سوال ۱

برنامه‌ی زیر به زبان پایتون را در نظر بگیرید که یک عدد از کاربر گرفته و مشخص می‌کند که آیا آن عدد زوج است یا فرد.

```
def main():
    number = int(input())
    result = 'Even' if number % 2 == 0 else 'Odd'
    print(result)
main()
```

فرض کنید که تنها دستورات input و print بلافاصله انجام نمی‌شوند و نیازمند صبر کردن برای I/O هستند. حالات (process states) مختلف پردازه این برنامه را در طول چرخه حیات آن بررسی کنید.

از آنجایی که پایتون یک زبان مفسری است تمامی کد های این زبان توسط یک instance از مفسر آن اجرا می‌شوند.

با دادن دستور اجرای این کد توسط سیستم عامل یک پردازه جدید برای اجرای مفسر ایجاد شده و در حالت new قرار میگیرد.

پس از مقداردهی های اولیه و لود شدن مفسر در حافظه و نسبت داده شدن کد به آن پردازه در حالت ready قرار میگیرد.

پس از تخصیص cpu به این پردازه، پردازه در حالت running قرار میگیرد

به محض رسید به input() پردازه درخواست I/O را ارسال کرده و در حالت waiting قرار میگیرد

پس از رسیدن I/O (وارد کردن یک عدد توسط کاربر) و انتقال آن به پردازه توسط سیستم عامل پردازه در حالت ready قرار میگیرد.

پس از تخصیص cpu به این پردازه، پردازه در حالت running قرار میگیرد

به محض رسیدن به دستور print پردازه درخواست I/O را ارسال کرده و در حالت waiting قرار میگیرد

بعد از انجام درخواست I/O توسط سیستم عامل (چاپ کردن result در std out) پردازه در حالت ready قرار میگیرد.

پس از تخصیص cpu به این پردازه، پردازه در حالت running قرار میگیرد و از آنجایی که کد دیگری برای اجرا ندارد پایان یافته و در حالت terminated قرار میگیرد.

سوال ۲

فرض کنید می‌خواهید یک سیستم‌عامل طراحی کنید که Process Control Block در آن تنها ۵ فیلد مختلف می‌تواند داشته باشد، این فیلدها را چه چیزی‌هایی انتخاب می‌کنید؟ فرضیات خود درباره سیستم عامل‌تان را بیان و دلایل خود را ذکر کنید. در چه مواردی ممکن است این سیستم عامل در context switch با مشکل مواجه شود؟
تذکر: هر فیلد PCB باید شامل یک عدد یا رشته یا آرایه‌ای از اعداد و رشته ها باشد، همچنین این اعداد/رشته ها نباید داده های multiplex شده باشند و باید داده خام و مستقیما قابل استفاده باشند، برای مثال نمیتوانید

program_counter * 10000 + pid

را به عنوان یک فیلد اختصاص دهید زیرا ۲ داده multiplex شدند تا یک داده را تشکیل دهند.

فیلد هایی که در یک PCB وجود دارند عبارتند از:

Process State, Process ID, Program Counter, CPU Registers, CPU-Scheduling Information, Memory-management Information, Accounting Information, I/O status information

از بین این فیلد ها برای اینکه بتوانیم context switch موفقیت آمیز بین دو پردازه داشته باشیم لازم است که فیلد های program counter, cpu registers , memory-management information و I/O status information را داشته باشیم (فیلد پنجم هر یک دیگر فیلد ها میتواند باشد و هر کدام بخش خاصی از فرایند را بهبود میدهند).

Program counter:

برای ادامه دادن یک برنامه باید بدانیم که دستور بعدی که باید اجرا کنیم چیست.

CPU registers:

مهم است که از دید برنامه در حال اجرا pre-empt شدن هیچ تاثیری نگذارد در نتیجه state cpu باید ذخیره و برای ادامه برنامه بازیابی شود

Memory-management information:

برنامه باید به متغیر های خود در ram دسترسی داشته باشد و محل آن ها را بداند.

I/O status information:

مانند متغیر های داخل رم برنامه باید به I/Oهایی که در دسترسش قرار دارند (مانند فایل های باز شده ویا سوکت های شبکه) دسترسی داشته باشد.

در راه‌حل بالا context switch به عنوان عملکرد اصلی‌ای که باید PCB به سیستم عامل ارائه دهد فرض شده، در صورتی که عملکرد اصلی‌ای که فرض کردید متفاوت باشد جواب نیز میتواند فرق کند.

سوال ۳ (بخش ۳.۴ کتاب)

۱) پردازه‌های independent و cooperating به چه پردازه‌های گفته می‌شود و چه ویژگی‌های دارند؟ توضیح دهید.

یک پردازه independent است اگر هیچ داده و اطلاعاتی با دیگر پردازه‌هایی که در سیستم در حال اجرا شدن هستند به اشتراک نگذارد.

یک پردازه cooperating است در صورتی که بتواند روی دیگر پردازه‌های در حال اجرا تاثیر بگذارد یا از آن‌ها تاثیر ببیند. به وضوح هر پردازه‌ای که با دیگر پردازه‌های موجود در سیستم اطلاعات به اشتراک بگذارد از نوع cooperating است.

۲) چرا به ارتباط میان پردازه‌ها نیاز داریم؟ ۳ دلیل برای آن بیاورید و آن‌ها را توضیح دهید.

۱) به اشتراک گذاری اطلاعات(Information sharing): از آنجایی که برنامه‌های متعددی ممکن است به یک اطلاعات خاص نیاز داشته‌باشند باید روشی برای به اشتراک گذاری این اطلاعات بین آن‌ها ایجاد کنیم.

۲) سرعت محاسبه(Computation speedup): یکی از روش‌های معمول افزایش سرعت انجام تسک‌ها شکاندن آن‌ها به تسک‌های کوچک‌تر است به شکلی که هر زیر مجموعه از تسک اصلی به صورت موازی با دیگر بخش‌ها اجرا شود. این اجرای موازی مستلزم به اشتراک گذاری اطلاعات میان تسک‌های کوچک است.

۳) ماژولار بودن(Modularity): اگر بخواهیم سیستم را به شکل ماژولار بسازیم(یعنی آن را به بخش‌های کوچک‌تر تقسیم کنیم) نیازمند کمک گرفتن از پردازه‌های cooperating هستیم.

۳) چگونه می‌توان با ایجاد ارتباط همکاری میان پردازه‌ها باعث سریع‌تر انجام شدن یک برنامه شد؟

اگر یک تسک را به تسک‌های کوچک‌تر(subtask) بشکنیم به شکلی که این بخش‌های کوچک به صورت موازی با هم اجرا شوند افزایش زیادی در سرعت اجرای برنامه ایجاد می‌شود. در این صورت هر subtask بخشی از کار را به شکل موازی با

subtaskهای دیگر انجام میدهد که به وضوح باعث سریع‌تر شدن برنامه می‌شود.
این مورد هم همانطور که گفته‌شد نیاز به پردازه‌های cooperating و ایجاد ارتباط میان پردازه‌ها دارد.

سوال ۴

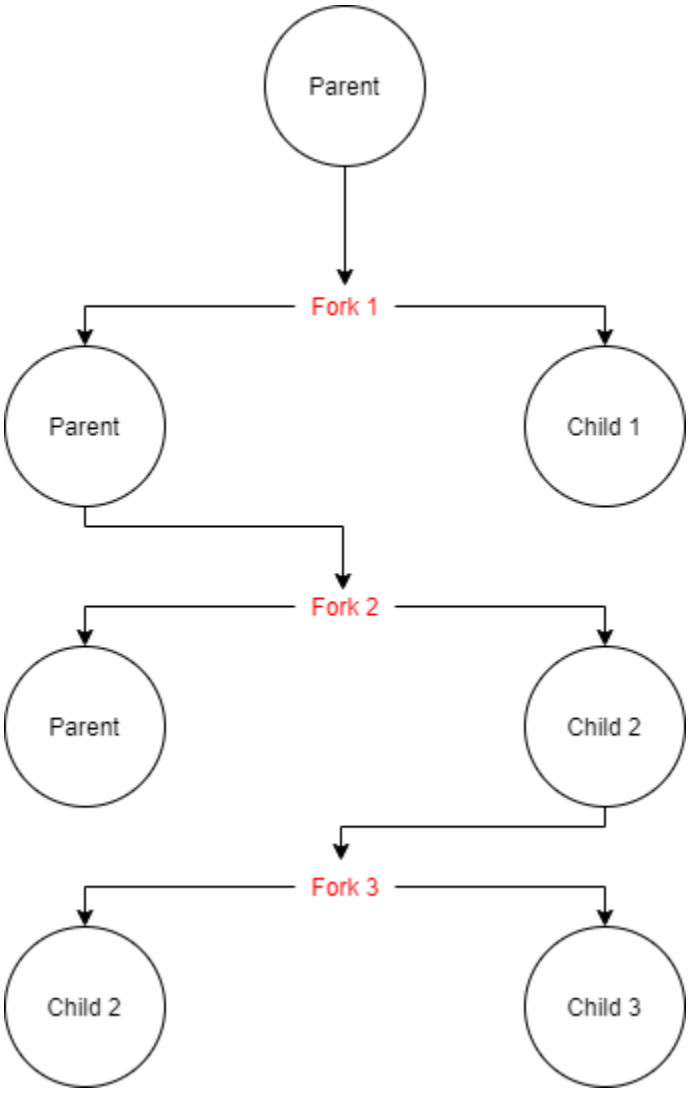
خروجی کدهای زیر را با رسم شکل درختواره پردازه‌ها توضیح دهید (سعی کنید ابتدا بر روی کاغذ تحلیل خود را انجام دهید (آنچه که در پاسخ تمرین ارسال خواهید کرد) و سپس برنامه را کامپایل و اجرا کنید):

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

int main()
{
    if (fork()) {
        if (!fork()) {
            fork();
            printf("1 ");
        }
        else {
            printf("2 ");
        }
    }
    else {
        printf("3 ");
    }
    printf("4 ");
    return 0;
}
```

در ابتدا برای پردازه P یک فرزند C1 ساخته می‌شود. پردازه پدر وارد قسمت if و فرزند وارد else می‌شود. پردازه پدر فرزند دیگری به اسم C2 ساخته و وارد else می‌شود. پردازه C2 فرزندی به اسم

C3 ساخته و با پرینت ۱ و ۴ کار خود را تمام می‌کند. پردازه C3 نیز ۱ و ۴ را پرینت می‌کند. پردازه پدر اعداد ۲ و ۴ را پرینت کرده و پردازه C1 با پرینت کردن ۳ و ۴ به اتمام می‌رسد. دقت کنید هیچ تضمینی در ترتیب پرینت شدن این اعداد وجود ندارد.



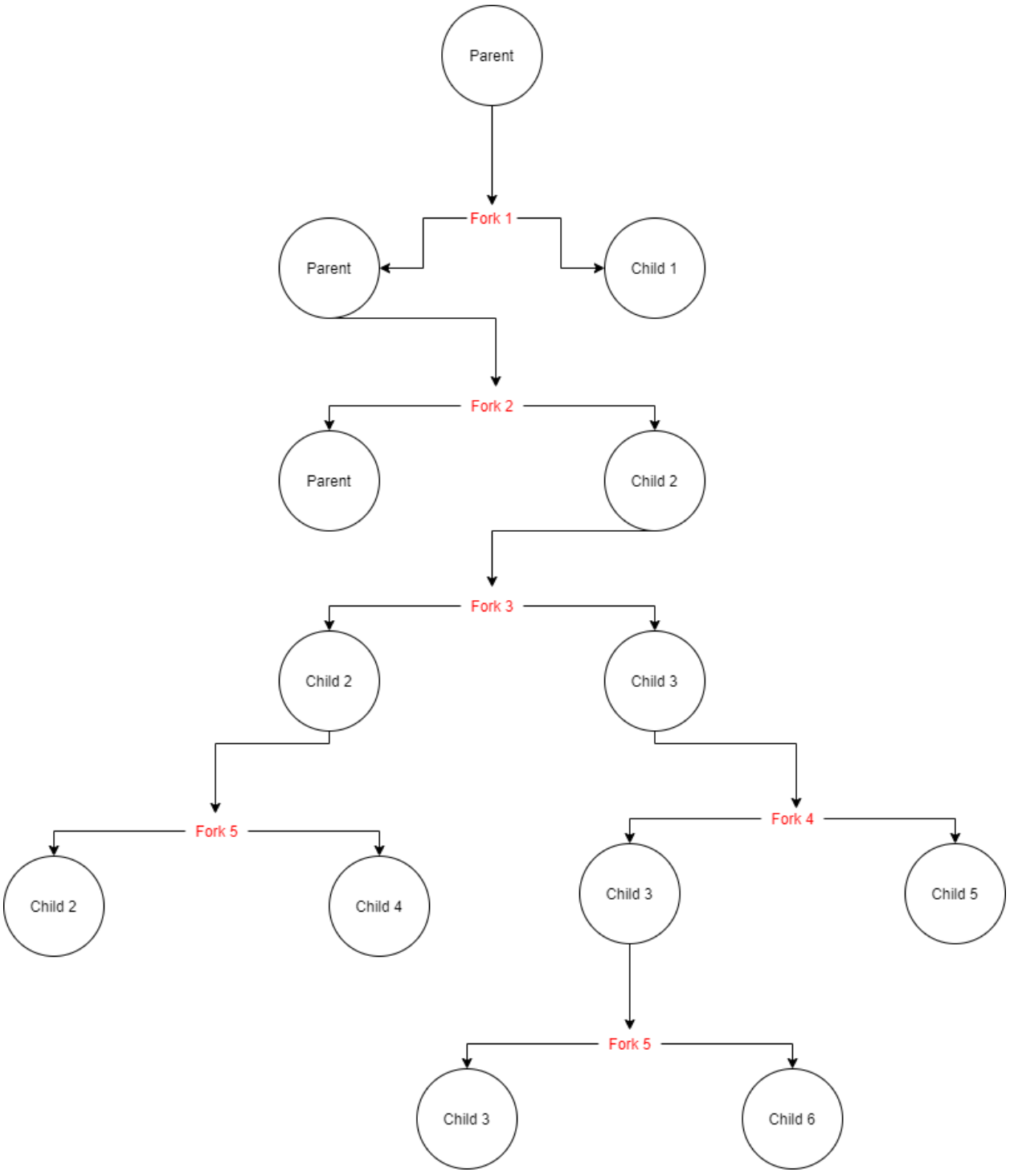
```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    if (fork() && (!fork())) {
        if (fork() || fork()) {
            fork();
        }
    }
    printf("2 ");
    return 0;
}
```

ابتدا پردازه پدر یک فرزند به اسم C1 می‌سازد. به دلیل وجود اپراتور && ادامه شرط دیگر برای فرزند اجرا نمی‌شود و تنها برای پردازه پدر بار دیگر اجرا شده و یک فرند دیگر به نام C2 ساخته می‌شود. در نتیجه پردازه پردازه C2 وارد شرط می‌شود و پردازه C1 و پردازه پدر از if خارج شده و دو بار عدد ۲ را چاپ می‌کنند. پردازه C2 وارد شرط شده و در اجرای اولین fork یک فرزند C3 را ایجاد می‌کند. سپس به دلیل true بودن و وجود اپراتور || وارد شرط می‌شود و در نتیجه با اجرای fork داخل شرط یک فرزند دیگر به اسم C4 ساخته و به کار پایان می‌دهد.

پردازه C3 وارد fork دوم شرط شده و فرزند C5 را ایجاد می‌کند و خود این بار وارد شرط می‌شود. در حالی که C5 خارج می‌شود. در ادامه تمامی آن‌ها عدد ۲ را پرینت کرده و اجرا تمام می‌شود.

نمی‌توان ترتیب اجرا شدن را تضمین کرد ولی به صورت کلی ۷ بار عدد ۲ پرینت می‌شود.


Parent
Fork 1
Parent
Child 1
Fork 2
Parent
Child 2
Fork 3
Child 2
Child 3
Fork 5
Fork 4
Child 2
Child 4
Child 3
Child 5
Fork 5
Child 3
Child 6

سوال ۵ (بخش ۳.۵ کتاب)

در مورد shared memory به سوالات زیر پاسخ دهید

1) این فضای ذخیره‌سازی مشترک در کجا قرار داد؟
این فضا در فضای آدرسی(address space) پردازه‌ای که آن را میسازد قرار دارد.

2) پردازه‌های دیگری که تمایل به استفاده از این فضا دارند چگونه میتوانند به آن متصل شوند؟
پردازه‌های دیگری که میخواهند به از این فضای اشتراکی استفاده کنند باید آن را به فضای آدرسی خود متصل کنند. البته به طور معمول سیستم‌عامل اجازه‌ی دسترسی به فضای آدرسی یک پردازه را به پردازه‌های دیگر نمی‌دهد و به همین خاطر این پروتکل نیاز دارد تا پردازه‌های درگیر قبول کنند که این محدودیت را بردارند.

3) پس از اتصال چند پردازه به این فضای مشترک چگونه میتوانند داده‌ها را با هم به اشتراک بگذارند و ارتباط برقرار کنند؟
این پردازه‌ها پس از اتصال به فضای اشتراکی میتوانند با نوشتن و خواندن داده‌های موجود در محیط اشتراکی به رد و بدل اطلاعات و داده‌ها بپردازند. از آنجایی که محدودیت مربوط به سیستم عامل هنگام اتصال پردازه‌ها به محیط اشتراکی برداشته شده‌است، فرم داده و مکان آن و به طور کلی مسئولیت آن با خود پردازه‌هاست و سیستم عامل دیگر نقشی ندارد.

4) در پروسه‌ی ارتباط پردازه‌ها با روش shared memory سیستم‌عامل چه نقشی دارد و تا چه حد دسترسی دارد؟
همانطور که گفته‌شد سیستم‌عامل به طور معمول اجازه‌ی دسترسی به فضای آدرسی دیگر پردازه‌ها را به یک پردازه نمیدهد به همین خاطر قبل از اتصال پردازه به فضای اشتراکی باید با کمک فراخوانی‌های سیستمی(system call) به سیستم عامل اطلاع دهند که قصد ایجاد چنین فضایی را دارند تا محدودیت‌های مربوط به آن برداشته‌شود. پس از برداشته‌شدن محدودیت‌ها سیستم عامل دیگر در فرایند انتقال داده نقشی ندارد و مسئولیت آن به عهده پردازه‌هاست.

5) در حین ارتباط با روش shared memory چه مشکلاتی ممکن است به وجود بیاید و چه بخشی مسئول حل این مشکلات است؟
ممکن است چند پردازه بخواهند به شکل همزمان روی یک مکان اطلاعات بنویسند

و مشکلاتی مانند race condition به وجود بیاید و یا پردازه‌ای بخواهد داده‌ی خاصی را بخواند ولی پیش از آن پردازه‌ی دیگری آن را تغییر دهد. از آنجایی که محدودیت‌های اعمالی از سیستم‌عامل پیش از شروع اشتراک گذاری برداشته شده‌اند دیگر سیستم عامل نقشی ندارد و مسئول جلوگیری از مشکلات و حل آن‌ها خود پردازه‌ها هستند.

6) مشکل producer-consumer در حین استفاده از shared memory چیست و چگونه میتوان آن را حل کرد؟

این روش یک الگوی مشترک میان پردازه‌های cooperating است. به این شکل که پردازه‌ی producer اطلاعاتی را تولید میکند که پردازه‌ی consumer از آن‌ها استفاده میکند.(مانند اتفاقی که در پروتکل client-server رخ میدهد.) حال مشکلی که ممکن است رخ دهد این است که چگونه این فرایند ایجاد شود به طوری که مثلا زمانی که اطلاعاتی وجود ندارد پردازه‌ی consumer شروع به خواندن نکند یا زمانی که جایی برای اطلاعات وجود ندارد پردازه‌ی producer شروع به نوشتن آن‌ها نکند. برای حل این مشکل می‌توان از یک بافر استفاده کرد به شکلی که producer در آن بنویسد و آن را پر کند و consumer از آن بخواند و آن را خالی کند. در این حالت باید producer و consumer به شکل سنکرون عمل کنند تا مشکلات ناشی از ناهماهنگی به وجود نیاید مثلا consumer قبل از نوشتن داده توسط producer شروع به خواندن آن نکند. حال ۲ نوع مختلف از بافر را میتوان استفاده کرد. نوع اول سایز محدودی ندارد. در این حالت دیگر تولید کننده نیازی به صبر کردن ندارد و هر زمان که خواست فارغ از میزان داده‌ی موجود در بافر میتواند در آن بنویسید زیرا محدودیت سایزی در بافر وجود ندارد اما از سمت دیگر مصرف کننده همچنان ممکن است در حالتی که بافر خالی است مجبور به صبر کردن شود. نوع دوم بافر سایز محدود دارد و هر دوی مصرف کننده و تولید کننده ممکن است مجبور به صبر کردن شوند زیرا در این حالت ممکن است بافر پر باشد و تولید کننده نتواند هر زمان که خواست در آن بنویسد. این بافر به صورت حلقوی پیاده‌سازی میشود و با داشتن دو متغیر in و out میتواند متوجه شد که بافر چقدر فضای خالی دارد و آیا امکان نوشتن روی آن وجود دارد یا نه.(اشکال ۳.۱۲ و ۳.۱۳)