in pegmatite.cfg

```
CONFIG_UEVENT_HELPER_PATH="/sbin/hotplug"
```

由于build Linux kernel的.config来自与pegmatite.cfg，所以"uevent helper" feature被enable了。

in config-3.18.7-yocto-standard

```
CONFIG_UEVENT_HELPER=y
CONFIG_UEVENT_HELPER_PATH="/sbin/hotplug"
```

但实际上"uevent helper"这个feature已经被udev取代。实际上完全不应该在G2 LSP中enable该feature。

in README file of udev-182

---

- The deprecated hotplug helper /sbin/hotplug should be disabled in the

  kernel configuration, it is not needed today, and may render the system

  unusable because the kernel may create too many processes in parallel

  so that the system runs out-of-memory.

---

从uevent helper相关kernel code上可以理解上面README的说明。

in lib/kobject_uevent.c

```c
/**
 * kobject_uevent_env - send an uevent with environmental data
 *
 * @action: action that is happening
 * @kobj: struct kobject that the action is happening to
 * @envp_ext: pointer to environmental data
 *
 * Returns 0 if kobject_uevent_env() is completed with success or the
 * corresponding error when it fails.
 */
int kobject_uevent_env(struct kobject *kobj, enum kobject_action action,
                       char *envp_ext[])
{
        struct kobj_uevent_env *env;
        const char *action_string = kobject_actions[action];
        const char *devpath = NULL;
        const char *subsystem;
        struct kobject *top_kobj;
        struct kset *kset;
        const struct kset_uevent_ops *uevent_ops;
        int i = 0;
        int retval = 0;
#ifdef CONFIG_NET
        struct uevent_sock *ue_sk;
#endif

        pr_debug("kobject: '%s' (%p): %s\n",
                 kobject_name(kobj), kobj, __func__);

        /* search the kset we belong to */
        top_kobj = kobj;
        while (!top_kobj->kset && top_kobj->parent)
                top_kobj = top_kobj->parent;

        if (!top_kobj->kset) {
                pr_debug("kobject: '%s' (%p): %s: attempted to send uevent "
                         "without kset!\n", kobject_name(kobj), kobj,
                         __func__);
                return -EINVAL;
        }

        kset = top_kobj->kset;
        uevent_ops = kset->uevent_ops;

        /* skip the event, if uevent_suppress is set*/
        if (kobj->uevent_suppress) {
                pr_debug("kobject: '%s' (%p): %s: uevent_suppress "
                                 "caused the event to drop!\n",
                                 kobject_name(kobj), kobj, __func__);
                return 0;
        }
        /* skip the event, if the filter returns zero. */
        if (uevent_ops && uevent_ops->filter)
                if (!uevent_ops->filter(kset, kobj)) {
                        pr_debug("kobject: '%s' (%p): %s: filter function "
                                 "caused the event to drop!\n",
                                 kobject_name(kobj), kobj, __func__);
                        return 0;
                }

        /* originating subsystem */
        if (uevent_ops && uevent_ops->name)
                subsystem = uevent_ops->name(kset, kobj);
```

```
	else
		subsystem = kobject_name(&kset->kobj);
	if (!subsystem) {
		pr_debug("kobject: '%s' (%p): %s: unset subsystem caused the "
			 "event to drop!\n", kobject_name(kobj), kobj,
			 __func__);
		return 0;
	}

	/* environment buffer */
	env = kzalloc(sizeof(struct kobj_uevent_env), GFP_KERNEL);
	if (!env)
		return -ENOMEM;

	/* complete object path */
	devpath = kobject_get_path(kobj, GFP_KERNEL);
	if (!devpath) {
		retval = -ENOENT;
		goto exit;
	}

	/* default keys */
	retval = add_uevent_var(env, "ACTION=%s", action_string);
	if (retval)
		goto exit;
	retval = add_uevent_var(env, "DEVPATH=%s", devpath);
	if (retval)
		goto exit;
	retval = add_uevent_var(env, "SUBSYSTEM=%s", subsystem);
	if (retval)
		goto exit;

	/* keys passed in from the caller */
	if (envp_ext) {
		for (i = 0; envp_ext[i]; i++) {
			retval = add_uevent_var(env, "%s", envp_ext[i]);
			if (retval)
				goto exit;
		}
	}

	/* let the kset specific function add its stuff */
	if (uevent_ops && uevent_ops->uevent) {
		retval = uevent_ops->uevent(kset, kobj, env);
		if (retval) {
			pr_debug("kobject: '%s' (%p): %s: uevent() returned "
				 "%d\n", kobject_name(kobj), kobj,
				 __func__, retval);
			goto exit;
		}
	}

	/*
	 * Mark "add" and "remove" events in the object to ensure proper
	 * events to userspace during automatic cleanup. If the object did
	 * send an "add" event, "remove" will automatically generated by
	 * the core, if not already done by the caller.
	 */
	if (action == KOBJ_ADD)
		kobj->state_add_uevent_sent = 1;
	else if (action == KOBJ_REMOVE)
		kobj->state_remove_uevent_sent = 1;

	mutex_lock(&uevent_sock_mutex);
```

```
        /* we will send an event, so request a new sequence number */
        retval = add_uevent_var(env, "SEQNUM=%llu", (unsigned long long)++uevent_seqnum);
        if (retval) {
                mutex_unlock(&uevent_sock_mutex);
                goto exit;
        }

#if defined(CONFIG_NET)
        /* send netlink message */
        list_for_each_entry(ue_sk, &uevent_sock_list, list) {
                struct sock *uevent_sock = ue_sk->sk;
                struct sk_buff *skb;
                size_t len;

                if (!netlink_has_listeners(uevent_sock, 1))
          ④
                        continue;

                /* allocate message with the maximum possible size */
                len = strlen(action_string) + strlen(devpath) + 2;
                skb = alloc_skb(len + env->buflen, GFP_KERNEL);
                if (skb) {
                        char *scratch;

                        /* add header */
                        scratch = skb_put(skb, len);
                        sprintf(scratch, "%s@%s", action_string, devpath);

                        /* copy keys to our continuous event payload buffer */
                        for (i = 0; i < env->envp_idx; i++) {
                                len = strlen(env->envp[i]) + 1;
```

```
                                  scratch = skb_put(skb, len);
                                  strcpy(scratch, env->envp[i]);
                        }

                        NETLINK_CB(skb).dst_group = 1;
                        retval = netlink_broadcast_filtered(uevent_sock, skb,
                                                            0, 1, GFP_KERNEL,
                                                            kobj_bcast_filter,
                                                            kobj);
                        /* ENOBUFS should be handled in userspace */
                        if (retval == -ENOBUFS || retval == -ESRCH)
                                retval = 0;
                } else
                        retval = -ENOMEM;
        }
#endif
        mutex_unlock(&uevent_sock_mutex);

#ifdef CONFIG_UEVENT_HELPER
      ①
        /* call uevent_helper, usually only enabled during early boot */
        if (uevent_helper[0] && !kobj_usermode_filter(kobj)) {
     ②
                struct subprocess_info *info;

                retval = add_uevent_var(env, "HOME=/");
                if (retval)
                        goto exit;
                retval = add_uevent_var(env,
                                        "PATH=/sbin:/bin:/usr/sbin:/usr/bin");
                if (retval)
                        goto exit;
                retval = init_uevent_argv(env, subsystem);
                if (retval)
                        goto exit;

                retval = -ENOMEM;
                info = call_usermodehelper_setup(env->argv[0], env->argv, ③
                                                 env->envp, GFP_KERNEL,
                                                 NULL, cleanup_uevent_env, env);
                if (info) {
                        retval = call_usermodehelper_exec(info, UMH_NO_WAIT);
                        env = NULL;     /* freed by cleanup_uevent_env */
                }
        }
#endif

exit:
        kfree(devpath);
        kfree(env);
        return retval;
}
```

kobject_uevent_env() function是/sys file system中的device产生的uevent与udevd沟通的渠道（另文叙述）。

①

在enable "uevent helper" feature的情况下，才需要下面的code。

这段code的作用就是当uevent产生后，需要user mode的application来读取这些产生在kernel mode的信息，以便在user mode create device相关信息，就是/dev目录下的内容。这个feature有很大问题。

当还处于kernel initialization阶段，即udevd运行以前，每当产生一个uevent就会create "/sbin/hotplug" process来运行。而创建device处于initcall level 3阶段，从/sys/kernel/uevent_seqnum的输出看(在disable udevd的情况下，因为这个数字就是kernel initialization阶段发送的uevent number)，在G2 LSP上都是超过1000，也就是在kernel初始化的末期的这么一点时间内，kernel要如此密集的create 1000多个process。这就是README中说的所谓

"it is not needed today, and may render the system

unusable because the kernel may create too many processes in parallel

so that the system runs out-of-memory."

另外，在G2 LSP的rootfs，根本就没有/sbin/hotplug这个文件或link，也就是这段code完全在浪费CPU,做无用功。

②

在udevd起来以后，udevd作为netlink listener，会make uevent_helper[] NULL.因为udevd完全可以取代/sbin/hotplug，这样这段code就没必要运行了。

③

在kernel mode 异步地启动"/sbin/hotplug"

④

无论是/sbin/hotplug还是udevd都是netlink listener。kernel通过netlink socket来向user mode application broadcast uevent。

在enable udevd的情况下，等系统启动后

```
root@granite2:~# cat /sys/kernel/uevent_helper
```

看不到什么东西。用hexdump可以看到"0x0a"

但在disable udevd的情况下，等系统启动后

root@granite2:~# cat /sys/kernel/uevent_helper

/sbin/hotplug

虽然在G2 LSP，/sbin/hotplug没任何东西，但由于没有application去write /sys/kernel/uevent_helper文件而使得在kernel中赋值的内容得到保留。

```
#ifdef CONFIG_UEVENT_HELPER
char uevent_helper[UEVENT_HELPER_PATH_LEN] = CONFIG_UEVENT_HELPER_PATH;
#endif
```

---

禁掉"uevent helper" feature，kernel一切正常。