**Kamil Lipski, G4I2**

# Systemy operacyjne (I)

Ćwiczenie IV – monitory

## 1. Treść zadania

Jest danych 2 producentów, którzy produkują losowe liczby i umieszczają je w 9-cio elementowym buforze FIFO. Jest też danych 2 konsumentów, którzy konsumują produkty z bufora (usuwają je), przy czym konsument pierwszy konsumuje tylko liczby parzyste, a drugi tylko liczby nieparzyste. Ponadto konsumenci mogą konsumować tylko, jeśli w buforze znajdują się co najmniej 3 produkty. Zadanie należy wykonać korzystając z monitorów.

## 2. Proponowane rozwiązanie

Rozwiązanie powyższego zadania na monitorach będzie różniło się w pewnym stopniu od jego rozwiązania za pomocą semaforów.

Jedną z najważniejszych różnic pomiędzy semaforem a monitorem jest kolejność budzenia procesów: w semaforach jest ona dowolna, natomiast w monitorach – ściśle określona. Ponadto, podniesienie semafora powoduje kontynuację wykonywania procesu, a po signalu (monitor) jest on jedynie wrzucany na stos oczekujących procesów. Warto również wspomnieć, że w przeciwieństwie do monitora, semafor nie jest mechanizmem strukturalnym (co jest jego wadą).

Ponownie należy pamiętać o zabezpieczeniu się przed próbą umieszczenia produktu w pełnym buforze przez producenta, jako że bufor nie jest nieskończony i posiada narzuconą liczbę wolnych miejsc – 9 (zgodnie z założeniami zadania). W przypadku konsumentów należy sprawdzać ilość elementów znajdujących się w buforze (trzeba zapewnić zgodność z założeniami zadania). Jeśli będą w nim mniej niż trzy elementy, pobranie elementu nie może nastąpić, a proces zostanie zawieszony (sytuacja analogiczna do klasycznego problemu producent – konsument, w którym pobranie elementu nie mogło nastąpić przy pustym buforze).

Bufor zostanie zaimplementowany jako kolejka FIFO.

Najważniejszą zmianą względem poprzedniego zadania będzie wprowadzenie dwóch nowych, „zaprzyjaźnionych” ze sobą klas wykorzystujących semafory: klasy *Monitor* i klasy *Conditions*, odpowiadającej za warunki synchronizujące (każda z nich będzie posiadała po jednym mutexie).

Klasa *Monitor* będzie złożona z metod: *enter()*, *leave()*, *wait()*, *signal()*, a klasa *Conditions* - z: *wait()*, *signal()*. Klasa *Conditions* będzie też zawierała zmienną zliczającą oczekujące procesy.

Ponadto, mam zamiar zaimplementować „rozszerzoną” klasę *enhancedMonitor* dziedziczącą po klasie *Monitor*, w celu sprostania warunkom zadania. Znajdą się w niej dodatkowo: bufor (jako element wymagający ochrony przez monitor), metody do jego obsłużenia oraz trzy zmienne typu *Conditions* dla producentów, konsumenta parzystego i konsumenta nieparzystego.

Funkcje producentów i konsumentów będą korzystały jedynie z metod publicznych klasy *enhancedMonitor* dotyczących operacji na buforze (*addElement()*, *removeElement()*).

Producenci produkują i wstawiają produkty do bufora tak długo, aż nie dostaną informacji o całkowitym zapełnieniu miejsc. Wtedy procesy, które pozostają bez przydziału (są zawieszone) czekają do momentu, aż zwolni się miejsce (i wybudzi je inny proces). Analogiczna sytuacja występuje w przypadku konsumentów.

W funkcji producenta losowe liczby będą generowane za pomocą funkcji *rand()*, a jako jej zarodek w przypadku pierwszego producenta będzie wstawiany czas pobrany w sekundach, a w przypadku drugiego – pierwszy element znajdujący się w buforze (jeśli takowego nie będzie, to pozostawię domyślny zarodek). Zostaną również zaimplementowane dwie funkcje konsumenta; każda z nich będzie „zainteresowana” innym typem produktu (tu: parzystością liczby). Będą się one różnić parametrem przekazywanym do wywoływanej w ich ciele funkcji *removeElement()*.

Ogólna zasada działania będzie polegała na tym, że proces będzie zajmował sekcję krytyczną; jeśli z jakichś powodów nie będzie mógł kontynuować swojej pracy (np. niespełnione warunki), proces opuści sekcję krytyczną i zostanie wstrzymany do momentu, aż obudzi go inny z działających procesów i odda mu sekcję krytyczną. Wybór procesu do wybudzenia będzie następował we wspomnianych wcześniej publicznych metodach klasy *enhancedMonitor*.