کلاس حل تمرین اول سیستم عامل
پاییز ۱۴۰۱

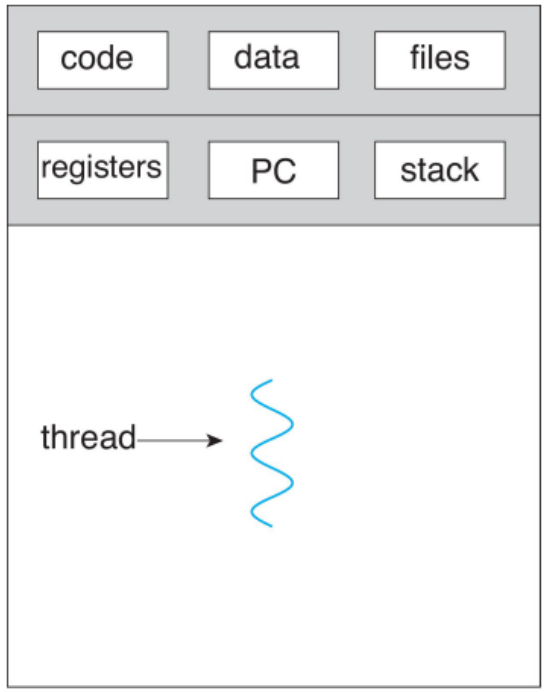
سروناز سروقد

سید محمدعلی میرکاظمی

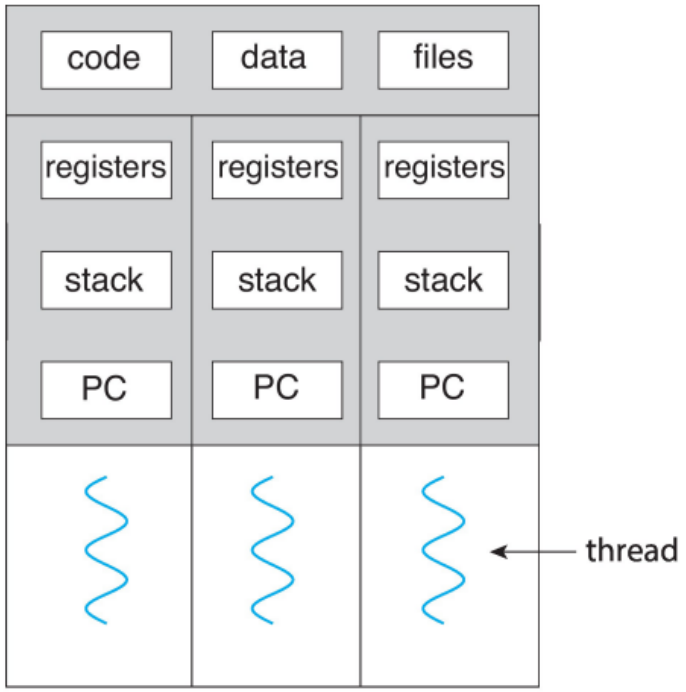
امیرحسین پولاد

پردازه و ترد

- پردازه به زبان ساده همان برنامه‌ایست که در حال اجراست.
- ترد جزئی از پردازه است و حاوی اطلاعاتی مانند «کجای برنامه اجرا می‌شود؟» و «مقادیر رجیستر ها چیست؟» و «استک برنامه کجای حافظه قرار دارد؟» است. یک پردازه می‌تواند یک یا چند ترد داشته باشد.



single-threaded process



multithreaded process

ایجاد پردازه جدید

- در سیستم عامل های مبتنی بر Unix ایجاد پردازه های جدید از طریق سیستم‌کال fork صورت می‌گیرد. (همه‌ی پردازه ها به همین صورت ایجاد شده‌اند!)
- این سیستم‌کال یک کپی کامل از پردازه‌ای که آن را صدا زده ایجاد می‌کند. خروجی این تابع، برای پردازه‌ای که تازه ایجاد شده صفر، و برای پردازه‌ای که فورک را صدا زده غیر صفر است.
- بعد از انجام این دستور، می‌توانیم با استفاده از دستور exec یک برنامه‌ی جدید به جای پردازه‌ی فعلی بیاوریم.

مثال: استفاده از fork و exec در یک برنامه‌ی واقعی

- فایل EduShell.c

مقایسه fork و pthread_create

- نکته‌ی مهم در مورد فورک: پس از فورک، یک کپی کامل از پردازه‌ی پدر ایجاد می‌شود. توجه کنید که این یک پردازه‌ی جدید و کاملا مستقل از پردازه‌ی پدر است و به جز File Descriptor ها **هیچ چیز** پردازه‌ی پدر و فرزند مشترک نیست.
- یکی از روش های ایجاد ترد جدید در سیستم عامل هایی که استاندارد POSIX را رعایت می‌کنند استفاده از تابع pthread_create است.

```
int pthread_create(pthread_t *restrict thread,
                   const pthread_attr_t *restrict attr,
                   void *(*start_routine)(void *),
                   void *restrict arg);
```

مقایسه fork و pthread_create

	fork	pthread_create
محل اجرا شدن/ادامه‌ی اجرا	هر دو از کد بعد از صدا زده شدن فورک	ترد جدید: تابع ورودی ترد قبلی: کد بعد از صدا زده شدن
استک	مستقل	مستقل
مقادیر متغیر های داخل استک	مستقل	مستقل
مقادیر رجیستر ها	مستقل	مستقل
متغیر های گلوبال	مستقل	مشترک
فضای آدرس	مستقل	مشترک
PID	مستقل	مشترک
Parent's PID	مستقل	مشترک
File Descriptor	مشترک	مشترک

مثال: خروجی برنامه با استفاده از pthread

- فایل های thread_test0.c و thread_test1.c و thread_test2.c

مثال: fork

با فرض اینکه تمام فراخوان های سیستمی کد زیر با موفقیت اجرا می‌شوند، خروجی را معین کنید:

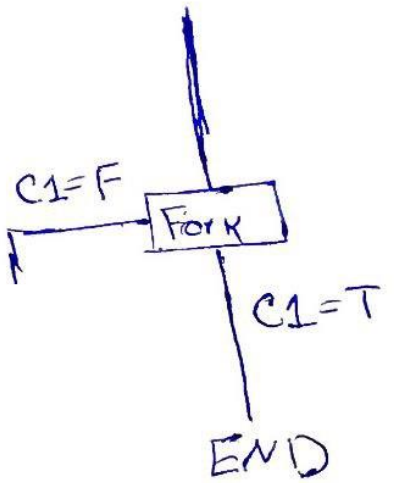
```
int main()
{
    fork() || fork() && fork() || fork() && fork() || fork();
    printf("+");
    return 0;
}
```

مثال: fork

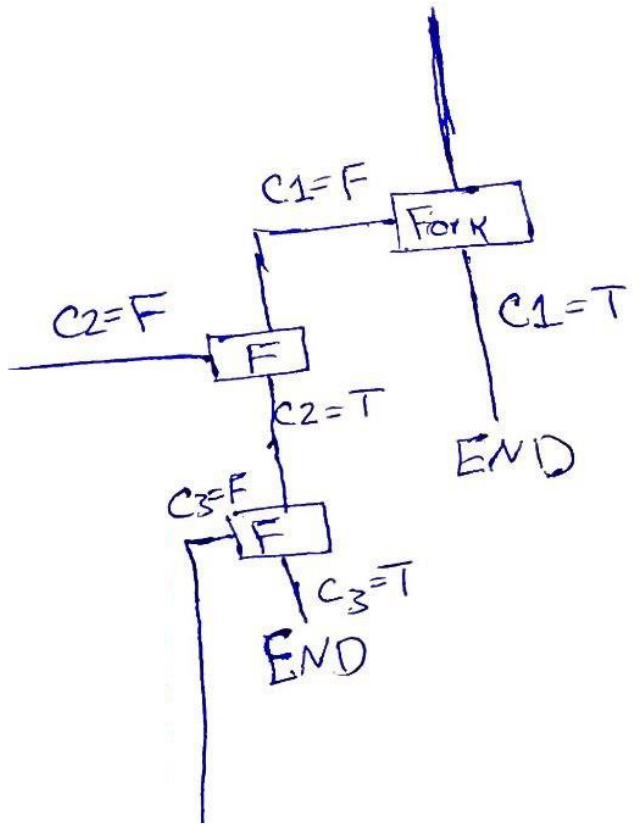
نکات اصلی سوال:

- خروجی فورک در پردازه‌ی والد بزرگتر از صفر و در پردازه‌ی فرزند صفر است.
- در زبان C اولویت && از || بیشتر است.
- Short-Circuiting داریم. یعنی در شرط a && b اگر a == 0 باشد دیگر b چک نمی‌شود و در شرط a || b اگر a == 1 باشد دیگر b چک نمی‌شود.

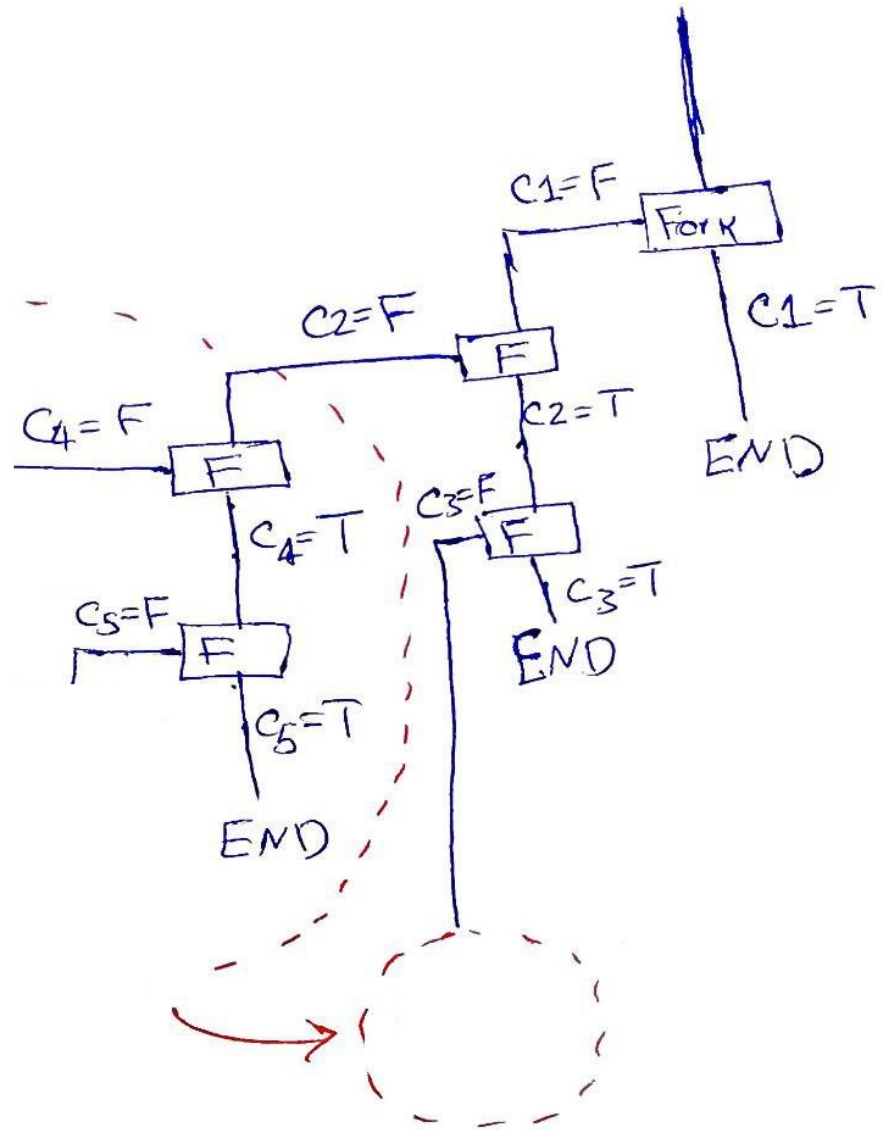
```
int main()

{
    C1          C2          C3          C4          C5          C6
    fork() || fork() && fork() || fork() && fork() || fork();

    printf("+");

    return 0;

}
```



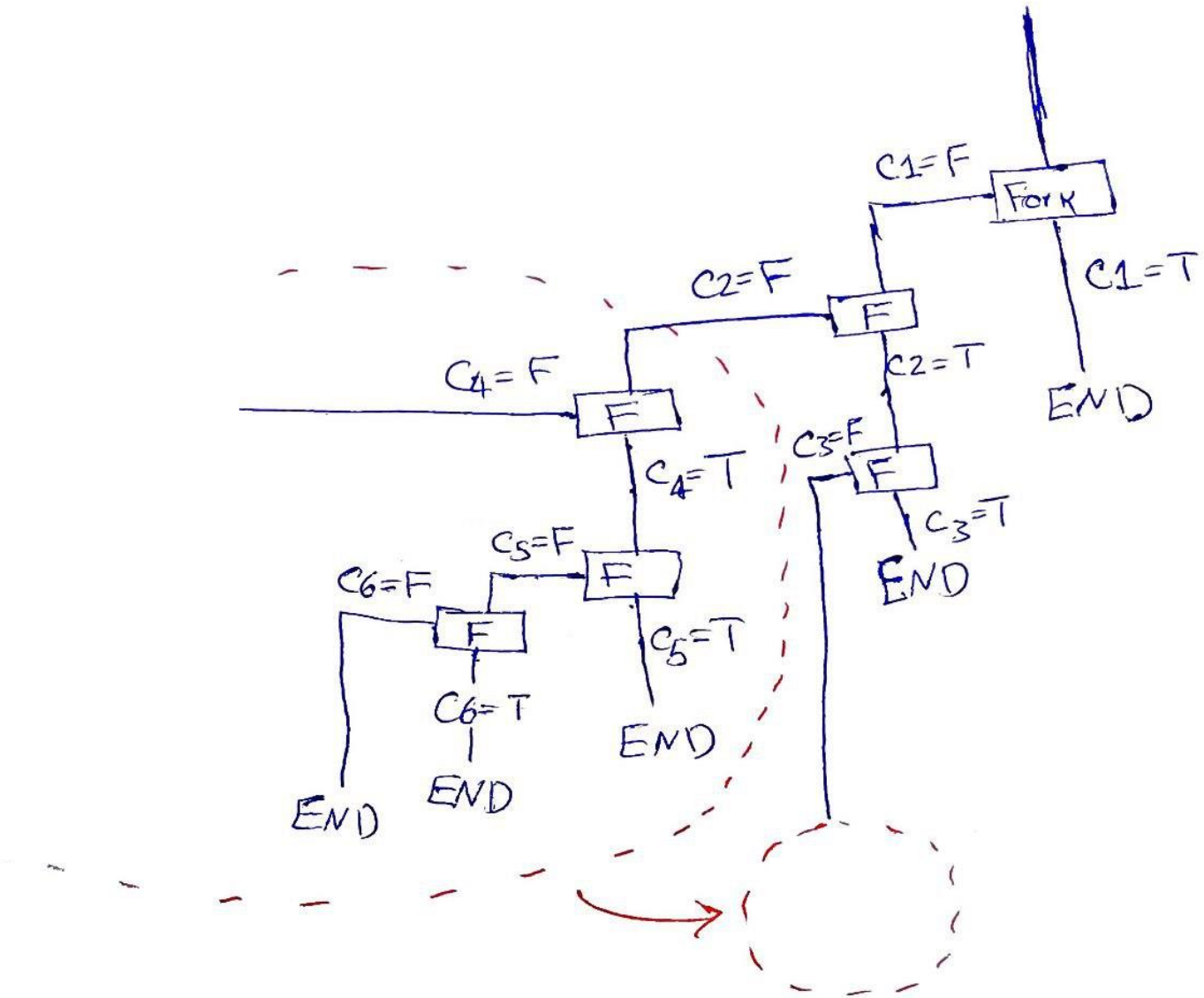
```
int main()
{
    fork() || fork() && fork() || fork() && fork() || fork();
    printf("+");
    return 0;
}
```



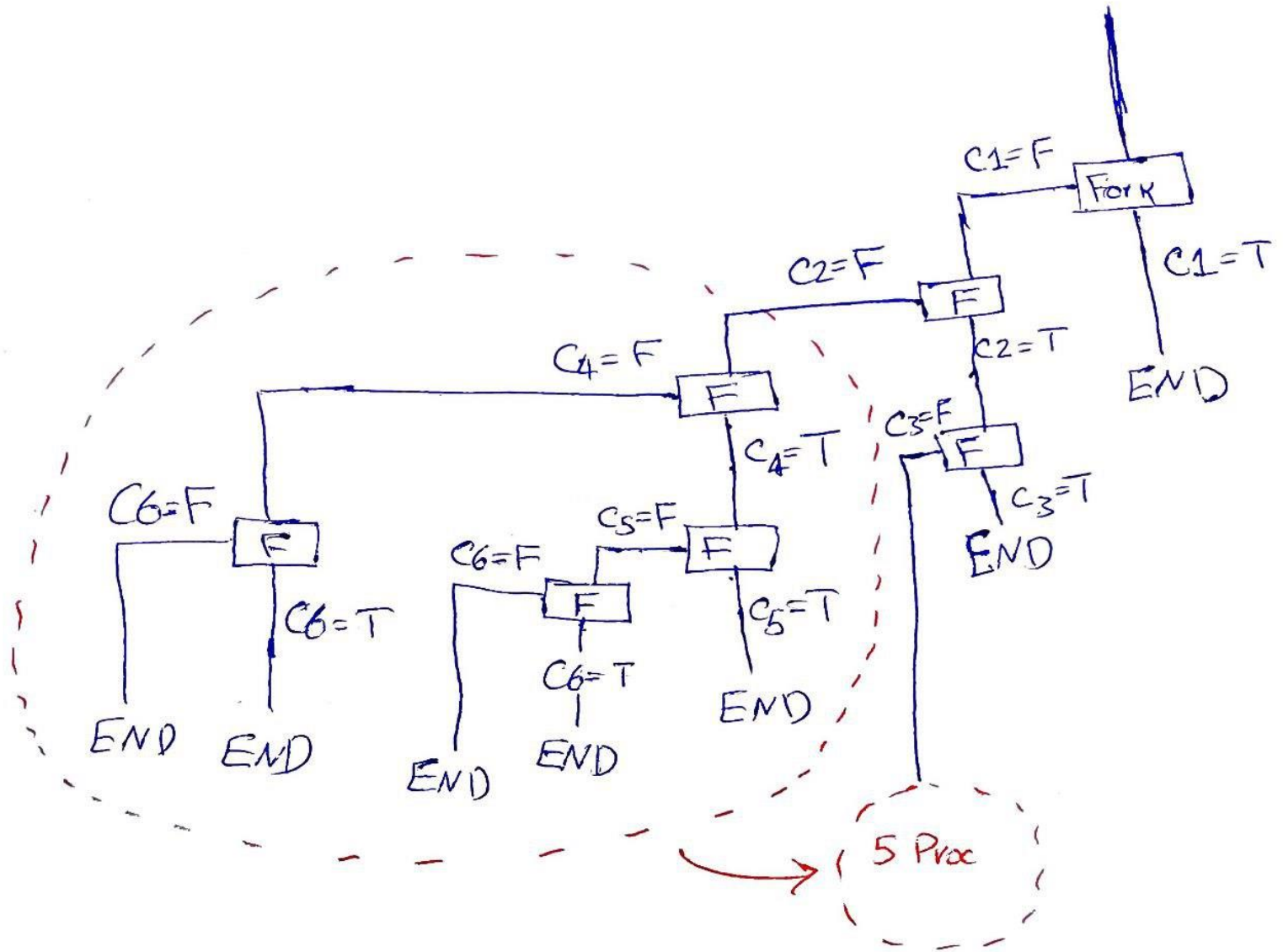
```c
int main()
{
    fork() || fork() && fork() || fork() && fork() || fork();
    printf("+");
    return 0;
}
```



```c
int main()
{
    fork() || fork() && fork() || fork() && fork() || fork();
    printf("+");
    return 0;
}
```



```c
int main()
{
    fork() || fork() && fork() || fork() && fork() || fork();
    printf("+");
    return 0;
}
```



مثال: fork

با فرض اینکه تمام فراخوان های سیستمی کد زیر با موفقیت اجرا می‌شوند، تعداد کل پردازه های ایجاد شده پس از اجرای قطعه کد زیر را به دست آورید.

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    int i;

    for (i = 0; i < 4; i++)
      fork();

    return 0;
}
```

مثال: fork

در هر بار اجرای حلقه، تعداد پردازه ها ۲ برابر می‌شود.

برای مثال در بار اول اجرای حلقه یک پردازه وارد حلقه می‌شود و بعد از ایجاد یک پردازه‌ی جدید ۲ پردازه از حلقه خارج می‌شوند.

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    int i;

    for (i = 0; i < 4; i++)
      fork();

    return 0;
}
```

دفعه‌ی بعد ۲ پردازه وارد حلقه می‌شوند، هر کدام یک پردازه‌ی جدید می‌سازند و ۴ پردازه از حلقه خارج می‌شوند.