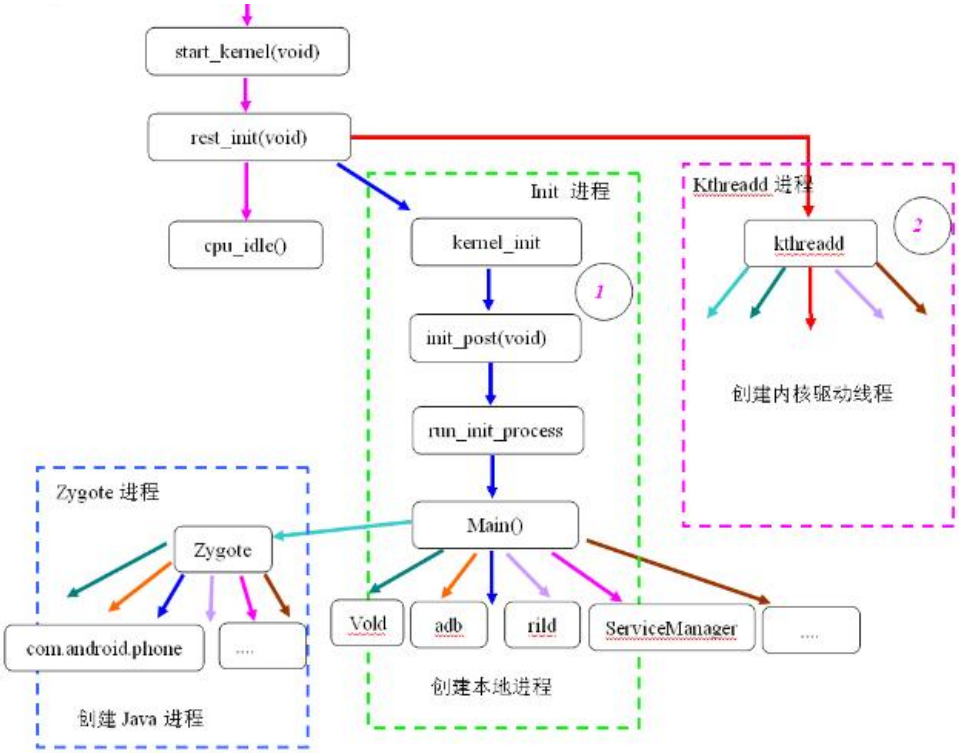
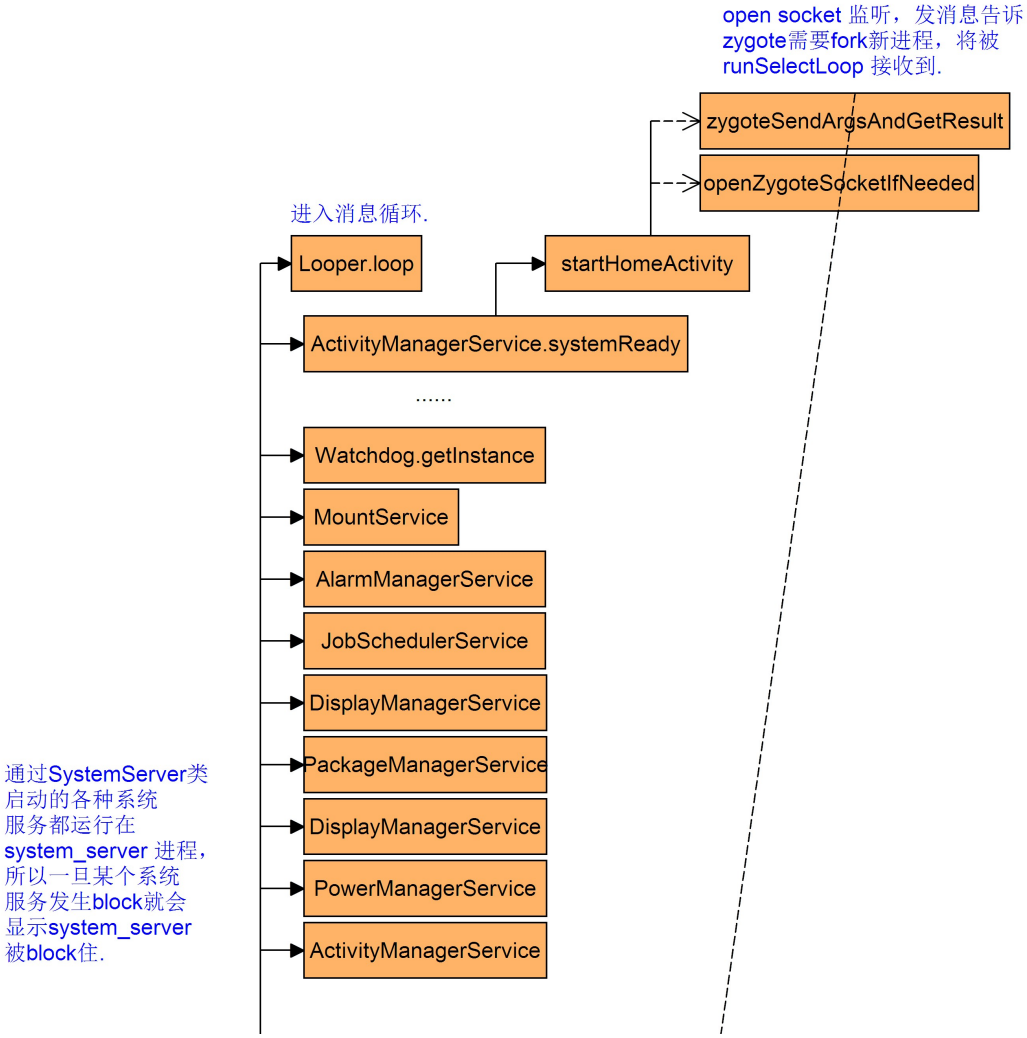
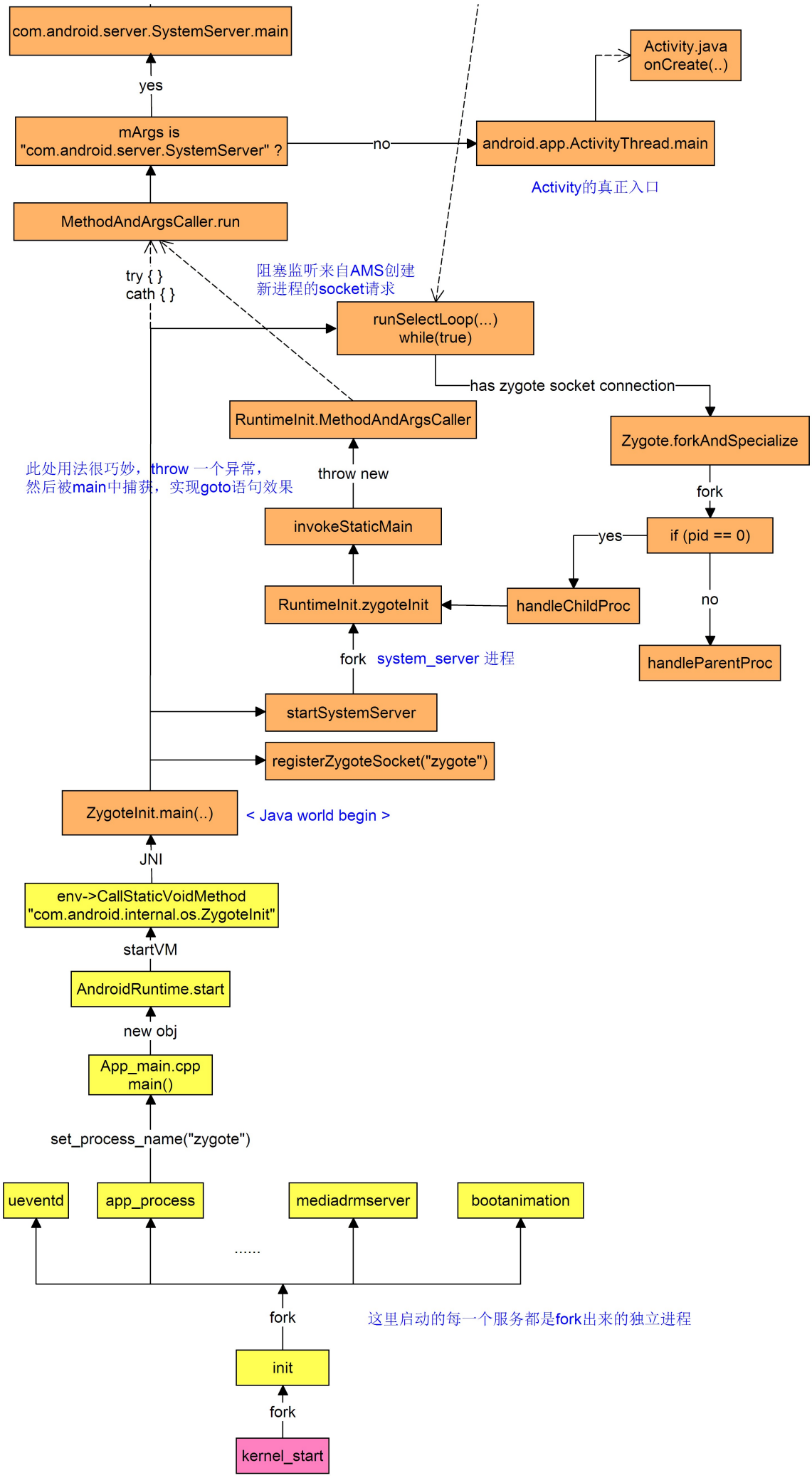
Android M 启动源码分析笔记之 - Init 进程

一、init进程模型



执行结构流程图：




com.android.server.SystemServer.main
yes
mArgs is "com.android.server.SystemServer" ?
no
Activity.java onCreate(..)
android.app.ActivityThread.main
Activity的真正入口
MethodAndArgsCaller.run
try { }
cath { }
阻塞监听来自AMS创建新进程的socket请求
runSelectLoop(...) while(true)
has zygote socket connection
RuntimeInit.MethodAndArgsCaller
Zygote.forkAndSpecialize
此处用法很巧妙，throw 一个异常，然后被main中捕获，实现goto语句效果
throw new
fork
invokeStaticMain
yes
if (pid == 0)
RuntimeInit.zygoteInit
handleChildProc
no
handleParentProc
fork
system_server 进程
startSystemServer
registerZygoteSocket("zygote")
ZygoteInit.main(..)
< Java world begin >
JNI
env->CallStaticVoidMethod "com.android.internal.os.ZygoteInit"
startVM
AndroidRuntime.start
new obj
App_main.cpp main()
set_process_name("zygote")
ueventd
app_process
mediadrmserver
bootanimation
.....
fork
这里启动的每一个服务都是fork出来的独立进程
init
fork
kernel_start

流程图比较清晰的展示了init到system_server的启动过程和zygote的孵化原理.接下来分以下几个阶段分析：

1、init进程干了什么事情？
2、zygote进程是如何启动的？如何成为Java world所有进程的父进程？
3、一个新app启动的基本流程，以Home程序为例.

二、源码分析
1、init启动流程：

```
int main(int argc, char** argv) {
/*
这个地方容易让人误解以为init进程跟ueventd/watchdogd进程是一个，
其实完全是独立的进程。查看system/core/init/Android.mk 可以看到：
LOCAL_POST_INSTALL_CMD := \
    $(hide) mkdir -p $(TARGET_ROOT_OUT)/sbin; \
    ln -sf ../init $(TARGET_ROOT_OUT)/sbin/ueventd; \
    ln -sf ../init $(TARGET_ROOT_OUT)/sbin/watchdogd
也就是说ueventd/watchdogd只是共用了这份代码，
启动进程时候输入参数作为区分而已.
*/
    if (!strcmp(basename(argv[0]), "ueventd")) {
        return ueventd_main(argc, argv);
    }

    if (!strcmp(basename(argv[0]), "watchdogd")) {
        return watchdogd_main(argc, argv);
    }

/* 初始化环境变量和文件系统,如果是stage 1启动，则需要创建挂载一些文件系统目录.
   这个stage 1 是表示从kernel启动init进程，stage 2 是init 自己启动自己！对，
   这里没错，就是自己启动自己！
*/
    add_environment("PATH", _PATH_DEFPATH);

    bool is_first_stage = (argc == 1) || (strcmp(argv[1], "--second-stage") != 0);

    if (is_first_stage) {
        mount("tmpfs", "/dev", "tmpfs", MS_NOSUID, "mode=0755");
        mkdir("/dev/pts", 0755);
        mkdir("/dev/socket", 0755);
        mount("devpts", "/dev/pts", "devpts", 0, NULL);
        mount("proc", "/proc", "proc", 0, NULL);
        mount("sysfs", "/sys", "sysfs", 0, NULL);
    }

...
//初始化klog输出.
    open_devnull_stdio();
    klog_init();
    klog_set_level(KLOG_NOTICE_LEVEL);

...

/* 下面这里要干的事情首先初始化属性系统，
然后取出系统设备树、
命令行等文件中的属性值写入到属性系统map.
导出填充kernel启动属性默认值等.
*/
    if (!is_first_stage) {
...
        property_init();
        process_kernel_dt();
        process_kernel_cmdline();
        export_kernel_boot_props();
    }

/* Selinux 初始化相关部分 */
    selinux_initialize(is_first_stage);
```

```
    if (is_first_stage) {
        if (restorecon("/init") == -1) {
            ERROR("restorecon failed: %s\n", strerror(errno));
            security_failure();
        }
        char* path = argv[0];
        char* args[] = { path, const_cast<char*>("--second-stage"), nullptr };

/* 这里的path就是 ./init, args就是传入--second-stage 然后重启init进程!这样做的目的
   猜想是基于权限方面的考虑?
*/
        if (execv(path, args) == -1) {
            ERROR("execv(\"%s\") failed: %s\n", path, strerror(errno));
            security_failure();
        }
    }

...

//创建epoll描述符，获取句柄注册epoll信号处理，用于父子进程通信.
    epoll_fd = epoll_create1(EPOLL_CLOEXEC);
    if (epoll_fd == -1) {
        ERROR("epoll_create1 failed: %s\n", strerror(errno));
        exit(1);
    }

    signal_handler_init();

/* 加载default.prop到属性系统，启动属性服务socket监听
*/
    property_load_boot_defaults();
    start_property_service();

/* 解析init.rc，将解析得来的各种service、on 填充到容器链表：service_list、action_list 中管理起来。
   init.rc解析原理不复杂，但是内容比较多，涉及大量指针链表操作.
   如果想详细了解细节，需另写一篇文章来描述，非本文主线，此略过.
*/
    init_parse_config_file("/init.rc");

/* 下面干的事情主要是将各种cmd添加到 action_queue 链表执行队列. */
    action_for_each_trigger("early-init", action_add_queue_tail);
    queue_builtin_action(wait_for_coldboot_done_action, "wait_for_coldboot_done");
    queue_builtin_action(mix_hwrng_into_linux_rng_action, "mix_hwrng_into_linux_rng");
    queue_builtin_action(keychord_init_action, "keychord_init");
    queue_builtin_action(console_init_action, "console_init");
    action_for_each_trigger("init", action_add_queue_tail);
    queue_builtin_action(mix_hwrng_into_linux_rng_action, "mix_hwrng_into_linux_rng");

/* 根据启动模式选择不同的触发cmd */
    char bootmode[PROP_VALUE_MAX];
    if (property_get("ro.bootmode", bootmode) > 0 && strcmp(bootmode, "charger") == 0) {
        action_for_each_trigger("charger", action_add_queue_tail);
    } else {
        action_for_each_trigger("late-init", action_add_queue_tail);
    }

/* 取出全部从init.rc中解析出来的属性触发条件（on property:sys.xxx=1）
   加入到属性map中关联起来.
*/
    queue_builtin_action(queue_property_triggers_action, "queue_property_triggers");

/* 进入循环执行 */
    while (true) {
        if (!waiting_for_exec) {
// 依次取出执行队列中的cmd执行
            execute_one_command();
// 依次取出执行队列中的service执行，fork新进程.
            restart_processes();
        }

/* time out 策略*/
        int timeout = -1;
        if (process_needs_restart) {
            timeout = (process_needs_restart - gettime()) * 1000;
            if (timeout < 0)
```

```
                timeout = 0;
        }

        if (!action_queue_empty() || cur_action) {
            timeout = 0;
        }

        bootchart_sample(&timeout);

/* 等待执行注册了epoll的回调函数，这里主要有两个：
一个是start_property_service中注册的，一个是signal_handler_init中注册的信号监听，
一旦收到来自子进程挂掉信号，根据设定的策略重启子进程.
*/
        epoll_event ev;
        int nr = TEMP_FAILURE_RETRY(epoll_wait(epoll_fd, &ev, 1, timeout));
        if (nr == -1) {
            ERROR("epoll_wait failed: %s\n", strerror(errno));
        } else if (nr == 1) {
            ((void (*)()) ev.data.ptr)();
        }
    }

    return 0;
}
```

init要干的事情小结：

1、first stage 初始化环境变量和各种文件系统目录，klog初始化等；

2、selinux 相关初始化完成，然后切换second stage重启init进程；

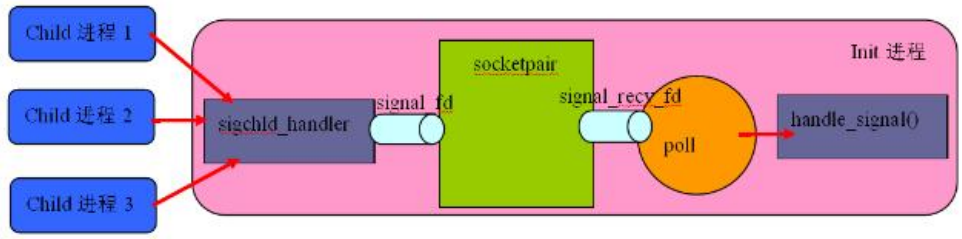
3、属性服务初始化，将各种系统属性默认值填充到属性Map中；

4、创建epoll描述符结合注册socket监听，处理显示启动进程和挂掉的子进程重启；

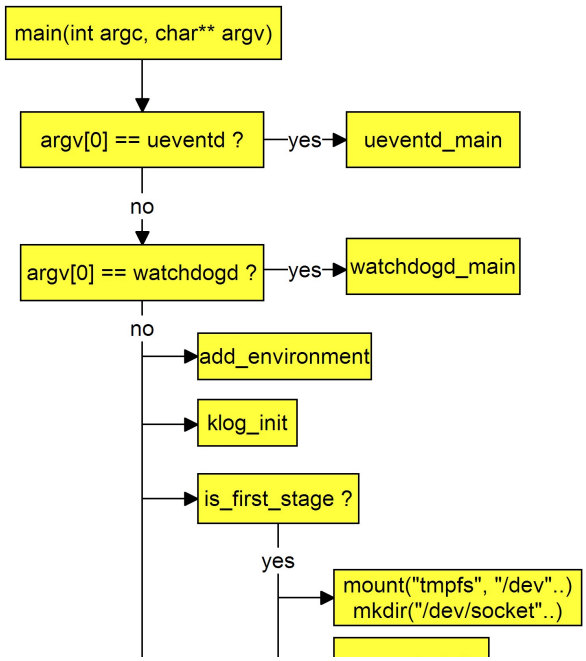
5、解析init.rc，把各种Action、service等解析出来填充到相应链表容器中管理；

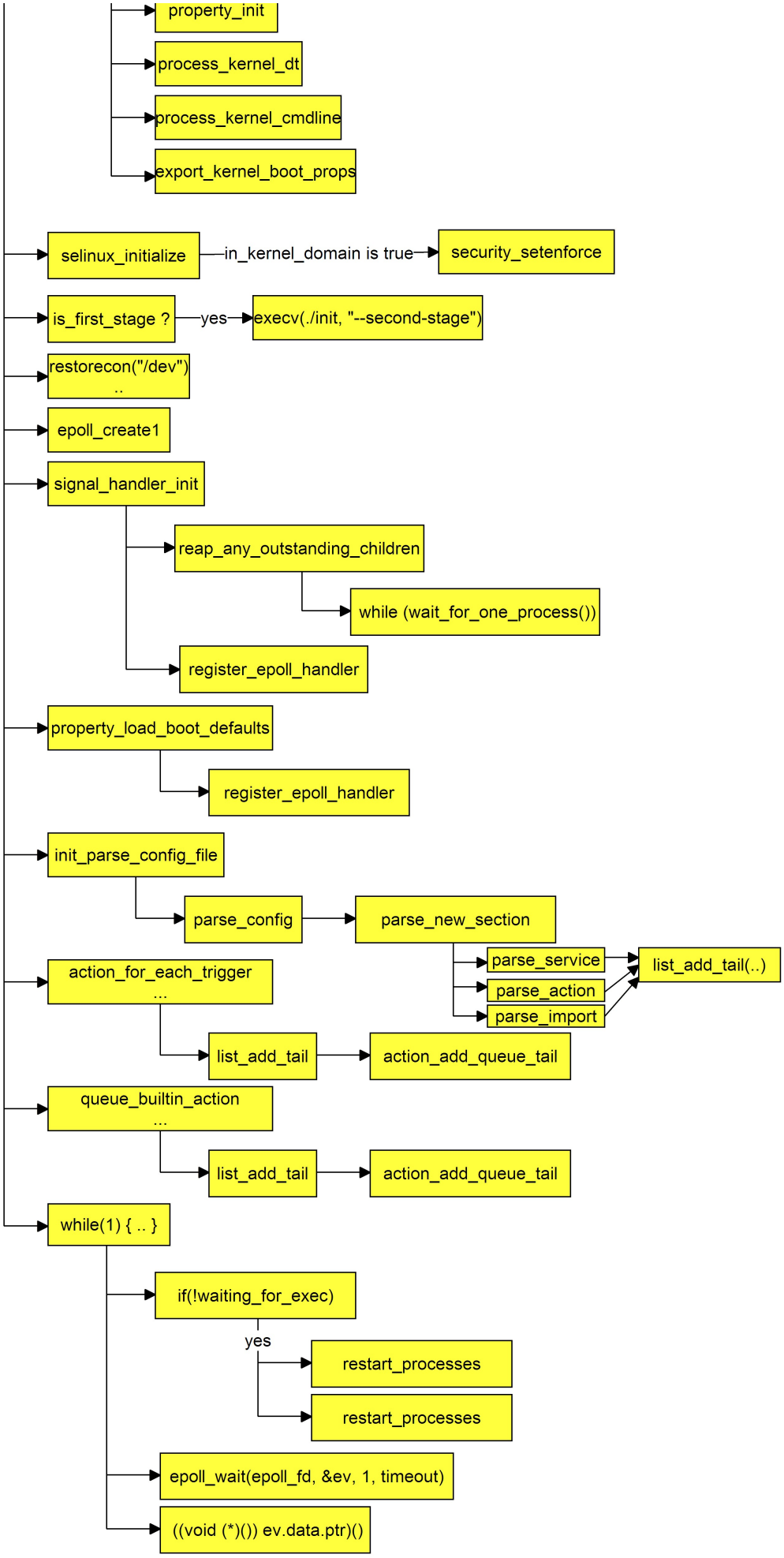
6、有序将early-init、init等各种cmd加入到执行队列action_queue链表中；

7、进入while(1)循环依次取出执行队列action_queue中的command执行，fork包括app_process在内的各种进程，epoll阻塞监听处理来自挂掉的子进程的消息，根据设定策略restart子进程.



源码流程图：




property_init
process_kernel_dt
process_kernel_cmdline
export_kernel_boot_props
selinux_initialize
in_kernel_domain is true
security_setenforce
is_first_stage ?
yes
execv(./init, "--second-stage")
restorecon("/dev")
..
epoll_create1
signal_handler_init
reap_any_outstanding_children
while (wait_for_one_process())
register_epoll_handler
property_load_boot_defaults
register_epoll_handler
init_parse_config_file
parse_config
parse_new_section
parse_service
parse_action
parse_import
list_add_tail(..)
action_for_each_trigger
...
list_add_tail
action_add_queue_tail
queue_builtin_action
...
list_add_tail
action_add_queue_tail
while(1) { .. }
if(!waiting_for_exec)
yes
restart_processes
restart_processes
epoll_wait(epoll_fd, &ev, 1, timeout)
((void (*)()) ev.data.ptr)()