代码：

## 1.

1. **static** **struct** proc\_struct \*
2. alloc\_proc(**void**) {
3. **struct** proc\_struct \*proc = kmalloc(**sizeof**(**struct** proc\_struct));
4. **if** (proc != NULL) {
5. //LAB4:EXERCISE1 YOUR CODE
6. /\*
7. \* below fields in proc\_struct need to be initialized
8. \*       enum proc\_state state;                      // Process state
9. \*       int pid;                                    // Process ID
10. \*       int runs;                                   // the running times of Proces
11. \*       uintptr\_t kstack;                           // Process kernel stack
12. \*       volatile bool need\_resched; // bool value: need to be rescheduled to release CPU?
13. \*       struct proc\_struct \*parent;                 // the parent process
14. \*       struct mm\_struct \*mm;               // Process's memory management field
15. \*       struct context context;                     // Switch here to run process
16. \*       struct trapframe \*tf;                       // Trap frame for current interrupt
17. \*       uintptr\_t cr3;      // CR3 register: the base addr of Page Directroy Table(PDT)
18. \*       uint32\_t flags;                             // Process flag
19. \*       char name[PROC\_NAME\_LEN + 1];               // Process name
20. \*/
21. proc->state = PROC\_UNINIT;  //设置进程为“初始”态
22. proc->pid = -1; //设置进程pid的未初始化值
23. proc->runs = 0;//初始化时间片
24. proc->kstack = 0;//内核栈的地址
25. proc->need\_resched = 0;//是否需要调度
26. proc->parent = NULL;//父节点为空
27. proc->mm = NULL; //内存管理初始化
28. memset(&(proc->context), 0, **sizeof**(**struct** context));//进程上下文初始化
29. proc->tf = NULL; //中断帧指针置为空，总是能够指向中断前的trapframe
30. proc->cr3 = boot\_cr3;//设置内核页目录表的基址
31. proc->flags = 0; //标志位初始化
32. memset(proc->name, 0, PROC\_NAME\_LEN); //进程名初始化
33. }
34. **return** proc;
35. }

## 2.

1. **int**
2. do\_fork(uint32\_t clone\_flags, **uintptr\_t** stack, **struct** trapframe \*tf) {
3. **int** ret = -E\_NO\_FREE\_PROC;
4. **struct** proc\_struct \*proc;
5. **if** (nr\_process >= MAX\_PROCESS) {
6. **goto** fork\_out;
7. }
8. ret = -E\_NO\_MEM;
9. //LAB4:EXERCISE2 YOUR CODE
10. /\*
11. \* Some Useful MACROs, Functions and DEFINEs, you can use them in below implementation.
12. \* MACROs or Functions:
13. \*   alloc\_proc:   create a proc struct and init fields (lab4:exercise1)
14. \*   setup\_kstack: alloc pages with size KSTACKPAGE as process kernel stack
15. \*   copy\_mm:      process "proc" duplicate OR share process "current"'s mm according clone\_flags
16. \*                 if clone\_flags & CLONE\_VM, then "share" ; else "duplicate"
17. \*   copy\_thread:  setup the trapframe on the  process's kernel stack top and
18. \*                 setup the kernel entry point and stack of process
19. \*   hash\_proc:    add proc into proc hash\_list
20. \*   get\_pid:      alloc a unique pid for process
21. \*   wakup\_proc:   set proc->state = PROC\_RUNNABLE
22. \* VARIABLES:
23. \*   proc\_list:    the process set's list
24. \*   nr\_process:   the number of process set
25. \*/
27. //    1. call alloc\_proc to allocate a proc\_struct
28. //    2. call setup\_kstack to allocate a kernel stack for child process
29. //    3. call copy\_mm to dup OR share mm according clone\_flag
30. //    4. call copy\_thread to setup tf & context in proc\_struct
31. //    5. insert proc\_struct into hash\_list && proc\_list
32. //    6. call wakup\_proc to make the new child process RUNNABLE
33. //    7. set ret vaule using child proc's pid
34. //第一步：申请内存块，如果失败，直接返回处理
35. **if** ((proc = alloc\_proc()) == NULL) {
36. **goto** fork\_out;
37. }
38. //将子进程的父节点设置为当前进程
39. proc->parent = current;
40. //第二步：为进程分配一个内核栈
41. **if** (setup\_kstack(proc) != 0) {
42. **goto** bad\_fork\_cleanup\_proc;
43. }
44. //第三步：复制父进程的内存信息到子进程
45. **if** (copy\_mm(clone\_flags, proc) != 0) {
46. **goto** bad\_fork\_cleanup\_kstack;
47. }
48. //第四步：复制父进程相关寄存器信息（上下文）
49. copy\_thread(proc, stack, tf);
50. //第五步：将新进程添加到进程列表（此过程需要加保护锁）
51. **bool** intr\_flag;
52. local\_intr\_save(intr\_flag);
53. {
54. proc->pid = get\_pid();
55. //建立散列映射方便查找
56. hash\_proc(proc);
57. //将进程链节点加入进程列表
58. list\_add(&proc\_list, &(proc->list\_link));
59. //进程数+1
60. nr\_process ++;
61. }
62. local\_intr\_restore(intr\_flag);
63. //第六步：一切准备就绪，唤醒子进程
64. wakeup\_proc(proc);
65. //第七步：设置返回的子进程号
66. ret = proc->pid;
67. fork\_out:
68. **return** ret;
70. bad\_fork\_cleanup\_kstack:
71. put\_kstack(proc);
72. bad\_fork\_cleanup\_proc:
73. kfree(proc);
74. **goto** fork\_out;
75. }