جواب سؤال‌های امتحان میانی درس سیستم عامل

۱ سیستم عامل به هر پردازه فضای آدرس مجزایی تخصیص می‌دهد و یک پردازه نمی‌تواند به حافظه‌ی پردازه‌ی دیگر دسترسی داشته باشد (مگر اینکه قبلا حافظه‌ی مشترک درخواست شده باشد).

۲ الف) برنامه‌نویسان کاربردی لازم نیست برای سخت‌افزارهای متفاوت موجود راه‌انداز بنویسند. ب) خیر. سیستم عامل وظایف دیگری مثل امکان استفاده‌ی همزمان چند پردازه از سخت‌افزار، محافظت پردازه‌ها و امکان تعامل بین پردازه‌ها را نیز انجام می‌دهد.

۳ در سیستم‌عامل‌های چندپردازه‌ای، منابع سیستم از جمله پردازنده بین پردازه‌ها تقسیم می‌شود. وقتی اجرای یک پردازه با درخواست I/O یا با کمک وقفه‌ی سخت‌افزاری قطعه‌ی Timer متوقف می‌شود و سیستم عامل پردازه‌ی دیگری را برای اجرا انتخاب می‌کند. از سوی دیگر، کارت شبکه با دریافت بسته‌ها (معمولا) وقفه‌ی سخت‌افزاری تولید می‌کند که توسط آن، اجرای پردازه قطع می‌شود و سیستم عامل در روال پاسخ به وقفه، بسته‌ها را از کارت شبکه دریافت می‌کند. اجرای سایر پردازه‌ها و پاسخ به وقفه‌های سخت‌افزاری موجب می‌شوند پردازنده زمان کمتری به یک پردازه اختصاص یابد.

۴ در صورتی که منابع مورد نیاز پردازه‌ها بیشتر از منابع موجود باشد (یا به علت جدال بر سر منابع موجب کندی اجرای آنها شود) درجه‌ی چند برنامگی می‌تواند کاهش یابد. زمانبند میان مدت این شرط را بررسی می‌کند و در صورت نیاز برخی از پردازه‌ها را به حافظه‌ی Swap انتقال می‌دهد.

۵ در صورت بروز خطا، سیستم عامل محتویات حافظه‌ی پردازه و مقدار رجیسترهای پردازه را در یک فایل (که به آن Coredump گفته می‌شود) قرار می‌دهد. با بررسی مقدار متغیرهای برنامه در لحظه‌ی خطا در این فایل (با ابزاری مثل یک Debugger) می‌توان دلیل بروز خطا را یافت.

۶ الف)

```
if (fork()) {
      for (i = 0; i < 50; i++)
            compute(i);
} else {
      for (i = 50; i < 100; i++)
```

```
            compute(i);
}
```

ب) پردازه‌ی اصلی و پردازه‌ی جدید می‌توانند روی دو هسته‌ی مجزا به صورت موازی اجرا شوند. ج) لوله، حافظه‌ی مشترک، ارسال پیغام، فایل.

۷ الف) با توجه به اینکه امکان دارد سرور هیچ‌گاه به درخواست‌های اول صف پاسخ ندهد، امکان قحطی (Starvation) وجود دارد. ب) بله. چون یک عنصر از اول صف نیز برداشته می‌شود، همه‌ی درخواست‌ها پس از تعداد محدودی پاسخ به سایر درخواست‌ها، پاسخ داده می‌شوند.

۸ بله. فرض کنید دو ریسمان در حال اجرای این تابع باشند. اگر ریسمان اول بعد از بررسی شرط if (ولی قبل از تغییر متغیر isfree) متوقف شود و ریسمان دوم اجرای خود را تمام کند و سپس ریسمان اول خاتمه یابد، هر دو ریسمان مقدار متغیر isfree را به یک تغییر می‌دهند و هر دو ریسمان مقدار ۱ را از تابع allocate برگشت خواهند داد.

۹ در این سؤال چند جنبه اهمیت دارند: محافظت دسترسی‌های همزمان به متغیرهای مشترک، منطق درست برای افزایش و کاهش موجودی، نبودن امکان بن‌بست و امکان تغییر موجودی توسط بیش از دو ریسمان. در صورتی که فرض شود فقط یک ریسمان موجودی حساب را افزایش و فقط یک ریسمان آن را کاهش می‌دهند، شبه کد زیر کافی خواهد بود (برای کسب حدود دو-سوم نمره‌ی سؤال کافی است).

```
Monitor Account {
      // condition variable for outstanding withdrawals
      condition cond;
      // account balance
      int balance;

      void increase(int n) {
            balance += n;
            cond.signal();
      }
      void decrease(int n) {
            while (balance < n)
                  cond.wait();
            balance -= n;
      }
}
```

راه حل قبل را می‌توان با استفاده از سمافور نیز پیاده‌سازی نمود اما باید به انحصار متقابل و درستی انتظار در تابع

decrease() دقت کرد. با وجود بیش از دو ریسمان راه حل بالا کافی نیست (چه مشکلاتی ممکن است رخ دهند؟). شبه کد زیر با فرض وجود حداکثر N ریسمان برای فراخوانی decrease() نوشته شده است. دقت کنید که راه حل ساده‌تری با فرض وجود تابعی برای بیدار کردن همه‌ی ریسمان‌های منتظر روی یک متغیر شرط می‌توان ارائه داد، که آن هم قابل قبول است.

```
Monitor Account {
      // condition variables for outstanding withdrawals
      condition cond[N];
      // the amount of outstanding withdrawals; zero means free
      int amount[N];
      // account balance
      int balance;

      void increase(int n) {
            int left;
            balance += n;
            left = balance;
            // waking up withdrawals that can be satisfied
            for (i = 0; i < N; i++) {
                  if (amount[i] && amount[i] >= left) {
                        left -= amount[i];
                        cond[i].signal();
                  }
            }
      }
      void decrease(int n) {
            int i = 0;
            // finding a free index in cond[] and amount[]
            while (amount[i])
                  i++;
            amount[i] = n;
            if (balance < n)
                  cond[i].wait();
            balance -= n;
            amount[i] = 0;
      }
}
```