به نام خدا

امیرفاضل کوزه‌گر کالجی ۹۹۳۱۰۹۹ تمرین ۲ سیستم عامل

(۱) با استفاده از API ، بسیاری از جزئیات واسط سیستم عامل از برنامه نویس پنهان می‌ماند و درگیر طراحی و پیچیدگی های آن نمی‌شود.

وقتی از API ها استفاده می‌کنیم، سیستم کال فراخوانی شد، سیستم کال مربوطه در هسته سیستم عامل را فراخوانی می‌کند و در نهایت وضعیت سیستم کال را برمی‌گرداند.

همچنین برنامه نویس دسترسی مستقیم به system call های درون kernel ندارد. و برای استفاده از API نیازی به دانستن نحوه implementation سیستم کال ها ندارد لذا می‌تواند برنامه‌اش را بدون دسترسی مستقیم داشتن به kernel جلو ببرد.

(۲) میدانیم که fork() روند اجرای برنامه را به دو قسمت تقسیم می‌کند به علت ایجاد Process جدید، تمام کدهای بعد از fork() دوبار اجرا می‌شوند (از آنجایی که در این برنامه وظایف فرزند و والد را جدا نکرده‌ایم) و نکته قابل توجه دیگر، مقداری است که fork() برمی‌گرداند که Process id فرزند است، اگر پروسه فرزند باشد مقدار صفر و اگر main Process باشد، مقدار مثبتی (اغلب id فرزند) خواهد داشت، حال به بررسی کد می‌پردازیم:

```
if(fork() && fork())
{
    fork();
}
```

a ← fork() اول
b ← fork() دوم
c ← fork() سوم

قطعه اول کد:

توجه به توضیحات مقدار fork() ها در مورد شرط می‌تواند مثبت یا صفر باشد، همچنین برای اینکه شرط and شود باید هر دو در true پروسه والدشان باشند، پس می‌توان گفت fork() درون if فقط یکبار اجرا می‌شود. اجرای یکباره آن به معنای ۲ بار اجرای کد زیرین آن می‌باشد. حال به شرط باطل بپردازیم. and به گونه‌ایست که اگر a صفر باشد توجهی به b نمی‌کند. پس فرض کنیم a صفر است، شرط باطل، و کد زیرین آن اجرا شود پس b چه صفر چه مثبت باشد نادیده گرفته می‌شود. فقط وقتی به b توجه می‌کنیم که a مثبت باشد و مقدار b تعیین کننده می‌شود.

برای اجرای کد زیرین این قطعه کد حالت زیر را داریم :

a	b	c
0	+/0	—
+	0	—
+	+	0
+	+	+

پس می توان گفت به طور کلی و با حساب مطالب قبلی،
قطعه کد دارای and (&&) ، ۴ بار که ~~خروجی~~ زیرینش را اجرا می کند

قطعه کد دوم :

```
if(fork() || fork())
{
    fork()
}
```

شرط اجرای کد داخل if آن است که حداقل یکی از میان a یا b ، مثبت باشد پس میتوان حالت هردو مثبت را نادیده گرفت . حالت اول : a مثبت بدون توجه به b ، حالت دوم : b مثبت بدون توجه به a ، پس در دو حالت c اجرایی شدد که می شود ۴ بار اجرای کد زیرینش .
حال برای عدم اجرای کد داخل if تنها وقتی که a و b صفر باشند امکان پذیر است که باعث می شود کد زیرینشان فقط ۱ بار اجرا شود .
(علت آنکه ۲ بار اجرا نمی شود آن است که این مقدار a یا b تغیر کند ، به درون شرط می رود) .
پس در کل ۵ حالت داریم که به شکل جدول زیر است :

a	b	c
+	+/0	+
+	+/0	0
+/0	+	+
+/0	+	0
0	0	—

قطعه اول ، کد زیرینش را ۴ بار اجرا می کند ، قطعه دوم که زیرینش را ۵ بار اجرا می کند .
با توجه به اینکه چاپ hello زیر هر دو قطعه است ، در کل ،

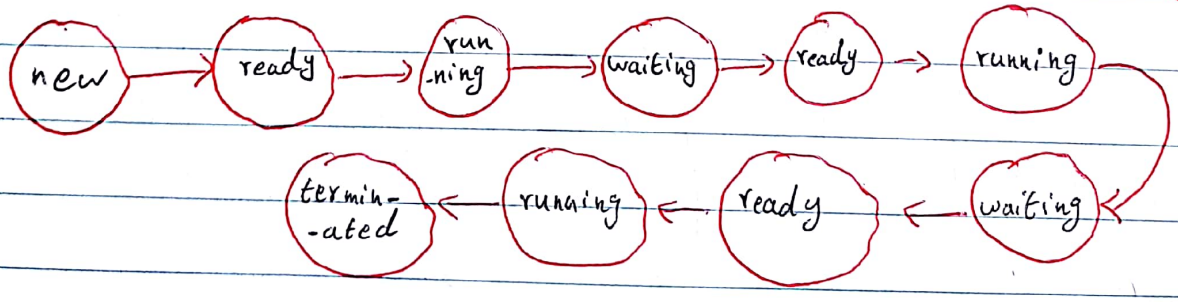
$$۴ \times ۵ = ۲۰$$

۲۰ بار در صفحه کنسول ، hello چاپ می شود .

۳)

- تغییر می‌کند، فرآیند فرزند تمام شده، سپس به ready queue می‌رود.
- به ready queue رفته و پس از اجرا شدن، والدش ترسط wait() باقی می‌شود و آن هم به ready queue می‌رود.
- در بدو ورود به سیستم درون Job queue می‌رود. هرگاه وارد استیت ready شد، به ready queue می‌رود.
- به I/O queue رفته و منتظر می‌ماند تا کار با دستگاه I/O تمام شود سپس به ready queue برمی‌گردد.
- اگر یک فرایند منتظر رخ دادن وقفه باشد، پس از رخ دادن وقفه، به ready queue بازگشته و ادامه می‌یابد. رخ دادن خود وقفه در حین اجرای فرایندی که منتظر آن نبوده یا از حالت run به ready می‌رود که باعث می‌شود به ready queue انتقال یابد.
- پس از پایان timeslice، به ready queue برگشته و منتظر می‌ماند تا دوباره اجرا شود.

۴)



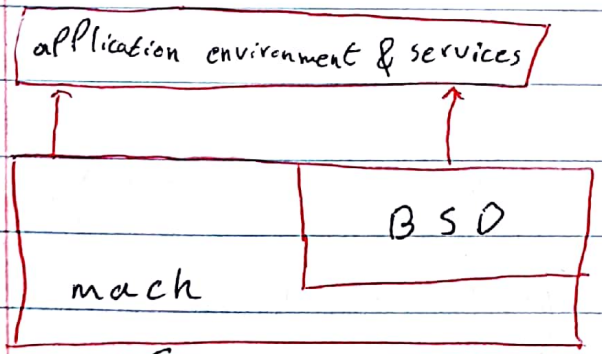
۵) الف) به علت ارتباط میان فضای کاربر (user space) با فضای هسته (kernel space)، سربار عملکرد ایجاد می‌شود و باعث کاهش عملکرد در برخی سیستم عامل ها مثل windows NT 4 می شود.

ب) Mach همانند دیگر رویکردها از message passing برای انتقال اطلاعات استفاده می‌کند، که عملکرد کمتری نسبت به shared memory دارد. برای حل این مشکل، Mach از فضای حافظه مجازی (virtual memory) استفاده می‌کند تا محتوای پیام های بزرگ را انتقال دهد. کپی کردن مجازی باعث می شود تا جلوگیری شود از کپی کردن واقعی محتوا.

ج) ساختار ماژولار، از آنجایی که از ماژول و کامپوننت های مختلف تشکیل شده است، به micro-kernel شباهت دارد و از آنجایی که قابلیت گفت‌وگو بین کامپوننت ها برقرار است به ساختار layered شباهت دارد.

- نیازی به استفاده از message passing ندارد پس از micro-kernel بهتر است
- هر کامپوننت می تواند مستقیما با دیگری حرف بزند، پس از layered بهتر است.

۶) Mac OS یا دقیقتر MacOS X، ساختار کلی اش hybrid است که ترکیبی از ساختارهای متفاوت است اما درباره هسته آن که darwin نیز نام دارد بر اساس mach micro kernel ایجاد شده است به شکل زیر:



و از آنجایی که راجع mach نیز پیشتر توضیح دادیم، به علت message-passing، برای انتقال اطلاعات باید محتوای پیام ها بین دو سرویس کپی شود، پس این مشکل سربار به وجود می آورد برای حل این شکل، mach از حافظه مجازی استفاده می‌کند و محتوای پیام ها در آن، به صورت مجازی کپی می شوند که باعث می شود عملیات کپی شدن واقعی انجام نشود و یا به کمترین بیفتد. از آنجایی که هر دو سرویس ها به این حافظه مجازی دسترسی دارند می توان گفت به نوعی shared memory پیاده سازی شده است.