SYSTEMY OPERACYNE – LABORATORIUM

Synchronizacja procesów z wykorzystaniem monitorów
Przygotował: Tomasz Bocheński

1. TREŚĆ ZADANIA:

Wyobraźmy sobie fabrykę, w której produkowany jest przedmiot P składający się z trzech elementów: X,Y,Z. W fabryce pracują N + M + O + 1 robotów (wątków):

- N robotów specjalizuje się w produkcji elementów X i po wyprodukowaniu elementu kładzie go na taśmie produkcyjnej (buforze) BP,
- M robotów specjalizuje się w produkcji elementów Y i po wyprodukowaniu elementu kładzie go na taśmie produkcyjnej (buforze) BY,
- robotów specjalizuje się w produkcji elementów Z i po wyprodukowaniu elementu kładzie go na taśmie produkcyjnej (buforze) BZ,
- 1 robot pobiera i składa elementy z taśm produkcyjnych BX, BY i BZ – tak powstały przedmiot kładzie na taśmie produkcyjnej (buforze) BP.

Przyjmijmy, że elementy X i Z to liczby naturalne mniejsze od 10, a element Y to znak z przedziału ‘a’-‘j’. Złożenie elementu P to operacja polegająca na stworzeniu ciągu znaków postaci XYZ. Należy przyjąć że taśmy produkcyjne mogą jednocześnie pomieścić najwyżej 5 elementów. Produkcja powinna zostać zakończona po zbudowaniu 10 elementów P.
Dla uproszczenia można przyjąć, że robot składający pobiera najpierw element z kolejki X, następnie z kolejki Y a na końcu z Z. Jeśli któraś z tych kolejek jest pusta, robot oczekuje na pojawienie się elementu.

W ramach zadania należy stworzyć program symulujący pracę fabryki przy pomocy N + M + O + 1 wątków (roboty) i 3 buforów (taśmy produkcyjne). Aby zapobiec przepełnieniu buforów i jednoczesnemu wykonywaniu operacji na buforze należy wykorzystać mechanizm monitorów. Należy zwrócić szczególną uwagę, aby żaden z elementów nie został „zdublowany” albo zagubiony.
Dodatkowo należy zasymulować czas pracy robotów przez uśpienie procesów na losową wartość z zakresu:

- <1, P> dla procesów produkujących element X,
- <1, R> dla procesów produkujących element Y,
- <1, S> dla procesów produkujących elementy Z.

Wartości M, N, P, R i S powinny być konfigurowalne przy uruchomieniu.

2. PRZYJĘTE ZAŁOŻENIA:

- elementy X,Y,Z to wartości typu char, gdzie X i Z to znaki z przedziału ‘0’ – ‘9’, a Y to znak z przedziału ‘a’ – ‘j’;
- taśmy produkcyjne mogą pomieścić najwyżej 5 elementów;
- produkcja zostaje zakończona po wyprodukowaniu 10 elementów typu P;
- robot składający pobiera najpierw element z kolejki X, potem z Y, potem z Z. Jeśli któraś kolejka jest pusta, robot oczekuje na pojawienie się elementów;
- czas pracy robotów symulowany jest przez uśpienie procesów na losową wartość z zakresu
- <1, P> dla procesów produkujących element X,
- <1, R> dla procesów produkujących element Y,

- <1, S> dla procesów produkujących elementy Z.
- Czas pracy robota pobierającego elementy z taśm produkcyjnych i składającego je w jedną całość jest równy 0 (gotowy element P powstaje od razu po zebraniu wszystkich trzech typów elementów);
- w zadaniu wykorzystany zostanie mechanizm monitorów (wykorzystana zostanie przykładowa klasa Monitor znajdująca się na stronie przedmiotu);
- powstanie klasa Buffer, która dziedziczyć będzie po klasie Monitor (szczegóły odnośnie tej klasy zostaną przedstawione później);
- wartości M, N, O, P, R, S będą wczytywane na początku, następnie nastąpi przejście do symulacji programu.

3. PROPONOWANE ROZWIAZANIE:

Proponuję stworzenie dwóch plików:
- Pierwszy plik będzie plikiem nagłówkowym. Znajdować się w nim będą:
  - ➔ definicje klas Semaphore, Condition oraz Monitor wraz z definicjami ich metod.
- Drugi plik będzie plikiem wykonywalnym. Znajdować się w nim będą:
  - ➔ definicja klasy Buffer wraz z definicjami jej metod
  - ➔ definicje funkcji robotX(), robotY(), robotZ(), oraz robotCreator(), zajmujących się odpowiednio produkowaniem elementów X, Y, Z, oraz składaniem ich w całość, produkowaniem elementu P
  - ➔ funkcja main().

4. WYKORZYSTANE STRUKTURY / KLASY ORAZ STAŁE:

Stałe:

```
#define BUFFER_SIZE 5                 /// stała opisująca wielkość bufora
const bool PRODUCTION = true;         /// użyta do pętli while(PRODUCTION)
const int productX = 48;              /// pomocnicza zmienna w produkcji elementów X
const int productY = 97;              /// pomocnicza zmienna w produkcji elementów Y
const int productZ = 48;              /// pomocnicza zmienna w produkcji elementów Z
```

Klasy Semaphore, Condition oraz Monitor zostaną zaimplementowane zgodnie z implementacją przedstawioną na stronie przedmiotu.
Dodatkowo stworzona zostanie klasa Buffer, która wyglądać będzie następująco:

```
class Buffer: public Monitor
{
private:
        std::string name;             /// nazwa bufora, np. bufor X
        int count;                    /// licznik opisujący ilość elementów w buforze
        char buffer[BUFFER_SIZE];     /// tablica elementów
        Condition full, empty;        /// zmienne warunkowe
        int first_full;               /// liczba opisująca indeks pierwszego zajętego miejsca
        int first_empty;              /// liczba opisująca indeks pierwszego wolnego miejsca
public:
        Buffer(std::string bufName);  /// konstruktor z parametrem – nazwą bufora
        void put(char item, int putTime);  /// metoda wkładająca element do bufora
        char get()';                  /// metoda wyjmująca element z bufora
```

```
}
```

5. SPOSOBY WYKORZYSTANIA MONITOROW W IMPLEMENTACJI FUNKCJI (KOD):

```
void put(char item, int putTime)
{
	enter();

	while(count == BUFFER_SIZE)
		wait(full);

	sleep(putTime);
	buffer[first_empty] = item;
	first_empty = (first_empty + 1) % BUFFER_SIZE;
	++count;

	if(count == 1)
		signal(empty);
	leave();
}

char get()
{
	char item;
	enter();

	while(count == 0)
		wait(empty);

	item = buffer[first_full];
	first_full = (first_full + 1) % BUFFER_SIZE;
	--count;

	if(count == BUFFER_SIZE - 1)
		signal(full);
	leave();
	return item;
}
```

6. IMPLEMENTACJA FUNKCJI REPREZENTUJĄCYCH ROBOTY:

```
void * robotX(void * data)
{
	char charProduct;
	int progressFactorX = 0;
	int seed = *((int*)(&data));

	srand(seed);

	while(PRODUCTION)
	{
		charProduct = (char)(productX + progressFactorX);
```

```
            progressFactorX = (progressFactorX + 1) % 10;
            int time = 1 + rand()%maxTimeX;

            bufferX.put(charProduct, time);
        }
        return EXIT_SUCCESS;
}
```

Analogiczne funkcje dla robotów Y oraz Z.

```
/// jako argument przekazywana jest liczba elementów które muszą zostać wyprodukowane aby
///zakończyć produkcję
void * robotCreator(void * data)
{
        char str[4];
        int d = *((int*)(&data));

        for(int i = 0; i < d; ++i)
        {
            str[0] = bufferX.get();
            str[1] = bufferY.get();
            str[2] = bufferZ.get();
            str[3] = '\0';
        }
        return EXIT_SUCCESS;
}
```

7. WIELOWĄTKOWOŚĆ:

W celu obsługi wątków zostaną wykorzystane następujące funkcje:

- int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void*), void *arg) – tworzy ona nowy wątek;
- int pthread_join(pthread_t thread, void **value_ptr) – wstrzymuje dalsze wykonywanie funkcji wywołującej wątek do momentu jego zakończenia;

Możliwe funkcje dodatkowe wykorzystane do obsługi wątków:

- int pthread_cancel(pthread_t thread) – wysyła żądanie do wątku aby zakończył on swoje działanie

8. REALIZACJA FUNKCJI MAIN:

- Na początku zostaną utworzone odpowiednie zmienne oraz wczytane zostaną argumenty funkcji podane przez użytkownika.
- Następnie stworzone zostaną wątki reprezentujące odpowiednie roboty, dla każdego robota powstanie inny wątek. Powstanie zatem M wątków robotów X, N wątków robotów Y, O wątków robotów Z oraz 1 wątek robotaCreator.
- Funkcje reprezentujące roboty X, Y i Z działać będą w nieskończonej pętli. Funkca robotCreator działać będzie w skończonej pętli for i wykona się tyle razy, aby stworzyć określoną wcześniej liczbę elementów.
- Po stworzeniu robotów wykorzystana zostanie funkcja pthread_join, która wstrzyma dalsze wykonywanie funkcji main aż do momentu zakończenia działania robotaCreator, czyli wyprodukowania określonej liczby produktów.
- W tym momencie trzeba będzie zatrzymać działanie reszty robotów. Jeśli chodzi o sposób

zakończenia reszty wątków to nie jest on jeszcze do końca ustalony. Jednak ze wszystkich przemyślanych przeze mnie sposobów wybrałem dwa, które wydają mi się najlepsze.

- ➔ Pierwszy sposób w jaki można by było to zrobić to dodanie w środku nieskończonych pętli fragmentu kodu if(...) { break; }, który umożliwiłby zakończenie działania wątku gdy zajdzie taka potrzeba.
- ➔ Drugi sposób to wykorzystanie funkcji pthread_cancel, która wysyła żądanie zakończenia działania wątku.

- Bez względu na sposób kończenia pracy wątków, na samym końcu funkcji main znajdować się będzie funkcja pthread_join, która w pętli będzie czekać na zakończenie działania wszystkich robotów.

9. SPOSÓB PRZETESTOWANIA POPRAWNOŚCI ZAIMPLEMENTOWANEGO ZADANIA:

Jako sposób testowania proponuję wyprowadzać na ekran komunikaty o włożeniu produktu do bufora, wyjęciu produktu z bufora oraz informację o wyprodukowaniu elementu końcowego. Wraz z każdym tego typu komunikatem będą wyświetlane bardziej szczegółowe informację, czyli jaki produkt jest wkładany/ wyjmowany z którego bufora, oraz w przypadku wyprodukowania elementu końcowego, jak on wygląda. Testowanie takie pozwoli na bieżące śledzenie działania wątków, dlatego wydaje mi się że powinno być wystarczające w tego typu zadaniu.

10. PROBLEMY I PRZEMYŚLENIA:

Z definicji monitor to zbiór procedur, zmiennych i struktur danych, które są zgrupowane w specjalnym module. W każdej chwili **tylko jeden proces aktywny może przebywać w monitorze**. Zatem nie można jednocześnie wkładać elementu do bufora oraz wyjmować elementu z bufora. Może to powodować duże opóźnienia w wyjmowaniu elementów z bufora, ponieważ aby dostać się do monitora wątek typu robotCreator będzie musiał czekać w kolejce wraz zresztą wątków. Zatem jeśli liczba wątków produkujących np. element X będzie większa od pojemności bufora X, oraz wątek typu robotCreator wyjmujący elementy z bufora zostanie stworzony jako ostatni, to dostanie on dostęp do bufora jedynie w momencie, gdy w buforze X nie będzie już miejsc i wszystkie wątki produkujące elementy X zostaną zawieszone. Wtedy będzie on jedynym wątkiem zdolnym do wejścia. Można by było próbować uniezależnić wkładanie do bufora od wyjmowania z bufora, jednak wtedy otrzymana struktura danych nie spełniała by definicji monitora.