# Synchronizacja - monitory

Jakub Robaczewski

Do implementacji monitorów wykorzystam semafory z biblioteki <semaphore.h>

*Zaimplementuję klasy:*

- Message – wiadomość
- Queue – kolejka, zawiera metody odpowiedzialne za dodawanie wiadomości VIP i zwykłych i wyjmowanie, dziedziczy po klasie Monitor
- Monitor – główna klasa odpowiedzialna za system monitorów, wykorzystuje semafory
    - Warunki: NotFull, NotEmpty

*Wątki:*

- Reader – czytelnik, pobiera wiadomości
- WriterVIP – pisarz, wysyła wiadomości priorytetowe
- Writer – pisarz, wysyła wiadomości

*Pseudokod:*

```
void put_message(Message mess, int priority){
      enter(); // Wejdź do bufora

      if(Messages.size() == QueueLength)
            wait(NotFull); // Jeśli NotFull nie spełnione, czekaj na spełnienie, inny proces może wejść
            do bufora

      if (priority == 0)
            Messages.push(mess);
      else
            Messages.pushVIP(mess);

      signal(NotEmpty); // Warunek NotEmpty spełniony, obudź procesy oczekujące

      leave(); // Wyjdź z bufora
}
```

*Pseudokod 2:*

```
Message get_message() {
      enter(); // Wejdź do bufora

      if(Alarms.size() == 0)
      wait(NotEmpty); // Jeśli NotEmpty nie spełnione, czekaj na spełnienie, inny proces może wejść do
      bufora

      Message mess = Messages.front();
      Messages.pop();

      signal(NotFull); // Warunek NotFull spełniony, obudź procesy oczekujące

      leave(); // Wyjdź z bufora
      return mess;
}
```

*Testowanie:*

- Wypisywanie czasu przy stworzeniu wiadomości oraz przy czasie wpisania do kolejki
- Wypisywanie aktualnego stanu procesu: kiedy oczekuje na spełnienie warunków, kiedy wykonuje, kiedy woła inne procesy (razem z aktualnym czasem)
- Wiadomości generowane przez pisarzy są w postaci {priorytet}_{id_wiadomości}_{id_pisarza}