Ćwiczenie 9

Ad. 1

Co by było, gdyby w problemie producenta-konsumenta pętle oczekujące, aż bufor nie będzie pusty/pełny umieścić również w sekcji krytycznej?
Umieszczenie pętli oczekującej w sekcji krytycznej należy do sytuacji, której należy unikać ponieważ powinna być ona jak najprostsza. Mogą wystąpić przypadki, w których wątki nie zostaną wpuszczone do sekcji krytycznej lub też nigdy jej nie opuszczą, blokując producentów i konsumentów, zakładając, że z jakiegoś powodu nie spełnią warunków opuszczenia pętli. W pętli natomiast działają sekcja wejściowa i sekcja wyjściowa. Tego rodzaju sytuacja może doprowadzać do zakleszczenia, niepotrzebnych opóźnień i marnowania zasobów.

Ad. 2

Instrukcja exchange zapewnia sprzętowe wsparcie dla takich działań jak synchronizacja dostępu do sekcji krytycznej. Wykorzystanie exchange należy do operacji atomowych, co oznacza, że żaden proces czy wątek nie powinien nie powinien zakłócić jej prawidłowego działania i wartość nie ulegnie nieporządanej zmainie. Możemy to zastosować definiując zmienną używając instrukcji exchange. Jeśli jest ustawiona wartość 0 to sekcja jest wolna, w przypadku wartości 1 jest ona zajęta i proces musi poczekać na wejście. Istotne jest żeby powyższa zmienna działała właśnie w sposób atomowy.

Ad. 3

Instrukcja test-and-set sprawdza wartość zmiennej binarnej (o nazwie lock) i ustawia ją na nową wartość w jednej, niewrażliwej na nie porządane zmiany operacji (atomowej).

Każdą pałeczkę reprezentuje zmienna lock (zmienna atomic). Wartość true oznacza, że jest zajęta, a false wolna). Każda z sąsiadujących pałeczek zarówno „po lewej”, jak i „po prawej” stronie filozofa musi być zwolniona aby rozpocząć posiłek. Ten moment jest kluczowy i wykorzystuje instrukcję test-and-set podczas zmiany wartości dostępu do pałeczek dla filozofa.

Filozof usiłuje zdobyć „zestaw” pałeczek rozpoczynając od lewej pałeczki, następnie prawej, z wyjątkiem ostatniego z nich, który stara się uzyskać obie w odwrotnej kolejności. Dzięki takiemu ograniczeniu powinniśmy uniknąć zakleszczeń.

```
// Instrukcja test-and-set
bool test_and_set(bool *lock) {
    bool old = *lock;
    *lock = true; // Ustawienie blokady
    return old;   // Zwrócenie poprzedniego stanu
}

// Tablica reprezentująca stan pałeczek (zainicjowane wszystkie dostępne)
bool sticks[5] = {false, false, false, false, false};
// Mutex dla synchronizacji
bool mutex = false;

// Funkcja wejścia do sekcji krytycznej
void enter_critical_section(bool *lock) {
    while (test_and_set(lock)); // Czekaj, aż lock zostanie zwolniony
}
```

```
// Funkcja wyjścia z sekcji krytycznej
void exit_critical_section(bool *lock) {
    *lock = false; // Zwolnienie locka
}
// Funkcja działania filozofa
void philosopher(int id) {
    int left = id;              // Indeks lewej pałeczki
    int right = (id + 1) % 5;    // Indeks prawej pałeczki

    while (true) {
        // Myślenie
        think();

        // Wejście do sekcji krytycznej
        enter_critical_section(&mutex);

        // Podnieś lewą pałeczkę, jeśli dostępna
        while (test_and_set(&sticks[left]));
        // Podnieś prawą pałeczkę, jeśli dostępna
        while (test_and_set(&sticks[right]));

        // Wyjście z sekcji krytycznej
        exit_critical_section(&mutex);

        // Jedzenie
        eat();

        // Zwolnienie widełek
        sticks[left] = false;
        sticks[right] = false;

        // Powrót do myślenia
    }
}
```

Ad 4

Program w języku java opierający się na problemie pięciu ucztujących filozofów z wykorzystaniem wątków i klasy Semaphore.

```
import java.util.concurrent.Semaphore;
public class Main {
    //ustalenie liczby filozofów i czas rozmyslania
    static final int MAX_RANDOM_TIME = 1500;
    static final int SLOT_LIMIT=1;
    private static final int LICZBA_FILOZOFOW = 5;

    public static void main(String[] args) {
    //wykorzystanie klasy Semaphore
        Semaphore[] sticks = new Semaphore[LICZBA_FILOZOFOW];
        for(int i = 0; i< LICZBA_FILOZOFOW; i++){
            sticks[i] = new Semaphore(1);
        }
        Thread[] filozofowie = new Thread[LICZBA_FILOZOFOW];
        for (int i =0; i<LICZBA_FILOZOFOW;i++){
            int filozofInd = i;
            filozofowie[i] = new Thread( ()-> {
                while (true){
                    try {
                        mysl(filozofInd);
                    } catch (InterruptedException e) {
                        throw new RuntimeException(e);
                    }
                    if (filozofInd % 2 == 0){
```

```
                        try {
                            sticks[filozofInd].acquire();
                            sticks[(filozofInd+1)% LICZBA_FILOZOFOW].acquire();
                        } catch (InterruptedException e) {
                            throw new RuntimeException(e);
                        }
                    }else {
                        try {
                            sticks[(filozofInd+1)% LICZBA_FILOZOFOW].acquire();
                            sticks[filozofInd].acquire();
                        } catch (InterruptedException e) {
                            throw new RuntimeException(e);
                        }
                        try {
                            jedz(filozofInd);
                        } catch (InterruptedException e) {
                            throw new RuntimeException(e);
                        }
                        sticks[filozofInd].release();
                        sticks[(filozofInd+1)% LICZBA_FILOZOFOW].release();
                    }
                }
            });
        }
        // działanie filozofów
        for (int i=0; i<4; i++){
            filozofowie[i].start();
        }
        System.out.println("piąty filozof czeka");
    }
    //jedzenie
    private static void jedz(int filozofInd) throws InterruptedException {
        System.out.println("filozof "+filozofInd+" je");
        Thread.sleep((int) (Math.random()*2000));
    }
    //rozmyslanie
    private static void mysl(int filozofInd) throws InterruptedException {
        System.out.println("filozof "+filozofInd+" mysli");
        Thread.sleep((int) (Math.random()*2000));
    }
}
```

Ćwiczenie 10
Ad 1.

first-fit

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1MB</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>P5</td><td>P5</td><td>P5</td><td>P5</td><td>|</td></tr>
<tr><td>P1</td><td>P1</td><td>P1</td><td>P1</td><td>P1</td><td></td><td></td><td></td><td></td><td></td><td>|</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>P6</td><td>P6</td><td>|</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>|</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>P7</td><td>|</td></tr>
<tr><td></td><td></td><td></td><td></td><td>P4</td><td>P4</td><td>P4</td><td>P4</td><td>P4</td><td>P4</td><td>|</td></tr>
<tr><td></td><td>P2</td><td>P2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>|</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>|</td></tr>
<tr><td></td><td></td><td>P3</td><td>P3</td><td>P3</td><td>P3</td><td>P3</td><td></td><td></td><td></td><td>10MB</td></tr>
</table>

Best-fit

										1MB
						P5	P5	P5	P5	|
P1	P1	P1	P1	P1						|
									P7	|
										|
										|
				P4	P4	P4	P4	P4	P4	|
	P2	P2								|
								P6	P6	|
		P3	P3	P3	P3	P3				10MB

Worst-fit

										1MB
						P5	P5	P5	P5	|
P1	P1	P1	P1	P1						|
								P6	P6	|
										|
										|
				P4	P4	P4	P4	P4	P4	|
	P2	P2								|
									P7	|
		P3	P3	P3	P3	P3				10MB

Ad. 2

Adres wytwarzany przez procesor jest zazwyczaj nazywany adresem logicznym (nazywany też adresem wirtualnym). Adres oglądany przez jednostkę pamięci (ten umieszczony w jej rejestrze adresowym) to adres fizyczny. Dzięki takiej formie zarządzania pamięcią możliwe jest zapewnienie procesom wystarczającej ilości.
Adresy logiczne i fizyczne są powiązane. Adresy logiczne są odwzorowane na adresy fizyczne i stosuje się do tego wiele technik.
Odbywa się to z użyciem sprzętowego wsparcia w postaci MMU (memory management unit).

Ad. 3
Jaka jest różnica między fragmentacją wewnętrzną i zewnętrzną?
Jeżeli nie ma wspólnego jednolitego obszaru do zaalokowania pamięci mamy do czynienia z fragmentacją zewnętrzną i konieczna będzie kompaktyfikacja, czyli scalanie wolnych obszarów. Powstaje, gdy w pamięci fizycznej dostępne są niewykorzystane luki, które są zbyt małe, aby wystarczyły na pomieszczenie nowego żądania pamięci przez proces.
W przypadku fragmentacji wewnętrznej mamy do czynienie gdy proces otrzymuje więcej pamięci

niż potrzebuje, ale nie jest możliwe zapewnienie odpowiednio niewielkiego przez co marnowana jest pamięć „wewnątrz” bloku.
Zewnętrzna to wynika z marnowania przestrzeni pamięci między blokami (poprzez ich „rozdrobnienie”), natomiast wewnętrzna poprzez niewykorzystane zasoby pamięci wewnątrz bloku przydzielonego dla procesu.

Ad. 4
MMU (Memory Management Unit) jest sprzętowym rozwiązaniem (jednostką) wspomagającym zarządzanie pamięcią. Adresy pamięci generowane przez programy są adresami wirtualnymi, co wymaga translacji na adresy fizyczne, dodatkowo korzysta się także z stronicowania. MMU obsługuje te czynności czyli translację adresów logicznych na fizyczne, zapewnia odizolowanie pamięci dla różnych procesów, wspiera obsługę obszaru wymiany (czyli segmentację, stronicowanie).

Ćwiczenie 11
Ad. 1

FIFO

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1	1	1	5	5	5	5	5	3	3	3	3	3	1	1	1	1
	2	2	2	2	2	2	6	6	6	6	6	7	7	7	7	7	7	3	3
		3	3	3	3	3	3	2	2	2	2	2	6	6	6	6	6	6	6
			4	4	4	4	4	4	1	1	1	1	1	1	2	2	2	2	2

14 braków stron

OPTMALNY

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1	1	1	1	1	1	1	1	1	7	7	7	7	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
			4	4	4	5	6	6	6	6	6	6	6	6	6	6	6	6	6

8 braków stron

LRU

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1	1	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	5	5	5	5	5	3	3	3	3	3	3	3	3	3
			4	4	4	4	6	6	6	6	6	7	7	7	7	1	1	1	1

10 braków stron

2nd CHANCE

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1	1	1	1	6	6	6	6	6	6	6	6	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	3
		3	3	3	3	5	5	5	5	5	5	7	7	7	7	7	7	7	6
			4	4	4	4	4	4	1	1	1	1	1	1	2	2	2	2	2

12 braków stron