Problem producenta i konsumenta reprezentowany graficznie w czasie rzeczywistym

Systemy czasu rzeczywistego
Projekt 1

Marcin Kądziołka, Dominik Tekiela, Maksymilian Świętoń, Szymon Żur

29 czerwca 2021

1 Opis projektu

Problem producenta i konsumenta to klasyczny informatyczny problem synchronizacji. W problemie występują dwa rodzaje procesów: producent i konsument, którzy dzielą wspólny zasób – bufor – dla produkowanych jednostek. Zadaniem producenta jest wytworzenie produktu, umieszczenie go w buforze i rozpoczęcie pracy od nowa. W tym samym czasie konsument ma pobrać produkt z bufora. Pierwszy problem może wystąpić, gdy producent i konsument będą próbowali uzyskać dostęp do tego samego miejsca w pamięci. Aby to rozwiązać, należy zabezpieczyć sekcję krytyczną tak, aby tylko jeden wątek miał do niej dostęp.

Drugim problemem jest taka synchronizacja procesów, żeby producent nie dodawał nowych jednostek gdy bufor jest pełny, a konsument nie pobierał gdy bufor jest pusty. Rozwiązaniem dla producenta jest uśpienie procesu producenta w momencie, gdy bufor jest pełny. Pierwszy konsument, który pobierze element z bufora budzi proces producenta, który uzupełnia bufor. W analogiczny sposób usypiany jest konsument próbujący pobrać z pustego bufora. Pierwszy producent, po dodaniu nowego produktu umożliwi dalsze działanie konsumentowi. Rozwiązanie wykorzystuje komunikację międzyprocesową z użyciem semaforów. Nieprawidłowa implementacja powyższego algorytmu może skutkować zakleszczeniem.

Nasz projekt przedstawia rozwiązanie tego problemu w sposób graficzny, równocześnie z wykonywanym kodem. Celem jest ułatwienie zrozumienia tego problemu i możliwość analizy zachowań obu procesów. Zastosowaniem tego programu mogłoby być przedstawianie go uczniom i studentom na lekcjach dotyczących współbieżności. Pomogłoby to szybciej przyswoić koncepty synchronizacji. Program mógłby być dostarczany razem z kodem źródłowym, tak aby studenci mogli zakomentować sekcje synchronizacyjne w kodzie, w celu analizy problemów synchronizacyjnych.

2 Kod źródłowy

Najważniejszym elementem kodu źródłowego są klasy producenta i konsumenta. Przedstawione poniżej są już poprawnie zsynchronizowane i to właśnie przebieg tego kodu można śledzić w programie.

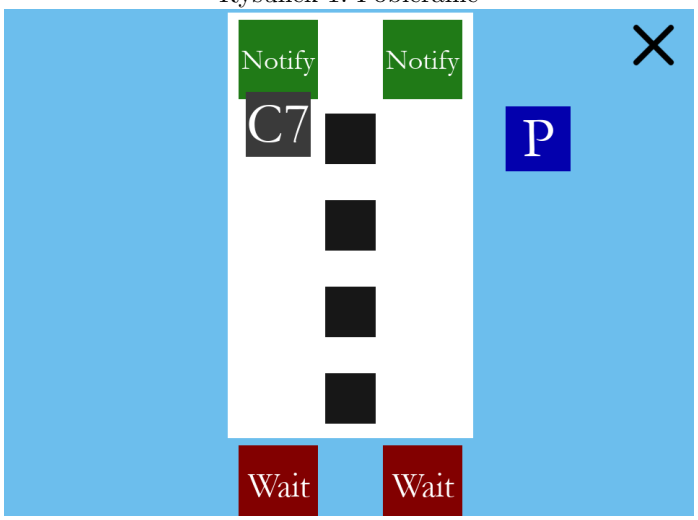
```
class ProducerThread(Thread):
    def run(self):
        global num
        global buffer

        nums = range(10)
        while True:
                # Zablokowanie dostepu innym watkom do sekcji
            condition.acquire()

            # Sprawdzenie, czy bufor jest zapelniony
            if len(buffer) == MAX_NUM:
                # Jesli jest, wstrzymanie watku,
                # az do czasu otrzymania powiadomienia od
                # konsumenta, o tym, ze pobral element
                # i mozna dodac kolejny
                condition.wait()

            # Produkowanie losowej liczby
            num = random.choice(nums)

            # Dodanie liczby do bufora
            buffer.append(num)

            # Powiadomienie konsumenta o tym,
            # ze w buforze znajduje sie produkt
            condition.notify()


            # Zwolnienie dostepu do sekcji
            condition.release()

            time.sleep(random.random())
```

```
class ConsumerThread(Thread):
    def run(self):
        global buffer
        while True:
                # Zablokowanie dostepu innym watkom do sekcji
            condition.acquire()

                # Sprawdzenie, czy bufor jest pusty
            if not buffer:
                # Jesli jest, wstrzymanie watku,
                # az do czasu otrzymania powiadomienia od
                # producenta, o tym, ze dodal element
                # i mozna go pobrac
                condition.wait()

            # Pobieranie elementu z bufora
            num = buffer.pop(0)

            # Powiadomienie producenta o tym,
            # ze w buforze znajduje sie miejsce
            condition.notify()

            # Zwolnienie dostepu do sekcji
            condition.release()

            time.sleep(random.random())
```

3 Wygląd i funkcjonalność aplikacji

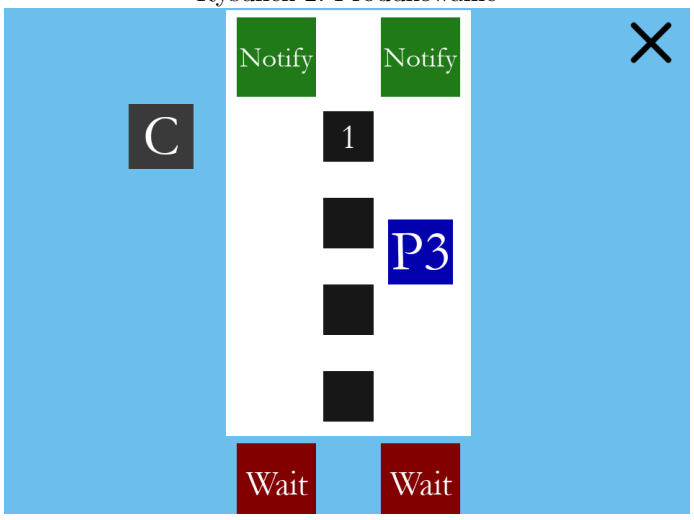
Poniżej przedstawione są zrzuty ekranu, obrazujące działanie aplikacji. Jest to kilka możliwych stanów, w których mogą znajdować się wątki producenta i konsumenta.

Rysunek 1: Pobieranie



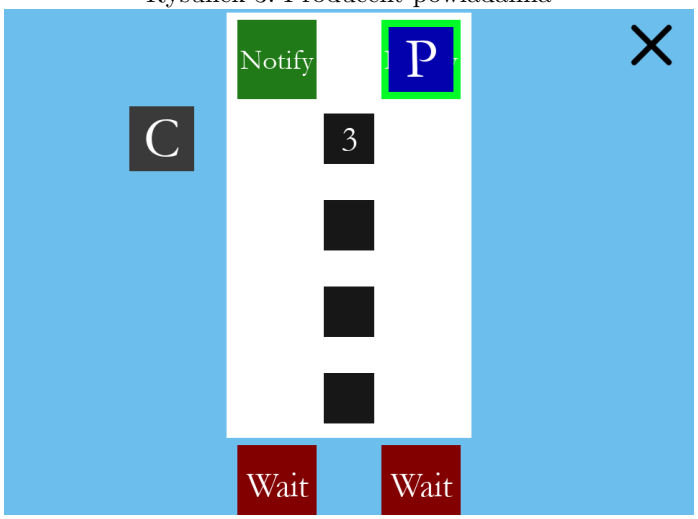
Konsument przed chwilą pobrał zawartość z bufora i udaje się do góry, aby powiadomić o tym producenta.

Rysunek 2: Produkowanie



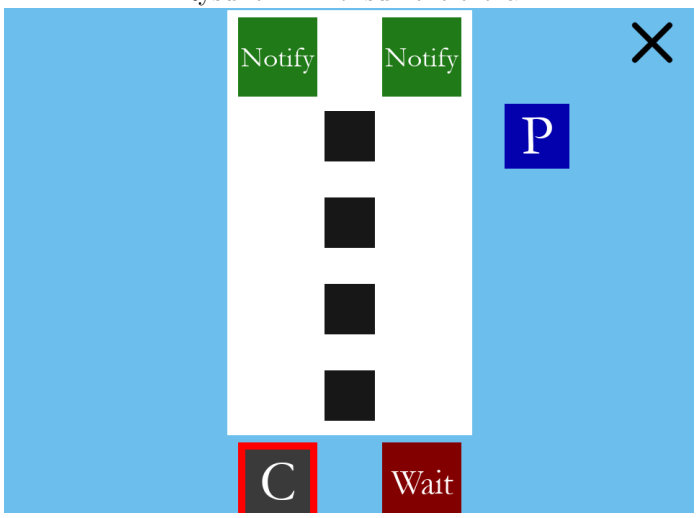
Producent znajduje się w strefie chronionej przez funkcję `condition.acquire()`, wyprodukował liczbę 3 i udaje się na miejsce drugie od góry, aby dodać ten produkt do bufora.

Rysunek 3: Producent powiadamia



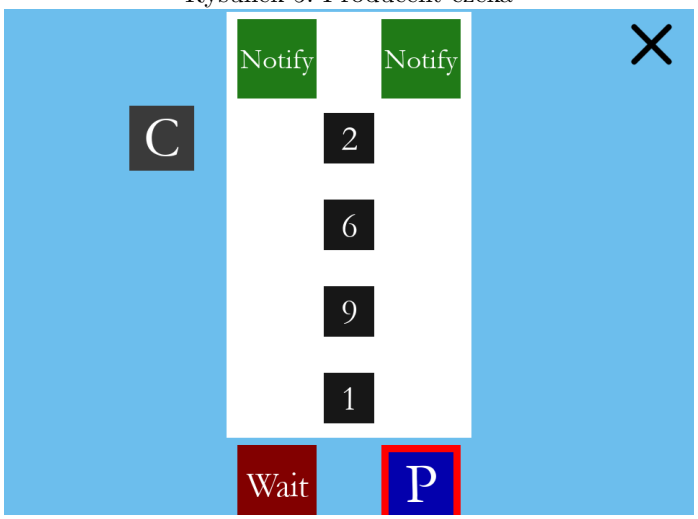
Producent dodał liczbę 3 na pierwsze miejsce bufora i powiadamia o tym konsumenta.

Rysunek 4: Konsument czeka



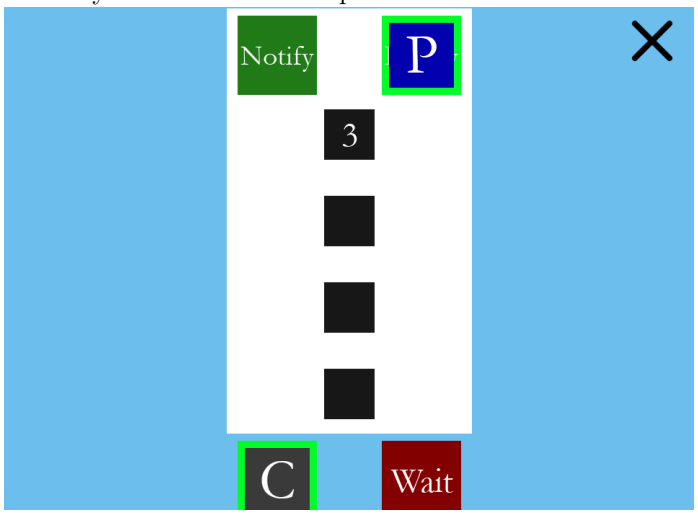
Konsument próbował pobrać produkt, ale bufor był pusty. Została uruchomiona funkcja `condition.wait()`, która zatrzyma ten wątek, aż do otrzymania powiadomienia od producenta. Funkcja ta zwalnia dostęp do zasobu, więc producent może "wejść" do bufora.

Rysunek 5: Producent czeka



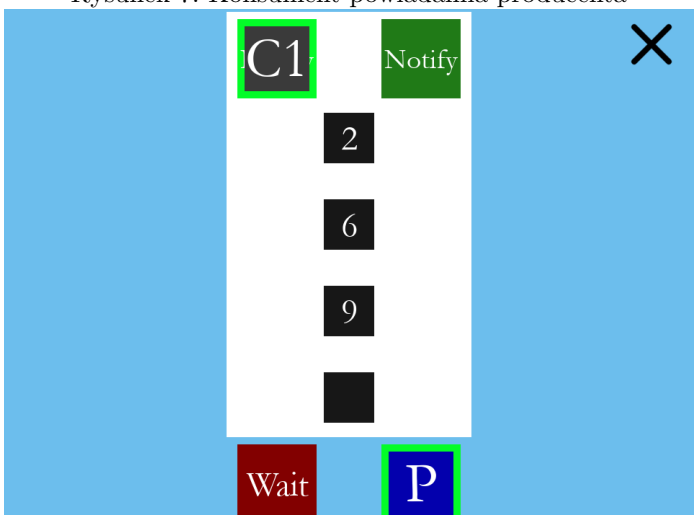
Producent chciał dodać kolejny produkt do bufora, ale ten jest pełny.Została uruchomiona funkcja `condition.wait()`, która zatrzyma ten wątek, aż do otrzymania powiadomienia od konsumenta. Funkcja ta zwalnia dostęp do zasobu, więc konsument może "wejść" do bufora.

Rysunek 6: Producent powiadamia konsumenta



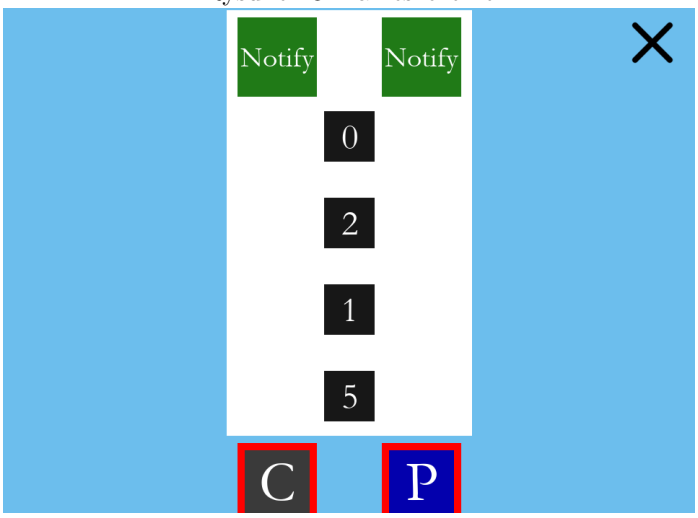
Producent właśnie dodał element do pustego bufora i powiadamia o tym konsumenta, który został zatrzymany. Teraz konsument może pobrać produkt.

Rysunek 7: Konsument powiadamia producenta



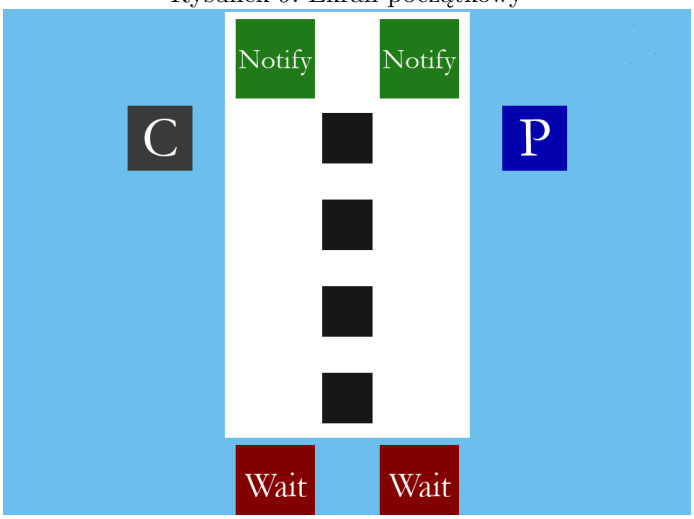
Konsument właśnie pobrał element z pełnego bufora i powiadamia o tym konsumenta, który został zatrzymany. Teraz producent może dodać produkt.

Rysunek 8: Zakleszczenie



Przed uruchomieniem tego programu zostały zakomentowane linijki kodu *# condition . notify ()* w obu klasach. Konsument chciał pobrać element, ale bufor był pusty, więc został zatrzymany. Producent dodając kolejne elementy nie powiadamiał konsumenta, więc bufor szybko się zapełnił. Przy próbie dodania do pełnego bufora on też został zatrzymany. Doszło do zakleszczenia.

Rysunek 9: Ekran początkowy



Te i inne przypadki można obejrzeć na żywo podczas wykonywania programu.

4 Użyte technologie

Program został napisany w języku Python. To dynamiczny obiektowy język programistyczny, rozprowadzany na otwartej licencji umożliwiającej zastosowanie go do zamkniętych komercyjnych projektów. Python jest aktywnie rozwijany i posiada szerokie grono użytkowników na całym świecie i jest jednym z najpopularniejszych języków według różnego rodzaju metryk mierzących ilość tworzonych projektów, bibliotek i preferencji programistów.

Dodatkowym atutem jest wiele przewodników, jak i dodatkowych bibliotek, czy narzędzi ułatwiających programowanie w tym języku.

Graficzny interfejs programu opiera się na bibliotece Pygame. To stworzona przez Pete Shinnersa biblioteka przeznaczona do tworzenia gier komputerowych oraz aplikacji multimedialnych w języku Python. Do działania wymaga biblioteki SDL, przy wykorzystaniu której dostarcza modułów pozwalających na wyświetlanie grafiki, odtwarzanie dźwięków, śledzenie czasu, obsługę myszy i joysticka, obsługę CD, czy renderowanie czcionek TTF.

Pygame jako nakładka na SDL jest wieloplatformowa i umożliwia pracę na różnych systemach operacyjnych m.in. na Windows, Linux, MacOS. Biblioteka Pygame stanowi wolne oprogramowanie i jest dystrybuowana na zasadach licencji LGPL.

5 Podsumowanie i wnioski końcowe

Aplikacja w sposób czytelny i przyswajalny prezentuje zagadnienia związane z synchronizacją i współbieżnością. Są to zagadnienia skomplikowane, jednak dzięki wizualnej reprezentacji bardziej przyswajalne.

Rozwojem programu może być wprowadzenie trybu interaktywnego, w którym użytkownik ręcznie steruje producentem lub konsumentem. Wtedy należałoby posiąść głębsze zrozumienie problemu, aby własnoręcznie sterować powiadamianiem, czekaniem i blokowaniem zasobów.

Kolejnymi krokami byłoby zrealizowanie w podobny sposób innych znanych problemów synchronizacyjnych takich jak problem pisarzy i czytelników, czy problem pięciu filozofów.