دفتر فنی **ایلیا**

با بیش از یکهزار عنوان جزوه در کلیه رشته های دانشگاهی

# سیستم بلادرنگ

استاد زمرّدیان

**کد: ۱۶۹**

| | |
|---|---|
| کپی و پرینت رنگی لیزری | پروژه و تحقیق |
| تایپ (جدول، فرمول، نمودار) | ترجمه (تخصصی و ...) |
| اسکن تا سایز **A3** | صحافی (فنر، سیمی فلزی، گالینگور) |
| پرس تا سایز **A3** | لوازم مهندسی و تحریر |
| تبدیل **Word** به **PDF** | بانک نرم افزار |

**شعبه ۱:** نبرد جنوبی – خیابان ده حقی – روبروی دانشگاه آزاد – پاساژ دانشجو – پلاک ۵ ۳۳۶۹۸۷۲۱

**شعبه شکوه:** خیابان انقلاب – روبروی لاله زار نو – بن بست شکوه – پلاک ۷ ۸۸۱۴۰۹۵۱

بسمه تعالی
بلادرنگ

Subject:
Year. Month. Date. 1/1

Real Time system & Software — استاد زمردیان :
Alan C. Shaw

جلسه اول ۹۰/۷/۶ :

المان های سخت افزاری :

برای روشن کردن کامپیوتر دکمه power را می‌زنیم چه اتفاقی می‌افتد؟
BIOS سیستم ورودی خروجی پایه
I,O های پایه را که برنامه هایشان را داریم. BIOS داخل ROM قرار دارد.
CPU با ROM کاری ندارد. حافظه اصلیش RAM است.

به جای ROM از PROM و E²PROM استفاده می‌شود. ولتاژ پاک کردن E²PROM ها با ولتاژ خواندن آنها از لحاظ سخت افزاری فرق دارد.

Electrical Erasable programmable Read only mem: E²PROM

E²PROM ها توسط جریان برق اطلاعاتشان پاک می‌شود.

امروزه از حافظه های Flash استفاده می‌شود. این حافظه ها با همان ولتاژ 5V هم نوشته می‌شوند هم خوانده می‌شوند هم پاک می‌شوند...!

سوال ← زمانی که PROM بود چطور این اتفاق می‌افتاد؟
جواب:
معلم در Flash می‌شود write و Read کرد و از زمانی که PROM به وجود آمد کدهای BIOS به وجود آمدند. یعنی اولین BIOS ها از نوع PROM بودند.
در PROM نمی‌شود نوشت!

در کنار این PROM یک ساعت هم همیشه کار می‌کند. زمان clock pulse که برای
هماهنگ سازی داشتیم!

منظورمان clock سیستم است که زمان روز و تاریخ و ساعت را نشان می‌دهیم.
Updat آن ساعت به وسیله‌ی این IC انجام می‌شود. این IC باید زمان را دائماً تغییر
بدهد تا بتواند زمان را نگه دارد.

پس در کنار این IC یک باتری است. در واقع از نوع RAM بود و یک باتری در کنار آن.
اصطلاحاً به این ساختار NVRAM می‌گوییم.

حافظه‌هایی که موقع PROM استفاده می‌شد، اطلاعات در داخل NVRAM ذخیره می‌کند
برای همین در آن زمان اگر باتری کامپیوتر را در می‌آوردید، حتی password setup هم
پاک می‌شد. الان دیگر اینطور نیست.

* NV مخفف Non-volatage ← وابسته به ولتاژ نیست.

* بین ۲ جور RAM داریم:
- Non volatage: وابستگی به ولتاژ ندارند، چون باتری همراهشان هست، به کار خود ادامه می‌دهند
- Volatage: با از بین رفتن ولتاژ و یا قطع برق، اطلاعات داخلش valid نیست!

ساعت سیستم در واقع یکی از حافظه‌هایی هست که نباید به ولتاژ وابستگی داشته باشد، وقتی این
حافظه‌شون را می‌گیریم، ساعت سیستم عددش عوض نمی‌شود. پس NVRAM است.

? تفاوت NVRAM و Flash چیست؟

جواب: در Flash ها تعداد Read و write محدود است. 100000 بار است، یعنی
اگر 100,000 بار در Flash تان write و Read کنید Flash خراب می‌شود
با اینکه این عدد، عدد کمی نیست، اما باز هم خراب می‌شود.
حالا تصور کنید این Read و write بخواهد هر ثانیه انجام شود، اگر

Subject:
Year. Month. Date. ۳ ( )

قرار بر این شد از Flash استفاده کنند این ساعت را نگه دارند و هر بار که بخواهد پاک شود ببرند روز تقریباً بعد از این Flash را دور بیندازیم. حال آن که ساعت کامپیوتر تا همیشه باید ادامه دارد ... !!!

در واقع password در این چیپ‌های سیستم روی Flash ذخیره می‌شود.
ما در سیستم‌های جدید هم این باتری 3V را می‌بینیم. اگر باتری تمام بشود ساعت به هم می‌خورد ولی password از بین نمی‌رود.

← وقتی که سیستم می‌خواهد بالا بیاید اولین اتفاقی که می‌افتد این است که باید یک برنامه‌ای اجرا شود. آن برنامه در داخل همین Flash قرار دارد. این Flash، Load می‌شود در داخل RAM اصلی.
حالا برای این که این برنامه بخواهد اجرا بشود باید pc (program counter) یا Ip (Instruction pointer) که می‌گوید برنامه‌ی ما در کجا قرار دارد (اول کار برنامه در آدرس مشخصی نوشته شده که شروع بشود به اجرا شدن) و Reset کردن آدرس را می‌برد روی صفر حافظه.
پس برنامه‌ی این Flash در داخل صفر حافظه نوشته می‌شود.
وقتی که شما کامپیوتر را روشن می‌کنید در واقع کامپیوتر یک بار Reset می‌شود و کامپیوتر وقتی Reset شد، pc می‌رود در حافظه و شروع می‌کند آدرس داخل و اجرا شدن برنامه. قبل از آن اطلاعات یک قسمت از Flash، کل Flash در داخل حافظه ریخته شده.
حالا شروع می‌کند برنامه را اجرا کردن.
اینها همه سخت‌افزاری است.
خب به هر حال، عملی انجام می‌شود بعد cpu را از یک حالت تعلیق درمیارد.
cpu می‌رود سراغ آدرس صفر را اجرا می‌کند.
خب بعد از اینکه آدرس صفر را اجرا کرد چه اتفاقی می‌افتد؟
یک قسمت از این برنامه مسأله سخت‌افزار باید باشد
پس دیگر شما cpu نداشته باشید سخت‌افزار تست نمی‌شود.
پس خود cpu است که دارد سخت‌افزار را تست می‌کند.

PAPCO

Subject:
Year. Month. Date. ۴ ( )

خب. سخت افزار تست کردن یعنی : post : power on self Test

یعنی بعد از روشن کردن دستگاه خودش را تست می کند. یکی از کارهای خیلی اساسی روی سخت افزارهای خیلی حرفه ای است.

خب وقتی self Test انجام شد و فهمید Hard درسته، ROM و cpu و ... همه چی درسته، بعد Boot می شود.
خب پس Poweron s.T فقط می گه که همه چی درست است فقط Test می کنه یا درست است یا نه!

IP را می برد به اول حافظه آدرس باید شروع کند به programming از اینجا دیگر کار سخت افزار تمام می شود.

اگر در حوزه سخت افزار کد می نویسید در حوزه سخت افزار نباید برنامه تمام شود. مثلاً کامپیوتر را روشن می کنید و شروع می کنید به اجرا کردن یکی از برنامه‌ها. حالا اگر مثلاً سیستم عامل تمام شود چه می شود؟ کامپیوتر خاموش می شود.
* پس برنامه تمام نمی شود و سیستم عامل حتماً در داخل یک while به شکل زیر قرار می گیرد

```
while(1)
{
    کدها و interrupt ها داخل این while قرار می گیرد
}
```

سوال: در برنامه های اسمبلی ۳ نوع End داریم:
۱- End برنامه
۲- End سگمنت
۳- ۲ خط کد می نویسیم که از برنامه خارج شویم

Subject:
Year. Month. Date. ۹۵ ( )

این ۳ تا هر کدام به یک دلیل است. End آخر برنامه به چه دردی می‌خورد؟

← جواب: این End آخر برنامه مانند علامتی است که برنامه تمام شد، دیگر کامپایل نکن!
۲ تا End دیگر در واقع interrupt برای تابع است.

* حالا می‌خواهیم ببینیم برنامه‌های windows چطور کار می‌شود و چرا برنامه‌های windows را نمی‌توانیم RealTime بدانیم؟

اگر برنامه‌ها به طور کلی سیستم تحت DOS باشد، سیستم اجرا از زمان شروع، در اختیار Application قرار می‌گیرد تا Exit را بزنیم. اگر خروج را نزنند OS نمی‌داند که این برنامه تمام شده و فکر می‌کند هنوز باید کدهای بعدی را اجرا کند. چون حافظه پاک نشده، کدهای بعدی هم معلوم نیست چیست و این کدها شروع می‌کنند به اجرا کردن.
در نهایت به هر صورت یا کامپیوتر هنگ می‌کند یا برنامه جدید بر روی برنامه قبلی خرابکاری می‌شه

پس کل سیستم از آن زمانی که کامپیوتر شروع می‌کند برنامه‌های کاربردی‌اش را اجرا کردن تا انتها دست کاربر است.

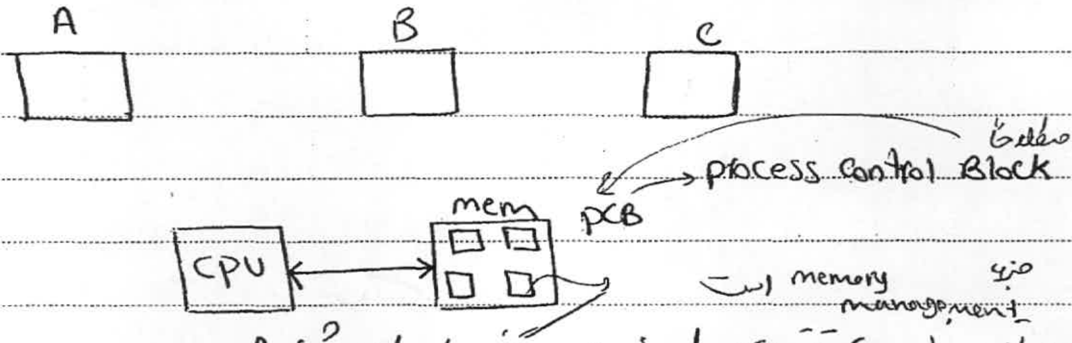
اما در windows: اینطور نیست. وقتی یک برنامه‌ی کاربردی اجرا می‌شود سیستم همیشه حول
آن برنامه ---- ؟

پس بهتر است وقتی می‌خواهیم یک مسأله RealTime را حل بکنیم آن را روی DOS حل کنیم،
چون می‌توانیم اختیار سیستم را به دست بگیریم، کارمان را انجام دهیم بعد آزادش کنیم.

پس روی DOS برنامه‌نویسی آسان‌تر است. حالا روی windows چه اتفاقی می‌افتد
Real Time

Subject:
Year.　Month.　Date. ۴/۱

در windows چه اتفاقی می‌افتد؟



به عبارتی برنامه‌های تحت windows چگونه اجرا می‌شوند؟

در windows، همانند multi Tasking داریم. دیگر یک برنامه نیست، چندین برنامه همزمان می‌خواهد اجرا شود. مثلاً در شکل بالا برنامه A,B,C را داریم و حافظه و cpu ...

واحد اجرایی کامپیوتر cpu است. cpu: central process unit

- cpu از بخش‌های زیر تشکیل شده است:

cpu {
۱) CU ← control unit
۲) ALU ← Aricmatic logic unit
Arithmatic => ?

ALU: عملیات منطقی و <=> و ... را انجام می‌دهد و جمع و تفریق و مقایسه و ...

کار CU: {
۱- کنترل کلی
۲- کنترل داخلی. مثلاً jmp یک دستور کنترلی است. باید بگوید که pc کجا برود.

اطلاعات کامل: Reg ها + stack هر برنامه + حافظه Heap و ...
هر قسمت از برنامه که بخواهد مستقلاً اجرا شود یک process است. پس مثلاً به برنامه process می‌گوییم.

مثلاً وقتی یک keyboard که Resource خارجی است وارد می‌شود OS حتماً باید بفهمد که این مربوط به کدامیک از A یا B یا C است؟
در GUI windows باید برنامه active بودن را بفهمد.

Real time sys. : از طریق sensor ها با محیط در تعامل است .

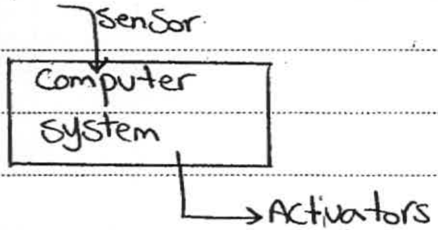
Subject:
Year. Month. Date. ۷(۱) ۹۰/۷/۲۰

جلسه دوم :

شبه توازی یعنی ما موازی می بینیم اما موازی اجرا نمی شوند .
سیستم عامل windows ، RealTime نیست اما لینوکس می تواند RealTime باشد
خیلی از آنها به صورت RealTime هستند و RealTime پیاده سازی شده اند .

---

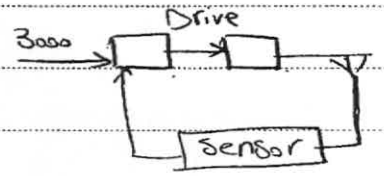
سیستم های بلادرنگ سیستم های کامپیوتری هستند که عمل monitoring ، پاسخ و یا کنترل محیط خارجی را بر عهده دارند . این محیط خارجی با استفاده از سنسورها ، عملگرها و رابطه های ورودی خروجی دیگر به سیستم کامپیوتری متصل می شود*

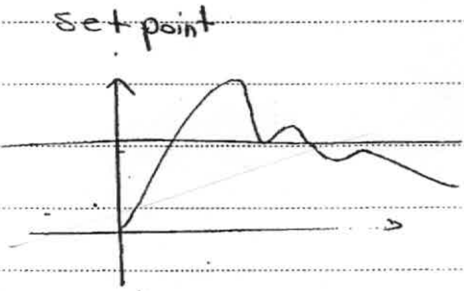


محیط خارجی که می تواند شامل وسایل فیزیکی و بیولوژیکی در هر ساختار و شکلی باشد .
اغلب انسان ها بخشی از محیط خارجی هستند که به صورت کلی هر وسیله طبیعی یا مصنوعی دیگری نیز می تواند بخشی از محیط خارجی باشد .
سیستم های کامپیوتری با شرایط و محدودیت های زمانی متعددی مواجه می شوند که این محدودیت ها توسط رفتارهای بلادرنگ محیط خارجی که مرتبط با سیستم های کامپیوتری هستند تحمیل می شود .

---

مثال برای این محیطها : مثل سیستم آسانسور . اینکه آسانسور حتما باید سر هر طبقه بایستد نه بالاتر یا پایین تر .





سیستم magler ( ∝ magnetic یعنی تعلیق مغناطیسی ( در قطارهای سریع السیر

multimedia
۱۰ صفحه

Subject:
Year. Month. Date. ۸/۱

کاربردی شده است. یعنی چون قطار به ریل وصل نیست و هیچ اصطکاکی ندارد با حداکثر سرعت ممکن حرکت می‌کند و با نیروی کم قطارهای بزرگ حرکت می‌کند.

---

* نام دیگری که به این سیستم‌ها داده می‌شود سیستم‌های واکنشی یا Re active system است به دلیل اینکه هدف اصلی این سیستم‌ها پاسخ دادن یا واکنش نشان دادن به سیگنال‌های است که از محیط خارج دریافت می‌شود.
سیستم کامپیوتری بلادرنگ می‌تواند به عنوان جزئی از یک سیستم بزرگتر باشد که در داخل کل سیستم جاسازی شده است.
به این نوع سیستم‌ها embeded system می‌گویند. مثل ماشین لباسشویی.

---

در خودروهای جدید ۸۲ تا embeded system وجود دارد.
در داخل موتور ماشین پیستون و سیلندر دائم و کروز کنترل (توضیح در Recorder)
ABS هزار بار در ثانیه عمل ترمز را انجام می‌دهد.
چون ضریب اصطکاک ایستایی > حرکتی است
ABS مخفف (Anti Block System) (سیستم ضد قفل) است. چرخ‌های ماشین را قفل نمی‌کند که ماشین سُر نخورد.
ABS های جدید روی تک تک چرخ‌ها کنترل دارد که لااقل یکی از چرخ‌ها Block نشود مثل وقتی‌که روی یخ ترمز کنیم، ماشین بچرخد.

---

* کاربردهای سیستم‌های بلادرنگ در همه جا وجود دارد و به عنوان بخشی از ساختارهای اقتصادی، دولتی و نظامی و ... قرار گرفته. به عنوان مثال سیستم‌های وسایل نقلیه از قبیل ماشین، مترو، هواپیما، قطار، کشتی، سیستم‌های کنترل ترافیک برای بزرگراهها، خطوط هوایی، خطوط دریایی و ...
کنترل فرایند نیروگاه‌های برق، کارخانجات، مواد شیمیایی و غیره.
سیستم‌های پزشکی از قبیل پرتودرمانی، مانیتورینگ بیمار (ضربان قلب و ...)
کاربردهای نظامی مثل ردیابی و فرماندهی و کنترل.

Subject:
Year. Month. Date. 9 ( )

ساخت سیستم ها با روبات

در صنعت ارتباطات ، تلفن ، رادیو و ماهواره ، بازی‌های کامپیوتری ، سیستم‌های چند رسانه‌ای ، مدیریت ساختمان یا BMS ( Building management system )

جلسه سوم : ۹۰/۷/۲۷

تفاوت نرم افزارهای بلادرنگ با نرم افزارهای عادی :

سخت افزار سیستم های بلادرنگ شامل واحدهای حافظه ـ پردازنده ـ وسایل جانبی و پردازنده ورودی خروجی مانند سنسورها و ادغام گرها می باشند.
بحث اصلی ما نرم افزار یا برنامه اجرایی است که بر روی این سخت افزار اجرا می شود

نرم افزارهای بلادرنگ از جهاتی با نرم افزارهای عادی تفاوت دارند.
① ابتدا می توان به نقش اصلی محدودیت های زمانی اشاره کرد. نه تنها باید جواب یا خروجی سیستم درست باشد بلکه می بایست باید در زمان تعیین شده انجام داده. به عبارت دیگر در واقع باید هم از نظر منطقی و هم از نظر زمانی درست باشد

deadline / deadline

نرم افزارهای بلادرنگ را می توان در طیفی بین Soft و Hard قرار داد.

سیستم های بلادرنگ Hard به سیستم هایی گفته می شود که بدون هیچ استثنائی باید به درستی در زمان های تعیین شده عمل کنند. اگر محدودیت زمانی نقض شد سیستم با شکست مواجه می شود

در طرف مقابل سیستم های بلادرنگ Soft قرار دارند علیرغم اینکه بعضی از محدودیت های زمانی را از دست دادند ، همچنان می توانند موفق باشند و تنها کارایی سیستم کم می شود.

PAPCO

Neda Dorin Mina

joy

Subject:
Year. Month. Date. ۱۵/۱

به طور کلی خاصیت‌های سیستم‌های Real Time :

۱_ Real Time بودن
۲_ Concurrency
۳_ Reliability
۴_ Fault tolerance
۵_ Testing & Certification

توضیح اون کلی جزوه رو بکون

تعریف مهندسی نرم‌افزار: راه و روش یا متدولوژی ایجاد نرم‌افزار می‌باشد. نحوه‌ی طراحی و ساخت بر اساس life cycle نرم‌افزار می‌باشد.

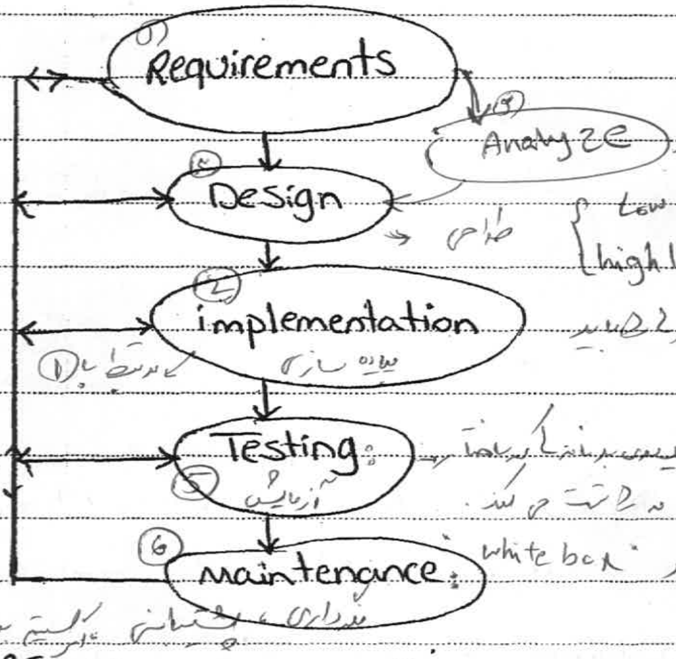
requirment ← life cycle

PAPCO

Subject: جلسه چهارم
Year. Month. Date. 11/10

## معماری نرم‌افزار Real Time System

## Life cycle


①
Requirements
④
Analyze
②
Design
طراحی
③
implementation
پیاده سازی
ارتباط با ①
⑤
Testing
آزمایش
⑥
maintenance
نگهداری، پشتیبانی

تعیین نیازمندی ها {
{ Low Level ① → OS
{ high level ②

در این مرحله اشکالات را باید تشخیص دهیم.

سیستم بر اساس نیازها برنامه را تست می کند.

به دو صورت Blackbox و white box هستند.

مثلاً عمل Sort برعکس درست اجرا شد یا خیر؟

همه مراحل تست می کنند.

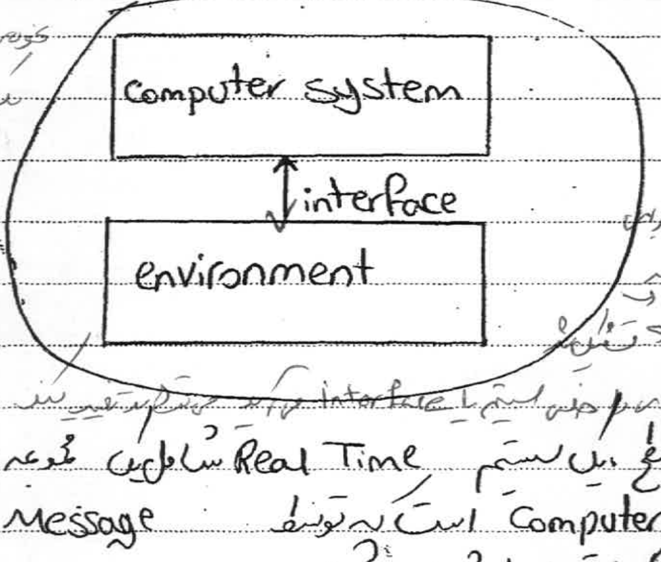
اگر سیستم بر اساس تغییرات نیاز قابل پیش بینی تغییر کرد

توصیف ساختاری سیستم، ساختار و روش های المان های تشکیل دهنده سیستم را مشخص می کنند.

میتوان مدیریت کرد

معماری نرم افزار سیستم های Real Time همچنین شامل انواع Component های نرم افزاری مختلف، نحوه سازمان دهی این Component ها و همچنین ارتباط بین آنها توسط مدل ها

نحوه ارتباط اجزای تشکیل دهنده یک سیستم نرم افزاری و روش تعامل بین اجزای مختلف تحت اندازه


computer system
interface
environment

محیط بسته:
هر کدام از اجزاء از اجزای داخلی تر تشکیل شده که هر کدام از State تشکیل
و با درون هایش به دو اجزای داخلی سیستم با interface ارتباط برقرار می کند.

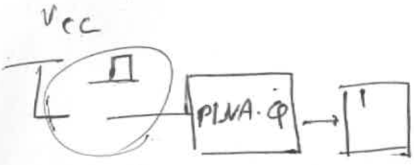
در بالاترین سطح این سیستم Real Time شامل مجموعه بسته از اجزای environment و computer system است که توسط Message ها، سیگنال ها و event ها می توانند با یکدیگر ارتباط داشته باشند.

معمولاً environment و computer system به اجزای کوچکتری نیز تقسیم

PAPCO

نحوه ارتباط اجزای نرم افزاری: ۱- stateBase این مدل ↑

مثلاً هی برنامه اگر statebase نگاه کنیم .



```
while (1)
{
  if(PINA.0==1)
  print i++;
}
```

(example برنامه در یک حلقه دائمی است در سیستم زیر :
وقتی پردازنده وقتی دکمه زده شد عدد 1 چاپ شود و وقتی دوباره زدن 2 شمارد .
این برنامه غلط : اصلاحش اینه:
که 1ms زمان باید بدهیم .

Subject:
Year. Month. Date. ۱۲

می شوند. (1- Ready 2- Running 3- blocked)

در هر زمان یک سیستم می تواند در یکی از State های خود قرار بگیرد (تعداد State ها می تواند نامحدود باشد)

معمولاً یک State شامل مقادیر خاص متغیرها و یا ارتباط بین متغیرها را نشان می دهد اگر این متغیرهای State تغییر کنند طوری که وضعیت سیستم تغییر کند به آن state transition گفته می شود.

event یا رخدادی که باعث Transition می شود، ممکن است interface event باشد که بین 2 سیستم environment و computer system اتفاق می افتد یا می توانند internal event باشند که تنها در داخل خود سیستم اتفاق می افتد.
به منظور طراحی و معماری می توان مجموعه ای از state ها را در یک گروه به نام super state یا mode قرار داد.

---

process < ① periodic ② sporadic

object های یک system را process می گویند.

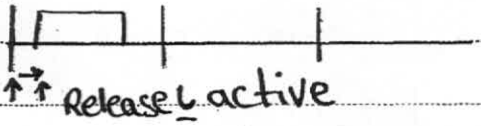
سیستم های Real Time معمولاً شامل 2 نوع process هستند :
یکی process های periodic دیگری process ها sporadic!
periodic process های در بازه زمانی ثابت و مشخص فعال می شوند. معمولاً در monitoring ، polling و Sampling استفاده می شود. برای مثال می توان توسط یک periodic process دمای یک مایع را هر 50 میلی ثانیه یکبار خواند. یا در atc یا air traffic Control فضا را توسط رادار در هر ۳ ثانیه یکبار Scan کرد.

process های sporadic :
در مقابل process های periodic ، process های sporadic هستند بر اساس event ها فعال می شند. این process ها توسط سیگنال های خارجی فعال

یا پروسس هایی که تکرار می شوند !

می‌شوند. process های sporadic می‌توانند در سیستم‌هایی که خطا را نشان می‌دهند مورد استفاده قرار بگیرند. یا تغییرات خارجی

یک periodic process، مقدار مشخصی از کار را در هر cycle انجام می‌دهد و معمولاً در ابتدای هر cycle، Active یا Release می‌شود.



به زبان Ada!

```
p:: loop wait-For-start-of-next-cycle ; p-code; end loop;
```

p-code، برنامه‌ی process است.

ساده‌ترین فرم برای مشخص کردن process به صورت سه‌تایی (c,p,d) است که

p ← period (زمان تکرار)

d ← dead line

c ← computation time است.

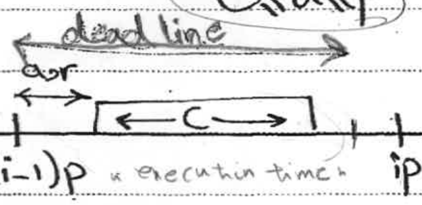
c (computation time): زمان اجرا، معمولاً تخمینی از بیشترین زمان اجرای process یا برنامه‌ی process یا همان p-code است. تخمین

p (period): همان cycletime است.

d (dead line): بیشترین زمانی است که می‌تواند process در اجرای کامل برنامه‌اش در cycle مشخص تأخیر داشته باشد

$c \leq d \leq p$

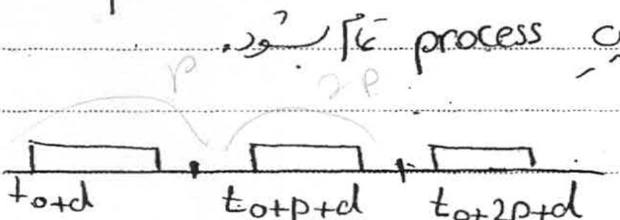
(زمان اجرای پروسس از d باید کمتر باشد.)



$(t_0 + (i-1) * p + d)$

زمان تمام شدن

زمانی که باید iامین cycle process تمام شود.



(نسبت به اولین زمان، ددلاین را مشخص می‌کنیم.)

2x $c \leq d \leq p$

Utalization: $\sum \frac{C_i}{P} < 1 \rightarrow$ مقدار بهره‌وری سیستم در دو حالت

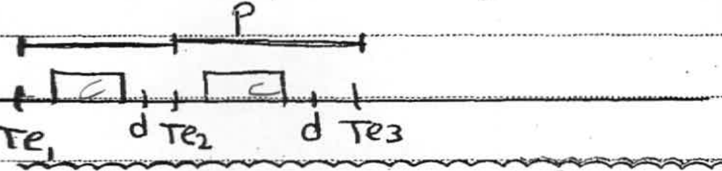
∞ می‌شود این قابلیت را بخوبی بدهد event که رو دارد ۱.

$c \le d \le p$

$c \le p \le d$

Subject:
Year. Month. Date. ١٤

period و deadline توسط مشخصات سیستم تعیین می‌شود. معمولاً p و d را یکسان می‌گیریم.



process های sporadic نیز با استفاده از ۳ تایی (c,p,d) با شرط $c \le d \le p$ نشان داده می‌شوند. در اینجا نیز c و d مانند process های periodic هستند. در زمانی که event اتفاق می‌افتد، اجرای process باید deadline مشخص شده‌اش به اتمام برسد یعنی اگر ما t به عنوان Completion Time، t باید $t_c < Te + d$ باشد که Te زمان وقوع رخداد است.

p، minimum زمان بین event هاست یعنی ۲ event متوالی حداقل در چه زمانی می‌توانند اتفاق بیفتند.

$c \le d \le p$

سه شکل دیگری از process های based to event یا بر اساس رخداد، process هایی است که رخدادهای آن به صورت هر زمان اتفاق می‌افتد. یا p=∞ بنابراین process ها process های air periodic می‌گوییم
Aperiodic

cyclic Executive:

یکی از کاربردی‌ترین روش‌ها برای پیاده‌سازی سیستم‌های RealTime، cyclic Executive است. این روش تنها برای process های periodic استفاده می‌شود. process های sporadic را می‌توان به صورت معادل periodic آنها استفاده کرد. هدف اصلی این روش این است که task های تعیین شده بتوانند در زمان تعیین شده زمان processor یا زمان پردازشگر را دریافت کرده و اجرا شود. پردازشگر می‌تواند بیش از یک عدد باشد ولی در این بخش از یک پردازشگر استفاده می‌کنیم. برای رسیدن به این هدف این الگوریتم به صورت صحیح مشخص می‌شود و قبل از اجرا زمان اجرای process ها به صورت feasible تعیین می‌شود.

# CE قبل از اجرا زمان اجرای process ها را مشخص می‌کند ← 1.pre_runtime

Subject:
Year. Month. Date. ۱۵/

روش cyclic Executive: براساس برنامه زمانبندی شده ی قبل از اجرا عمل می کند به خاطر اینکه عامل Task های کنترلی periodic هستند، CE تنها یک الگوریتم periodic یا cyclic است.

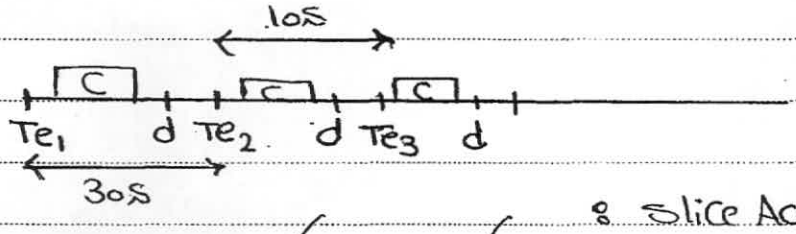
این روش بسیار معمول است به خصوص در زمان های قدیم. به خاطر این که ساده است اجرای آن کارآمد است و رفتار قابل پیش بینی بالایی دارد. در مقابل، بسیار سخت پذیر است، انعطاف پذیری ندارد که نگهداری و تغییر در طراحی بسیار مشکل است.

90,8,11 "جلسه ششم"

تعاریف و خصوصیات CE:

برای آن که CE به راحتی اجرا و پیاده سازی شود آن را به قطعات کوچک تقسیم می کنیم که در پایین توضیح داده شده:

cyclic executive
(CE)
prerun-time
periodic (c,p,d)



Slice Action (BLOCK):

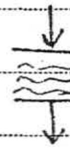
برنامه ای که باید توسط الگوریتم CE زمانبندی شود را S بنامیم. کد را به قسمت های کوچکتر $S = S_1, S_2, \dots, S_n$ تقسیم بندی می کنیم و آن ها را Slice Action یا Scheduling Block می نامیم. هر slice در الگوریتم CE به عنوان یک واحد مطرح می شود. این slice ها nonpreemptible هستند.

توانایی توخالی شدن
قبلی را ندارند.

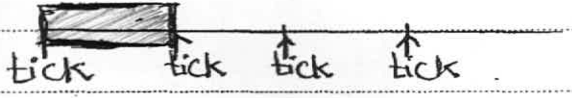
نمی شود از ایشان
CPU را گرفت

Subject:
Year. Month. Date. 19,( )

یعنی در cpu اگر به process طعام، یک process دیگر با اولویت بالاتر بیاید، اگر process اول قابلیت Block شدن داشته باشد، process جدید وارد می‌شود اولی می‌رود یعنی اگر process اول این قابلیت را نداشته باشد، می‌تواند تا وقتی کارش تمام نشده از آن cpu گرفت‌ها) ⟸ nonpreemtive

Straight Line Code  یعنی یک procedure یک ورودی دارد یک خروجی
هر slice ، complitionTime محدودی دارد
loop دائم ندارند.

Major schedule : کار k تا $s_1 \ldots s_n$ را بدانیم cpu همشون اختصاص بدهیم تا بتوانند این slice ها تمام اجرا شوند و محدودیت‌های زمانی تعیین شده برای هر slice رعایت شود، به آن Major schedule می‌گوییم.
زمانی که طول می‌کشد تا Major schedule کارش را تمام کند به این زمان MCT یا Major Cycle Time می‌گویند.
Major schedule را به قطعات کوچکتر تقسیم می‌کنیم به هر قطعه Minor cycle گویند.
زمانی که برای این قطعات نشان می‌دهند را mct یا minor cycle time می‌گویند.
* زمانی که می‌خواهیم زمان‌بندی کنیم از زمان تقسیم‌بندی ما در ابتدای mct است.



mct طول Frame است.

اگر ما major schedule را داخل loop دائم بیندازیم، به‌طور مرتب اجرا می‌شود.
وسیله کامل یک برنامه را نشان می‌دهد

EX: ۴ process periodic A,B,C,D که به صورت (c,p,d) مشخص شده‌اند را در نظر بگیرید.

A(1,10,10)
B(3,10,10)
C(2,20,20)
D(8,20,20)

۸ شکسته شود به 2و6
4و4 گذاشت
یا هیچ

PAPCO

طول دوره اجرای کل پردازش‌ها یا slice ها: MCT

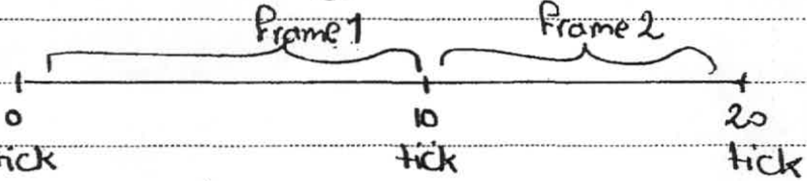
Subject:
Year. Month. Date. ۱۷/

زمان اجرای هر slice :mct

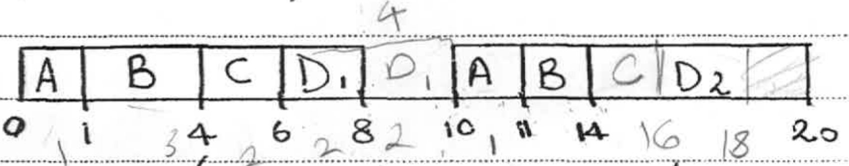
حل: می‌خواهیم یک برنامه بنویسیم. اولین کاری که می‌کنیم باید MCT را مشخص کنیم. چه زمانی را در نظر بگیریم که هر process حداقل یک بار اجرا شود؟

MCT=20

mct=10 } ~ * یعنی تقسیم داده می‌شود



باید برنامه را تقسیم کنیم تا هر process را در هر زمان مشخص اجرا کنیم (صفحه قبل با این شرایط)



* D₂ را نمی‌توانیم اجرا کنیم چون جا کم دارد، برای این Frame وسط اجرا هم cpu نگیریم سر Frame، cpu را می‌گیریم.

قسمت دوم اجرای برنامه: A اجرا می‌شود چون period‌ش 10 است.

B
11-4=7+3=10

برنامه: CE یک تابع است

CE:: حالت کلی

const نوع mct

```
mct: constant := 10; minor cycle time
next_time: time := clock + mct; next_tick_time
Frame_Number: integer := 1;
loop delay_until next_time;
* Frame_Number := (Frame_Number + 1) mod 2;
 case Frame_Number is
 when 0 => A; B; C; D1;
 when 1 => A; B; D2
 end case;
```

کدام مقدار می‌گیرد؟ ۱ یا ۲؟
از صفر شروع می‌شود یعنی ترتیب را می‌نویسند

صبر کند: این زمان برسد و بعد کار را ادامه دهد

Subject:
Year. Month. Date. ۱۸/۲

②

```
next_time := next_time + mct
if clock > next_time then
        Handle_Frame_over run;
        end if;
        end loop;
```

18 20

اگر بزرگتر بود یعنی در Frame بیشتر رفته
یعنی زمانبندی درست انجام نشده
مدیریت خطا انجام می‌شود
زمان اجرا deadline ها را در نظر بگیرد

---

* MCT باید چه خاصیت‌هایی داشته باشد؟

ت ۴۰۲

$MCT = Lcm(P_1, P_2, ..., P_n)$

حداقل یکبار اجرا شود، بزرگتر از همه باشد
کوچکترین مضرب مشترک period کل periodic processها

MCT = 20

mct = 10

* ساده‌ترین شرط برای mct :
(c,p,d)

$mct \geqslant C_i \quad i = 1, ..., n$

$mct \leqslant d_i$

$C_i \leqslant mct \leqslant \overset{d}{P_i}$

$\frac{MCT}{mct} = n \in \mathbb{N} = mct$

2

ب ۴۰۲

$mct + (mct - gcd(mct, p_i)) \leqslant d_i$

۱۰ + (۱۰ − gcd(۱۰,۱۰)) ≤ 10 29
10
۱۰ + (۱۰ − ۱۰) ≤ ۱۰ 20 29

تست مثال قبل : ۱۰ + (۱۰ − ۱۰) ≤ ۲۰

A(1,10,10) C(2,20,20)
B(3,10,10) D(8,20,20)

MCT = Lcm(10,10,20,20)

MCT = 20

$mct \geqslant C_i \Longrightarrow mct \geqslant 8$ mct = d

$mct \leqslant d_i \Longrightarrow mct \leqslant 10$ اینجا فقط ۸، ۹، 10

$\frac{MCT}{mct} = n \in \mathbb{N} \Longrightarrow \frac{20}{8, 9, 10} = 2$ mct بین 10 ✓

PAPCO

20 (2,3,6) 6

Subject:
Year. Month. Date. 19

درسته که نسبتاً زیاده اما شرط ۴ هم همین طوری چک می‌کنیم :

$\underset{10}{mct} + (mct - gcd(mct, p_i)) \leq \underset{10}{d_i}$

جلسه ششم ۹۰٫۸٫۱۸

مثال : periodic process ۳ ، $A(\overset{}{\underset{3}{2}},10,10)$ ، $B(5,10,10)$ ، $C(3,25,25)$

C P A

با فرض اینکه فاز پردازش A به ۲ بخش $A_1$ و $A_2$ تقسیم می‌شود در بدترین حالت

? $A_2=1, A_1=2$ ← C

B به ۲ بخش $B_1$ و $B_2$ با فاز های ۲ و ۳ تقسیم و C تقسیم نمی‌گردد.
Major schedule توسط CE را برای process های فوق پیاده‌سازی کنید.

$A(\underset{}{\overset{3}{2}},10,10) \rightarrow \begin{cases} A_1(2,10,10) \\ A_2(1,10,10) \end{cases}$

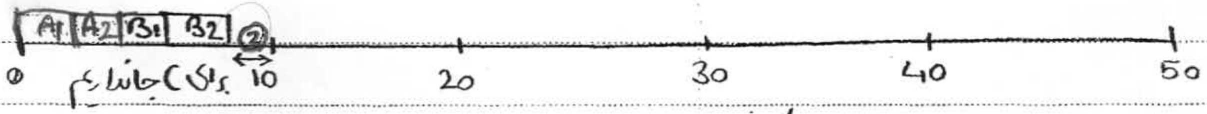
$B(5,10,10) \rightarrow \begin{cases} B_1(2,10,10) \\ B_2(3,10,10) \end{cases}$

$MCT = LCM(p_1, p_2, p_3) = 50 = 2 * 5^2$

$\left.\begin{array}{l} ① \; mct \geq c_i \Rightarrow mct \geq \cancel{3}\,5 \\ ② \; mct \leq d_i \Rightarrow mct \leq 10 \end{array}\right\} \Rightarrow mct\,\{\cancel{3}, \cancel{4}, 5, 6, 7, 8, 9, 10\}$

③ $\frac{MCT}{mct} \in \mathbb{N} \Rightarrow mct = \{5, ⑩\}$

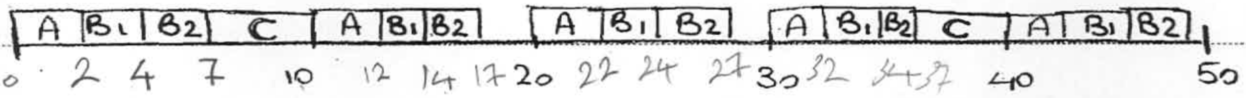
④ $mct + (mct - gcd(mct, p_i)) \leq d_i$ صدق می‌کند!



PAPCO

اگه process C رو تقسیم بکنیم می‌شه در 2 تا جدول اجرا کرد، این روش درست نیست!!! پس A رو تقسیم می‌کنیم!

Subject:
Year. Month. Date. ۲۰/( )


A
B1
B2
C
A
B1
B2
A
B1
B2
A
B1
B2
C
A
B1
B2
0
2
4
7
10
12
14
17
20
22
24
27
30
32
34
37
40
50

switch()
{when 0 → A, B1;

EX : ۳ process داریم $C(3,22,22)$ , $B(2,20,20)$ , $A(1,14,14)$

می‌خواهیم اینها را به روش CE زمانبندی کنیم . mct و MCT را بدست آورید

$MCT = LCM(p_1, p_2, p_3) = LCM(14, 20, 22)$

$2*7 \quad 2^2*5 \quad 2*11$

$MCT = 2^2 * 7 * 5 * 11 = 1540$

① $mct \geq C_i \Rightarrow mct \geq 3$

اگه ۱ ، 3 رو حساب کنید

$\boxed{\;3\;|\;}$

اینجا دیگه ابتدای frame یکی از کارها بیاد

② $mct \leq d_i \Rightarrow mct \leq 14$

پس mct می‌تونه در بازه {3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14} باشه
x ✓ ✓ x ✓ x x ✓ ✓ x x ✓

③ $\frac{MCT}{mct} \in N$

پس چون بیشتریشون حذف شد
$mct = \{4, 5, 7, 10, 11, 14\}$

$mct + (mct - gcd(mct, p_i)) \leq d_i$

7 9 10

$mct \{4, 5, 7, \cancel{10}, 11, 14\}$

PAPCO

Subject:
Year. Month. Date. ۲۱

روانجیم mct های بزرگ‌تر با قضیه

برای عدد 14 : 8

$$\begin{cases} 14 + (14 - \gcd(14, p_i)) \leq 14 \\ 14 + (14 - \gcd(14, 20)) \leq 14 \end{cases}$$

2

$20 \not\leq 14$ mct ≠ 14

شرط deadline لحاظ

$$\begin{cases} 11 + (11 - \gcd(11, p_i)) \leq d_i \\ 11 + (11 - \gcd(11, 14)) \not\leq 14 \end{cases}$$

1

mct ≠ 11

$$\begin{cases} 10 + (10 - \gcd(10, p_i)) \leq d_i \\ 10 + (10 - \gcd(10, 14)) \not\leq 14 \end{cases}$$

$18 \not\leq 14$ mct ≠ 10

(14 , 20 , 22)

mct = {4 , 5 , 7}

8 10 14

اگه بریم سمت بیشتر به ۲ برابرش از تمام di ها کوچک‌تر بود قبوله!

<

حالا کدوم از این ۳ تا مناسب‌تره؟! (7)!

(12)

اما اگه اول PA بشه 15 به جای 14 اون وقت دیگه اول 7 از ~~MCT~~ mct حذف میشه و دیگه ۴ می‌مونه

و تعداد فریم‌ها عوض می‌شه!

پس این نرم‌افزار Flexibility نداره!

CE نسبت به تمام ساختارها ساده است. مثل context switching نداره!

اما یکی از مشکلات اساسی که می‌تونه ایجاد بشه نداشتن Flexibility است!

Subject:
Year. Month. Date. ۲۲/۱

مزایای CE : (cycle Executive)

۱، ساده است.

۲، زمان context switching نمی‌گیرد.

۳، حافظه موردنیاز برای context switching پایین است

۴، بخاطر ساختار preruntime این الگوریتم رفتار آن قابل پیش بینی است

معایب : (C)

۱، معمولاً Computation Time در نظر گرفته برای process ها صحیح نیست و به همین خاطر missed deadline اتفاق می‌افتد.

۲، الگوریتم کاملاً غیر انعطاف پذیر است. تغییرات اندک در period ها، deadline ها یا computation time ها باعث می‌شود تا مجبور شویم کل الگوریتم را تغییر دهیم

روش دیگری که از CE ساده‌تر است و امروزه از آن استفاده می‌شود:

---

FOREGROUND _ BACKGROUND:

برای process های پریودیک است. حالت ساده‌شده‌ی این روش process ها را به دو مجموعه تقسیم می‌کند

① FORE GROUND (FG)
② BACKGROUND (BG)

① مربوط به process های Real Time critical است. (FG)
② ~ ~ non critical است. (BG)

| FG | BG |
|---|---|
| nonpreemptible | preemptible می‌تونه CPU گرفته بشه! |

سیستم ابتدا مجموعه FG را اجرا می‌کند و اگر زمانی باقی ماند به BG می‌رسد!

مشخص کردن زمان از طریق timer interrupt صورت می‌گیرد!

که Dispatcher وظیفه دارد با وقوع timer interrupt، process های در حال اجرای BG را به تعلیق دوباره و نوبت را به process های FG بدهد

| | | | | | |
|---|---|---|---|---|---|
| FG1 | FG2 | FG3 | .... | FGn | BG |

(MCT = mct)

اگر Computation time تغییر کند از زمان BG استفاده می‌کند و می‌تواند به کارش ادامه دهد ( CE اینجوری نبود) که process ها دارای period time ثابتی هستند و همچنین برابر با deadline است.

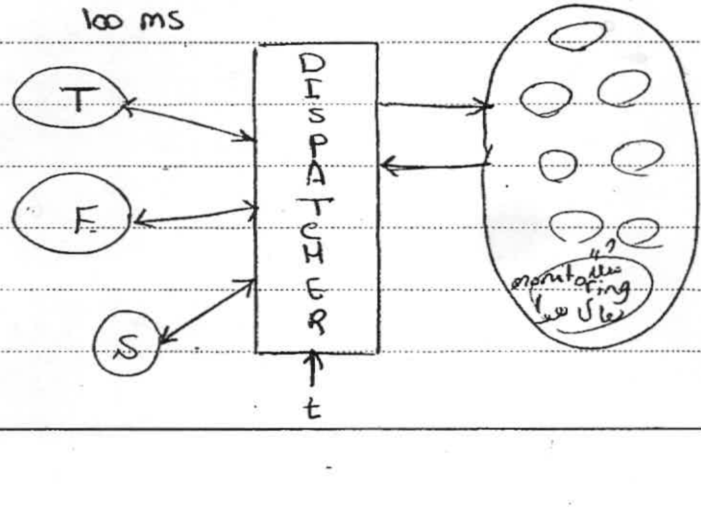
یعنی deadline time ثابتی هم دارند

در زمان Idle، الگوریتم CE کار مفیدی نمی‌کرد اما این یکی در زمان Idle، BG ها را اجرا می‌کند!

BG مثلاً می‌تونه در سیستم مثلاً sensor های کنترل خطا را انجام بدهد مثل شکل زیره
"کنترل دمای سیال راکتور هسته‌ای"

| T | F | S |
|---|---|---|
| Time controling | Fault controling | کارهای سنسوری داخلی سیستم |

| | | | |
|---|---|---|---|
| T | F | S | BG |



Subject:
Year. Month. Date. ۲۴

مثلاً در Case های server میشه چندتا power suply گذاشت و به صورت online تعویض کرد

کلاس فوق العاده

E: فرض کنید سیستمی دارای ۳ process پریودیک است

CPd

A(2,9,9) $3^2$

B(4,12,12) $3*2^2$

C(1,15,15) $3*5$

الف/ MCT را بدست آورید.

ب/ با فرض اینکه بلاک‌ها هم process ها هستند مقادیر ممکن برای mct را بدست آورید.

ج/ یکی از جواب‌های بدست آمده از قسمت ب را برای رسم major cycle استفاده کنید.

$MCT = 3^2 * 2^2 * 5 = 180$

① $mct \geq c_i \Rightarrow mct \geq 4$

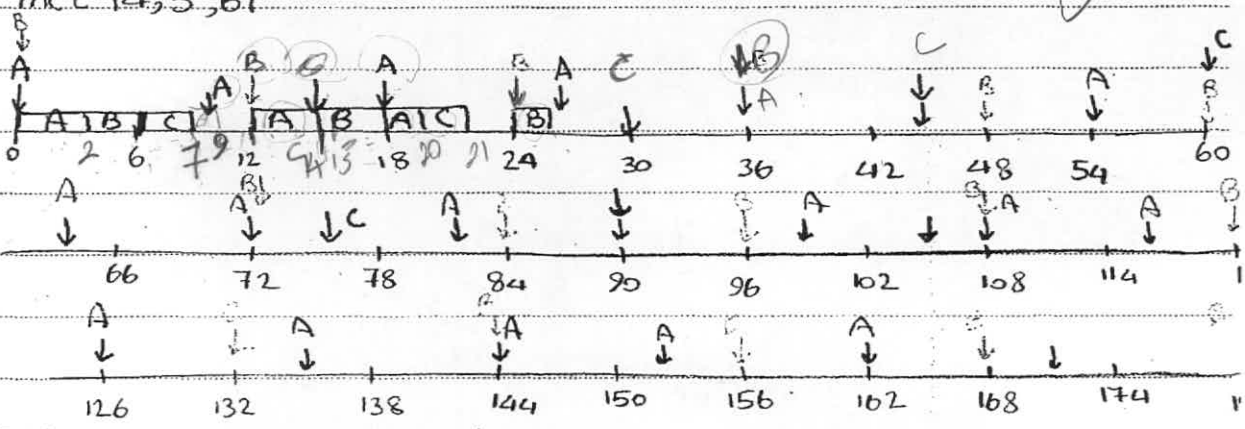
② $mct \leq d_i \Rightarrow mct \leq 9$

③ $\frac{MCT}{mct} \in \mathbb{N} \Rightarrow mct = \{4, 5, 6, 9\}$

④ $9 + (9 - gcd(9,12)) \leq 12$ $15 \not\leq 12$ $mct \neq 9$

$6 + (6 - gcd(6,9)) \leq 9$ 3

mct {4, 5, 6}



A
B
C
0
2
6
7
9
12
13
18
24
30
36
42
48
54
60
66
72
78
84
90
96
102
108
114
126
132
138
144
150
156
162
168
174

اول به اونهایی که [illegible] می‌دهیم به deadline شون زودتره [illegible]

Requiement & design speecification

Subject:
Year. Month. Date. ۲۵

## جلسه هفتم

۹۰٫۸٫۲۵

ساختار OS ای که در سیستم‌های Real Time وجود دارد به ۴ لایه تقسیم می‌شود.

| |
|---|
| ۱/ Application |
| ۲/ Resource Management |
| ۳/ Kernel (OS) |
| ۴/ Machine |

مربوط به سخت افزار [illegible]

امروزه ساختار کلی OS ای که در اغلب سیستم‌ها استفاده می‌شود operating system است که تجاری

OS

هم‌زمانی تر در آن پیاده‌سازی شده است.

شکل فوق لایه‌های ساختار کد OS را نشان می‌دهد

Kernel سطح process ها را با دستگاه هم‌روندی یا هم‌زمانی پیاده‌سازی می‌کند. روش‌های هم‌زمانی process ها، زمان‌بندی و switching هم‌چنین سرویس‌های timer و مدیریت interrupt ها در این لایه است.

Resource Management سطح مدیریت ذخیره‌سازی داده‌ها، کنترل دستگاه‌های ورودی و خروجی و دیگر کارهایی که مربوط به تخصیص منابع می‌شود بر عهده این لایه است.

process های periodic و sporadic به سیستم‌های Real Time تعریف می‌شود در بالاترین لایه OS یعنی Application قرار دارد. زمان‌بندی و محدودیت‌های دیگر مانند اهمیت process ها، کرای بودن آنها به اولویت process ها تبدیل می‌شود.

اولویت دادن به process ها می‌تواند به صورت صریح یا ضمنی باشد!

در اولویت صریح: به صورت مستقیم به process ها اولویت داده می‌شود.

در اولویت ضمنی: می‌تواند process ها به طور مثال براساس زمان Activation اولویت داده شود.

Subject:
Year. Month. Date. ۲۹/

یعنی process ی که زمان Activation آن پایین (کم) باشد با اولویت بالاتری اجرا شود.

سیستم‌ها می‌توانند به صورت ① Fix priority ② Dynamic priority باشند.

در نوع Fix priority اولویت process ها قبل از اجرا یا در زمان initialize شدن process ها تعیین شود و در حین اجرا هیچ تغییری نمی‌کند.
در سیستم‌های Dynamic priority اولویت process ها در حین اجرا می‌توانند تغییر کنند.

---

سیستم‌ها می‌توانند به صورت I non_preemptive یا II preemptive باشند.

به سیستم‌هایی non_preemptive گفته می‌شود که در زمان اجرای process اجازه آن به process دیگر زمان cpu داده نشود.

در سیستم‌های preemptive اگر process ی با اولویت بالا از process در حال اجرا وجود داشته باشد، باعث می‌شود تا process اولویت بالا آن را از cpu خارج می‌کند.

---

## Requirements & Design Specification :

زبان‌های مدل‌سازی برای بیان Requirement و Design تأثیرات مهمی در زمان پیاده‌سازی، درست بودن، کارآیی و قابلیت نگهداری سیستم‌ها دارند.
یکی از خواصی که باید این زبان‌ها داشته باشند این است که هیچ ابهامی وجود نداشته باشد بنابراین مشتری و برنامه‌نویس می‌توانند در رفتار و عملکرد سیستم به توافق برسند.

وقتی همه دنیات تنها و غریب بود وقتی همه جا با تو احساس وحشت بود

Subject:
Year. Month. Date. ۲۷

۱. برای اینکه زبان مدلسازی نباید ابهام داشته باشد، نمی‌توان از زبان های طبیعی استفاده کرد و به جای آن دارای ساختار منظم تر مثل زبان برنامه‌نویسی باشند یا به صورت یک زبان گرافیکی مثل فلوچارت

۲. این زبان باید امکان اثبات درستی آن قبل از پیاده سازی سیستم را داشته باشد.

۳. قابلیت scalability داشته باشد یعنی بتوان سیستم را از اجزای کلی تر تشکیل داد و سپس آن را به جزئیات تقسیم کرد تا در دید اول کل کارایی سیستم مشخص شود.

این زبان های مدلسازی را به ۲ دسته تقسیم می‌کنیم:

① imperative , operational

(operational) عملیاتی

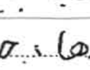
② Descriptive Declerative

توصیفی

① زبان های imperative یا operational رفتار سیستم را با استفاده از الگوریتم ها توصیف می‌کنند. این الگوریتم ها شامل مجموعه‌ای از دستورالعمل ها یا عملیات است که توسط یک یا چند عامل اجرا می‌شود.

رفتارها

② مشخصاً خصوصیاتی که باید توسط سیستم به وجود بیاید را توصیف می‌کنند. این روش ها ساختاری بر پایه ریاضیات دارد مانند تئوری مجموعه‌ها، جبر و منطق و ... به دلیل اثبات [illegible] در این نوع زبان ها می‌توان به راحتی عملکرد سیستم ها را اثبات کرد ولی برای پیاده سازی استفاده از این زبان ها ساختارها مشکل است.



PAPCO

Descriptive
Declerative

**Subject:**
**Year.** **Month.** **Date.** ۲۸

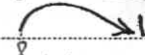
DFD مخفف : Data Flow Diagram

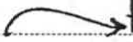
یک روش گرافیکی مناسب برای نمایش پردازشها(فرایندها) و توابع سیستم است . ورودی، خروجی و تولید و ذخیره سازی داده ها توسط توابعی نشان داده می شود . هم چنین ارتباط بین ورودی ها و خروجی ها نیز در این ساختار نشان داده می شود.

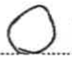
نمادهایی که در این روش استفاده می شود به صورت زیر است :

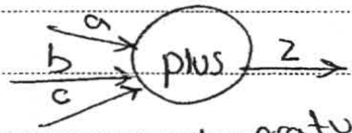
(arrow) نمادها

مثال

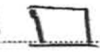
1 
Data Flow
جریان داده ها نشان می دهد

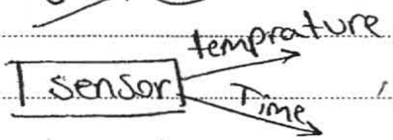

temprative → Temprature

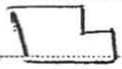
2 
Function

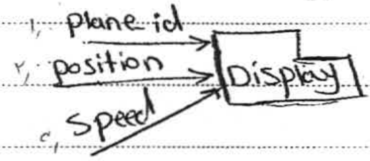


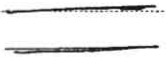
می تواند یک یا چند ورودی داشته باشد

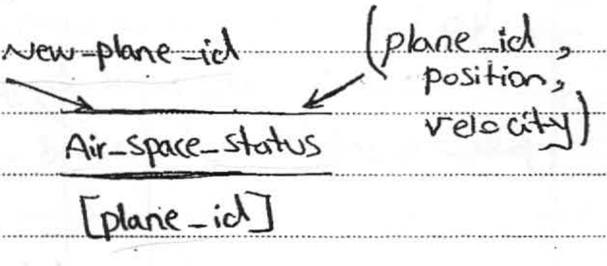
3 
Input
ورودی برای سیستم


Sensor
temprature
Time
sensor به عنوان input است .

4, 
output


1, plane-id
2, position
3, speed
Display

5, 
storage
محل ذخیره سازی داده ها
می تواند به صورت ورودی و خروجی باشد


New-plane-id
(plane_id, position, velocity)
Air-space-status
[plane-id]

مانند دیتابیس (DB)

data flowdiagram

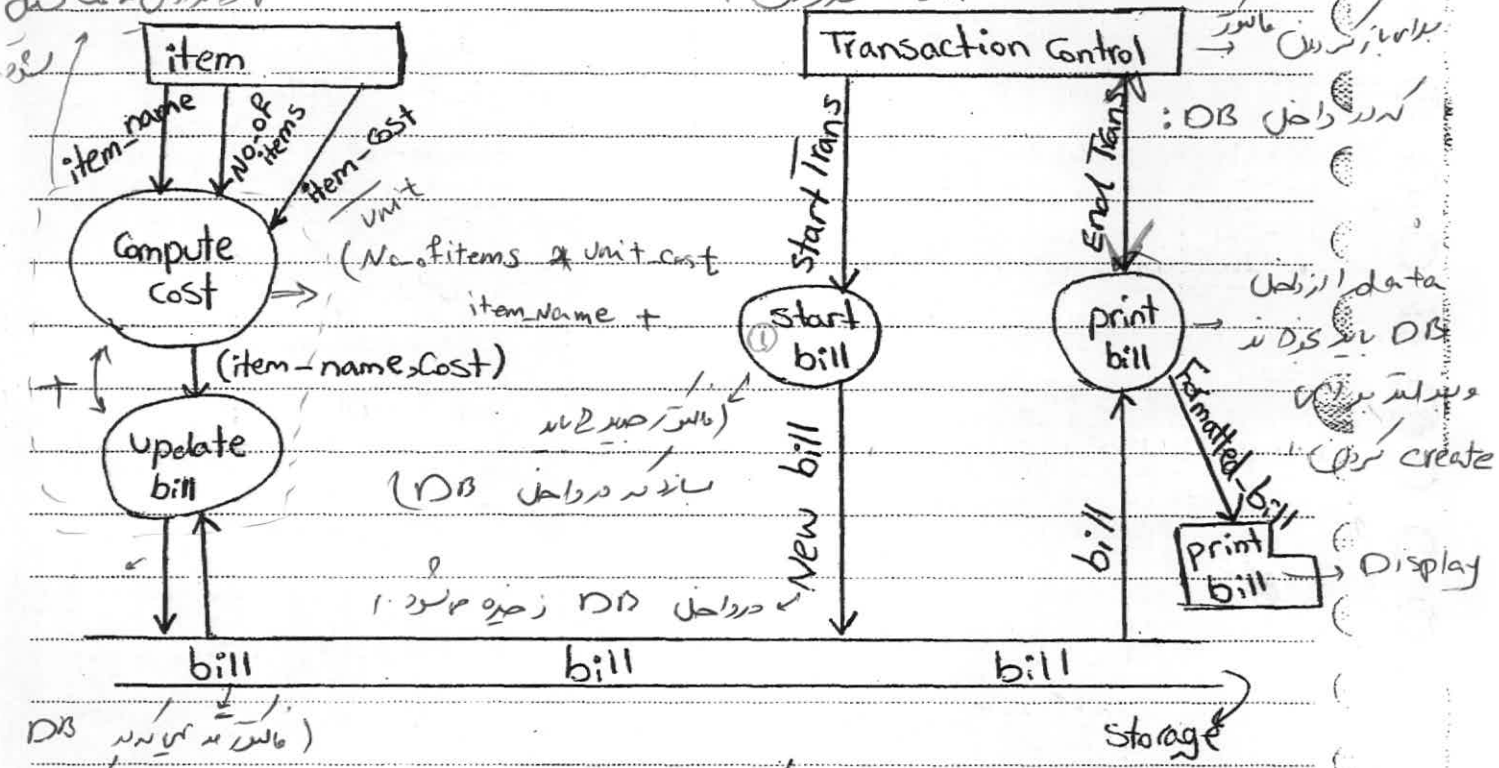
Subject:
Year. Month. Date. ۲۹

جلسه هشتم ۶٥,۹,۲

EX: سیستم فروش یک فروشگاه

با استفاده از DFD : Function و پارامترهایش

برای صادر کردن:


item
item_name
No-of items
item-cost
unit
Compute cost
(No-of items * unit_cost
item_Name +
(item-name, Cost)
Update bill
bill
bill
bill
Storage
Transaction Control
Start Trans
End Trans
start bill
①
New bill
print bill
bill
Formatted-bill
print bill
Display
برای باز کردن فاکتور
که در داخل DB:
data را از داخل DB باید بگیرد
و بعد از این
create کردن
(فاکتور جدید که باید
بسازد در داخل DB)
در داخل DB ذخیره می‌شود.
(فاکتور جدید می‌آید DB
که مقدار و فاکتور جدید
در DB می‌گذارد)

توضیح نمودار چپ: این فروشگاه یک کالا دارد (item).
این کالاها یک سری خصوصیات دارند (فیلدها)
در compute cost محاسبه می‌کند.
بعد فاکتور را چاپ می‌کند و بعد از update در داخل storage
قرار می‌دهد

سمت راست: برای اینکه فاکتور را ببندیم و صادر کنیم باید Transaction control را انجام
بدهیم.
برای print Bill چاپ کردن! و به این ترتیب فاکتور جدید وارد
DataBase می‌شود

ادامه در جزوه عیب های DFD : { ۱- ابهام ؛
*
۲- خط اشتراک :
نمی‌توانیم تصمیم بگیریم
آیا درست یا خیر؟! و ...

planeid

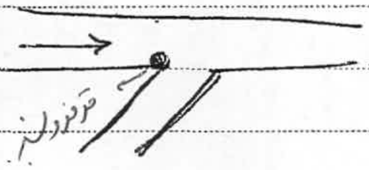
وقتیکه ستون J ام انتخاب شد اقدامات ستون J ام را انجام می دهیم.

Action J

if $a_{IJ}$ = 'X' then $a_1$ , ...

خودم ← یعنی تو ستونهای راست هرجا X داریم اقدامات A ها را انجام بدیم!

EX: ورودی یک اتوبان توسط چراغ راهنمایی کنترل می شود تا سرعت ماشین هایی که از اتوبان عبور می کنند از حد آستانه 30 kph کمتر شود، چراغ ورودی به اتوبان قرمز می شود. صف تشکیل شده در پشت چراغ قرمز توسط سنسوری مشخص می شود. اگر این طول بزرگتر از 100 متر بود آن گاه به پلیس منطقه اعلام می شود.



می خواهیم این چراغ را با Tabular language بنویسیم.

Conditions:

~~Average speed > 30~~

C1 : سرعت متوسط > 30

C2 : ~~چراغ قرمز~~ Averege speed < 30

C3 : طول صف > 100

(Entry Quene >۱۰۰)

(meter light Activated)

Actions:

A1 : چراغ قرمز شود

A2 : چراغ سبز شود

A3 : به پلیس اعلام کند

(Alert -police)

۳ تا Condition ، ۸ تا Rule داریم.

PAPCO

Subject: yes,No $\rightarrow 2^3$

Year. Month. Date. ۳۱

* اگر ۳ تا شرط داشته باشیم 8 تا حالت میتوانیم داشته باشیم !.

$$C_{ij} \begin{cases} \rightarrow \text{Yes} \\ \rightarrow \text{NO} \end{cases}$$ Conditions * مقادیر داخل جدول

حتماً اجرا کن $\rightarrow$ X Do it

$$A_{ijk} \begin{cases} \rightarrow \text{X} \quad \text{Do it} \\ \rightarrow \text{—} \quad \text{Don't Do it} \end{cases}$$ Actions * مقادیر داخل جدول

حتماً نکن $\rightarrow$ — Don't Do it

~~*~~

توضیح جدول صفحه قبل :

قسمت بالای جدول، condition و قسمت پایین جدول بخش Action های جدول است. هر ستون جدول قانون ستون i اُم جدول است که با شرایط ستون i اُم محافظت می‌شود! (یعنی gaurd شده است)

*** یعنی condition ِ $r_{ij}$ می‌شه gaurd ِ action ِ $r_{ij}$

* برای اینکه بتوانیم gaurd را ارزیابی کنیم که آیا T یا F است و باید از دستورات زیر استفاده کنیم :

If condition j then action j

$$\Big(\big(C_{1j}='y'\big) \,\&\&\, C_1\Big) \text{ or } \big(C_{1j}='N'\big) \,\&\&\, \big(\text{NOT } C_1\big) \qquad \text{and}$$

$$\Big(\big(C_{2j}='y'\big) \,\&\&\, C_2\Big) \text{ or } \big(C_{2j}='N'\big) \,\&\&\, \big(\text{NOT } C_2\big) \qquad \text{and}$$

$$\vdots$$

$$\underbrace{\Big(\overbrace{\big(\underbrace{C_{nj}='y'}_{①}\big) \,\&\&\, (C_n)}^{②}\Big) \text{ or } \overbrace{\big(\underbrace{C_{nj}='N'}_{③}\big) \,\&\&\, \big(\underbrace{\text{NOT } C_n}_{④}\big)}^{⑤}\Big)}_{⑥}$$

ترتیب پرانتزها $\uparrow$

وقتی قطار داره وارد میشه ، countcL

Subject:
Year. Month. Date. ۳٤

$2^4 = 16$ حالت

?

٤ شرط داریم ۱۶ تا قانون !!!

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TE | C1 | Y | Y | Y | Y | Y | Y | Y | Y | N | N | N | N | N | N | N | N |
| TL | C2 | Y | Y | Y | Y | N | N | N | N | Y | Y | Y | Y | N | N | N | N |
| C=0 | C3 | Y | Y | N | N | Y | Y | N | N | Y | Y | N | N | Y | Y | N | N |
| C=1 | C4 | Y | N | Y | N | Y | N | Y | N | Y | N | Y | N | Y | N | Y | N |
| | | ☺ | ☹ | ☺ | ☺ | ☺ | X | Don't care | | | | | | | | | |
| open Gate | A1 | | | | | | — | — | — | | ق | X | — | | ق | ق | ق |
| close Gate | A2 | ق | ق | ق | ق | ق | X | — | — | ق | | — | — | ق | | | |
| C++ | A3 | | | | | | X | X | X | | | — | — | | | | |
| C-- | A4 | | | | | | — | — | — | | | X | X | | | | |
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |

زمانی که احتمالات یا نمونه‌ی شرط به هم وابسته هستند

Don't care = X

حذف منطقی : چون هم‌زمان C==0 , C==1 نداریم.
← خانه‌های اول، ۹ و ۱۳

حذف فیزیکی : قطار وارد میشه . Gate بازه ← بسته می‌شه!
خارج میشه Gate بسته ← باز می‌شه!
یعنی انگار کلاً Gate کاری نکرده، نه باز شده نه بسته!
پس این ستون هم حذفه!
← خانه‌ها ۲، ۳، ۴

حذف ذکی : هیچ قطاری وارد یا خارج نشه ، نه هم تعداد تغییر نکنه پس حذف می‌شود
← خانه‌های ۱۴ و ۱۵، ۱۶

X حذف نازکی : اصلاً این خانه کلاً غلطه! چون نمی‌شه C==0 ، Count-- باشه و قطار خارج باشه .
اگر صفر باشه نمی‌شه leave شه دستکاری بشه!
← خانه ۱۵

PAPCO

برای پیاده‌سازی هر ستون یک if می‌نویسیم:

```
1) if((C11 =='yes') && (C1) || ((C11=='N')
        && not C1) && ((C21 =='y') && C2)
   || ((C21=='N') && not C2) && ((C31 =='y') && C3) || ((C31=='N') &&
        not C3))
2) Then if (A1 =='x') then do A1 ;
        if (A2 =='x') Then do A2 ;
        if (A3 =='x') Then do A3 ;
```

Subject:
Year. Month. Date. ۳۴

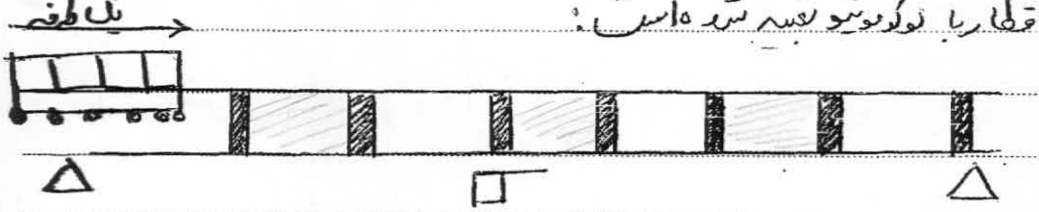
| | | | ? | ? | | | ? | ? |
|---|---|---|---|---|---|---|---|---|
| ۳۰ تا ورود C1 | Y | Y | Y | Y | N | N | N | N |
| فراغت C2 | Y | Y | N | N | Y | Y | N | N |
| حداقل سن C3 | Y | N | Y | N | Y | N | Y | N |
| A1 | — | — | ~~X~~ | ~~X~~ | — | — | X | X |
| A2 | X | X | — | — | ~~X~~ | — | — | — |
| A3 | X | — | X | — | X | — | X | — |

A تعداد

C ها تو چند متفاوت باشند.

قوانینی که اقدام یکسانی دارند حذف می‌شود.

EX: تقاطع ریل راه‌آهن با جاده توسط راهبر Gate کنترل می‌شود. با استفاده از یک جدول، کنترل کننده‌ای را طراحی کنید که گیت را باز و بسته کند.
برای این منظور ۲ Sensor با فاصله مناسب جهت تشخیص ترن ورود و خروج قطار با لوکوموتیو تعبیه شده است:



Conditions:

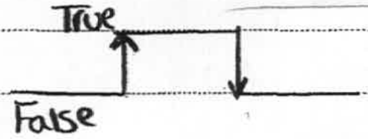
| | | |
|---|---|---|
| C1: TE | Tran Entering | ورود قطار |
| C2: TL | Tran Leaving | خروج قطار |
| C3: Counter == 0 | | تعداد قطارهایی را که در محوطه هستند می‌شمارد |
| C4: Counter == 1 | (بیشتر هم نمی‌تواند باشد) | |

Actions:
A1 : open Gate
A2 : Close Gate
A3 : count ++
A4 : count --

. if gaurd and Event then perform action.

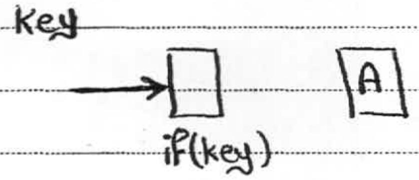
Subject:
Year. Month. Date ۳۹/ / )

↑ event را نشان می‌دهد. لحظه تغییر حالت را نشان می‌دهد.



تغییر حالت یعنی :

* برنامه :



```
while (1)
 {
  if(key)
  {
   printA
   while(key);
  }
 }
```

وقتی کلید را فشار می‌دهم فاصله‌ای دارم، آزاد کردن کلید توسط کاربر است. برنامه‌ها ثانیه طول می‌کشد...
سیستم هنگ کرده است.

زمانیکه key را فشار می‌دهم True یا False (جایشان) عوض می‌شود. اگر بنویسیم

```
if(key)
printA
```

همان آزاد شدن کلید دست کاربر است:

* برنامه جدید :

متغیر Boolean f را تعریف کردیم
True است.

```
bool f=true
while(true)
{ if(key&&f)
  { print A;
    f=false;
  }
  else if(!key)
    f=true;
}
```

① set vect:
② get vect:

اگر C میتوان Interrupt را نوشت!

(سرعت اجرای اینترپت‌ها دقیق‌تر از polling است!)

8054
2
(4) فواصل

توضیح خانه ۷: به فشار تو فضاست، پمپ بسته است، پس Counter را یه واحد اضافه کنیم

" " ۱۲: چون سطح فشار در محدوده است. باید پمپ بسته بشه.

" " " خره ۳ یا ۲ ... می‌تونه باشه.

| | | | | |
|---|---|---|---|---|
| $\overline{IE}$ | Y | Y | N | N |
| $\overline{IL}$ | N | N | Y | Y |
| C=0 | Y | N | N | N |
| C=1 | N | Y | Y | N |
| OG | — | — | X | — |
| CG | X | — | — | — |
| C++ | X | X | — | — |
| C-- | — | — | X | X |

Don't Care های صفحه قبل را یکی تغییر نیستم اینجا بسته

جلسه ۹۰.۹.۹

## SCR : (Software Cost Reduction)

اجزای اصلی تشکیل دهنده ی SCR، مدها، شرایط (Conditions)، eventها و Action هاست.

مد مجموعه‌ای از state های مرتبط به هم است و شرایط (Condition) مجموعه‌ای از گزاره هستند که می‌توان آن را به صورت T یا F ارزیابی کرد.

تغییر در سیستم بر اساس تغییر متغیرهای حالت است که معمولاً از طریق ورودی سیستم که سوگ از محیط است. نتیجه آن تغییر Condition به حالت T یا F می‌باشد.

این تغییر event نامیده می‌شود که در جدول SCR به صورت زیر نمایش داده می‌شود:

↑C1 when C2     C1 از حالت F به T تغییر حالت می‌دهد زمانیکه C2 درست باشد

↓C1 when C2

PAPCO

خداحافظ سی‌جی !

Subject:
Year. Month. Date. ۳۸

تمرین ۵: کنترل چراغ راهنمای را در تقاطع فرعی و اصلی کنترل می‌کند. یک جفت چراغ هر مسیر چراغ می‌تواند سبز یا قرمز شود ولی ۲ جفت با یکدیگر نمی‌توانند سبز باشند. در حالت عادی، چراغ معمولی قرمز و سبز می‌شود. (هر ۴۵ ثانیه)

زمانیکه آمبولانس به تقاطع نزدیک می‌شود سیگنال amb_enter کنترل باید چراغ را

road

این آدرس نشان می‌دهد که روی جاده اصلی است یا فرعی

برای آمبولانس، مسیر سبز را برای آمبولانس نگه دارد؟ زمانیکه آمبولانس از ۴ راه خارج می‌شود سیگنال amb_Exit ارسال می‌شود که کنترلر باید به صورت عادی چراغ را تغییر دهد.

با استفاده از SCR های sporadic کنترل شده فوق را طراحی کنید.

## State Machines

* ساختار نوعی از state ماشین‌ها در مدل‌سازی و طراحی سیستم‌های کامپیوتری استفاده می‌شود.

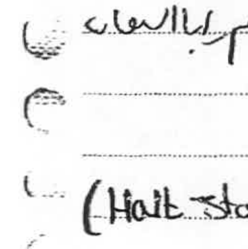
این ساختار برای توصیف رفتار مدارات switching نیز مورد استفاده قرار می‌گیرد به علاوه یکی از اصلی‌ترین علوم کامپیوتر، تئوری automata نیز بر پایه این ماشین است.

یک FSM الگویی دارای تعداد محدودی حالت و توابع Transition یا Next state است که نگاشتی از حالت‌ها و ورودی‌ها به حالت‌ها است.

"FSM: Finite State Machine"

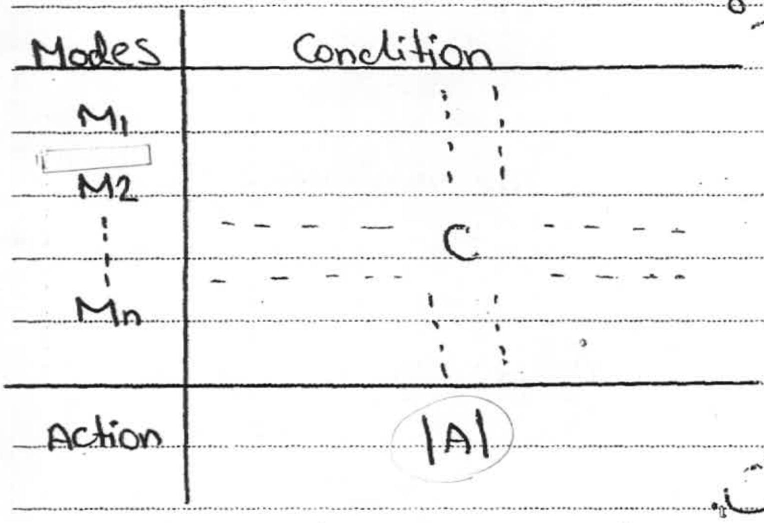
در هر زمان سیستم در یک حالت قرار می‌گیرد. اگر event ی اتفاق بیفتد، سیستم را با استفاده از توابع Transition از حالتی به حالت دیگری برد.

یک ماشین معمولاً یک start state و صفر یا چندین حالت پایانی دارد. (Halt state)



جدول SCR دو نوع است. یکی برای توابع periodic و یکی برای توابع Sporadic

SCR برای توابع periodic :



هر تابع یک جدول شرایط

شبیه DesicionTable است.

{E when C}

نکته: ① Sporadic ها event Base هستند.

② periodic ها Polling هستند.

$C_1$ when $C_2$
↑
False → True

$C_1 \downarrow$ when $C_2$
True → False

تمرین: کنترل دمای یک منبع با درجه C داده شده است. دمای منبع T در هر p واحد زمانی نمونه برداری می‌شود.

- اگر T > C باشد، اهرمی در منبع به اندازه مقدار ثابتی به سمت بالا حرکت می‌کند و اقدام mu (move up) اجرا می‌شود.
- اگر T < C باشد، اهرم به سمت پایین حرکت کرده و اقدام md (moveDown) انجام می‌شود.
- اگر اهرم در n پریود زمانی حرکت نکند سیستم وارد فازی شده به آن keep_stable گفته می‌شود.

اگر سنسور اعداد T < -273 یا T > 500 را در ۳ پریود زمانی متوالی نشان دهد سنسور خراب است و در اینجا خرابی سنسور اعلام می‌شود سیستم به Error-Action ها را اجرا کند. ساختار کنترل شده فوق را با استفاده از Desicion Table ها بیان کنید.

ورودی
og
cg
o-o
og

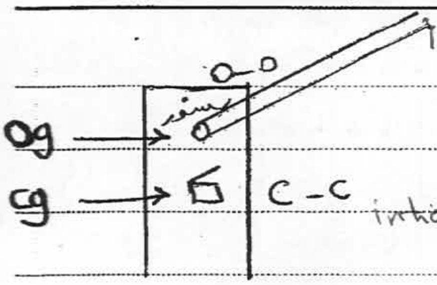
```
while(1){
  if((state==1)&&(og)){
    State=2
    So=1;
    Sc=0;
  }
  if((state==1)&&(cg)){
    Sc=0;
    Sc=1;
```

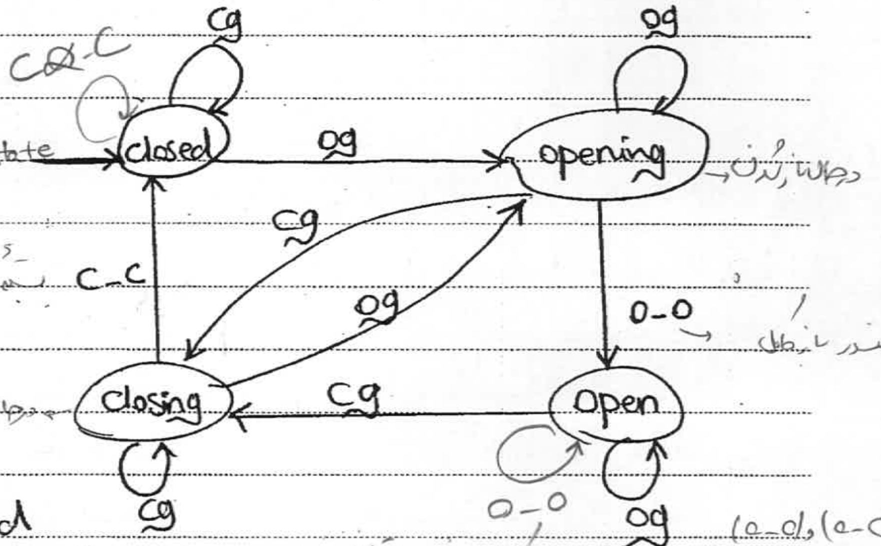
خروجی
$S_o$
$S_c$

برنامه‌ی دیاگرام بالا : Q

Subject:
Year. Month. Date. ~~Eg~~

* (این‌ها 2 سنسور og ، cg داریم.)





fault tolerant : تحمل در برابر خطا
بسته شدن
(با وجود خطا همچنان ... بتوانند)
در حال بسته شدن ، یا اگر اتفاقی افتاد
(لحظه‌ای در حال باز شدن بسته شود.)

f(closed, cg) → closed (o-o),(o-c)

* nostate خطا دارد برنامه بخواهیم بنویسیم باید به صورت همه داریم این ها رو.

FSM ها در تئوری به صورت Finite automata هستند. با استفاده از گراف مدل

ورودی هر FSM به صورت مجموعه‌ای از سمبل‌های محدود در نظر گرفته می‌شود.

سمبل‌ها می‌توانند به روش‌های مختلفی تعبیر کنند.

برای مثال کاراکترها، عددها، داده‌های ورودی یا کلمات.

یا مثلاً Transaction اتفاق بیفتد ورودی مخصوص برای آن تعریف می‌شود

FSM ها را به صورت 5 تایی $M=(Q, I, f, q_0, F)$ مجموعه

نشان می‌دهند Q مجموعه محدودی از حالت‌ها است. تابع شروع پایانی

I : مجموعه محدود از سمبل‌های ورودی (Input set)

f : machine function یا Transition ها Next state Function هم هستند

نگاشت $f: Q \times I \longrightarrow Q$

F={open, closed}

$q_0$ : حالت شروع (Start state)

Q{closed, opening, open, closing}

F : زیرمجموعه‌ای از Q که مجموعه‌ای از حالت‌های پایانی است.

I={cg, og, c-c, o-o}

$q_0$=closed

یک مجموعه‌ای از ورودی‌های ماشین M رشته گفته می‌شود. رشته‌ای توسط M

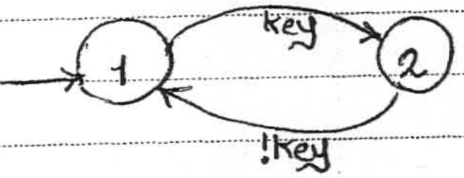
$f: Q \times I \rightarrow Q$ , f(close, og) → opening

Bit stuffing: یک فریم داریم که اگر 6 تا 1 پشت هم بیایند یعنی شروع.

Subject:
Year.  Month.  Date. ۳۹

یک FSM از حالت شروع اجرا می شود. بر اساس ورودی ها و Transition Function تغییر می کند تا به حالت پایانی برسد.

ماشین های متناهی FSM ها را (statemachine ها) با State Diagram نمایش می دهیم.



```
if((f==1) && (key))
 {
  f=2;
  print A;
 }
if ((f==2) && (!key))
{
 f=1
}
```

---

## روش Byte stuffing :

بایت به بایت ارسال می شود اما محدودیت دارد. 256 بیشتر ارسال نمی شود! (توضیح)

---

EX: رفتار کنترل شده ی یک تقاطع راه آهن را با استفاده از ساختار state Machine یا State Diagram نشان دهید.

گیت می تواند یکی از ۴ حالت Closed ـ opened ـ opening ـ Closing باشد.
ورودی از طریق فرمان های Close Gate یا open Gate سیستم وارد می شود و
هم چنین سنسورهای O_O باز کامل و C_C بسته کامل آن را نشان می دهد

حل می کنیم:

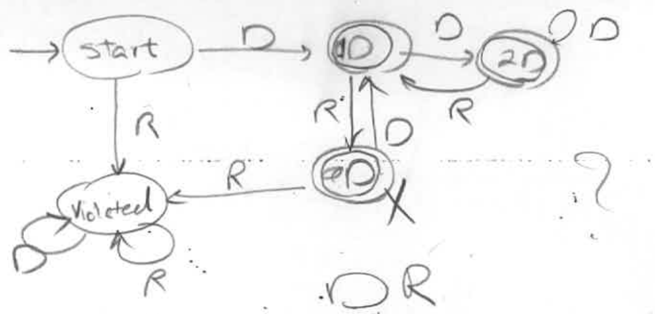
PAPCO

سوال) برای ستون (تسو) دگیذیری حالت بالا را با statemachine رسم کن ۱.۰

هوش مصنوعی

I={D,R}

Q={violated, start, 1D, 2D, ۴D}



Subject:
Year. Month. Date. ۴۲ ( )

FSM :

I={D,R}

$q_0$={start}

F={Deposited, Removed}

از این رو:

توابع انتقال

f(start, D) → Deposited
f(Start, R) → violated
f(Deposited, R) → Removed
f(Deposited, D) → Deposited
f(Removed, D) → Deposited
f(Removed, R) → violated
f(violated, x) → violated x in I

D قابل قبول است زیرا:

حالت پایانی

$$f(\overbrace{start,}^{s_0}\overbrace{D)=}^{x_0}\overbrace{Deposited}^{F}$$

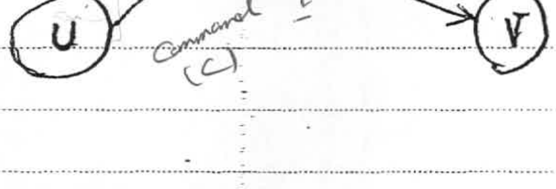
$$\underbrace{f(f(f(\overset{D}{start}, D), D), R)}_{DDR} = Removed$$

برای حل مشکلات FSM ساختار transition را عوض می‌کنیم:

[illegible]

---

: Extended FSM



به این ماشین هایی که ساختار اینطوری دارند می‌گویند:

input/Action/output

---

① یکی از مشکلات این است که خروجی با state ها مشخص هستند در صورتیکه مهم نباشد

② یکی هم $D^nR^n$ را با حالات این نوع نمی‌توانیم پیاده سازی کنیم.

state diagram را رسم کنید

Subject:
Year. Month. Date. ٤٢

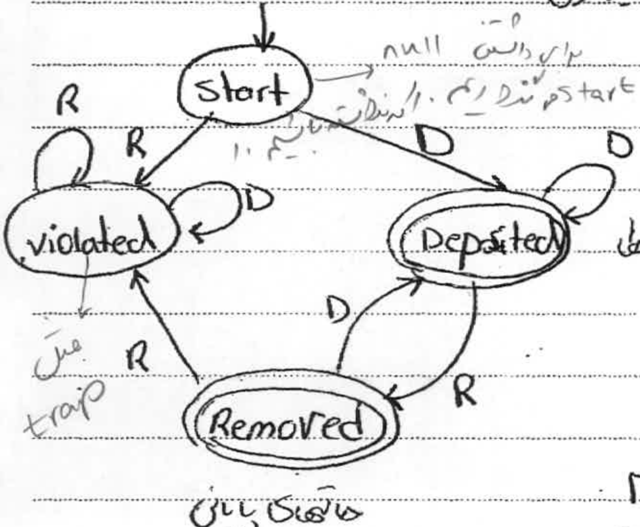
حالت پذیرفته شده.

رشته‌ای accepted یا Recognized هست که از $Q$ با توالی ورودی های
آن رشته به حالت پایانی یا Halt state ختم شود.

اگر $f(\ldots f(f(q_0,x_0),x_1)\ldots x_n) \in F$ یعنی قابل قبول است.
$(x_0 x_1 \ldots x_n)$ مثلا $q_0$ حالت قبول است.

جلسه دهم : 90/4/14

EX : فرض کنید که یک محیطی وجود دارد که دارای محدودیتی است که هر عملیات Remove R باید پس از حداقل یک عمل Deposit D انجام شود. یک FSM طراحی کنید که بتواند این سیستم را نشان دهد.
ذخیره کردن



حالتهای پایانی

روش کنترل خطا: مشخص می‌کند reset شده شود پس از خطا (violate) یعنی نتواند با D به Deposit که حالتهای خطا برود. وگرنه همه رشته‌ها را می‌پذیرفت.

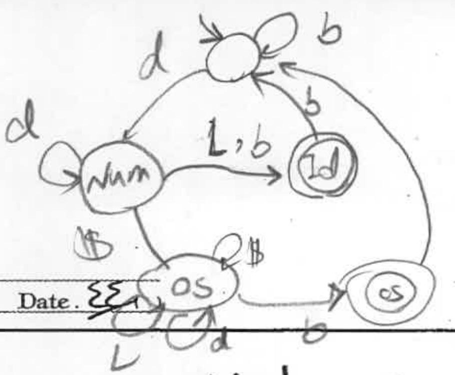
جملاتی که توسط این ماشین پذیرفته می‌شود:
D_DR_DDR_DRDDRDR
چون overwrite کردیم با چند D

غیرقابل قبول : R_DDRR
DDR
DDDRRR ← قابل قبول نیست
(فقط 2 عنصر دارد ما …) نسبت به موقعیت دارد!
* اگر حافظه در عنصر باشد چه کنیم؟! DDRR (2 ورودی 2 تا خروج ✓)

* جملاتی قابل قبول هستند ما را از حالت شروع به حالت های پایانی ◎ برسانند.
$I = \{D,R\}$

FSM
$M=(Q,I,f,q_0,F)$
Q = {start, Deposited, violated, Removed}

PAPCO



Subject:
Year. Month. Date. ٤٤

Finite state machine

۱/ تمرین: یک FSM رسم کنید ۳ object identifier یا
شناسه‌ها، unsign integer (اعداد صحیح بدون علامت) و
other string با استفاده از ورودی I شامل مجموعه I={L, d, b, $}

letter — حروف الفبا
Digit — ارقام
Blank — خط فاصله
علامت‌های خاص — ! و # و ؟ و + و / و --

FSM

با استفاده از Blank، Object ها از هم جدا می‌شوند

---

۲/ تمرین: یک بافر محدود شرایط زیر را در نظر بگیرید که متناوباً رشته‌هایی که با ۲ بار Deposit
انجام می‌شوند با ۲ بار Remove پذیرفته شود.
با استفاده از FSM ها این را پیاده سازی کنید

DR

DD RR

DD R DRR

FSM

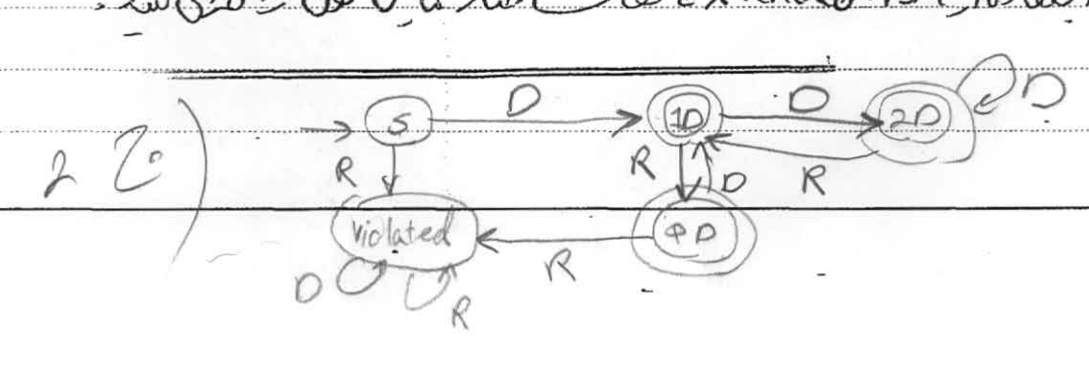
---

۳/ تمرین: یک mouse یک کلیدی را در نظر بگیرید. با استفاده از فشردن و
آزاد کردن کلید می‌توان نقطه و خط را پیاده سازی کرد. نقطه زمانی است که
updown متوالی در مدت زمان کمتر از td واحد زمانی اتفاق بیفتد و خط،
زمانی است که down و up متوالی بیشتر از td واحد زمانی باشد

راهنمایی: سیستم شامل ۳ ورودی D و U و tick است.
خروجی: Dot و Dash و eol است که eol بعد از ع
نقطه و خط تولید شود.
end of line

---

با استفاده از Extended FSM ها ساختار تمرین فوق را طراحی کنید.



(gaurd) g : یک گزاره منطقی است که می‌تواند به صورت True یا False ارزیابی شود

I : ورودی

a : مجموعه‌ای از Action ها

O : خروجی

g → I / a / a

یعنی اگر شرط درست باشد و ورودی I را بگیرد، Action a اتفاق بیفتد، خروجی O را تولید کند.
ساختار Transition ها: اگر در state u باشیم و شرط g برقرار باشد و ورودی I اتفاق بیفتد، آن گاه اقدامات داخلی a انجام شده، خروجی O تولید شود و سپس به حالت v خواهیم رفت.

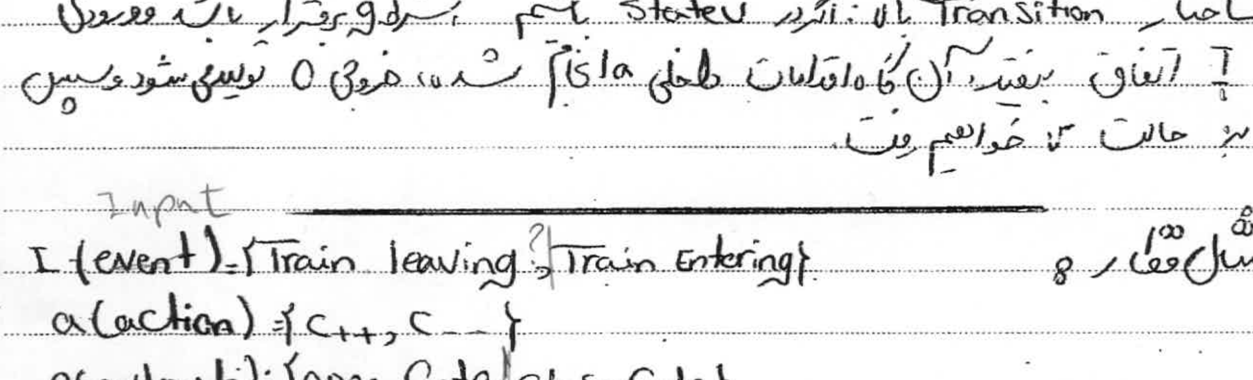
---

مثال قطار :

Input
I (event) = {Train leaving ; Train Entering}
a (action) = {C++, C--}
O (output) : {open Gate | Close Gate}
مجموعه‌ی حالت‌ها
Q = {No Train, one or more Train}

Extended FSM

حالات سیستم را با استفاده از شرایط بدست می‌آوریم.


g : I / a / O
TE
Train - Entering / Count:=1 / Close-Gate
No Train
One or more Train
TE / Count++
(Count=1) → TL / Count=0 / OG
I a O
(Count>1) → TL / Count--

یعنی باید Count=1 است
اتفاق بیفتد و Train leaving
open Gate بشه اسم Count=0

اگر تعداد قطارها
از یکی بیشتر بود

* محدودیت داریم پس نمی‌توانیم با n تا state تعداد سیستم را پیاده‌سازی کنیم با finite state machine ما.

CRSM

Subject:
Year. Month. Date. ۲۴

action ها بدین علامت نشان با خط از هم جدا می‌شوند.

? نشان دهنده دریافت پیام به عنوان ورودی است.

! به عنوان ارسال پیام یا خروجی است.

شکل کلی Transition ها به صورت c[τ] → g و نشان داده می‌شود که g همان (gaurd) گزاره منطقی است که می‌تواند درست یا غلط ارزیابی شود.

① input/output command ② c نیز (Command) که می‌تواند یک دستور ورودی خروجی یا io command باشد یا internal command و τ نشان دهنده محدودیت زمانی است.

③ internal command می‌تواند هر نوع محاسبه‌ای که با استفاده از یک زبان برنامه نویسی قابل بیان است باشد. ابزار به صورت [illegible] نیست ، loop داریم [illegible] برنامه [illegible]

هیچ محدودیتی در ساختار و پیچیدگی این برنامه وجود ندارد ولی باید برنامه ی کامپیوتری باشد. است

اگر Gaurd صحیح باشد می‌توان آن را حذف کرد.

I/O در CRSM چگونه طراحی شد؟

هنگامیکه CRSM ها ارتباط برقرار می‌کنند ارتباط‌ها به صورت همزمان و نقطه‌ای با استفاده از یک کانال دو طرفه است که ۲ ماشین را به هم متصل کرده است. هر کانال دارای یک پیامی است که ساختار پیام‌ها که از طرف کانال ارسال یا دریافت می‌شود را مشخص می‌کند.

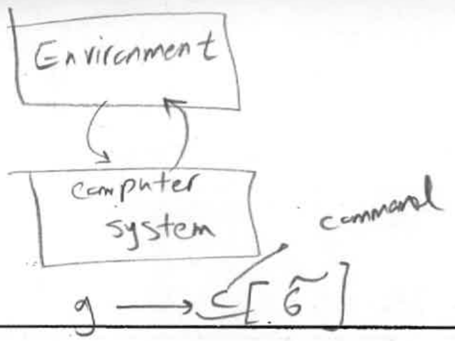
① name

یک کانال همچنین دارای یک اسم یا نشانه منحصر به فرد است که به صورت name (type) ② مشخص می‌شود.

position (x_y_coordinates)

Calendar_clock (calendar_time)

Deposit (character)



## system of state Machines:

مدل‌های State Machine برای توصیف، آنالیز و شبیه‌سازی رفتارهای سیستم‌های Real Time توسعه یافته‌اند.

ما ۲ ساختار از State Machine های پایه‌ای CLRS و CRSM را معرفی می‌کنیم:

[illegible]

### * CRSM (Communication RealTime state Machine):

زبان CRSM یک زبان کامل، قابل اجرا و منفرد طراحی شده. در این زبان، اجرای موازی داریم می‌توانند با استفاده از Message passing انتقال پیام با یکدیگر ارتباط داشته باشند. زمان در این زبان در هر transition وجود دارد.

توصیف سیستم با استفاده از CRSM به صورت یک شامل ۲ بخش environment و computer system است. داده ارسال شده و دریافت شده، پیام به صورت کامل در سیستم وجود دارد.

در نتیجه بیان مشخصات کامل یک سیستم شامل ۲ CRSM External Environment یا محیط خارجی و computer system است.

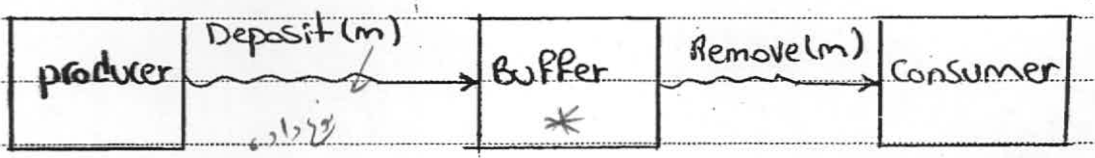
یک CRSM، یک State Machine است که برای Transition های آن، gaurded command در نظر گرفته شده باشد.

مثال: Gate Controller مثال قبل به عنوان یک CRSM است. با این تفاوت که در CRSM ورودی، خروجی و action ها با علامتی مشخص می‌شوند.
پس TE? (train Entering) ورودی و Count:=1 اقدامات
و خروجی Close_Gate!

با ? ورودی مشخص شده با ! خروجی و به جای / از خط استفاده می‌شود.

Subject:
Year. Month. Date. ٤٨

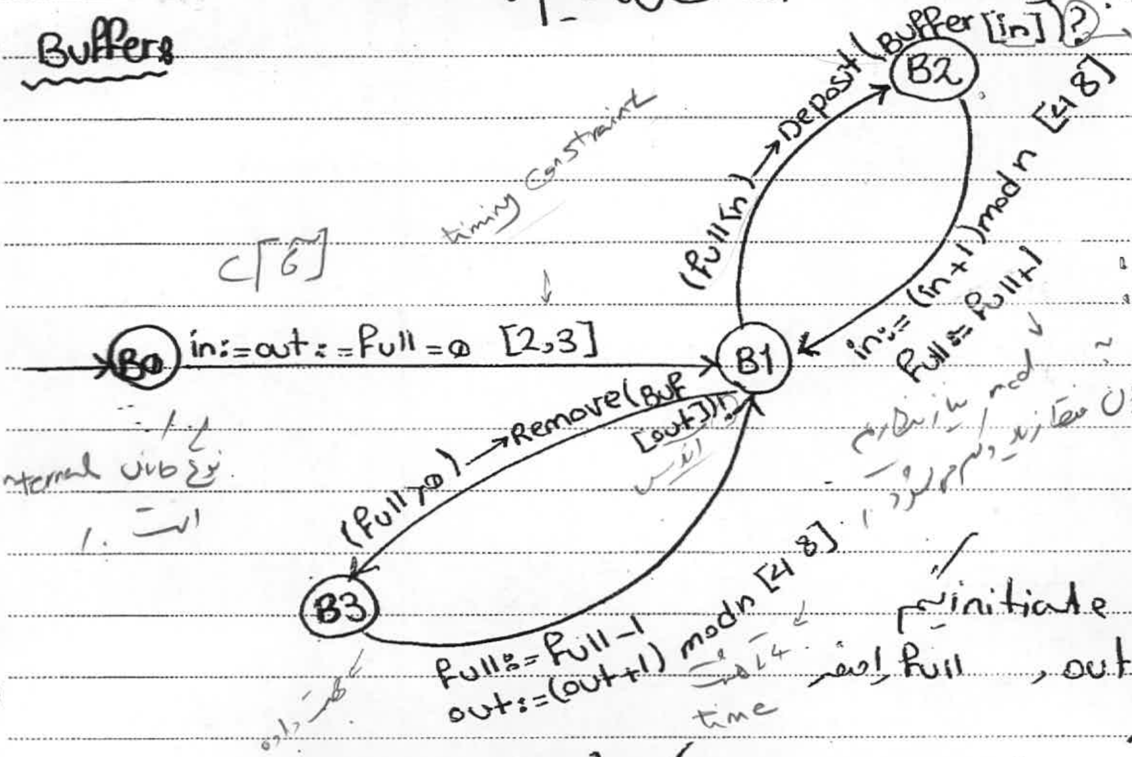
[illegible] سیستم را [illegible]


producer
Deposit(m)
نوع داده
Buffer
*
Remove(m)
Consumer

m : نوع داده‌ای (Type) داده‌ای که باید انتقال داده شود. [illegible] طول پیام است.

حالا بافر را با استفاده از CRSM نشان می‌دهیم :

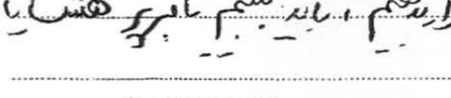
**Buffers**


B0
in:=out:=Full=0 [2,3]
B1
(Full<n) → Deposit(Buffer[in])?
B2
ورودی
in:=(in+1)mod n [4,8]
Full:=Full+1
(Full>0) → Remove(Buf[out])!
آدرس
B3
Full:=Full−1
out:=(out+1) mod n [4,8]
time
timing constraint
C[6]
نوع حالت internal است.
چون متغیر ها را داریم mod [illegible]
ظرف داده

- اول باید initiate کنیم in , out و Full را صفر می‌گذاریم.
- Transition B0 و B1 به شرط ندارد.
- این command ها از نوع internal Command هستند.
- از B1 به B2 می‌خواهیم این عنصر را در B2 قرار دهیم، باید ببینیم بافر پر هست یا نه

in و out اندیس هستند

یعنی اگر 15 را producer فرستاد، B1 تا B2 15 را ارسال می‌دهد



Full تعداد عناصر پر شده

Subject:
Year. Month. Date. ٤٧/١

I/O command

بدون ارسال عدد

open_Gate

Train_leaving → که داده‌ای را انتقال نمی‌دهد

position((14.5, 2.87))

emp → Calender_clock(((mon, Feb, 11), (10, 15, pM)))

calender time

emp → Deposit("h")

character

Open Gate

ورودی Command پیامی را دریافت می‌کند و ساختار آن به صورت زیر است

name(target)?

که name: نام کانال

target: متغیری است که با نوع کانال سازگاری دارد

داده

برای ارسال پیام از دستوری به شکل زیر استفاده می‌شود:

name(Message)!

درحالی‌که name: اسم کانال

Message: مقدار ثابتی است که با نوع کانال مطابقت دارد

Target := Message

جلسه ... ۱,۲۳

8 Bounded Buffer

EX: سیستم Real Time ای را در نظر بگیرید که ارتباط بین ۲ Consumer process و producer را از طریق بافر محدود فراهم می‌سازد.

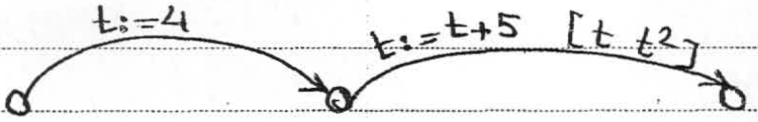
تولیدکننده پیام

producer المان‌های داده‌ای را از طریق کانال insert و Deposit می‌کند درحالیکه Consumer المان‌ها را براساس ~~First in First off~~ out (FIFO) استخراج می‌کند.

Subject:
Year. Month. Date. ۵۵/۱

این عبارت بین ارزیابی به صورت $[t_{min} < t_{max}]$ نشان داده می‌شود. منظور این صورت expr اشتباه است

اگر در عبارت $[\delta] = [expr_L \; expr_U]$ از متغیرهای استفاده شود که در اجرای Action آن Transition تغییر کند مقدار آن متغیرها در زمان محاسبه Expression ها مقدار قبل از Action است.



* شرط عدم قطعیت در این زبان.

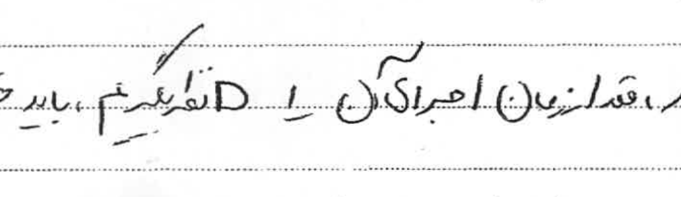
* همیشه باید $0 \leq t_{min} < t_{max}$ باشد.

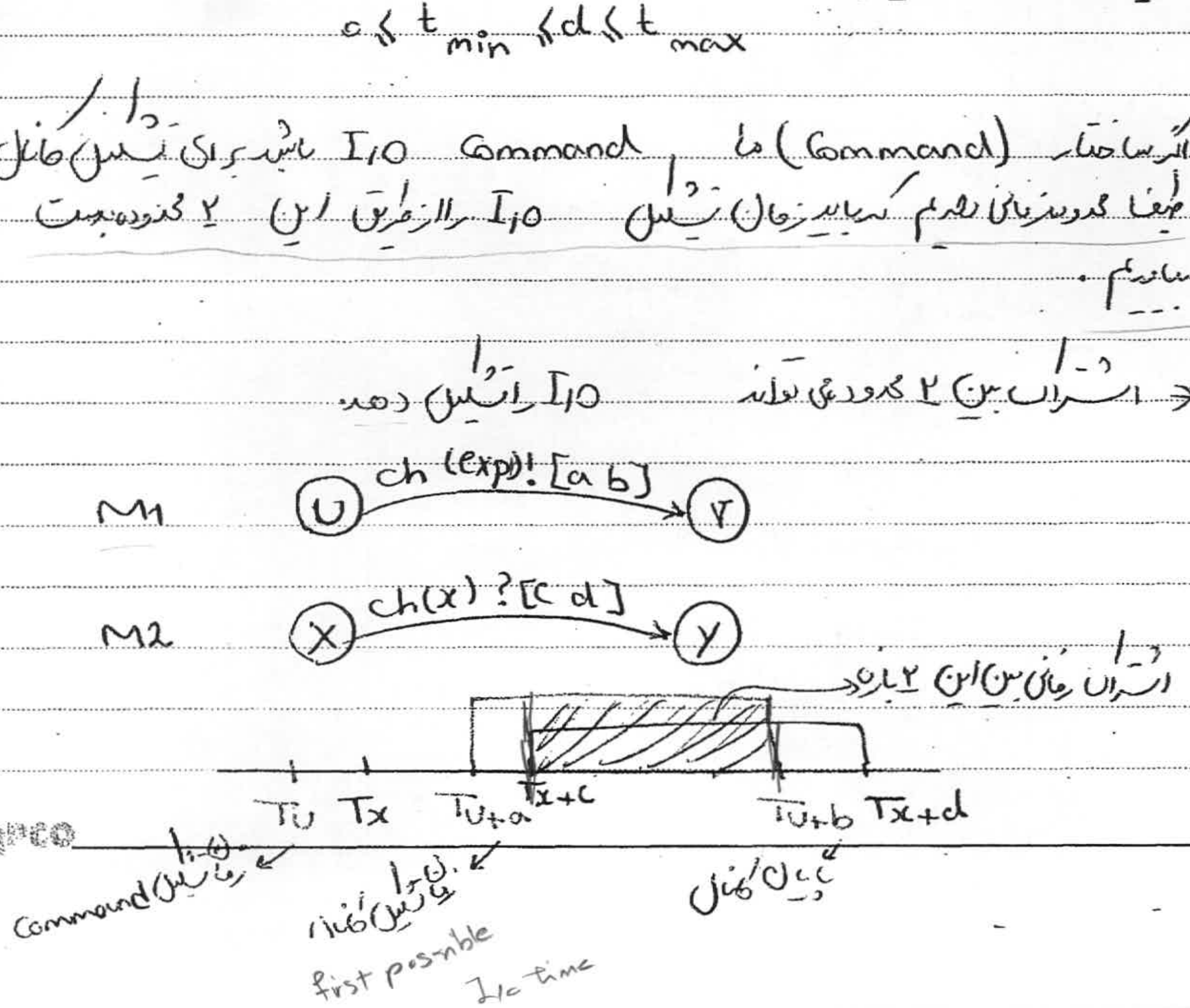
اگر C یک دستور داخلی باشد، مقدار زمان اجرای آن را D فرض کنیم، باید حتماً شرط زیر را داشته باشیم:

$$0 \leq t_{min} \leq d \leq t_{max}$$

* اگر ساختار (Command) ما Command I,O باشد برای تبادل حالت ۲ فقط محدودیت‌های زمانی که باید زمان تبادل I,O را از طریق این ۲ محدوده به دست بیاوریم.

* اشتراک بین ۲ محدوده می‌تواند زمان تبادل I,O را نشان دهد.

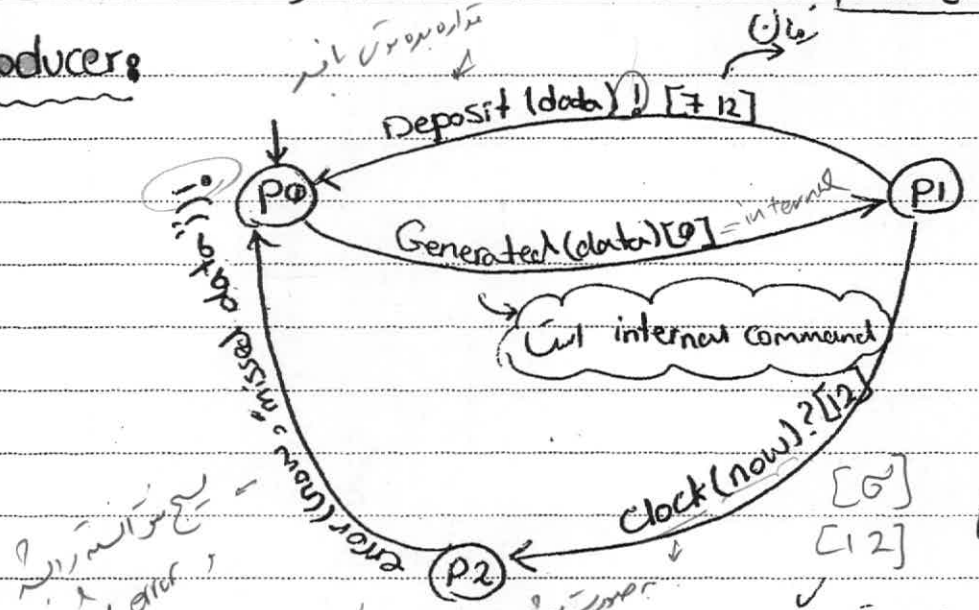

M1
U
ch(exp)! [a b]
V
M2
X
ch(x)?[c d]
Y


اشتراک زمانی بین این ۲ بازه
Tu
Tx
Tu+a
Tx+c
Tu+b
Tx+d
زمان تبادل Command
آغاز زمان تبادل
first possible I/O time
پایان تبادل

اگر کانال شلیک شود

Buf[in]:=Data

## producer:


قرار ده برو توی بافر
زمان
Deposit (data) ! [7 12]
P0
P1
P2
Generated (data) [0] = internal
internal command است
clock (now) ? [12]
[0]
[12]
error (now) ، missed ، ok
ok
هیچ سوالی درایه error شده

مثلاً [7 12] یعنی در این بازه زمانی باید عملیاتش را انجام دهد

برای کنترل این موضوع از حالت همین بودن باید خارج شود سپس باید زمان سیستم را بخوانیم.

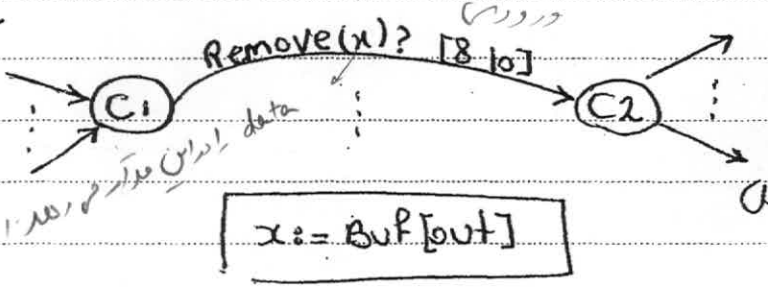
به صورت پیش فرض در کلاک هم نداریم!

این حالت ها در سطح برنامه نویسی با "Exception" انجام می‌شود! errohandling

مثلاً [12] یا [0] یعنی بازه پایین و بالایش مثلاً برای [0] میشه [0 0] است. یعنی Exact

---

## Consumer:


ورودی
Remove(x)? [8 10]
C1
C2
data را از این برمی‌دارد

x:= Buf[out]

اگر کانال شلیک شود

---

## Timing, clock:

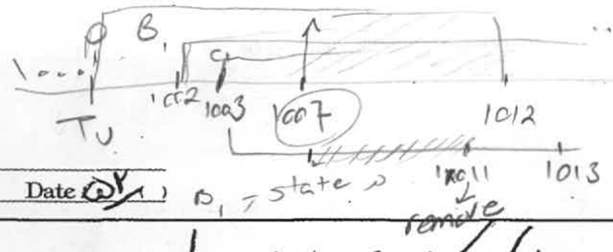
محدودیت Timing که با C[δ] نشان داده می‌شود شامل ۲ عبارت است که نشان دهنده محدوده پایین و بالای Transition می‌باشد یعنی

$[\delta] = [expr_L \quad expr_U]$

L ← حد پایین    U ← حد بالا

یک عبارت که زمان را برمی‌گرداند = expression

۷) سؤال) اگر producer در زمان 1000 در state P₁ باشد و consumer در زمان 1003 در state C باشد، چه زمانی کانال تشکیل می‌شود؟! و در زمان 1002 در state B₁ .
وقتی زمان نداریم منظور [۰ ∞] ...



1000
Tu
B₁
1002
1003
1007
1012
1011
1013
P₁ = state در
remove

Subject:
Year. Month. Date ۵/۲

وقتی در P1 است 12 واحد زمانی به طول می‌کشد تا بتواند کانال تشکیل بدهد.
Clock از زمانیکه RTC شروع می‌شود تا بی‌نهایت است.
اشتراک‌شان میشه First possible I/O Time

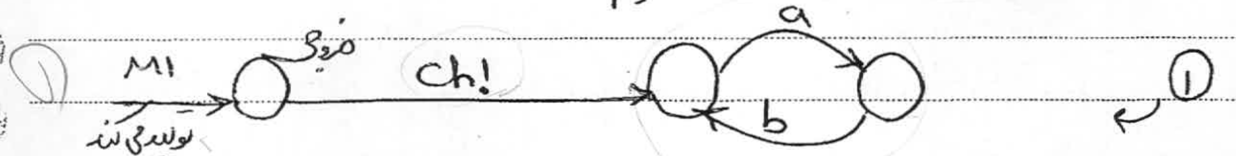
---

## 8. Dead Lock

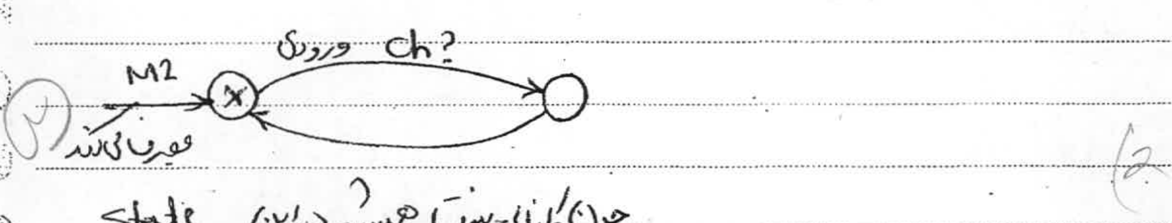
1- standard deadlock
2- time dead lock

اگر در CRSM یک state خاص باشیم و نتوانیم از این state خارج شویم اصطلاحاً Deadlock اتفاق افتاده.

۲ نوع Dead Lock اساساً داریم:



①
M1
تولید می‌کند
خروجی
Ch!
a
b
①

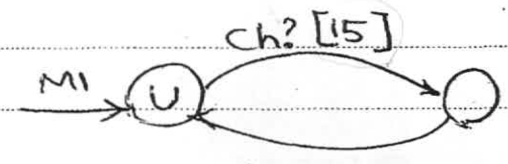


②
M2
مصرف می‌کند
ورودی
Ch?
x
②

چون کانالی نیست همیشه در این state می‌ماند و می‌گوییم Deadlock اتفاق افتاد.

② Time Deadlock

از لحاظ زمانی به Deadlock نگاه می‌کند.



M1
U
Ch? [15]
M2
x
Ch! [8-16]

PAPCO

Subject:
Year. Month. Date. ۵۱/

$T_{x+c}$ $T_{u+b}$

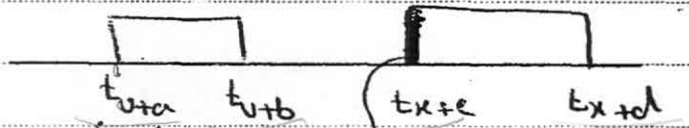
$$\boxed{t = \max(T_{u+a} , T_{x+c})}$$

$t < t_{u+b}$ , $t \leq t_{x+d}$ → آیا همان تشکیل شدن است؟!

ماشین M1 از $t_{u+a}$ تا $t_{u+b}$ می‌تواند کانال تشکیل بدهد.
ماشین M2 باید این کانال تولید شده را مصرف کند و از $t_{x+c}$ تا $t_{x+d}$ کانال تشکیل می‌دهد.

اگر این ۲ اشتراک زمانی نداشته باشند کانال اصلاً تشکیل نمی‌شود. در اولین زمانی که امکان تشکیل کانال وجود دارد کانال تشکیل می‌شود یعنی در t که در بالای صفحه نوشتم.

مثلاً:

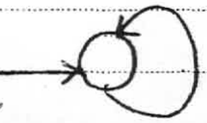


زمان شروع تشکیل کانال
پس $t \geq t_{u+b}$
پس کانال تشکیل نمی‌شود!

I/O command

---

* در CRSM‌ها نیاز به زمان واقعی است فرض می‌کنیم که ماشین $RTC_m$ وجود دارد که CRSM آن به صورت زیر است:
(Real Time Clock)



clock(rt)! [0 ∞] → همه حالات

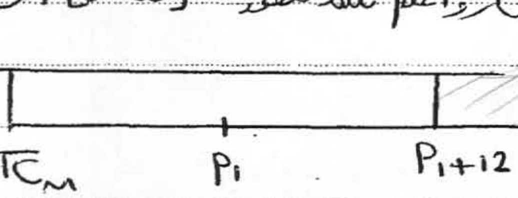
timing روی این برنامه

clock(now)?[12]

now=rt

* (همان تشکیل شدن همین کانال است)

در سیکل‌های قبل یا همین جا نیز زمان را اعلام کند منظور [0 ∞] است.



PAPCO

clock(x) ?

Subject:
Year. Month. Date ۵۴)

اضافی: Im(m) ← initialize
Am(m,d)
↳ (ـو.)

Ah(mL,ts) دستور Ah یک خط پیام اضافه می‌شود.

۹۰,۹,۳۰ : جلسه دوازدهم

state Charts:

مشکلات CRSM ها:

اگر سیستم پیچیده باشد با CRSM نمی‌توان بیان کرد.

حسن statechart:

۱- state هایی که باهم در ارتباط هستند را داخل یک superstate می‌توان جمع بندی کرد.

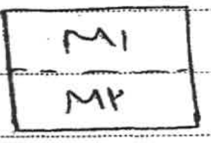
super state یا mode

برای transition هایش CRSM است مثل: Extended FSM
gaurd , input event/ command/ output event
action
I/O command
Internals
در اینجا نشان نداریم

۲- statechart ها می‌توانند ساختار ماشین‌های موازی را هم داشته باشند. ارتباط این ۲ ماشین توسط "BroadCast event" انجام می‌شود. ۲ ماشین می‌توانند بصورت موازی داخل یک superstate انجام شوند. کارهای ترتیبی هم می‌توان داخل این superstate انجام داد.

Subject:
Year. Month. Date. دوشنبه

اگر guard (شرط) همیشه درست باشد می‌توان حذف کرد

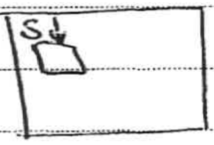


EX: count=1 → train leaving | count:=0/openGate ,

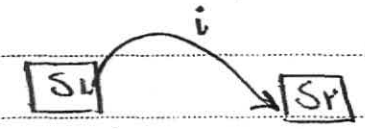
هر superstate یک start state یا entry state دارد

داخل superstate می‌توانند چند state یا superstate باشند.

* اگر ۲ تا superstate داشته باشیم که بین آنها transition اتفاق افتاده باشد:



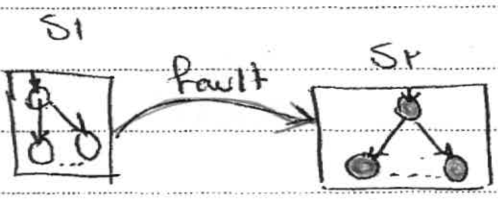
یعنی اگر در هر قسمتی از S1 باشیم، اگر event i اتفاق افتاد به start state در S۲ وارد می‌شویم.



در شکل مقابل S1 به S۲ از طریق Transition i متصل است. مفهوم این شکل این است که تمام state های S1 از طریق Transition i به entry state S2 متصل هستند.

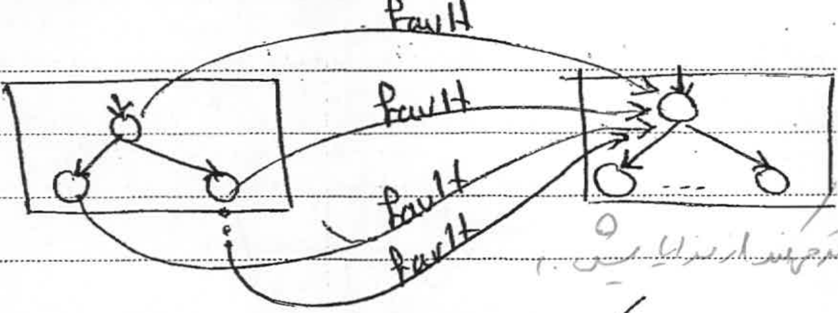
کاستی این روش نسبت به FSM ها:

۳- در state chart ها در هر state ای در S۱ باشیم با رخداد خطا به start state موجود در S۲ می‌رویم.



تولید پیام را فرت می‌کند و توسط کانال پیام را به ماشین های دیگر انتقال می‌دهد.

Subject:
Year. Month. Date.



ساختار قبل
با استفاده از FSM؛

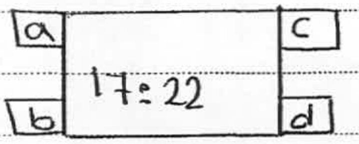
خود منظورم راحت‌تر هست از این است.

در اینجا برای تک تک state ها باید کنترل داشته باشیم و fault را اجرا دهیم ولی در روش state chart فقط یک fault قرار میدهیم

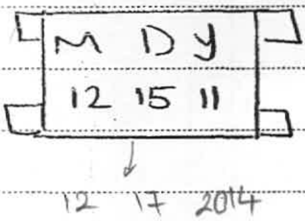
**Watch Example**

EX: فرض کنید ساعت دیجیتالی با موقعیت های ساعت، زمان، تاریخ، Alarm و hourly chime داریم که توسط کلید A و B و C و D کنترل میشوند.
رخداد ورودی

حال فرض میکنیم که برای ساده سازی این ساعت به ۳ mode (superstate)
یعنی: Display _ Normal و clock update ،
set Alarm chime تقسیم کرده‌ایم. میخواهیم عملکرد ساعت را با استفاده از
state chart ها نشان دهیم.



این ساعت ۲ مود نمایش میدهد



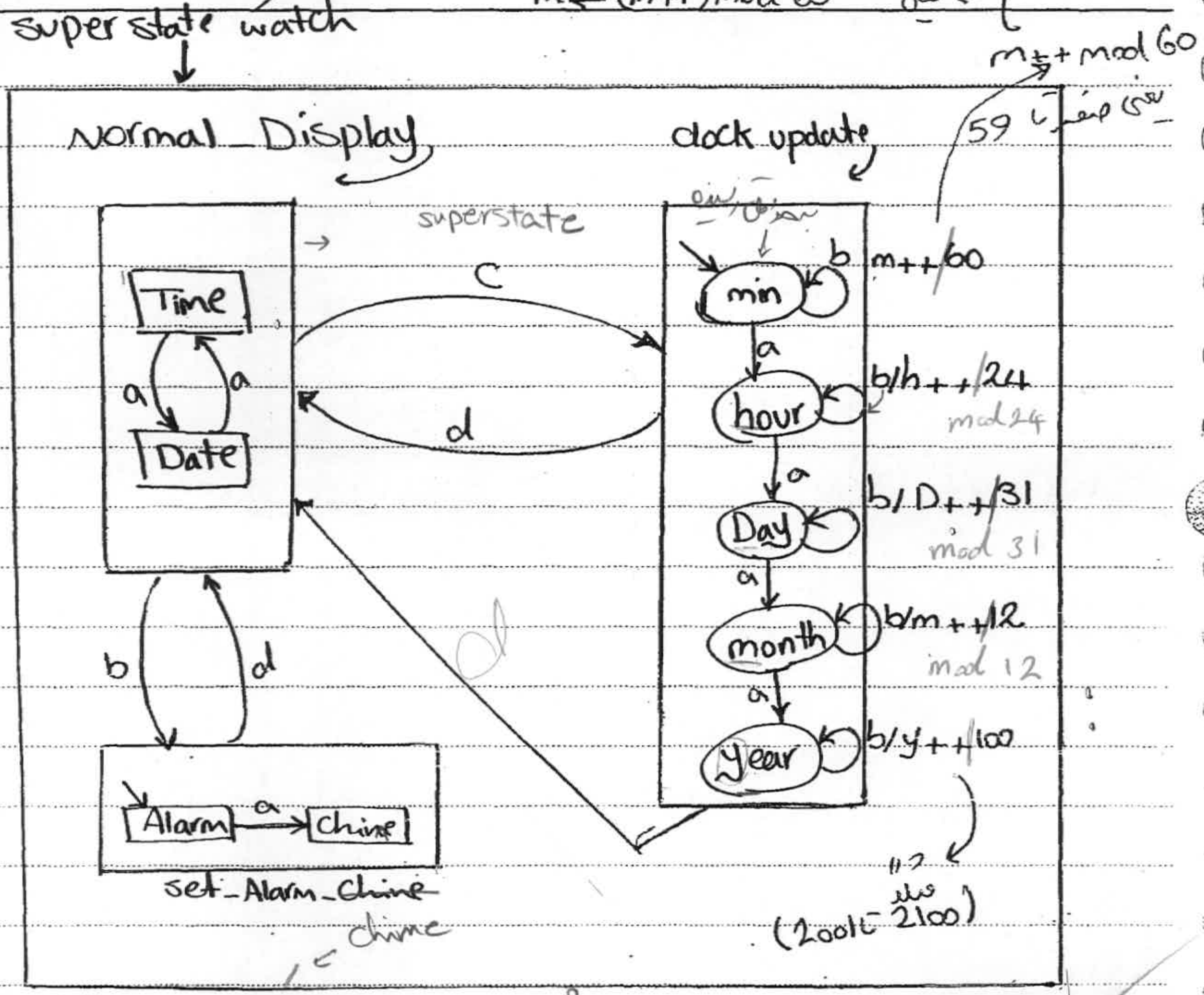
در واقع:

Normal_Display → Time: ساعت
→ Date: تاریخ

Subject:
Year. Month. Date.

$m \leftarrow (m+1) \bmod 60$ یعنی

$m++ \bmod 60$ یعنی صفر تا 59


super state watch
Normal_Display
clock update
superstate
Time
Date
a
a
C
d
b
d
Alarm
a
Chime
set_Alarm_Chime
chime
min
hour
Day
month
year
a
a
a
a
b
m++/60
b/h++/24
mod 24
b/D++/31
mod 31
b/m++/12
mod 12
b/y++/100
(2001 تا 2100)
سال

۱. set کردن هم حالت

CRSM : حالتی که هم‌زمان ...

هر state می‌تواند از superstate ها تشکیل شود.

توضیحات : برای رفتن از حالت نمایش ساعت به تاریخ دکمه a را باید فشار دهیم و برعکس

□ : superstate

○ : State ساده ← همه اینها یعنی stateDiagram هستند.

* برای تغییر ساعت و تاریخ از mode clock update استفاده می‌کنیم

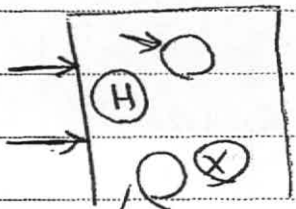
* با زدن دکمه c از حالت Normal_Disp می‌توانیم به clock_update برویم

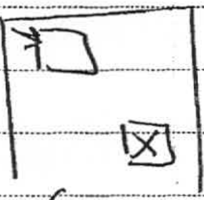
* با زدن دکمه b دقیقه یک واحد افزایش پیدا می‌کند. (برای تغییر دقیقه).

Subject:
Year. Month. Date. ۵۸

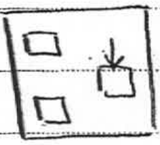
در هر state ای که هستیم اگر بخواهیم خارج شویم کنسل می‌کنیم.

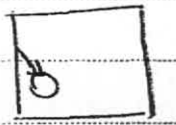
* گاهی موارد لازم است وضعیت های را ذخیره کنیم تا در موقع برگشت بتوان (
از حالت قبلی، سیستم را کار خود ادامه دهد. برای این منظور
از state chart ها، entry with history استفاده می‌شد.



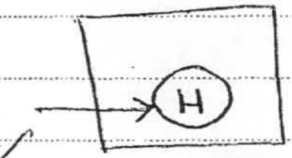


chime: زنگ‌ها و به ساعت‌هایی گفته می‌شود که سر هر ساعتی که می‌شود (۱، ۲، ۳، ...) زنگ می‌زند.
Alarm: ساعت را تنظیم می‌کنیم برای زنگ زدن.
chime: برای این است که تنظیم کنیم سر هر ساعت زنگ بزند یا نه یک متغیر است
که وضعیت از حالت False به True تبدیل شود.

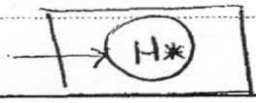




* مهم: entry with History:
اگر بخواهیم حالت‌های سیستم را ذخیره
* یعنی که در صورت برگشت به super state دیگر خارج شده از قبل خروج
آن دوباره از entry with history استفاده کنیم. مثلاً:



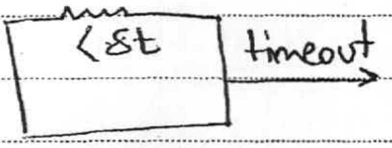
* اگر بخواهیم کلیه level های زمان خروج حفظ شود تا موقع برگشت
الگوریتم کار سیستم اجرا شود از ساختار entry with Deap history
استفاده می‌کنیم.



FSM < extended
CRSM
Statechart

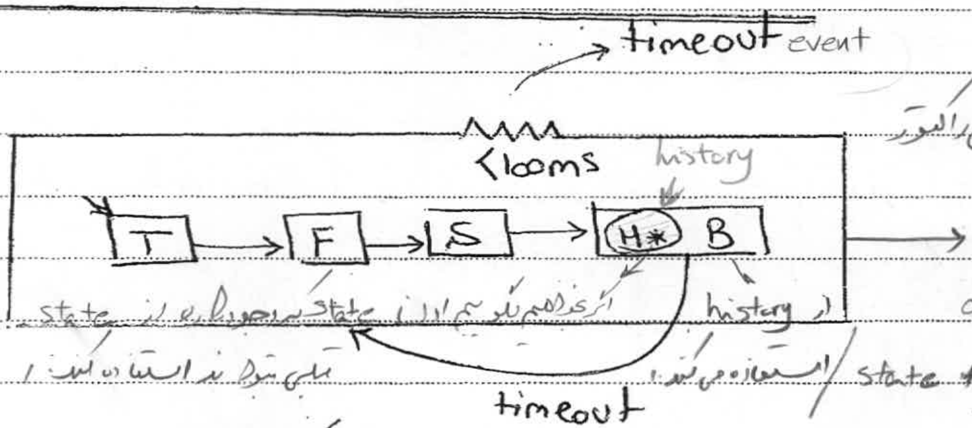
Subject:
Year. Month. Date. ۵۹

*مفهوم دیگری در statechart ها برای کنترل زمان به نام timeout وجود دارد
ساختار آن به شکل زیر است:



این به این مفهوم است که در بدو ورود به superstate timeout، Timer شروع به کار کرده و به مدت زمان δt > در state باقی می‌ماند و اگر زمان از δt بیشتر شد آنگاه از طریق transition Timeout خارج می‌شود.

مثال / [illegible]



timeout event
< 100ms
history
T
F
S
H*
B
timeout

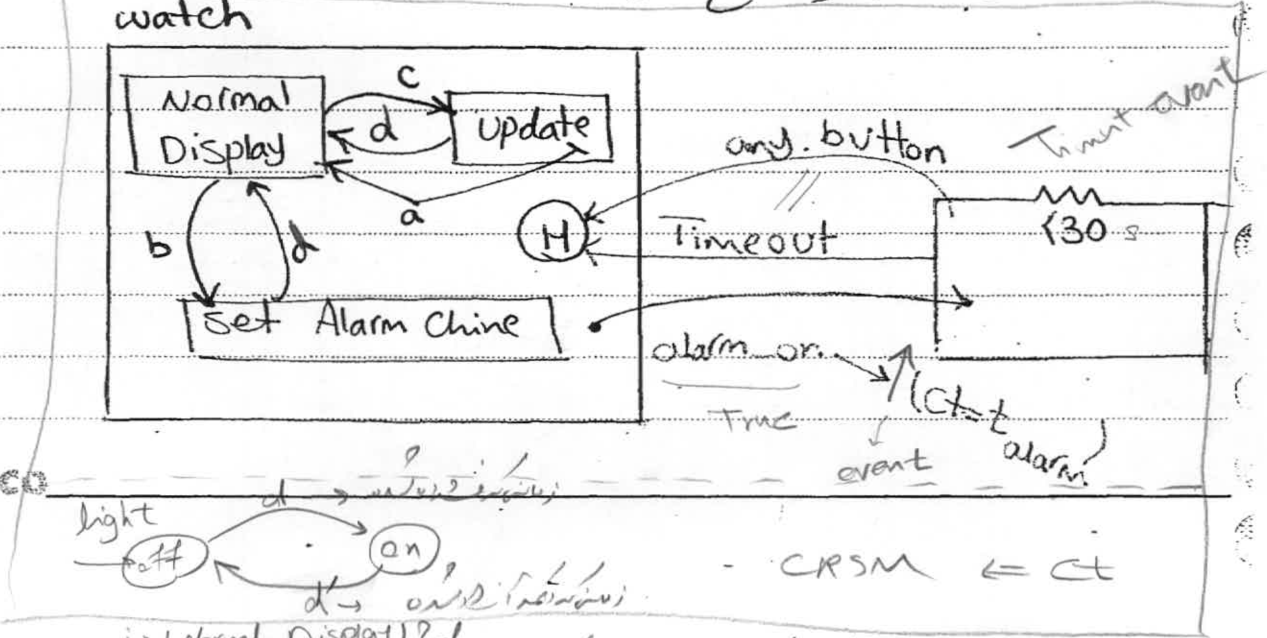
اگر بخواهیم بگوییم از آن state که بوده دوباره از state شروع شود باید استفاده کنیم.

از history استفاده می‌کنیم.

H* ⇐ deephistory

H ⇐ فقط آخرین state را به خاطر می‌دارند.

مثال / می‌خواهیم ساختار ساعت را مجهز به سیستمی کنیم که در زمان مشخص سیستم زنگ را فعال کند و اگر تا مدت زمان 30 ثانیه زنگ توسط کلیدی قطع نشد به صورت اتوماتیک قطع شود.



watch
Normal Display
Update
c
d
a
b
d
Set Alarm Chine
H
any. button
Timeout
Timeout event
< 30 s
alarm_on
True
Ct=t alarm
event
light
off
on
d
d'

d → زمانی که در حالت ... است

d' → زمانی که آمده ... شده

CRSM ⇐ Ct

in(Normal-Display)?d

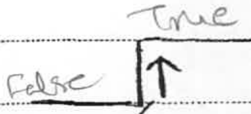
(انتظار داریم بدهیم)

Subject:
Year.    Month.    Date. ۷/۹

ثانیه $t_{alarm}$ 12:20

12:20 زمان ما (۱۱:۰۰)

Alarm روشن شد!



در لحظه‌ای که True ← False بشه

این مثال می‌شه.

trick کردن (·) → بر اساس تغییر حالت

---

## Deterministic Scheduling: (زمان‌بندی معین)

مشخص قطعی → اعداد قطعی

≠ Non Deterministic

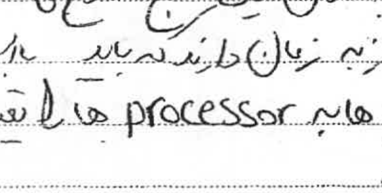
اسامی = step stochasting ⇒ بر مبنای احتمالات

تعاریف اولیه:

Scheduling: زمان در سیستم‌ها به عنوان یک منبع مطرح می‌شود. بین process ها برای اینکه اجرا شوند نیاز به زمان دارند که باید با استفاده از الگوریتم‌های زمان تخصیص process ها به processor ها را تعیین کنیم. به این عمل Scheduling می‌گویند.

به Scheduling ی Feasible گفته می‌شود که با شرایط و مشخصات process های که برای scheduling مشخص گرفته شده است بتواند cpu تخصیص دهد. در صورتیکه نتواند در زمان مناسب پاسخ دهد ← missed deadline

Optimal: به الگوریتمی Optimal گفته می‌شود که در صورت وجود راه حل برای تخصیص پردازنده، process های تعیین شده بتواند توسط آن الگوریتم راه حل Feasible شود.



Subject:
Year. Month. Date. ۹۱

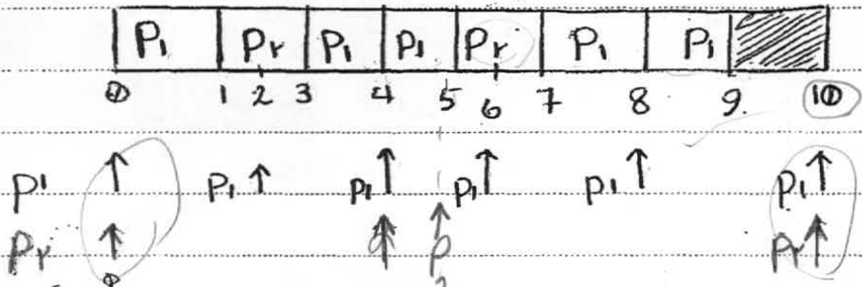
## روش priority

مثال ۱ فرض کنید $p_1$ و $p2$ دو فرایند periodic هستند دارای ویژگی $\overset{c}{\text{Compute-time}}$ و $\overset{p}{\text{periodic-time}}$ و $\overset{d}{\text{Deadline}}$ هستند

$$P_1=\overset{c\ p\ d}{(1,2,2)}$$
$$P_2=\underset{c\ p\ d}{(2,5,5)}$$

این دو فرایند از یک پردازنده مشترک استفاده می‌کنند. فرض کنید که فرایندها دارای اولویت زمان‌بندی ثابت $\pi_1$ و $\pi_2$ هستند به طوریکه $\pi_1>\pi_2$. فرایندها در زمانیکه آماده هستند مطابق اولویت آنها (اولویت بالاتر در ابتدا) زمانبندی می‌شوند. هر دو فرایند در اولین زمان یعنی $t=0$ فعال می‌شوند و فرایندها در زمان اجرا preempted می‌شوند.

← با فرض داشتن یک پردازنده با اختیار اولویت ثابت با اولویت $\pi_1>\pi_2$ و همچنین با فرض اینکه ۲ process در زمان صفر Active می‌شوند و preemptible هستند آیا process فوق طبق الگوریتم قابل زمان‌بندی هست یا خیر؟ Feasible هست

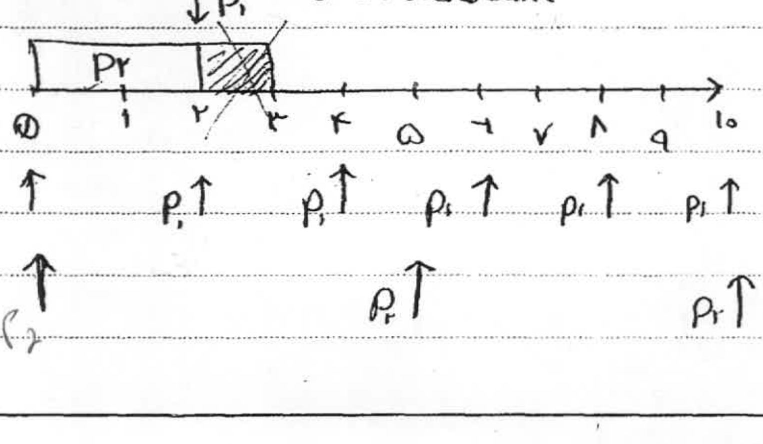


P1
P2
P1
P1
P2
P1
P1
0
1
2
3
4
5
6
7
8
9
10
P1
P2
P1
P1
P1
P1
P1
P2
P1
P2

10 ← ک.م.م دو پریودس = (2,5)

اگر $\pi_2>\pi_1$ باشد

هر دو تا در زمان صفر Active هستند



P1
missed deadline
P2
0
1
2
3
4
5
6
7
8
9
10
P1
P1
P1
P1
P1
P2
P2
P2

Feasible نیست

cpu utilazation: $\frac{1}{2}+\frac{2}{5}=\frac{9}{10}=\%90$

$$\boxed{\sum_{i=1}^{n} c_i/P_i}$$

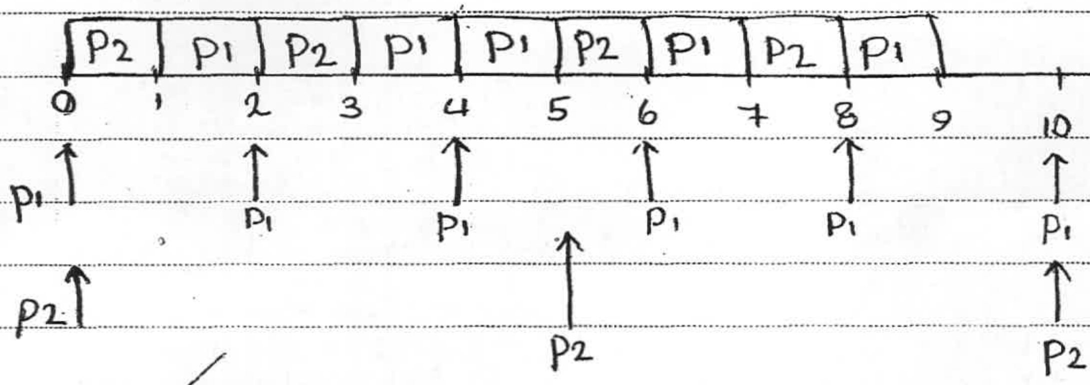
Subject:
Year. Month. Date. ۹۴

# Round Robin

8 در این روش فرایندها اولویت ندارند
هر کدام زودتر بیاید زودتر اجرا می‌شود.

(preemted)

Timeinte یکی است ، فرق نمی‌کند p1 یا p2.



از این به بعد برای مسائل scheduling فرضیات زیر را در نظر بگیرید (حالت ساده)

فرض اول: سیستم‌هایی که بررسی می‌کنیم همگی برنامه‌ای هستند.
مجموعه‌ای process های periodic هستند به صورت ۳تایی
معین شده‌اند. به شکل $p_i=(c_i,p_i,d_i)$

$[p_i=(c_i,p_i,d_i) : p_i=d_i]$

فرض دوم: همه process های زمان به فرض صفر در نظر گرفته می‌شود برای اینکه
اول همه آماده هستند

فرض سوم: به غیر از موارد گفته شده پروسس‌های در نظر گرفته شده preemptible
هستند.

مرحله‌ای در زمان $T_0$ با start می‌شوند ، فرض
ادامه در صفحات بعد می‌شود

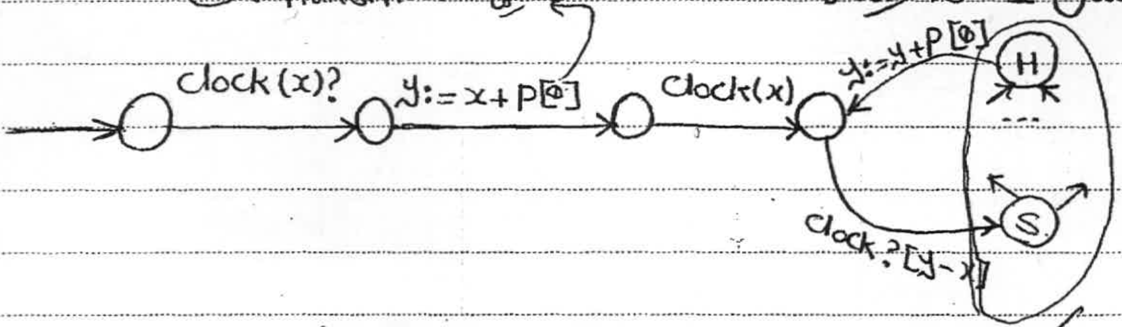
[illegible]

Subject:
Year.  Month.  Date. ۹۳

« جلسه حل تمرین »

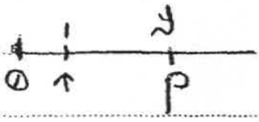
جواب مثال ۱ صفحه ۵۳:

Transition زمان ۰ است


Clock(x)?
y := x + p[0]
Clock(x)
y := y + p[0]
H
S
Clock?[y-x]

کنترل‌سنتزای که در زمان period ش active نشده‌اند Timing است

الان ما ساعت را از سیستم گرفتیم.



$y = x + p$

مثلا $p = 10$   $y = x + p$

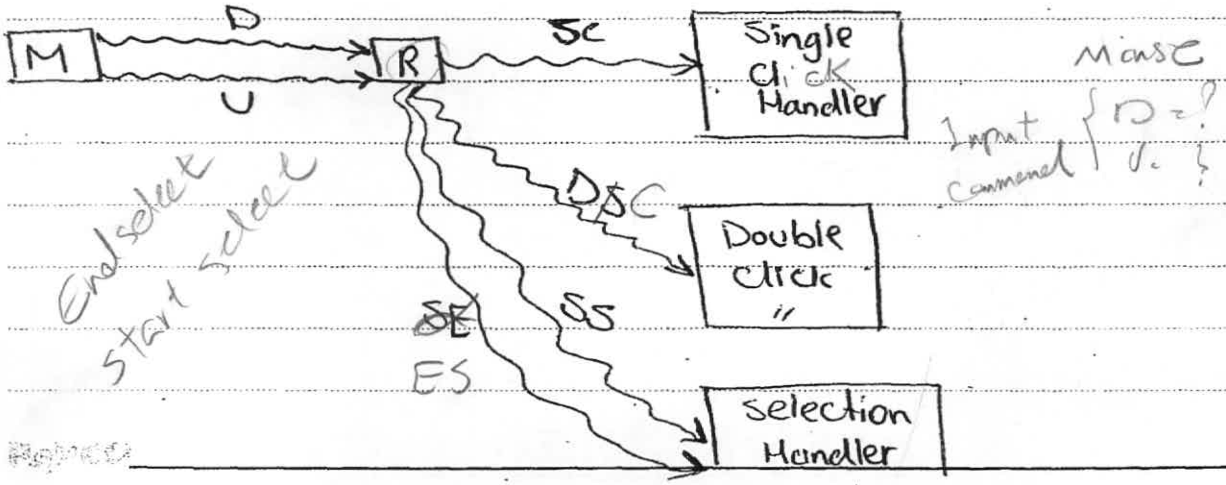
۹ لحظه زمانی باید سپری بشه تا I/O ش فرا رسد.

[0 ∞]   Time I10 در First possible اتفاق می‌افتد یعنی ۹ و [9 9]

Clock?[9] = Clock?[9 9]

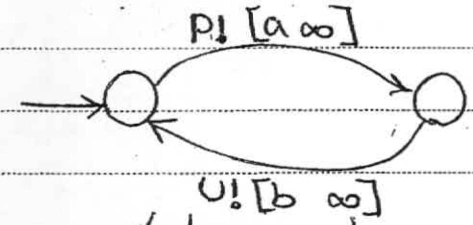
---

جواب مثال ۲ صفحه ۵۳:


M
D
U
R
SC
Single Click Handler
DSC
Double Click
SS
~~SE~~
ES
Selection Handler
End select
start select
Mouse
Input command
D = ?
U = ?

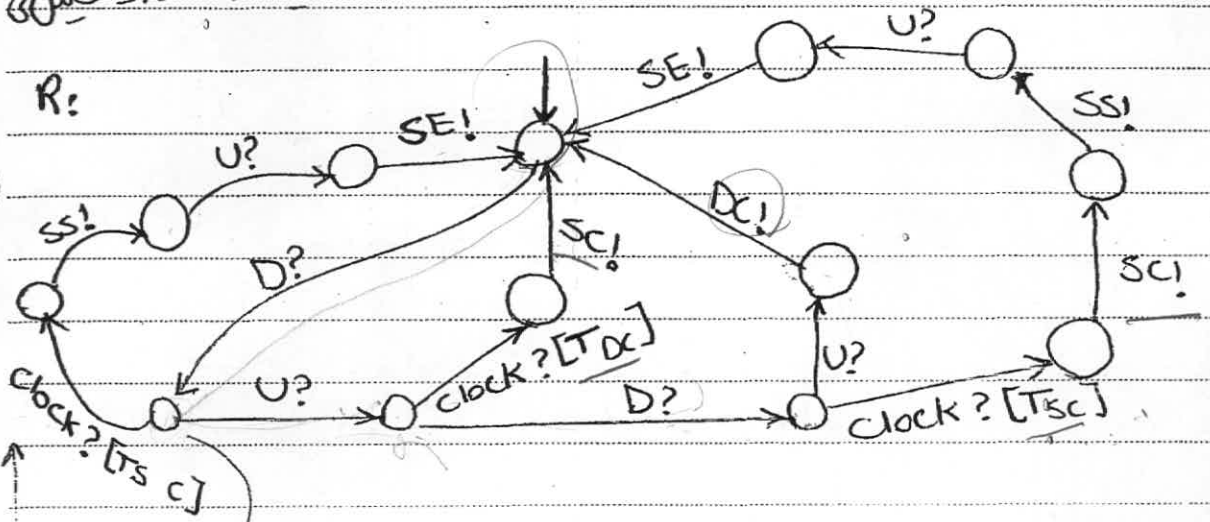
Subject:
Year. Month. Date. ۲۴ مهر

M:



واسه اینکه بگیم در این را I10 سیگنال شده

R:



SE!
U?
SE!
SS!
SS!
D?
SC!
DC!
SC!
U?
U?
clock ? [T DC]
D?
clock ? [Tsc]
clock ? [Ts c]

موقع movies اینجا لازم
دوره به این —
State

Tsc
TDC

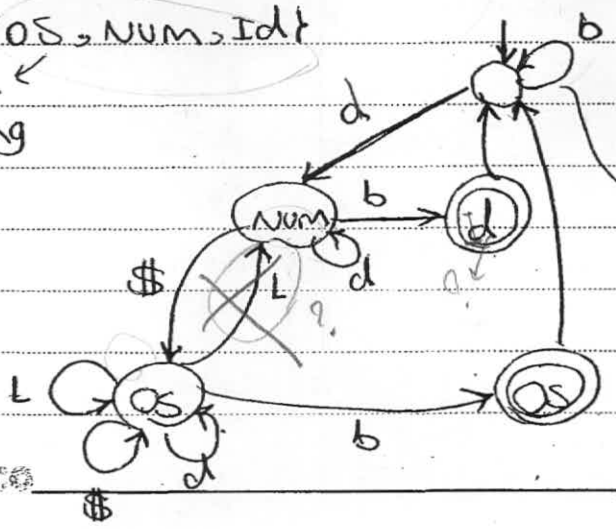
---

I = {d, L, b, $}

O = {OS, NUM, Id}

other
string

تمرین ۱ ص ۴۴ ۵



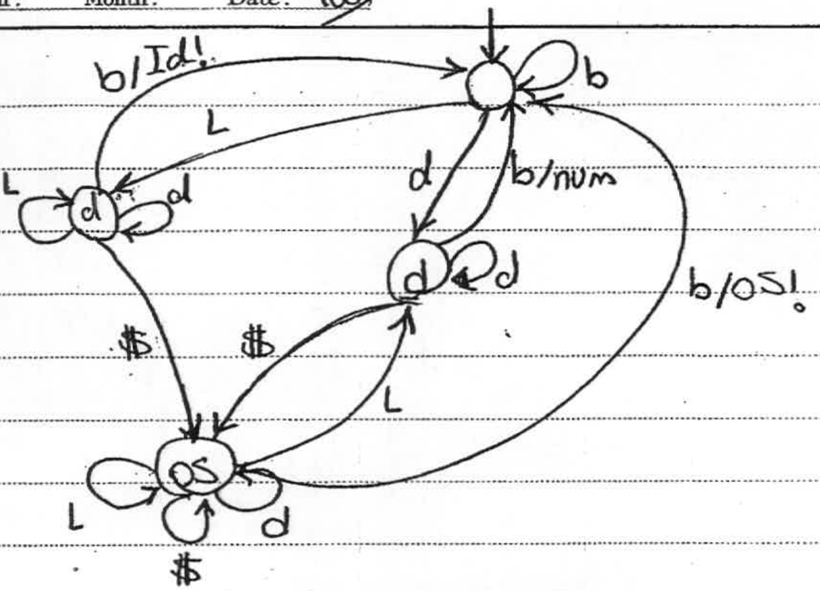
b
d
b
NUM
d
Id
$
L
L
OS
OS
b
d
$

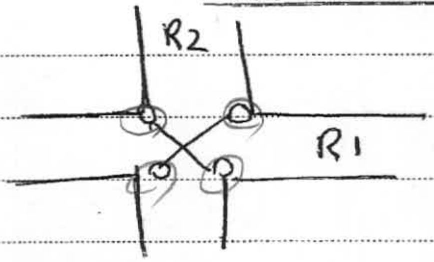
در ابتدا یا d یا c، b، $

یعنی blankها که اضافه حروف

Subject:
Year. Month. Date. ۷۵/۷

روش سوم


b/Id!
b
L
d
d
L
d
b/num
d
d
b/OS!
$
$
L
OS
L
d
$



تمرین ⑤ ص ۳۸ :

① amb ent $R_1$
② amb ent $R_2$
③ amb exit $R_1$
④ amb exit $R_2$
⑤ 45 sec

Rules

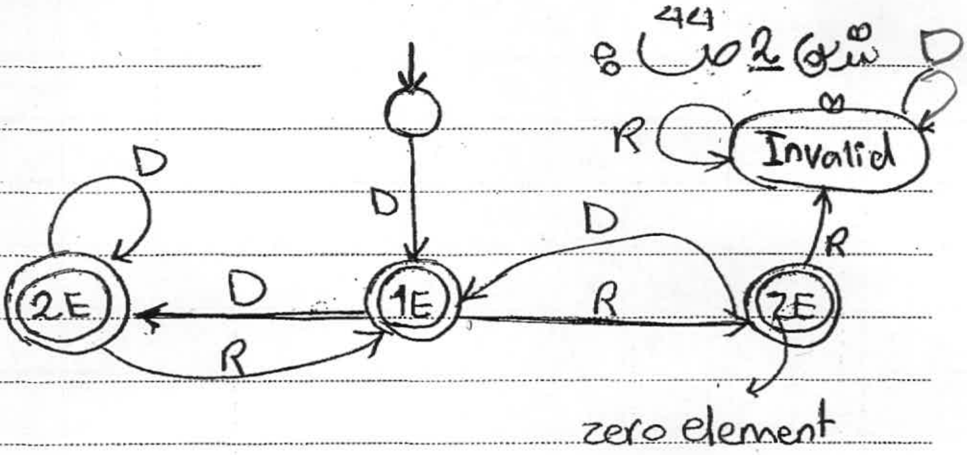
↑ علامت event L5

| | | | | |
|---|---|---|---|---|
| R2 قرمز S1 | ↑45sec | | amb ent R1 | amb ent R2 |
| R1 قرمز S2 | | 45 sec | amb ent R1 | amb ent R2 |
| R1 سبز S3 | | amb exit R1 | | |
| R2 ورود S4 | amb exit R2 | | | |
| | S2 | S1 | S3 | S4 |
| Action | | | | |

Subject:
Year. Month. Date. ۹۴/ /

DR
DDRR
DDR DRR

تمرین 2 ص ۴۴ :


D
D
D
R
R
D
R
R
D
2E
1E
ZE
Invalid
zero element

تمرین ۱ ص ۳۷ :

| | |
|---|---|
| C | T |
| Sample | P |
| T>C | MU |
| T<C | MD |
| دما تغیر نکند | KS |
| T<-273 | ۳ پریود متوالی |
| T>500 | Error |

صورت سوال

جواب : ص بعد

Subject:
Year. Month. Date. ٦٧ ( )

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| T=C | Y | N | N | N | N | Y | N | N |
| T>C | N | Y | N | Y | N | N | Y | N |
| T<C | N | N | Y | N | Y | N | N | Y |
| T<-273 | N | N | N | N | Y | N | N | Y |
| T>500 | N | N | N | Y | N | N | Y | N |
| EA=3 | N | N | N | N | N | N | Y | Y |
| KSnum=0 | N | N | N | / | / | Y | / | / |
| MU | — | ~~X~~ | ~~X~~ | — | — | — | — | — |
| MD | — | ~~X~~ | ~~X~~ | — | — | — | — | — |
| EA | — | — | — | — | — | — | X | X |
| KS | — | — | — | — | — | X | — | — |
| EC++ | — | — | — | X | X | — | — | — |
| CSnum++ | X | — | — | — | — | — | — | — |
| EC=0 | X | X | X | — | — | X | — | — |
| KSnum=0 | — | X | X | X | X | — | X | X |

۱۴ حالت پیش میاد. یعنی حالت های از اول علاوه به دو تا هستند.

N و Y

تمرین ۳ ص ۴۴ 8

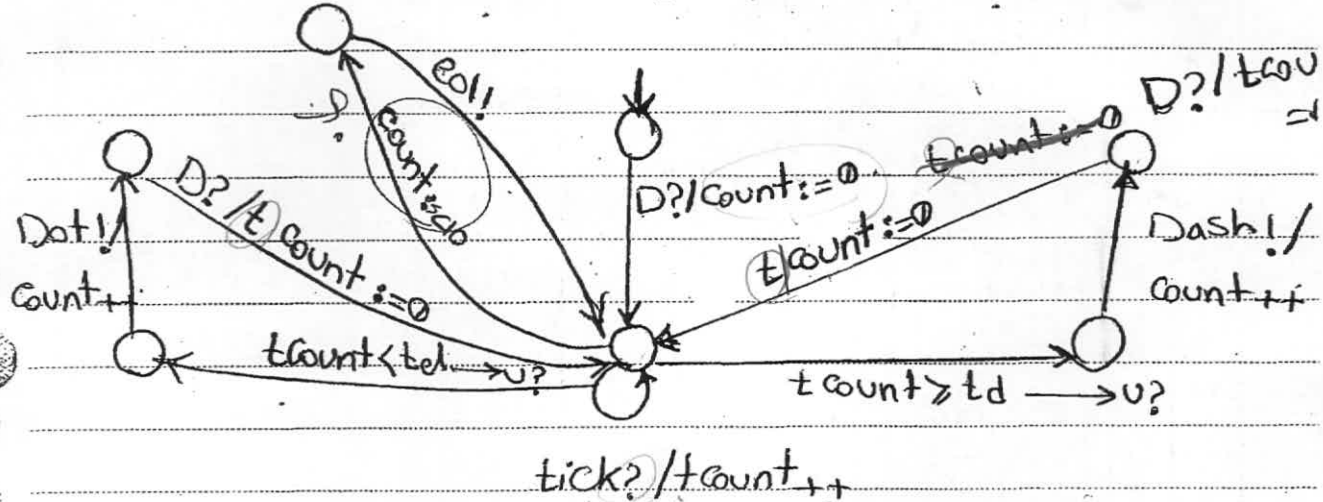
ورودی { D, U, tick

خروجی { ۰, −, eol

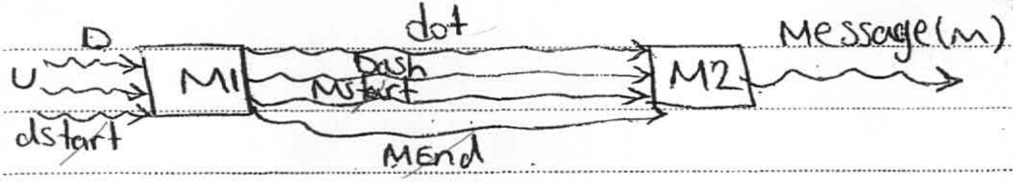
D <td U → ۰ خروجی

D >td U → − خروجی

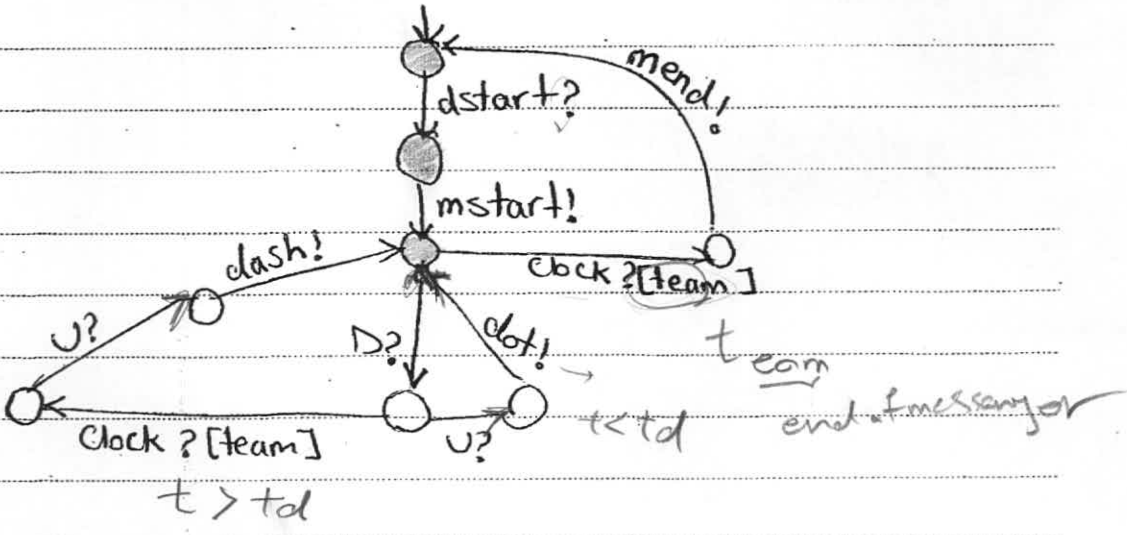
در نقطه شروع تعداد نقطه و خط هایی که می گیرد باید صفر شه!
eol خروجی تعداد نقطه و خط به 40 رسید.


eol!
count:=0
D?/count:=0
D?/tcou
~~tcount:=0~~
tcount:=0
Dash!/
count++
Dot!
Count++
D?/tcount:=0
tcount<td → U?
tcount≥td → U?
tick?/tcount++

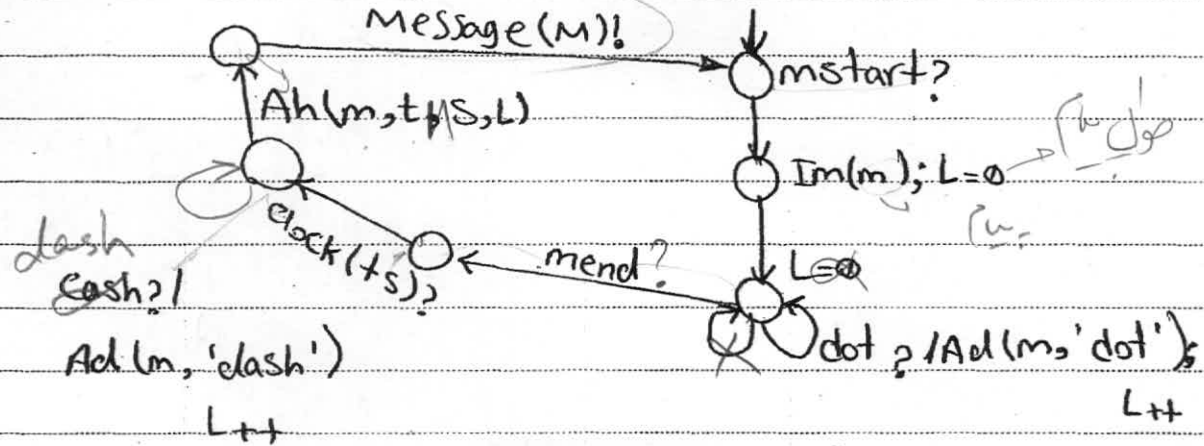
تمرین ۳ ص ۵۳ :

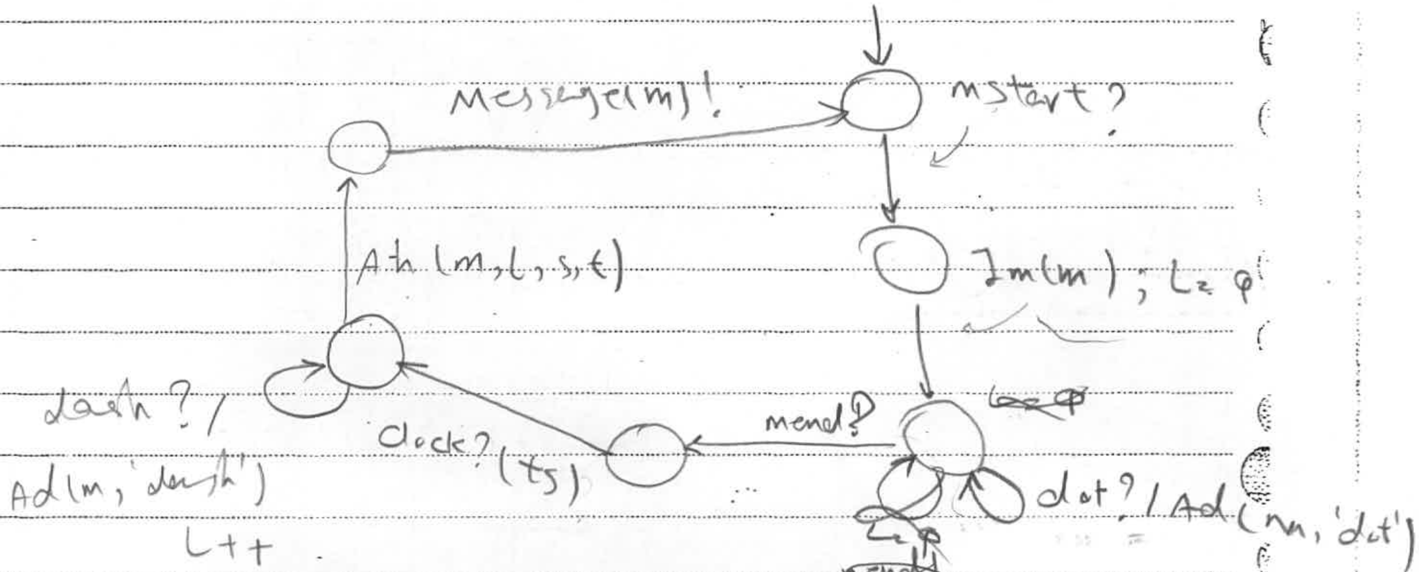

D
U
dstart
M1
dot
Dash
Mstart
Mend
M2
Message(m)

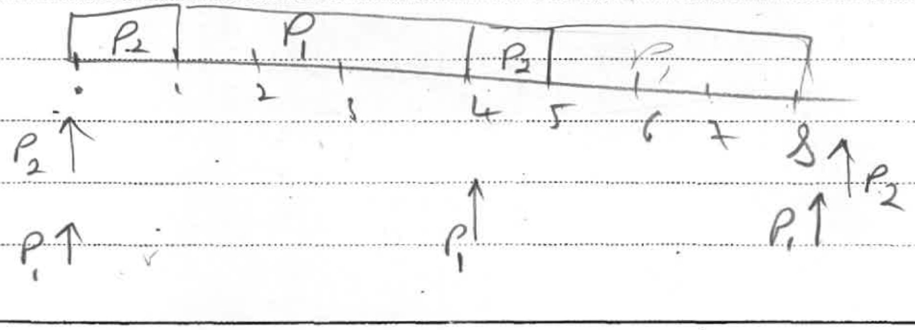
M1:


mend!
dstart?
mstart!
clock?[team]
dash!
U?
D?
dot!
Clock?[team]
U?
t>td
t<td
team
end.fmessanger

M2e


Message(M)!
mstart?
Ah(m,t,S,L)
Im(m); L=0
L=0
~~cash?~~ dash?/
Ad(m, 'dash')
L++
clock(ts)?
mend?
dot?/Ad(m,'dot');
L++


Message(m)!
mstart?
Ah(m,L,s,t)
Im(m); L=φ
dash?/
Ad(m,'dash')
L++
clock?(ts)
mend?
dot?/Ad(m,'dot')
L++


P2
P1
P2
P1
0
1
2
3
4
5
6
7
8
P2
P2
P1
P1
P1

EDF ① } Dynamic
SCT ②
LST ③

① : RM : اولویت با پریود کوتاه‌تر

② : LCT : اولویت با compute time کمتر

=> fixed priority

Fixed priority :

١) یکی از معروف‌ترین این الگوریتم‌ها Rate monotonic (RM) است. ـ نرخ یکنواخت ـ

اولویت process ها براساس period آن‌ها تعریف می‌شود. process با period کوتاه‌تر اولویت بالاتری دارد.

این الگوریتم optimal است. یعنی اگر الگوریتم RM feasible نباشد برای پراسس‌ها feasible نیست.

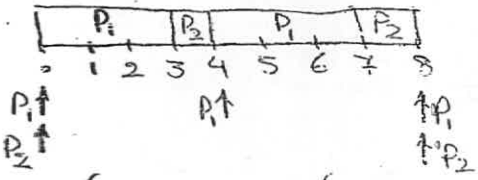
نرخ اجرا بیشتر

هرچه پریود کمتر اولویت بالاتر. یعنی هر کدام زودتر سرویس احتیاج داشته باشند آن‌ها بالاتر.

اگر اولویت ثابت P1 باشد

مثال: این مثال توسط Round Robbin، failed شد. می‌خواهیم ببینیم آیا توسط RM زمان‌بندی هست یا نه؟

P1 (3, 4, 4)
P2 (1, 8, 8)



اولویت با اونی که زمان period کوتاه‌تری دارد یعنی زمان execution کمتر => اولویت P1 بیشتر از P2 است.

ایده خوبی است چون process هایی که زمان از cpu می‌گیرند سریع cpu را می‌گیرند و پس می‌دهند.

2) least compute time (LCT): آنی که زمان اجرای کوتاه‌تری دارد اولویت بالاتری دارد. هرچقدر cpu time کمتر زودتر اجرا شود.

در مثال: اگر اولویت P2 بالاتر باشد دیگر feasible نیست. RM از LCT بهتر عمل می‌کند.

Dynamic priority : دینامیک (d-t) را حساب می‌کنیم

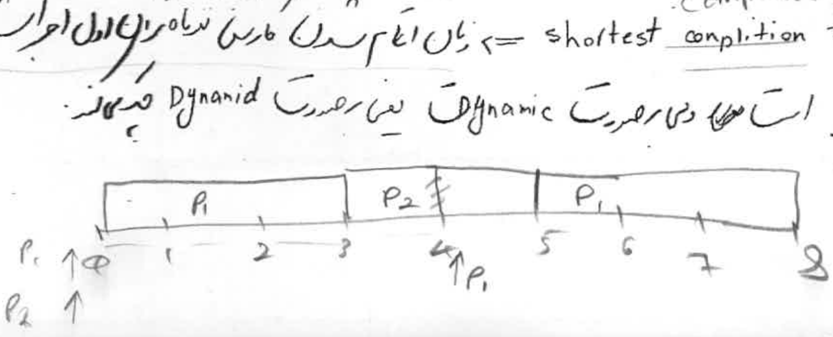
1) earliest deadline first (EDF): زمان لازم برای اینکه process تمام شود deadline. هر کدام زودتر Deadline ش تمام شود اولویت بیشتری دارد.

آنکه deadline کمتری دارد اولویت بالاتری داشته باشد.

3) shortest completion time (SCT) => زمان اتمام شدن کارش کوتاه‌تر اول اجرا شود. شبیه LCT است منتها در هر لحظه Dynamic یعنی در هر لحظه Dynamic چک می‌کند.

computation

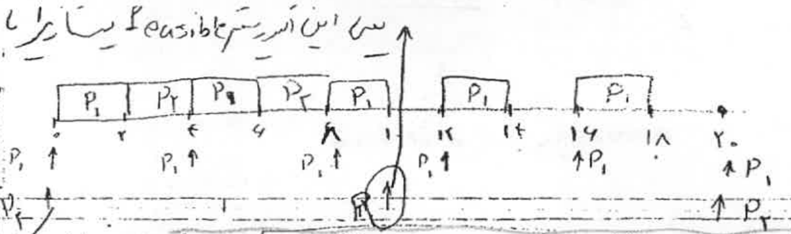
همه اجراها بررسی کنیم که اگر اولویت بدهیم:



تعیین مثال پایین بر روش RM :

$P_1$ بیشتر زیاد بر بودن تکرار ۲۰. بسته تر است.

یعنی این الگوریتم feasible نیست زیرا در زمان ۱۰ بار $P_2$ اجرا نشده است


P₁
P₂
P₂
P₂
P₁
P₁
P₁
۰
۲
۴
۶
۸
۱۰
۱۲
۱۴
۱۶
۱۸
۲۰
P₁
P₁
P₁
P₁
P₁
P₁
P₂
P₂

هیچ الگوریتمی وجود ندارد برای feasible نه زیرا اینترشده Optimal است.

مثال $P_2$ دو periodic process: $P_1(2,4,4)$ و $P_2(5,10,10)$ را می‌خواهیم زمانبندی کنیم

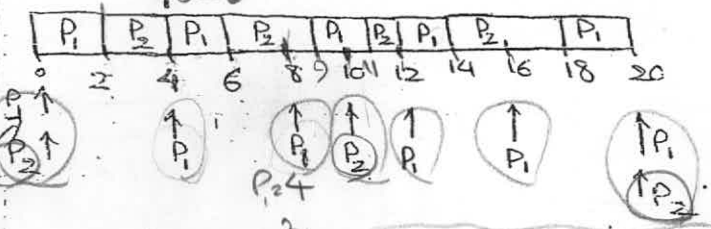
$-d-t$

deadline $P_2 = 6$ 10

deadline $P_1 = 4$ ✓ $8(d-t)$

$d-t$

اولویت با $P_1$ چون deadline کمتری دارد


P₁
P₂
P₁
P₂
P₁
P₂
P₁
P₂
P₁
0
2
4
6
8
9
10
11
12
14
16
18
20
P₁
P₂
P₁
P₁
P₂
P₁
P₁
P₁
P₂
P₂4

deadline ها را با ... چه کنیم اینجا؟

تا زمان شدیم، اتمام زمان دهد

اولویت‌ها را هم در Dynamic ها

## LST

مثال LST: least ~~slack~~ stack time — کمترین زمان سستی —

slack

مثل EDF است در زمان کارشیدن آن هم وجود دارد (اینجا $d-t$) مثل deadline است

محاسبه زمان سستی: $(d-t)-c'$

$(s-t)$

زمان deadline — زمان جاری

مقدار زمان لازم برای اجرای $P$

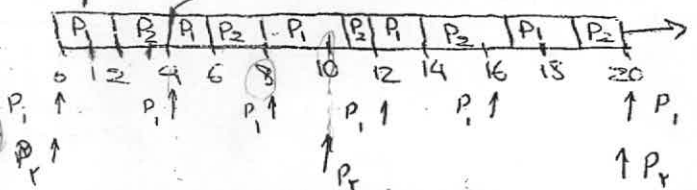
مقدار computation که مانده time

برای $P_2$: $(20-12)-4$

در زمان 1: $P_1 = 4-1-1 = ②$

$P_2 = 10-4-3 = ③$

مثال $P_2$: در زمان‌های ... باید ... است


P₁
P₂
P₁
P₂
P₁
P₂
P₁
P₂
P₁
P₂
0
1
2
4
6
8
10
12
14
16
18
20
P₁
P₁
P₁
P₁
P₁
P₁
P₂
P₂
P₂

تمام شد. context switching

$P_2 = 10-1-$

$2 + P_1 = ~~4-2-0~~ = ②$

$P_2 = 10-2-5 = ③$

$P_1: (4-0)-2 = ②$ ✓ slack time

$P_2: (10-0)-5 = 5$

در زمان صفر

$10-3-4=3$

~~$4-3-2$~~

اگر $P_1$ یک واحد اجرا شود زمان تعلل تغییر نمی‌کند همان $2$ می‌ماند. در صورتی که وقتی $P_1$ اجرا می‌شود زمان تعلل $P_2$ کمتر می‌شود.

$P_2: (10-1)-5 = 4$

$P_1: (8-4)-2 = 2$ ✓

$P_2: (10-4)-3 = 3$

در زمان $t=4$

$P_1 = (12-8)-2 = ②$

$P_2 = (10-8)-5 = 3$

$t = 8$

$d \quad t$

$P_2: (20-10)-5 = 5$

$P_1: 2$ ✓

$t = 10$

$(12-10)-0 = ②$

اگر در نظر گرفته شد non preemtive است!

نکته ۱- RM جزء الگوریتم های optimal است. یعنی الگوریتمی که اگر وجود داشته باشد می‌تواند اجرا شود. و هر الگوریتم Fixed داشته باشیم که بتواند process ها را زمانبندی کند RM هم می‌تواند اگر وضعیت feasible زمانبندی کند. با استفاده از هر الگوریتم دیگه ای شد حتما جواب بگیرید با

RM هم می‌شود جواب بگیرید ۱۰۰

در Dynamic، EDF و LST جزء الگوریتم های optimal هستند.

نکته ۲- اگر داشته باشیم n تعداد process ها است

$$\lim_{n\to\infty} n(2^{1/n}-1) = \lim_{n\to\infty} \frac{2^{1/n}-1}{1/n} = \frac{-\frac{1}{n^2}\ln 2}{-\frac{1}{n^2}} = \ln 2 \simeq 0.69$$

اگر بی‌نهایت process داشته باشیم و میزان مصرف cpu یعنی (u) کمتر یا مساوی ۶۹٪ باشد آنگاه با استفاده از الگوریتم های fixed priority می‌توان آنها را زمانبندی کرد. در غیر این صورت

$$U \le n(2^{1/n}-1)$$

معلوم نیست.

$$70 \le 69$$

به همین دلیل برای سیستم هایی که می‌خواهیم استفاده کنیم U را ۶۹٪ در نظر می‌گیریم.

* شرط لازم fix proity:

اگر تعداد process ها n باشد. اگر این عدد $n(2^{1/n}-1)$ بزرگتر از U باشد یعنی با استفاده از fixproity U قابل اجرا می‌باشد.

$$\boxed{n(2^{1/n}-1) \ge U}$$

می‌توان زمانبندی کرد

اگر شرط بالا صدق نکند ممکن است شاید باشد.

مثال: آیا الگوریتمی داریم که بتواند بدون مشکل اجرا شود این P ها؟ (example

c p d

$P_1(10,20,20)$

$P_2(~~10~~\ 6, 30, 30)$

$$\frac12+\frac13 = \frac56 = \%83$$

$\frac{10}{20}+\frac{20}{60}$

$\frac12+\frac13$

مهمترین زمان سخت fix-priority ۱۰۰٪ میشود

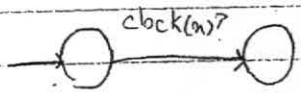
$\frac{6}{30}$ $\frac15+\frac12$

$U \simeq \frac{c}{p}$

۲)

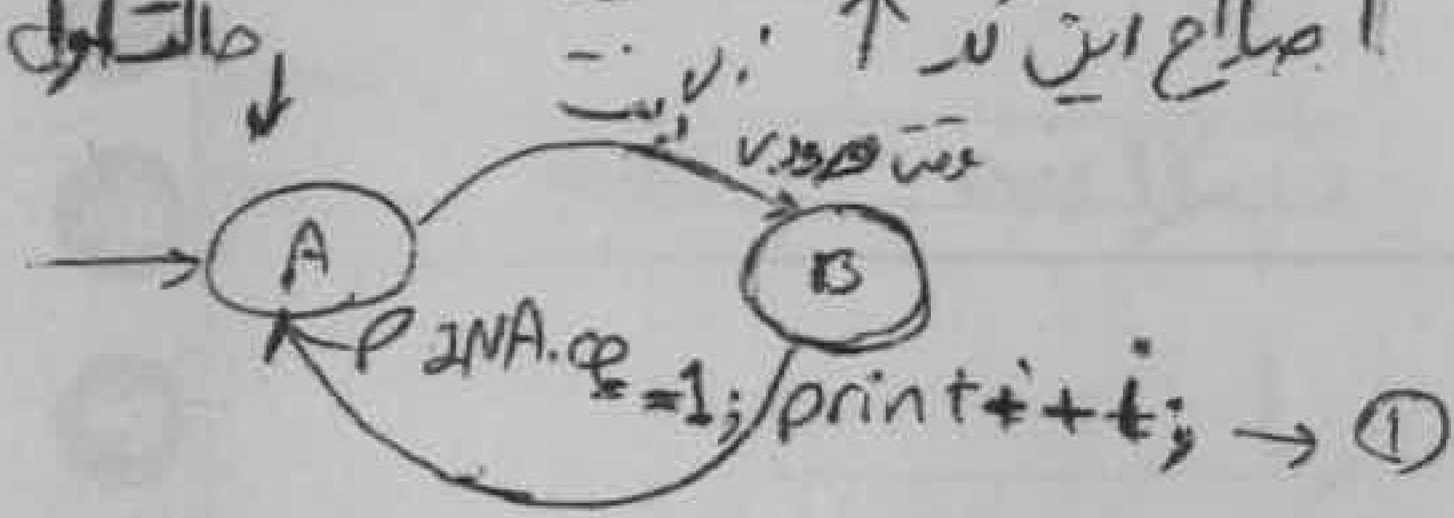
۱-۲۰-۹ ۱۰

CRSM را برای سیستم احتمالی با استفاده از زبان فرآیندها پراسس را بصورت فرآیند اجرا کنید ؟



۵
۱۹

StateBase


اصلاح این کد ↑ !!
حالت اول
زدن
A
B
PINA.0 ==1; print i++;
→ ①

② ← PINA.0 == 0

این حالت حالت ثابت است

```
bool j = false;
i = 0 ;
while(1) {
 if (PINA.0 == 0){
    j = false;
 if (PINA.0 == 1 && j == false){
    j = True;
 } } print i++;
```