Zad. 1

Moc zużywana przez pracujący procesor: $P = CV^2 * f$

Chcemy, aby czas wykonania algorytmu pozostał taki sam, ale zużywał mniej mocy. Wiemy, że algorytm na dwóch rdzeniach wykona się 2 razy szybciej. Zatem możemy to uzyskać zmniejszając dwukrotnie częstotliwość: $f_2 = f_1/2$

Korzystając z liniowych zależności:
$C = x * n$, *x – stała, n – liczba tranzystorów*

$V = y * f$, *y – stała, f - częstotliwość*

Możemy teraz porównać moc obydwóch procesorów:
$P_2 = x * n_2 * y^2 * f_2^3 = x * 2 * n_1 * y^2 * (\frac{1}{2} * f_1)^3 = x * 2n_1 * y^2 * \frac{1}{8} * f_1^3 = \frac{1}{4} * x * n_1 * y^2 * f_1^3 = \frac{1}{4}P_1$

Pokazaliśmy, że jest możliwość zaprojektowania procesora o takim samym czasie wykonania, ale zużywającego znacznie mniej mocy.

Zad. 2

Problem sekcji krytycznej: Jest to problem, w którym dwa procesy (tutaj: smoki Alicji i Boba) chcą jednocześnie korzystać ze wspólnego zasobu (jeziora), ale tylko jeden może to zrobić, aby uniknąć konfliktów.

Alicja:

1. Podnieś flagę
2. Dopóki flaga Boba jest w górze
    a. Jeśli napis wskazuje Bob:
        i. Obniż flagę
        ii. Czekaj aż napis wskaże Alicja
        iii. Podnieś flagę
3. Wypuść smoka
4. Poczekaj aż smok wróci
5. Ustaw napis na Bob
6. Obniż flagę

Bob:

1. Podnieś flagę
2. Dopóki flaga Alicji jest w górze:
    a. Jeśli napis wskazuje Alicja:
        i. Obniż flagę
        ii. Poczekaj aż napis wskaże Bob
        iii. Podnieś flagę
3. Wypuść smoka
4. Poczekaj aż smok wróci
5. Ustaw napis na Alicja
6. Opuść flagę

Zad. 3

Problem producenta-konsumenta polega na tym, że konsument nie może zastać pustych półek, a producent nie może uzupełnić półek, dopóki nie będą puste.

Alicja:

1. Czeka, aż Bob strąci jej puszkę
2. Wpuszcza zwierzaki do jeziora
3. Gdy wrócą sprawdza stan jeziora. Jeśli jest puste to ustawia swoją puszkę z powrotem i ciągnie za sznurek strącając puszkę Boba

Bob:

1. Czeka, aż Alicja strąci jego puszkę
2. Dokłada jedzenie do jeziora
3. Ustawia swoją puszkę z powrotem i ciągnie za sznurek strącając puszkę Alicji

Zad. 4

1. Bezpieczeństwa
2. Żywotności
3. Żywotności
4. Żywotności
5. Żywotności
6. Bezpieczeństwa
7. Żywotności

Zad. 5

Spośród ‘n’ więźniów, niech jeden pełni rolę przywódcy, odpowiedzialnego za zliczanie ilości więźniów, którzy już odwiedzili pokój.

Każdy z pozostałych więźniów będzie miał proste zadanie, aby tylko włączyć przycisk, jeżeli jest wyłączony i jeżeli jeszcze tego nie zrobił.

Przywódca za to za każdym razem jak wejdzie do pokoju, jeżeli przycisk jest włączony to dodaje 1 do liczby więźniów, którzy już byli w pokoju i wyłącza przycisk, w.p.p. nic nie robi.

Jeżeli przywódca doliczy się ‘n-1’ więźniów w pokoju ogłasza, że każdy był tam co najmniej raz.

Zad. 6

Podobna strategia jak w zadaniu 5, z minimalnymi zmianami. (Zakładamy, że jest n więźniów)

Przywódca:

1. Jeżeli przycisk jest włączony - liczy i wyłącza go
2. Jeżeli przycisk jest włączony - nic nie robi
3. Jeżeli zliczył „2n-2” - ogłasza, że wszyscy odwiedzili pokój

Więzień:

1. Jeżeli jest wyłączony i nie włączał jeszcze przycisku dwukrotnie – włącza go
2. Jeżeli przycisk jest włączony – nic nie robi

Zad. 7

```java=
public void run() {
      Random random = new Random();
      while (true) {
         try {
            sleep(random.nextInt(1000));
            sleep(100);
            System.out.println("Philosopher " + id + " is hungry");
            left.get();
            right.get();
            left.put();
            right.put();
         } catch (InterruptedException ex) {
            return;
         }
      }
   }
```

Każdy filozof najpierw sięga po lewy widelec, a dopiero później po prawy. Ponieważ mamy tyle samo filozofów i widelców to zakleszczenie nastąpi, kiedy każdy filozof weźmie do ręki lewy widelec, wtedy nie będzie żadnego prawego widelca wolnego do wzięcia.

Rozwiązanie:
Indeksujemy widelce. Filozofowie biorą najpierw widelec z mniejszym indeksem, a dopiero potem z większym. Zakleszczenie teraz nie nastąpi, ponieważ n-ty filozof zamiast lewego widelca będzie chciał wziąć prawy zatem będzie on na niego czekał, dzięki temu n-1-ty filozof będzie miał dostępny prawy widelec. Gdy skończy, odłoży widelec z najwyższym numerem, a następnie z niższym, umożliwiając kolejnemu filozofowi zabranie drugiego widelca itd.

```java=
public void run() {
```

```
        Random random = new Random();
        while (true) {
          try {
            sleep(random.nextInt(1000));
            sleep(100);
            System.out.println("Philosopher " + id + " is hungry");
            if (left.id < right.id){
              left.get();
              right.get();
              right.put();
              left.put();
            }
            else {
              right.get();
              left.get();
              left.put();
              right.put();
            }
          } catch (InterruptedException ex) {
            return;
          }
        }
      }
```