حوری دهش 9821413

سوال 1:

الف)

Device Controller: این یک بخش سخت افزاری است و درونش بافر، رجیستر و... قرار دارد. یک بافر دارد برای اینکه اطلاعاتی که می خواهد از دیوایس ها وارد سیستم شود یا خارج بشود از طریق این بافر صورت می گیرد و یکسری رجیستر هم دارد که کنترل این دیوایس را فراهم می کند.

Device driver: این یک بخش نرم افزاری است و کاری که انجام می دهد این است که دستوراتی که از نرم افزارها دریافت می کند را به زبانی که دستگاه متوجه آن می شود ترجمه می کند و همینطور یک ارتباط بین نرم افزار با دستگاه های سخت افزاری از طریق باس سیستم برقرار می کند.

تفاوت:

Device Controller برای مدیریت و کنترل دستگاه های سخت افزاری مورد استفاده قرار میگیرد و یک بخش سخت افزاری است در حالی که Device driver یک بخش نرم افزاری است و وظیفه اش ارتباط نرم افزار با دستگاه های سخت افزاری و ترجمه دستورات نرم افزار به فرمتی که دستگاه متوجه ان بشود هست.

ب)

Multiprocessor Systems: در این سیستم از چندین واحد پردازشی برای انجام کارها و اجرای برنامه ها استفاده می کنند و این واحد های پردازشی روی یک چیپ یا سیستم کامپیوتری قرار دارند.

Clustered Systems: در این سیستم ما چندین کامپیوتر داریم که برای ارتباطشون با هم از طریق یک شبکه بهم وصل شده اند (البته کامپیوترهای کناری در حالت کلی بهم وصل شده اند و این موضوع برای مواردی است که دو کامپیوتر کنار هم نباشند) و همینطور این کامپیوترها ممکن است از منابع مشترکی بهره برداری کنند.

تفاوت:

در Multiprocessor Systems چندین پردازنده به صورت موازی کار می کنند، همینطور در صورت خرابی یکی از پردازنده ها سایر پردازنده ها ممکن است به کارشون ادامه بدهند در حالی که در Clustered Systems چون ممکن است از چندین کامپیوتر مختلف تشکیل شده باشد برای مدیریت خرابی ان باید تدابیری صورت بگیرد و قابلیت اطمینان در این سیستم ها ممکن است پیچیده تر باشد و همینطور در این سیستم می توان از چندین کامپیوتر به صورت موازی برای کارها استفاده کرده اما ارتباط بین اعضای خوشه ممکن است مسائلی مانند تاخیر را به وجود بیاورد که روی کارهای پردازشی که انجام می دهد این موضوع تاثیرگذار است.

ج)

Kernel Mode: یکی از حالت های سیستم عامل است که به هسته یا هسته اصلی سیستم عامل دسترسی دارد در این حالت می توان به تمام منابع سخت افزار دسترسی داشت و همینطور این بخش مسئول اجرای کدهای حیاتی سیستم و مدیریت منابع سخت افزار است.

User Mode: یکی از حالت های سیستم عامل است که به برنامه ها دسترسی دارد در این حالت نرم افزار کاربر اجرا میشود و همینطور در این بخش محدودیت هایی برای دسترسی به منابع سخت افزار داریم و همینطور اگر جایی نیاز به کرنل مد داشتیم یک سیستم کال فراخوانی میکنیم که به مد کرنل برود و خط هایی که مربوط به کرنل است را انجا اجرا می کند و زمانی که کارش در کرنل مد به پایان رسید به مد یوزر برمیگردد و بقیه برنامه را اجرا می کند.

تفاوت:

در Kernel Mode ما به تمام منابع سخت افزار دسترسی داریم و این بخش عملیات های حیاتی سیستم را انجام می دهد و همینطور mode bit ان صفر است در حالی که در User Mode ما محدودیت هایی برای دسترسی به منابع سخت افزار داریم و mode bit ان یک است.

سوال 2:

مشکل اول:

باعث مشکلات امنیتی میشود به این صورت که اگر یک حمله کننده موفق شود که به بخش مموری حساس دسترسی پیدا کند، او ممکن است بتواند کدهای سیستم عامل را تغییر دهد یا اطلاعات حساس را دزدی کند به عبارت دیگر این روش نمی‌تواند جلوی حملات امنیتی را بگیرد پس در عمل حمله کنندگان با تکنیک های مختلف ممکن است به بخش های این مموری دسترسی پیدا بکنند.

مشکل دوم:

سختی در ارتقا و توسعه سیستم دارد به این صورت که نگهداری کدهای سیستم عامل در بخش های خاص مموری ممکن است باعث شود که ارتقا و توسعه سیستم به سختی انجام شود و زمانی که نیاز به به روزرسانی یا تغییر در کدهای سیستم عامل وجود دارد این روش ممکن است مانع از اعمال تغییرات و اصلاحات لازم شود در نتیجه این کار باعث کاهش کارایی و امنیت سیستم میشود.

سوال 3:

وجود مدهای عملیاتی بیشتر از دو حالت ممکن است باعث شود که امکانات و کاربردهای اضافی برای سیستم عامل ها و نرم افزارهای کابری ارائه شود و به انها کمک کند همینطور این مدها به مدیران سیستم و توسعه دهندگان کمک میکند تا بتوانند عملکرد و امنیت سیستم را افزایش و بهبود دهند. برخی از کاربردهای مدهای بیشتر می تواند به صورت زیر باشد:

Supervisor Mode: این حالت به عنوان یک مد مخصوص Kernel mode شناخته می شود و به سیستم عامل اجازه می دهد که به سطح بیشتری از کنترل و دسترسی به منابع سخت افزاری دست پیدا کند. این حالت ممکن است برای انجام عملیات های پیچیده تری مورد استفاده قرار بگیرد.

Hypervisor Mode: این حالت برای مجازی سازی سیستم عامل ها و ایجاد ماشین های مجازی استفاده می شود. در این حالت یک نرم‌افزار Hypervisor به عنوان سیستم عامل اصلی اجرا میشود و VMs از Kernel mode خود استفاده می کنند و این امکان را فراهم می کند تا چندین سیستم عامل مستقل به صورت موازی روی یک پردازنده فیزیکی اجرا شوند.

سوال 4:

توضیح DMA:

فناوری DMA (Direct Memory Access) به دستگاه‌های سخت‌افزاری امکان می‌دهد تا به مموری سیستم بدون نیاز به مداخله مستقیم از سوی واحد پردازشگری (CPU) دسترسی پیدا کنند. عملکرد DMA به شکل زیر انجام می‌شود:

DMA به دستگاه های سخت افزاری این امکان رامی دهد تا با مموری سیستم بدون نیاز به مداخله مستقیم از سوی CPU دسترسی داشته باشند و این کار به این صورت انجام میشود که زمانی وقفه داده می شود که مشخص بشود این دیوایس قرار است یک سری اطلاعات را روی رم بفرستد پس یک ارتباط مستقیم بین دیوایس و مموری برقرار می شود به اسم DMA و دیگر به CPU کاری ندارد پس CPU می تواند بقیه برنامه هاش ا اجرا کند و همزمان این اطلاعات هم داره از دیوایس به مموری منتقل میشود. به عبارت دقیق تر:

برای شروع عملیات DMA، نیاز به تنظیم DMA Controller و پیکربندی آن وجود دارد. DMA Controller وظیفه مدیریت عملیات DMA را دارد. این تنظیمات شامل اطلاعاتی مانند منبع داده (مبدا)، مقصد داده (مقصد)، تعداد داده‌ها، حجم داده‌ها، و نحوه انجام انتقال داده است بعد از تنظیم DMA Controller، عملیات انتقال آغاز می‌شود. DMA Controller سیگنال‌های مناسب را به دستگاه‌های سخت افزاری و مموری می فرستد تا انتقال داده ها آغاز شود سپس دیوایس های

سخت‌افزاری مبدا داده‌ها را از مکان مبدا به مکان مقصد که همان مموری سیستم است منتقل می‌کند بعد از انجام عملیات انتقال داده، DMA Controller و دستگاه‌های سخت افزاری می توانند سیگنال تکمیل عملیات یعنی اتمام انتقال را به CPU اعلام کنند.

الف)

به وسیله DMA Controller این کار را انجام می دهد که به عنوان میان CPU و دیوایس های سخت افزاری عمل میکند و امکان انجام عملیات انتقال داده ها را بدون مداخله مستقیم از سوی CPU فراهم می‌کند که در بالا دقیق تر این را توضیح دادم که به چه صورت انجام میشود.

ب)

CPU از اتمام عملیات DMA از طریق DMA Controller به دست می اورد به این صورت که وقتی DMA Controller عملیات انتقال داده ها را به مقصد یا منبع مورد نظر به طور کامل انجام داد ان زمان عملیات DMA تکمیل می شود به عبارت دیگر DMA Controller تعیین می کند که تمام داده ها به مقصد منتقل شده اند یا از منبع به مموری سیستم کپی شده‌اند بعد از ان DMA Controller یک سیگنال تکمیل عملیات به پردازنده ارسال می کند این سیگنال به عنوان تاییدیه اتمام عملیات DMA عمل می کند و وقتی پردازنده این سیگنال را دریافت کرد به ان واکنش نشان میدهد این واکنش ممکن است شامل بررسی وضعیت عملیات DMA، تایید انجام عملیات و ادامه اجرای برنامه باشد.

ج)

اجرای همزمان عملیات DMA و برنامه ها نظر اصولی ممکن است و در واقعیت اتفاق می افتد با این حال این همزمانی ممکن است مشکلاتی را هم به وجود بیاورد:

- رقابت برای منابع به این صورت که همزمانی DMA و برنامه های کاربری ممکن است به رقابت برای منابع منجر شود به بیان دیگر اگر برنامه های کاربری نیاز به دسترسی به مموری داشته باشند و در همان زمان DMA هم به مموری دسترسی داشته باشد رقابت ممکن است منجر به تداخل و اختلال در عملکرد سیستم شود در مواردی ممکن است نیاز باشد تا اولویت های مختلفی برای دسترسی به منابع تعیین شوند.
- مورد بعدی تداخل و تنظیم همزمان است یعنی اجرای همزمان نیازمند مدیریت مناسب تداخل و تنظیم منابع است. به عنوان مثال ممکن است نیاز باشد تا سیستم های تداخلی مانند قفل ها یا سمافورها برای همزمانی مناسب بین عملیات DMA و برنامه‌های کاربری استفاده شوند.

اولویت بین عملیات DMA و برنامه ها وابسته به نیازهای سیستم و تنظیمات مشخص می‌شود. در بسیاری از موارد عملیات DMA به عنوان اولویت بالاتر در نظر گرفته می شود زیرا معمولا برای انجام

عملیات مشخصی مهمتر است اما در مواردی که برنامه های کاربری نیاز به دسترسی فوری به منابع دارند ممکن است برنامه‌های کاربری اولویت بالاتری داشته باشند.

سوال 5:

الف)

یکپارچه:

- کارایی: تمام عملیات سیستم عامل در یک کرنل بزرگ پیاده سازی میشود و همین امر سبب می شود که تبادل داده بین ان ها به شکل مستقیم انجام شود و همینطور از نظر کارایی این ساختار عملکرد بهتری نسبت به ساختار میکروکرنل دارد.
- انعطاف پذیری: این معماری محدودیت های انعطاف پذیری دارد چون تغییرات بزرگی در سیستم عامل نیاز به تغییر کرنل دارند.

لایه ای:

- کارایی: سیستم‌عامل های با ساختار لایه‌ای معمولا عملکرد بهتری از نظر کارایی ارائه می دهند به علت اینکه همه عملیات سیستم به صورت مستقیم در کرنل اجرا می شوند و تبادل داده بین لایه‌ها به شکل بسیار سریع و با کمترین هزینه انجام می‌شود.
- انعطاف پذیری: انعطاف پذیری در این ساختار کمتر است زیرا تغییرات در سیستم عامل باید در کرنل اصلی اعمال شوند که ممکن است پیچیده و خطا پذیر باشد.

میکروکرنل:

- کارایی: میکروکرنل‌ها معمولا در اجرای عملکرد نسبت به ساختار لایه ای و یکپارچه کارایی پایین تری دارند و این به علت افزایش تعداد تبادلات داده بین ماژول های جداگانه است و همینطور باید از طریق پیام از بقیه برنامه ها استفاده کرد.
- انعطاف پذیری: میکروکرنل‌ها بیشترین انعطاف پذیری را دارند زیرا امکان اضافه و حذف ماژول ها به سیستم و انجام تغییرات بدون نیاز به تغییر کرنل اصلی انجام میشود.

ب)

شباهت:

- تقسیم وظایف: هر دو معماری به تقسیم وظایف نرم افزار به بخش های کوچکتر و ماژول ها یا لایه ها متصل به یکدیگر می پردازند.

- کاهش وابستگی: هر دو معماری به کاهش وابستگی میان ماژول ها یا لایه ها و تعاملات بهینه بین انها تاکید می کنند.
- جداگانگی منطقی: هدف اصلی هر دو معماری ایجاد جداگانگی منطقی بین ماژول ها یا لایه ها است تا بهترین تفکیک وظایف و مسئولیت ها برای هر بخش داشته باشند.

تفاوت:

- در معماری ماژولار تبادل اطلاعات بین ماژول ها ممکن است به صورت مستقیم و از طریق APIها صورت گیرد ولی در معماری لایه ای انتقال اطلاعات بین لایه ها از طریق رابط های لایه ای و محدود شده انجام می شود.
- معماری ماژولار به دلیل ماژول های جداگانه و ارتباط مستقیم انها ممکن است پیچیدگی بیشتری داشته باشد ولی معماری لایه‌ای معمولا سازمان دهی مشخص تری دارد و کاهش پیچیدگی را دارد.
- در معماری ماژولار نرم افزار به صورت یک توده از ماژول ها سازماندهی می شود و هر ماژول مسئولیت های خود را دارد و از طریق واسط‌ها یا APIها با ماژول های دیگر ارتباط برقرار میکند ولی در معماری لایه ای نرم افزار به صورت لایه ها سازماندهی می شود و هر لایه وظایف خاص خود را دارد و اطلاعات و درخواست ها از یک لایه به لایه بالاتر انتقال می یابند.

سوال 6:

الف)

ابزار strace

ب)

برنامه:

```c
#include <stdio.h>

int main(){
        char username[50];


        printf("user name: ");
        scanf("%s", username);

        printf("user name:%s \n", username);
        //write(STDOUT_FILENO, username, strlen(username));

        //free(username);

        return 0;
}
```

سیستم کال ها:

```
oem@hoori:~$ strace ./cc
execve("./cc", ["./cc"], 0x7ffe0626af50 /* 50 vars */) = 0
brk(NULL)                               = 0x556951b61000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd920fd470) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=75237, ...}) = 0
mmap(NULL, 75237, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f67df454000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\300A\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\30x\346\264ur\f|Q\226\236i\253-'o"..., 68, 880) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2029592, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f67df452000
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\30x\346\264ur\f|Q\226\236i\253-'o"..., 68, 880) = 68
mmap(NULL, 2037344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f67df260000
mmap(0x7f67df282000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7f67df282000
mmap(0x7f67df3fa000, 319488, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) = 0x7f67df3fa000
mmap(0x7f67df448000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f67df448000
mmap(0x7f67df44e000, 13920, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f67df44e000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f67df453540) = 0
mprotect(0x7f67df448000, 16384, PROT_READ) = 0
mprotect(0x556950d8a000, 4096, PROT_READ) = 0
mprotect(0x7f67df494000, 4096, PROT_READ) = 0
munmap(0x7f67df454000, 75237)           = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...}) = 0
brk(NULL)                               = 0x556951b61000
brk(0x556951b82000)                     = 0x556951b82000
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...}) = 0
write(1, "user name: ", 11user name: )           = 11
read(0, 0x556951b616b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0,
```

ج)

1. execve("./cc", ["./cc"], 0x7ffe0626af50 /* 50 vars */) = 0 : این سیستم کال نشان دهنده اجرای برنامه ای با نام `cc/.` است. "execve" برای اجرای یک برنامه استفاده می شود.
2. write(1, "user name: ", 11user name:) = 11 : این سیستم کال برای نوشتن داده ها به ترمینال (فایل با ملزومات `1`) استفاده می شود.
3. close(3) = 0 : این سیستم کال برای بستن فایل با ملزومات `3` استفاده می شود.

سوال 7:

1. انتقال پارامترها به عنوان رجیسترها یا Register Passing:

در این روش پارامترهای مورد نیاز برای اجرای system call به صورت مستقیم در رجیسترهای معین CPU قرار می گیرند سپس نرم افزار سیستم کال را صدا می زند و سیستم عامل به پارامترها در رجیسترها دسترسی دارد و اقدامات مورد نیاز را انجام میدهد. همینطور این روش سریعترین روش انتقال پارامترها به سیستم عامل است زیرا بدون نیاز به دسترسی به حافظه برای خواندن پارامترها از ادرس های حافظه مشخص، مستقیما از رجیسترها استفاده می کند.

2. انتقال پارامترها به عنوان بلوک های حافظه یا Memory Block Passing:

در این روش پارامترها به عنوان یک بلوک حافظه در ناحیه ای مشخص از حافظه ذخیره می‌شوند سپس سیستم عامل پس از فراخوانی system call به این ناحیه دسترسی دارد و از ان پارامترها را می خواند.

3. انتقال پارامترها به عنوان پشته یا Stack Passing:

در این روش پارامترها در پشته قرار می گیرند. نرم افزار پارامترها را در پشته قرار داده و سپس سیستم کال را صدا می زند، سیستم عامل سپس از پشته برای دریافت پارامترها استفاده می کند. این روش به نوعی از جلوگیری از تداخل با دیگر رجیسترها استفاده می کند و همینطور انعطاف پذیرتر از انتقال پارامترها به عنوان رجیسترها است.

سوال 8:

در سیستم‌های با معماری SMP ممکن است داده موجود در حافظه اصلی مقادیر مختلفی در کش های محلی پردازنده ها داشته باشد که این مسله به علت ماهیت مشترک حافظه و دسترسی همگن به حافظه در سیستم‌های SMP رخ می‌دهد.

داده ها می‌توانند مقادیر مختلفی در کش های محلی داشته باشند به علت عدم همگرائی (Cache Coherency) و ترتیب دسترسی به حافظه. برای مثال:

فرض کنید دو پردازنده A و B در یک سیستم SMP وجود دارد و یک متغیر به نام "x" در حافظه وجود دارد که ابتدا مقدارش 5 است. پردازنده A دسترسی به متغیر "x" دارد و مقدار 5 را از حافظه می خواند و این مقدار در کش محلی پردازنده A ذخیره می شود. همزمان پردازنده B نیز دسترسی به متغیر "x" دارد و مقدار 5 را از حافظه می خواند و این مقدار در کش محلی پردازنده B ذخیره می شود.

در این مرحله، همه چیز به نظر مرتب و هماهنگ است اما اگر پردازنده A مقدار متغیر "x" به 10 تغییر دهد، این تغییر در کش محلی پردازنده A اعمال می‌شود. در این لحظه، کش محلی پردازنده A دارای مقدار 10 برای متغیر "x" است اما مقدار متغیر "x" در حافظه همچنان 5 است. پس در حالت کلی مقدار x به صورت های زیر می باشد:

در حافظه = 5

در کش محلی پردازنده B = 5

در کش محلی پردازنده A = 10

این موضوع به علت عدم همگرائی کش ها رخ می‌دهد برای حل این مشکل، سیستم‌های SMP از مکانیزم‌های همگرایی مانند "MESI" (Modified, Exclusive, Shared, Invalid) استفاده می کنند و این مکانیزم ها به پردازنده ها اجازه می دهند تا به درستی اطلاعات را با هم به اشتراک بگذارند و همگرا شوند.

سوال 9:

الف)

در Arduino به صورت اولیه و بدون تغییر در سیستم عامل، امکان Multitasking و اجرای همزمان چندین وظیفه به صورت استاندارد وجود ندارد.

ب)

در Arduino و برنامه نویسی ان برنامه ها با اجرای یک تابع به نام setup شروع به کار می کنند. این تابع setup یک بار در ابتدای اجرای برنامه اجرا می شود و برای تنظیمات اولیه و مقدماتی برنامه استفاده می شود به عبارت دیگر، این تابع برای تنظیم پورت ها، تنظیمات سریال، اتصال به پریفرال‌ها و سایر تنظیمات اولیه برای میکروکنترلر استفاده می شود.

پس از اجرای تابع setup، برنامه به اجرای یک تابع به نام loop می پردازد. این تابع loop یک حلقه بی پایان تشکیل می دهد و برنامه در داخل این حلقه به صورت مکرر اجرا می شود. در واقع تمام عملکرد اصلی برنامه و کنترل میکروکنترلر در داخل این حلقه و توابع مرتبط با ان قرار دارد پس عملکرد اصلی برنامه در Arduino در داخل تابع loop قرار دارد و این تابع به صورت بی پایان اجرا می‌شود هرچه که در داخل این تابع قرار داده شود، به صورت مکرر توسط میکروکنترلر اجرا می‌شود و برنامه به صورت نمایشی شبیه به یک "حلقه اصلی" عمل می کند تا زمانی که برنامه به کمک دستورهای خروجی یا توقف های مشخص متوقف شود.

همینطور در Arduino و سخت افزارهای مشابهی که برای کنترل میکروکنترلرها به کار می روند مفهومی مانند سیستم عامل جایگاهی ندارد و این سخت افزارها به عنوان میکروکنترلرها عمل می کنند و برنامه ها به صورت بی سیستم عامل و بدون واسطه بر روی ان ها اجرا می شوند.

ج)

تمام اون مواردی که برای قسمت ب گفتم شامل این قسمت هم میشود اگر بخواهم چیزی بیشتر از اون بگویم به صورت زیر میشود:

در برنامه‌های Arduino می توان توابع دیگری نیز تعریف کرد که به صورت مستقل، از تابع های setup و loop اجرا میشوند. این توابع می توانند برای انجام وظایف خاص و جداگانه به کار بروند. به عنوان

مثال، اگر بخواهیم یک عملیات خاص را انجام دهیم می توانیم ان را در یک تابع جداگانه تعریف کنیم و آن را در داخل تابع های setup یا loop فراخوانی کنیم.

در مجموع Arduino به توسعه دهندگان اجازه می دهد که برنامه هایی با عملکرد مکرر اجرایی را ایجاد کنند و از اجزای مختلف سخت افزاری و پریفرال‌ها استفاده کنند. توابع setup و loop، میکروکنترلر Arduino را به توسعه دهندگان ارائه می دهند تا بتوانند ارتباط با سخت افزار و اجزا خارجی را مدیریت کنند و وظایف مختلف را اجرا کنند.

همچنین برای ارتباط با سخت افزار در Arduino می‌توان از کتابخانه های مختلفی استفاده کرد که توسط جامعه Arduino توسعه داده شده اند. این کتابخانه ها، ماژول ها و سنسورها را با برد Arduino ارتباط می دهند و کدهای اماده برای ارتباط و کنترل این سخت افزارها را فراهم می کنند. به عنوان مثال برای اتصال به ماژول های Wi-Fi، می توان از کتابخانه های Wi-Fi موجود برای Arduino استفاده کرد، این کتابخانه ها دسترسی به شبکه های بی سیم را فراهم می کنند و امکان ارسال و دریافت داده ها را از طریق اتصال Wi-Fi فراهم می کنند.

سوال 10:

تفاوت:

API یک واسط نرم افزاری برای تعامل برنامه ها با یکدیگر یا با سیستم‌های دیگر است ولی ABI یک موازی بین برنامه ها و سیستم عامل یا سیستم های عاملی است که برنامه ها به صورت دودویی با ان ها تعامل دارند.

API به برنامه نویسان این امکان را می دهد تا از خدمات و عملکردهای سیستم های دیگر یا کتابخانه های نرم افزاری به راحتی استفاده کنند بدون اینکه به جزئیات پیاده سازی دسترسی داشته باشند ولی ABI شامل قواعد و استانداردهایی است که تعیین می کنند چگونه برنامه ها باید اطلاعات را در حافظه ذخیره و به ان ها دسترسی داشته باشند، چگونه توابع و روش ها فراخوانی شوند و مسائل دیگری مربوط به اجرای برنامه ها.

API معمولا در سطح منطقی و تعامل برنامه به برنامه تعریف می شود و برای برنامه نویسان در زبان های برنامه نویسی مختلف قابل استفاده است ولی ABI برای اطمینان از سازگاری میان برنامه ها با سیستم های عامل و معماری های مختلف استفاده می شود یعنی برنامه های کامپایل شده برای یک ABI می‌توانند بر روی سیستم هایی با همان ABI اجرا شوند.

پس API در سطح منطقی تعامل برنامه به برنامه را تعریف می کند و برای توسعه برنامه ها استفاده می شود و ABI در سطح دودویی اجرای برنامه ها توسط سیستم عامل و معماری سخت افزار تعیین می کند و اطمینان می دهد که برنامه ها بتوانند به درستی واحد دودویی تفسیر شده اجرا شوند.