功时返回进程ID，失败时返回-1.

父进程Parent Prcess：fork函数返回子进程ID

子进程Child Process：fork函数返回0.



1. #include <stdio.h>
2. #include <unistd.h>
4. **int** gval = 10;
5. **int** main(**int** argc, **char** \*argv[]){
6. pid\_t pid;
7. **int** lval = 20;
9. gval++;
10. lval += 5;
12. pid = fork();
14. **if**(pid == 0){
15. //child
16. gval += 2;
17. lval += 2;
18. }**else**{
19. gval -= 2;
20. lval -= 2;
21. }
23. **if**(pid == 0){
24. printf("child gval = %d, lval = %d \n",gval,lval);
25. }**else**{
26. printf("parent gval = %d, lval = %d \n",gval,lval);
27. }
28. **return** 0;
29. }

4. 僵死Zombie进程：进程已死，但仍旧占用系统资源

a. 子进程终止方式：exit函数传递的参数和执行return并返回值。

b. 僵死进程的产生：exit的传递值和return返回值会传递给**[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "_blank)**，操作系统不会销毁子进程，直到这些值传递到父进程。

c. 如何传递值到父进程：操作系统不会主动把值传递到父进程，而是父进程主动请求，函数调用。如果父进程不主动请求，则子进程会一直处于僵死状态，所以父进程要负责收回子进程。

5. 利用wait销毁僵死进程：父进程主动请求获取子进程的返回值。

1. #include <sys/wait.h>
3. pid\_t wait(**int** \* statloc);
5. 成功时返回终止的子进程ID，失败返回-1

子进程终止参数保存在wait的参数中，里面还有其他信息，需要通过宏分离有效值。

WIFEXITED：子进程注册终止，返回TRUE

WEXITSTATUS：返回子进程的返回值

**调用wait函数时，如果没有已终止的子进程，那么程序blocking，直到有子进程终止，所以调用wait需谨慎**。

1. 示例二：wait销毁子进程
2. #include <stdio.h>
3. #include <unistd.h>
4. #include <stdlib.h>
5. #include <sys/wait.h>
7. **int** main(**int** argc, **char** \*argv[]){
8. **int** status;
9. pid\_t pid;
11. pid = fork();
13. **if**(pid == 0){
14. **return** 3;
15. }**else**{
16. printf("Child1 PID :%d \n",pid);
17. pid = fork();
19. **if**(pid == 0){
20. exit(7);
21. }**else** {
22. printf("Child2 PID :%d \n",pid);
23. wait(&status);
25. **if**(WIFEXITED(status)){
26. printf("Child one : %d \n",WEXITSTATUS(status));
27. }
29. wait(&status);
31. **if**(WIFEXITED(status)){
32. printf("Child two : %d \n",WEXITSTATUS(status));
33. }
35. sleep(5);
36. }
37. }
39. **return** 0;
40. }

7. waitpid函数销毁僵死进程

1. #include <sys/wait.h>
3. pid\_t waitpid(pid\_t pid, **int** \* statloc, **int** options);
5. 成功时返回终止子进程ID，失败返回-1
6. pid：等待终止的目标子进程ID，若传递-1，则与wait函数相同，等待任意子进程
7. statloc：与wait函数的statloc参数一致
8. options：传递头文件sys/wait.h中声明的常量WNOHANG，**即使没有终止子进程也不会阻塞，而是返回0并退出函数**

8. 示例三，waitpid

1. #include <stdio.h>
2. #include <unistd.h>
3. #include <sys/wait.h>
5. **int** main(**int** argc, **char** \*argv[]){
6. **int** status;
7. pid\_t pid;
9. pid = fork();
11. **if**(pid == 0){
12. sleep(5);
13. **return** 24;
14. }**else**{
15. **while**(!waitpid(-1,&status,WNOHANG)){
16. sleep(1);
17. puts("sleep 1sec.");
18. }
20. **if**(WIFEXITED(status)){
21. printf("child send %d \n",WEXITSTATUS(status));
22. }
23. }
25. **return** 0;
26. }