به نام خدا

اصول طراحی سیستم های نهفته
هموورک اول
۱۸ اردیبهشت ۱۳۹۹

استاد درس : دکتر محسن راجی
نویسنده : حسین دهقانی پور - ۹۵۳۲۲۵۰

فهرست مطالب

This document is written by LaTeX.

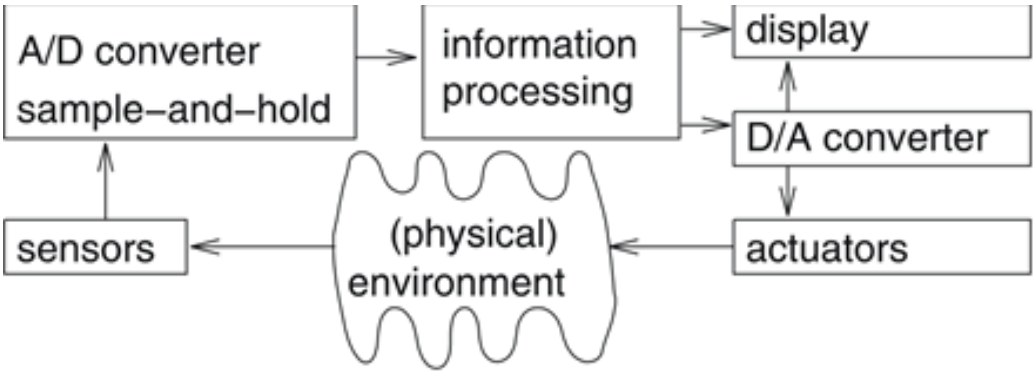
۱ سوال اول

چرا باید یک سیستم نهفته از نظر اندازه کد نرم افزاری بهینه و کارآمد باشد؟

چون نمیتونیم حجم زیادی از حافظه سیستم رو بزاریم فقط برای پردازنده علی الخصوص برای سیستم هایی که روی یک چیپ خلاصه میشن.

۲ سوال دوم

راجع به این شکل توضیح دهید.



توضیح دادن این سوال خودش یه جلسه کامل کلاسه ☺ . بطور خلاصه این شکل داره در مورد سخت افزار Cyber Physical System ها و Embedded System ها صحبت میکنه. بطوری کلی داره میگه ما از طریق یه سری سنسور های موجود در محیط اطراف (Physiscal Environmet) یه سری اطلاعات دریافت میکنیم. این اطلاعات از نوع سیگنال آنالوگ هستن و سیستم ما فقط با سیگنال دیجیتال میتونه کار کنه برای همین این سیگنال های دریافتی رو میدیم به یه مبدل تبدیل آنالوگ به دیجیتال که به صورت Sample-and-Hold کار میکنه.
سپس سیگنال های تبدیل شده به سیگنال دیجیتال وارد واحد پردازشی (Information Processing Unit) میشه. حالا اطلاعات دریافتی از محیط پردازش شدن و یه نتیجه ای دربر دارن . ما میتونیم با استفاده از اون اطلاعات یه سری کارا انجام بدیم. اگر هدف فقط نشون دادن نتایج به کاربر باشه که میایم روی یه صفحه نمایش (Display) اطلاعات رو نشون میدیم. ولی اگر قرار باشه که به فرض با استفاده از نتایج یه کارای جدی تری انجام بدیم (مثل وندینگ ماشین که یه چیپسی بندازه پایین و یا یه دماسنج که با استفاده از دمای محیط بیاد میزان آب پاشی باغچه رو تنظیم کنه) میایم نتایج دیتای آنالیز شده رو میدیم به یه مبدل دیجیتال به آنالوگ و بعد اون سیگنال آنالوگ رو میدیم به یه چیزی بنام Actucator که وظیفش اینه که یه کارایی روی محیط انجام بده (مثلا سرعت آبپاش رو کم کنه یا اون میله ای که چیپس رو نگه داشته به اندازه ۹۰ درجه بچرخونه که چیپس بیفته پایین) و این چرخه بصورت یک لوپ مدام در حال تکرار شدنه.

۳ سوال سوم

راجع به ارتباط از نوع حافظه اشتراکی (Shared Memory Communication) توضیح دهید و مشکل این نوع ارتباط را با توجه به تکه کدهای زیر شرح دهید.

```
thread a {
  u = 1; ..
  P(S)  //obtain mutex
  if u<5 {u = u + 1; ..}
  // critical section
  V(S)  //release mutex
}
```

```
thread b {
  ..
  P(S)  //obtain mutex
  u = 5
  // critical section
  V(S)  //release mutex
}
```

مشکلی که سر این سوال مطرح شد درمورد دوتا Component بود که از shared Memory برای communication استفاده میکردن. بحثی که مطرح بود این بود که این دوتا کامپوننت یه متغیر مشترکی تحت عنوان u دارن. طبق کد پایین ما انتظار داریم که u بیشتر از ۵ نشه ولی در حالتی که اگر سمافور وجود نداشته باشه مقدار u میتونه بیشتر از ۵ باشه . به این صورت که تو ترد a داشته باشیم $u = 4$ و تا سر شرط if بیایم پایین . شرط چک میشه true برمیگرده که بله u برابر با ۴ هست و کوچکتر از ۵ هست . ما الان شرط رو چک کردیم و اوکی بود ولی تو لحظه اجرا میتونه یه Context Switch اتفاق بیفته و بریم توی ترد b . توی ترد b مقدار u درجا برابر با ۵ میشه و وقتی برمیگردیم به ترد a دیگه شرط چک نمیشه (چون قبلا چک شده و true برگشت داده شده) پس یه راست میره برای اضافه کردن و مقدار u میشه ۶ . این مشکلی بود که توی shared memory بوجود میومد. برای پیشگیری از این مشکل یه ابزاری اومد وسط تحت عنوان سمافور که وظیفش این بود که اون تیکه هایی از کد رو که متغیر های مشترکی دارن رو در یه زمان محدود کنه فقط به یه ترد (تا مانع بشه از اینکه دو تا ترد همزمان روی یه متغیر کار کنن و دیتای غلط رو برگردونن) سمافور(کدهای خاکسرتی رنگ) کارش این بود که اگر ترد a داره روی متغیر u یه سری پردازش انجام میده دسترسی بقیه ترد هارو به u ببنده و این کار رو از طریق mutex انجام میده. یعنی ترد a میره تو mutex و در رو به روی بقیه ترد ها میبنده و با ارامش کارش رو انجام میده. وقتی هم که کارش انجام شد میاد بیرون و mutex رو واگذار میکنه به یکی دیگه و بقیه هم به همین منوال کارشون رو انجام میدن. گرفتن میوتکس از طریق دستور $P(S)$ و رها کردن میوتکس از طریق دستور $V(s)$ انجام میشه.