

1、 龙芯芯片使用的是 MIPS 架构，具体内核（SOC）有 ARM（32 位），X86（64 位），51（8 位）。

2、 列举几个开源（opensource）的硬件和软件？

硬件：Arduino；ESP32；OpenMV；树莓派；ESP8266。

软件：Nginx；Linux；Storm；Tomcat；Spark；RT-Thread；Arduino IDE；

3、 根文件系统镜

****.img

根文件系统镜像文件是：rootfs-yaffs2.img

2305201452

呆@西西弗斯

4、 子、父进程

PID：进程号，唯一地标识一个进程，他们都是非零的正整数。

PPID：父进程号。

getpid（）：获得进程号。

getppid（）：获得父进程号。

5、 pipe

pipe（pipe_fd）：建立管道的函数。

6、 线程同步 互斥 信号

在 POSIX 中线程同步的方法，主要有互斥锁（mutex）和信号量。

7、 mount umount

mount -t vfat /dev/sda1 /mnt：挂载 U 盘。

umount /mnt：卸载 U 盘

8、

insmod：加载模块。

rmmod：卸载模块。

9、 pmon 作用

pmon 的作用是引导龙芯派操作系统内核。在 PMON 中可以初始化硬件；加载操作系统内核；监控和调试操作系统；诊断和修复系统故障；升级和更新操作系统。

10、交叉编译

交叉编译：指在一个平台上生成另外一个平台的可执行代码。

11、 TFTP

set al tftp://193.169.2.215/vmlinux

g

12、 root

root 用户的解释，以及有什么作用？

答: **root**, 也称为根用户, 是 **Unix** 设备系统中的唯一的超级用户, 因其可对根目录执行读写和执行操作而得名。其相当于 **Windows** 系统中的 **SYSTEM** 用户。其具有系统中的最高权限, 如启动或停止一个进程, 删除或增加用户, 增加或者禁用硬件, 新建文件、修改文件或删除所有文件等等。

语句

sudo: 管理员身份运行

su: 切换到 **root** 身份

apt-get update: 更新源

cat: 打印某个文件。

gcc -v: 查看 **gcc** (编译器) 版本。

mkdir: 以 **root** 权限创建目录

which: 显示给定命令的绝对路径

2、 编 pmon

编译 **PMON** 的命令是什么? 最后生成什么文件?

答: **cd /Workstation/tools/pmon/pmon-ls1x-openloongson/zloader.ls**

make cfg all tgt=rom CROSS_COMPILE=mipsel-linux-

最后生成 **gzrom.bin**。

3、 图形化配置

mymake menuconfig

4、 chmod

chmod777 将文件权限修改为可读写

chmod755 文件所有者可读可写可执行 其他用户可读可执行

5、 mem xxx

memcpy: 内存复制。

memset: 内存设置。

6、 strxxx

strcmp: 字符串比较。

strcpy: 字符串复制。

7、 文件四大类、描述符

普通文件、链接文件、目录文件、设备文件

文件描述符: 为非负整数, 指文件的索引值, 指向进程打开记录表; 当需要读写文件时, 也需要把文件描述符作为参数传递给相应函数。

8、 进程和线程 (关系和区别)

进程是系统中程序执行和资源分配的基本单位。相应地, 线程是一个进程内的基本调度单位, 也可以成为轻量级进程。线程是在共享内存中并发的多道执行路径, 它们共享一个进程的资源, 如文件描述符和信号处理。因此, 就大大减少了上下文的切换开销。一个进程内的多线程共享一个用户地址空间。由于线程共享了进程的资源 and 地址空间。因此, 任何线程对系统资源的操作都会给其他线程带来影响, 这样一来就要实现多线程之间的同步。

9、 进程和程序 (关系区别)

进程是一个程序一次执行的过程。

它和程序的本质区别是, 程序是静态的, 它是一些保存在磁盘上的指令的有序集合, 没有任何执行的概念。而进程是一个动态的概念, 它是程序执行的过程, 包含了动态创建、调度和消亡的整个过程。

10、 无名管道特点

无名管道具有如下特点:

(1) 它只能用于具有亲缘关系的进程之间通信, 例如父子进程或者兄弟进程之间。

(2) 它是一个半双工的通信模式, 具有固定的读端口和写端口。

(3) 管道也可以看成是一种特殊的文件, 对于它的读写也可以使用普通的 read 和 write 函数。但它不是普通的文件, 并不属于其他任何文件系统, 并且只存在于内存中。在 Linux 的文件属性中带有 p (pipe) 的文件就是管道文件。一个进程向管道中写的内容被管道的另一端的进程读出。写入的内容每次都添加在管道缓冲区的末尾, 并且每次都是从缓冲区的头部读出数据。

11、互斥锁步骤

互斥锁的操作主要包括以下几个步骤。

- 1 互斥锁初始化： pthread_mutex_init 。
- 2 互斥锁上锁： pthread_mutex_lock 。
- 3 互斥锁判断上锁： pthread_mutex_trylock 。
- 4 互斥锁解锁： pthread_mutex_unlock 。
- 5 消除互斥锁： pthread_mutex_destroy。

12、 信息量 6 个函数

信号量其实就是一个非负的整数计数器，是操作系统中所用的 PV 原语，主要应用于进程或线程间的同步与互斥。其工作原理也很简单，PV 原语就是对整数计数器信号量 sem 进行操作，一次 P 操作使 sem 减一，而一次 V 操作使 sem 加一。当信号量 sem 的值大于等于零时，该线程具有访问公共资源的权限；相反，当信号量 sem 的值小于零时，该线程就阻塞直到信号量 sem 的值大于等于 0 为止。

信号量的操作函数说明如表 7.3 所示。

表 7.3 信号量操作函数

名称	作用
sem_init	用于创建一个信号量，并能初始化它的值
sem_wait	相当于 P 操作，将信号量的值减一，会阻塞进程
sem_trywait	相当于 P 操作，将信号量的值减一，立刻返回，不会阻塞
sem_post	相当于 V 操作，将信号量的值加一，同时发出信号唤醒等待进程
sem_getvalue	获得信号量当前值
sem_destroy	删除信号量

实例源码：sem.c，将 7.6.3 节互斥锁的实例简单地变换一下，使用信号量的机制来对 lock_var 操作，使信号量为 1，其实就相当于互斥锁了。

```
/* sem.c */
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <semaphore.h>

/* 定义一个信号量 */
```

13、 IP 地址

TCP 传输层 udp

IP 网络层

tcp/ip 协议

14、 GPIO 导出

- echo 32 > /sys/class/gpio/export 将 echo 32 设为导出
- ls /sys/class/gpio/gpio32 显示
- echo out > /sys/class/gpio/gpio32/direction 置为输出
- echo 0 > /sys/class/gpio/gpio32/value 置零

15、 PWM 调整值

- 将 PWM0 占比调到最大（高电平占 100%）：
echo 255 > brightness
- 将 PWM0 占比调到最小（低电平占 100%）：
echo 0 > brightness

16、 tftp 服务
没找到

17、 fileio.c

```
编写 Makefile 文件（在虚拟机上）：
fileio : fileio.o
mpicc -linux-gcc -o fileio fileio.o
fileio.o : fileio.c
mpicc -linux-gcc -c fileio.c -o fileio.o
clean :
rm fileio.o fileio

上述 4 行字必须存为 Makefile 文件，注意第 2、4 行必须以 Tab 键缩进，不能以空格键缩进。放入与 fileio.c 相同目录下，然后执行 make 命令即可编译程序，执行“make clean”可清除编译出来的结果。

将生成的 fileio 通过 tftp 传入开发板，修改权限，运行（在开发板上）。
[root@Loongson/]# tftp -r fileio -g 193.169.2.215
fileio 100% ..... 9338 0:00:00 ETA
[root@Loongson/]# chmod a+x fileio
[root@Loongson/]# ./fileio
open file:hello.c fd = 3
write: hello,I'm Loongson.This is file io test!
read from hello.c and the content is hello,I'm Loong
close hello.c
[root@Loongson/]# ls
bin fileio lib mps6050.ko sys usr
dev hello.c linuxrc opt test2c var
devdemo.ko home hot-cound proc testmpu6050 vartdev.ko
ds3231.ko i2cutils memdev.ko root tmp
etc kshuff mmf shui tools
[root@Loongson/]# cat hello.c
hello,I'm Loongson.This is file io test!
[root@Loongson/]#

文件描述符为 3，是因为通常每一个进程都首先打开 0、1、2 文件，也就是标准输入、标准输出、标准出错处理，所以这里打开的 fd 为 3。
```

18、 pipe.c




图 7.3 进程与管道关系示意图

实例源码：pipe.c，通过 pipe 函数创建无名管道，如果成功创建则打开两个文件描述符，分别是 fd[0] 和 fd[1]，其中 fd[0] 固定用于管道读端，fd[1] 固定用于管道写端。无名管道的关闭只需要将这两个文件描述符关闭即可，就像关闭普通文件描述符那样通过 close 函数分别关闭各个文件描述符。

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main()
{
    int pipe_fd[2];
    if(pipe(pipe_fd) < 0)
    {
        printf("Pipe creat error!\n");
        exit(1);
    }
    else
    {
        printf("Pipe creat success!\n");
        close(pipe_fd[0]);
        close(pipe_fd[1]);
    }
}
```

虚拟机中编译：

```
mpicc -linux-gcc pipe.c -o pipe
```

程序使用 pipe 函数创建一个无名管道，之后再将其关闭，执行结果如下：

```
root@Loongson:/# ./pipe
```