Programowanie współbieżne
Zadanie numer 3
Monitory

Cel zadania

Celem zadania jest poznanie monitorowego mechanizmu synchronizacji procesów i zasad jego działania.

Podstawa zaliczenia

1. Zademonstrowanie działającego programu.
2. Wykazanie że przedstawiony w zadaniu mechanizm jest monitorem
3. Oddanie sprawozdania

Sprawozdanie

W sprawozdaniu powinny się znaleźć następujące informacje:
1. Sposób rozwiązania problemu,
2. Wykazanie, że rozwiązanie problemu jest poprawne,
3. Wykazanie, że przedstawiony w zadaniu mechanizm jest monitorem,

Pytania kontrolne

1. Co to jest Monitor?
2. Dlaczego Monitor uważany jest za narzędzie wygodniejsze od semafora?
3. Dlaczego SIGNAL powinien być ostatnią instrukcją Monitora?
4. Czy dwie różne procedury tego samego Monitora mogą wykonywać się współbieżnie?

Teoria

Monitory

Semafor wprowadzono w celu dostarczenia synchronizującego mechanizmu pierwotnego, który nie wymaga aktywnego oczekiwania. Semafor jest mechanizmem pierwotnym niskiego poziomu, ponieważ nie jest strukturalny. Gdybyśmy mieli zbudować wielki system używając wyłącznie semaforów, odpowiedzialność za poprawne użycie semaforów byłaby rozproszona między wszystkie osoby implementujące system. Gdyby jedna z nich zapomniała wywołać Signal (S) po sekcji krytycznej, to w programie mogłaby wystąpić blokada, a przyczynę błędu trudno byłoby zlokalizować.

Monitory dostarczają strukturalnego mechanizmu pierwotnego dla programowania współbieżnego, skupiającego odpowiedzialność za poprawność w kilku modułach. Monitory są uogólnieniem monitora monolitycznego (lub *jądra,* lub *nadzorcy)* spotykanego w systemach operacyjnych. Sekcje krytyczne, takie jak przydzielanie urządzeń wejścia-wyjścia i pamięci, kolejkowanie żądań wejścia-wyjścia itd,, są skupione w uprzywilejowanym programie. Zwykłe programy żądają *usług* wykonywanych przez centralny monitor, uprzywilejowane zaś są wykonywane w specjalnym trybie sprzętowym zapewniającym, że ich wykonanie nie będzie zakłócane przez zwykłe programy. Dzięki separacji systemu i programów użytkowych, zwykle jest jasne, kto ponosi winę, gdy system przestaje działać (chociaż określenie dokładnej przyczyny może okazać się niezwykle trudne).

Monitory są zdecentralizowaną wersją monitora monolitycznego. Zamiast jednego programu obsługującego wszystkie żądania obsługi, wymagające dzielonych urządzeń lub struktur danych, możemy zdefiniować oddzielny monitor dla każdego obiektu lub grupy obiektów powiązanych ze sobą. Procesy żądają usług od różnych monitorów. Jeśli ten sam monitor jest wywoływany przez dwa

procesy, to implementacja gwarantuje, że te procesy będą obsłużone w takiej kolejności, by zapewnić wzajemne wykluczanie. Jeśli są wywoływane różne monitory, ich wykonywania mogą być przeplatane.

Składnia monitora opiera się na kapsułkowaniu (ang. *encapsulation)* elementów danych i działających na nich procedur w pojedyncze moduły. Interfejs monitora składa się ze zbioru procedur, które operują na danych ukrytych w module. Różnica między monitorem a zwykłym modułem, takim jak pakiet (ang. *package)* w Adzie, polega na tym, że monitor nie tylko ochrania wewnętrzne dane przed nieograniczonym dostępem, lecz także synchronizuje wywołania procedur występujących w jego interfejsie. Implementacja zapewnia wzajemne wykluczanie wykonań tych procedur.

Monitor grupuje w jednym miejscu wszystkie zmienne dzielone i wszystkie wykonywane na nich operacje. Dostęp do zmiennych spoza monitora możliwy jest wyłącznie poprzez wywołanie procedury monitorowej, która wykona odpowiednie działania na tych zmiennych. Główną cechą monitora jest to, że odwołania do procedur monitorowych wzajemnie się wykluczają. Potocznie mówi się o procesie, który rozpoczyna wykonywanie procedury monitorowej, że „wchodzi do monitora", gdy wykonuje procedurę- „przebywa wewnątrz monitora", natomiast gdy kończy wykonanie procedury, to „wychodzi z monitora". Ze względu na wzajemne wykluczanie odwołań do procedur, w monitorze może przebywać tylko jeden proces. Konsekwencją jest kolejka wejściowa procesów do monitora.

Procesy mogą również warunkowo opuszczać monitor, aby poczekać na spełnienie warunku umożliwiającego kontynuację działań. Do tego celu służą specjalne zmienne kolejkowe (typ conditional), na których wykonywane są dwie operacje:

- **wait(kol)** – zawieszenie wywołującego procedurę procesu w kolejce kol z jednoczesnym opuszczeniem monitora i umożliwienie wejścia do niego innemu procesowi,
- **signal(kol)** – w przypadku istnienia niepustej kolejki procesów związanej ze zmienną kol wznowienie pierwszego z nich; proces wraca do monitora i kontynuuje obliczenia od pierwszej instrukcji za wait.

Operacje wait i signal mogą być wykonywane wyłącznie przez procedury monitorowe. Z jednym monitorem może być związane kilka kolejek zawieszonych procesów. Zauważyć należy, że proces zawieszony może być wznowiony wyłącznie przez inny proces wykonujący procedurę monitorową.

Symulacja monitorów w pomocą semaforów

W celu zasymulowania monitorów za pomocą semaforów będziemy potrzebować semafora S, do zapewnienia wzajemnego wykluczania procedur monitora, i po jednym semaforze W_ semafor dla każdej zmiennej warunkowej. Zakładamy, że są to semafory z kolejką oczekujących, aby móc zapewnić kolejkowanie FIFO wymagane przez definicję monitora. W przeciwnym razie musielibyśmy jawnie zaprogramować kolejkę wstrzymanych procesów.

Potrzebujemy ponadto licznika dla każdej zmiennej warunkowej, gdyż znaczenie Signal (W) zależy od tego, czy kolejka jest pusta, czy też nie. Ponieważ nie ma żadnego sposobu odczytania tego z semafora, musimy sami zaprogramować licznik.

Każda procedura monitora będzie zaczynać się instrukcją Wait (S) i kończyć Signal (S).

Każde wystąpienie Wait (W) tłumaczymy na

```
W-Licznik := W-.Licznik -f- l;
Signal(S) ;
Wait(W-semafor) ;
Wait(S) ;
W-licznik : == W_licznik - l/
```

Definicja Wait (W) wymaga przywrócenia dostępu do monitora instrukcją Signal (S). Naturalnie, aby uniknąć blokady, musimy to robić przed oczekiwaniem na spełnienie warunku.

Każde wystąpienie Signal (W) tłumaczymy na

```
if W-licznik > O then Signal(W.semafor) ;
end if;
```

To jednak nie symuluje poprawnie wymagania natychmiastowego uruchomienia. Pamiętajmy, że założyliśmy, iż Signal (W) będzie ostatnią instrukcją w procedurze, tak więc Signal (S) nastąpi zaraz potem:

```
if W-licznik > O then
Signal(W_semafor) ;
end i f ;
Signal(S) ;
```

Wysłanie sygnału do semafora związanego z warunkiem wznowi wstrzymany proces. Jednak ciągle jest możliwe wplecenie instrukcji trzeciego procesu między Wait(W-semafor) a Wait (S). Ten trzeci proces mógłby wykonać. Wait (S) przed wznowionym procesem i pogwałcić wymaganie natychmiastowego uruchomienia. Wysłanie sygnału do semafora z kolejką oczekujących jest potrzebne jedynie do wznowienia procesu znajdującego się na początku kolejki w celu dania mu pierwszeństwa przed innymi wstrzymanymi procesami; nie jest zaś potrzebne po to, by wymusić natychmiastowe uruchomienie tego procesu.

Rozwiązanie polega na tym, by nie przywracać dostępu do monitora podczas wysyłania sygnału, lecz by pozostawić to wznowionemu procesowi. Parę składającą się z instrukcji Signal (W) i z końca procedury monitora przetłumaczymy teraz tak:

```
if W_licznik > O then
Signal(W_semafor) ;
else
Signal(S) ;
end if;
```

a Wait (W) **uprości się do**

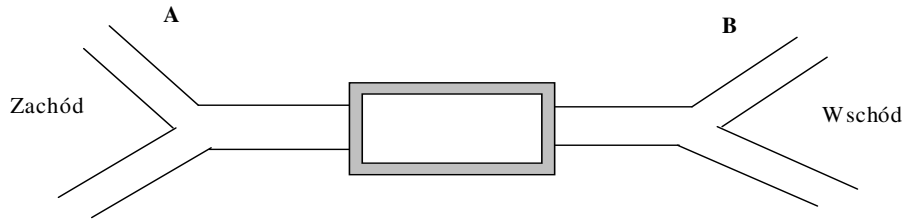
```
W_Licznik := W-Licznik + l;
Signal(S) ;
Wait(W-semafor) ;
```

W_licznik : == W_licznik - l;

Warianty zadań

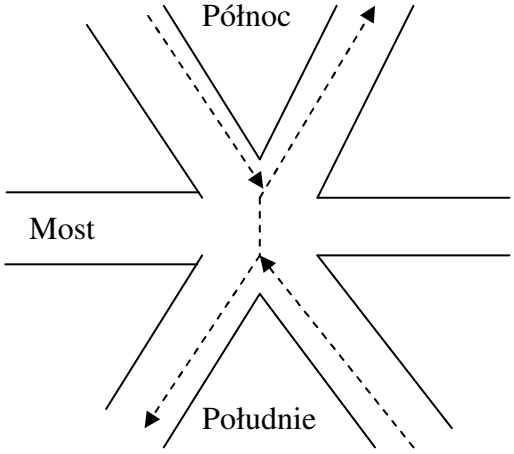
1. Zaprojektować działanie windy według podanych dalej reguł. Pasażerowie przychodzą na pietra i jeśli winda jest bezczynna, to przywołują ją. W przypadku gdy winda znajduje się w ruchu, pasażer musi czekać na swoim piętrze, aż stanie się bezczynna i wówczas będzie można ją przywołać. Może się zdarzyć, że winda zatrzyma się na tym piętrze ponieważ ktoś wysiada. W takim przypadku mogą wsiąść pasażerowie czekający, którzy jadą w tym samym kierunku co winda. Winda staje się bezczynna na piętrze , na którym opuszczą ją wszyscy pasażerowie oraz nikt na nią nie czeka.
2. W systemie są dwa bufory cykliczne b1 i b2. Procesy P1, ..., Pn produkują cyklicznie, niezależnie od siebie, porcje informacji i wstawiają je do bufora b1. Proces S pobiera porcje z bufora b1 i przetwarza je na jedną porcję wstawianą do bufora b2. Proces K czeka na całkowite wypełnienie bufora b2 po czym konsumuje cały bufor na raz.
3. Napisać program odwzorowujący synchronizację samochodów pod załadunek koparką. Należy założyć istnienie n koparek i samochodów o małej i dużej ładowności. Jedna koparka obsługuje równocześnie dwa małe pojazdy lub jeden duży. Dodatkowo można przyjąć, że operatorzy koparek maja przerwę obiadową między godziną 13 a 14, a niecierpliwi kierowcy rezygnują jeśli nie podjadą na stanowisko w ciągu pół godziny.
4. Sekretarka i dyrektorzy. Ustalony zbiór N procesów producenta wstawi komunikaty do skrytki o N miejscach. Proces Pi wstawia swoje komunikaty zawsze w i-te miejsce. Pojedynczy proces konsumenta usuwa komunikaty zgadnie z algorytmem karuzelowym, co oznacza, że jeżeli skrytkę przedstawiamy w postaci bufora cyklicznego, to proces konsumenta będzie poruszał się wokół bufora zgodnie z ruchem wskazówek zegara, pobierając komunikaty z niepustych miejsc. Jeśli i-te miejsce jest puste konsument przechodzi do następnego miejsca itd.
5. Zakład fryzjerski zatrudnia m fryzjerów i n manikiurzystek. Do zakładu tego klienci przybywają pojedynczo. Każdy z nich zgłasza kierownikowi chęć ostrzyżenia się , a niekiedy zrobienia manikiuru. Jeśli nie ma miejsca w poczekalni wówczas niektórzy klienci rezygnują z usługi. Po strzyżeniu i ewentualnym zrobieniu manikiuru klient płaci i opuszcza zakład. Strzyżenie zajmuje fryzjerom różną ilość czasu, zrobienie manikiuru trwa zawsze 15 minut. Strzyżenie lub manikiur może rozpocząć się w dowolnej chwili. Na przykład robienie manikiuru klientowi (strzyżenia) może rozpocząć się przed rozpoczęciem strzyżenia (robienia manikiuru), po rozpoczęciu strzyżenia (robienia manikiuru) lub też po całkowitym zakończeniu strzeżenia (robienia manikiuru). Jednakże klient, który początkowo prosił o zrobienie manikiuru, po strzyżeniu może opuścić zakład nie domagając się całkowitego wykonania usługi. Rola kierownika zakładu polega nie tylko na przyjmowaniu zgłoszeń od klientów. Zarządza on również fryzjerami i manikiurzystkami kierując do nich klientów.
6. Jednokierunkowa droga z zachodu na wschód przecina rzekę, na której nie ma mostu. Przeprawę na drugi brzeg obsługuje prom. Obowiązują pewne zasady dotyczące kursowania promu. Prom w jedną drogę przewozi samochody, a z powrotem wraca pusty. Odpływa z lewego brzegu (zachód), gdy liczba samochodów będących już na pokładzie osiągnie pewien ustalony limit (pojemność promu) lub gdy upłynął ustalony czas oczekiwania na samochody (próg cierpliwości). W drugim przypadku prom może odpłynąć tylko wtedy, gdy na pokładzie jest przynajmniej jeden samochód. Jeśli jest nadal pusty, to czas oczekiwania liczy się od nowa. Samochody wjeżdżają i zjeżdżają z promu pojedynczo.

7. Zsynchronizować przejazd pociągów po jednotorowej linii kolejowej z mijanką, na której znajdują się dwa tory.

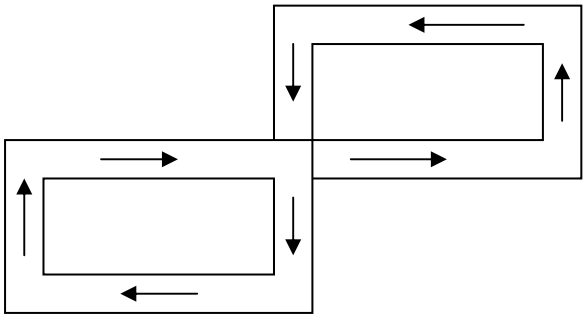


Po odcinku A-B mogą jechać jednocześnie co najwyżej dwa pociągi z tego samego kierunku lub jeden ze wschodu i jeden z zachodu. Na torach mijanki i odcinkach dojazdowych mieści się jeden skład pociągu. Z dwóch pociągów, np. wjeżdżającym w punkcie A i zjeżdżającym z mijanki w kierunku A pierwszeństwo ma pociąg z mijanki.

8. Samochody jadące z północy i południa muszą przejechać przez most na rzece. Niestety na moście jest tylko jeden pas ruchu.. Tak więc w dowolnej chwili przez most może jechać tylko jeden lub kilka samochodów podążających z tego samego kierunku (lecz nie z przeciwnych kierunków.



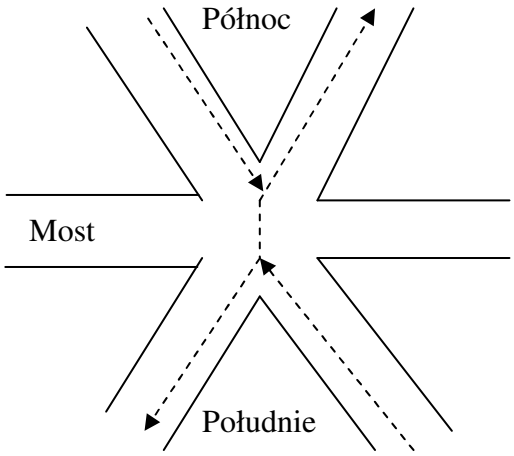
9. Po krzyżującej się trasie krążą samochody. Przejazd przez skrzyżowanie jest możliwy tylko na wprost. Zaproponuj rozwiązanie tego problemu uwzględniając bezpieczeństwo przejazdu przez skrzyżowanie i jak najkrótszy czas oczekiwania.



10. W systemie dostępnych jest N identycznych zasobów. Działają w nim dwie grupy procesów. Pierwszą stanowi N+1 procesów, które mogą wykorzystać pojedynczo dowolny z tych zasobów, drugą N procesów korzystających zawsze z tego samego zasobu. Rozwiązać problem przydziału zasobów.
11. W systemie jest M zasobów typu A oraz N zasobów typu B. Procesy działające w tym systemie należą do jednej z trzech grup:
    - Procesy żądające jednego zasobu typu A
    - Procesy żądające jednego zasobu typu B
    - Procesy żądające jednego zasobu typu A i typu B

Korzystanie z zasobu wymaga wyłącznego dostępu do niego. Zaplanować strategię przydziału tak aby nie zagłodzić żadnej z grup.

12. Samochody jadące z północy i południa muszą przejechać przez most na rzece. Niestety na moście jest tylko jeden pas ruchu.. Tak więc w dowolnej chwili przez most może jechać tylko jeden lub kilka samochodów podążających z tego samego kierunku (lecz nie z przeciwnych kierunków. Kierunek ruchu przez most powinien zmieniać się za każdym razem, gdy przekroczyło most 10 samochodów jadących z jednego kierunku, podczas gdy samochody podążające z przeciwnego kierunku oczekiwały na przejazd.



13. W Urzędzie Pocztowym są trzy kasy, które wykonują ten sam zakres świadczeń. Każda z kas w danej chwili może obsługiwać tylko jednego klienta. Do wszystkich kas jest jedna wspólna kolejka klientów oczekujących na obsłużenie. Zaproponuj rozwiązanie problemu uwzględniając, że klient który jest pierwszy w kolejce może zostać obsłużony przez dowolną kasę, przy której nie ma innego klienta.
14. Zaproponuj rozwiązanie problemu wsiadających i wysiadających pasażerów autobusu. Rozwiązanie ma uwzględniać, że tylko część pasażerów autobusu wysiada na przystanku i wysiadający z autobusu pasażerowie mają pierwszeństwo przed pasażerami wsiadającymi.