Daria Woźnicka *180744*
Adam Wiśniewski *180734*

Programowanie współbieżne
Sprawozdanie 3
Spotkania – wariant 5

1. Cel zadania

Celem zadania jest zapoznanie się mechanizmem spotkań służącym do synchronizacji i komunikacji zadań.

Wariant piąty zadania przedstawia problem fryzjerów i minikurzystek.

2. Rozwiązanie problemu

Program został napisany w języku C#.

Procesami symulującymi zadania wywołujące będą klienci salonu (klasa Client), natomiast procesami symulującymi zadania przyjmujące – fryzjerzy i manikurzystki (klasa Barber i klasa Manicurist). Zadania wywołujące żądają wykonania usługi – strzyżenia (zawsze) i manicure (czasem) – udostępnianym im przez fryzjerów i manikurzystki.

Po wprowadzeniu przez użytkownika liczby fryzerów i manikurzystek pracujących w salonie oraz pojemności poczekalni, tworzone są obiekty fryzjerów, manikurzystek oraz kierownika salonu, a następnie uruchamiane wątki fryzjerów i manikurzystek.

```
[Klasa Logic:]

public int liczbaFryzjerow, liczbaManikiurzystek, pojemnoscPoczekalni;

public void TakeNumbers(out int liczbaFryzjerow, out int liczbaManikiurz
ystek, out int pojemnoscPoczekalni){
            Console.WriteLine("Podaj liczbę fryzjerów: ");
            liczbaFryzjerow = int.Parse(Console.ReadLine());
            Console.WriteLine("Podaj liczbę manikiurzystek: ");
            liczbaManikiurzystek = int.Parse(Console.ReadLine());
            Console.WriteLine("Podaj pojemność poczekalni: ");
            pojemnoscPoczekalni = int.Parse(Console.ReadLine());
}
```

```
[Klasa Program:]

logic.TakeNumbers(out logic.liczbaFryzjerow, out logic.liczbaManikiurzys
tek, out logic.pojemnoscPoczekalni);
Console.WriteLine("Licbza fryzjerów: " + logic.liczbaFryzjerow);
Console.WriteLine("Liczba manikiurzystek: " + logic.liczbaManikiurzystek
);
Console.WriteLine("Pojemność poczekalni: " + logic.pojemnoscPoczekalni);
```

```
Boss szefu = new Boss(logic.liczbaFryzjerow, logic.liczbaManikiurzystek,
logic.pojemnoscPoczekalni);
```

```
[Klasa Boss:]

public Boss(int barbersCount, int manicuristsCount, int waitingRoomCapac
ity){
      for (int i = 0; i < barbersCount; i++){
          barbers.Add(new Barber(i, this));
          barbers[i].Start();
      }
      for (int i = 0; i < manicuristsCount; i++){
          manicurists.Add(new Manicurist(i, this));
          manicurists[i].Start();
      }
      waiting_room = new List<Client>(waitingRoomCapacity);
}
```

```
[Klasa BaseThread, po której dziedziczą pracownicy salonu i klienci:]

abstract class BaseThread
{
  private Thread thread;
  protected BaseThread(){
       thread = new Thread(new ThreadStart(RunThread));
  }
  public void Start(){
       thread.Start();
  }
  public abstract void RunThread();
}
```

Metoda RunThread zaimplementowana w klasie Barber i klasie Manicurist (dzięki powyższej implementcji klasy BaseThread wywoływana przy każdym uruchomieniu wątku fryzjera/manikurzystki (Start()):

```
private ManualResetEvent entryBegin = new ManualResetEvent(false);
private AutoResetEvent entryEnd = new AutoResetEvent(false);

public override void RunThread(){
     while(true){
        entryBegin.WaitOne(); //uśpienie wątku serwera, jeśli zdarzenie
entryBegin nie było sygnalizowane
        entryBegin.Reset(); //ustawia zdarzenie entryBegin jako nie sygn
alizowane
        entryEnd.Set(); //zakonczenie spotkania i odblokowanie wątku wyw
olujacego
     }
}
```

entryBegin.WaitOne() jest odpowiednikiem instrukcji accept z języka Ada – sprawia, że

wątek fryzjera/manikurzystki jest gotowe na rozpoczęcie spotkania z wątkiem wywołującym.

W naszym programie klienci wchodzą do salonu pojedynczo, co 10 minut (w skali programu – 1 sekundę).

```
[klasa Program]

System.Timers.Timer timer = new System.Timers.Timer();
  timer.Elapsed += (sender, e) => { counter++; HandleTimerElapsed(szefu,
counter); };
  timer.Interval = 1000;
  timer.Enabled = true;

  if (Console.Read() == 'q')
        timer.Enabled = false;

static void HandleTimerElapsed(Boss boss, int counter){
       CreateClient(boss, counter);
}

private static void CreateClient(Boss boss, int counter){
       Client client = new Client(counter, boss);
       if (client.manicure_wanted)
           Console.WriteLine("Przybył klient " + client.name + " na strz
yżenie i manicure");
       else
           Console.WriteLine("Przybył klient " + client.name + " tylko n
a strzyżenie");

       client.Start();
}
```

Po stworzeniu klienta jego wątek jest uruchamiany (client.Start();).

```
[klasa Client]

public override void RunThread(){
       boss.TakeClient(this);
}
```

Klient po uruchomieniu wątku żąda obslugi u kierownika salonu (boss.TakeClient(this)), która przekierowuje klienta do poczekalni. Jeśli jest w niej miejsce, klient zajmuje ostatnie miejsce w poczekalni, jeśli nie, czeka na to miejsce losową ilość czasu (od 0 do 20 minut – w skali programu do 2 sekund).

```
[klasa Boss]

public void TakeClient(Client client){
       bool wantManicure = client.manicure_wanted;
       bool entered = false;
```

```
       entered = CheckWaitingRoom(client);

       if (!entered){
          Console.WriteLine("Klient " + client.name + " wychodzi.");
          ClientLeaves(out client);
       }
       else{
          if (lastBarber < barbers.Count - 1)
               lastBarber++;
          else
               lastBarber = 0;

          barbers[lastBarber].Haircut(client);

          if(wantManicure){
             if (lastManicurist < manicurists.Count - 1)
                  lastManicurist++;
             else
                  lastManicurist = 0;
             manicurists[lastManicurist].Manicure(client);
          }
       }
}
```

Z poczekalni klient jest od razu wysyłany do odpowiedniego fryzjera (kierownik salonu wysyła klientów do każdego fryzjera po kolei), a jeśli oczekuje manicure'u, również do odpowiedniej manikurzystki.

```
[klasa Barber]
public void Haircut(Client client){
       Random rand = new Random();
       int time = rand.Next(1000, 3001);
       Console.WriteLine("Klient " + client.name + " idzie do fryzjera "
+ name);

       lock (this){
          Console.WriteLine("Fryzjer " + name + " rozpoczyna strzyzenie
(spotkanie) klienta " + client.name);
          Console.WriteLine("Strzyzenie bedzie trwac " + time / 100 + "
minut");

          if (boss.waiting_room.Contains(client))
                boss.waiting_room.Remove(client);
          entryBegin.Set(); // sygnalizacja zdarzenia entryBegin (czyli
rozpoczęcia spotkania Haircut)
         entryEnd.WaitOne(); //uśpienie wątku wywołującego do czasu zako
ńczenia spotkania
          Thread.Sleep(time); //strzyżenie trwa między 10-30 minut (1-3 s
ek)
          Console.WriteLine("Fryzjer " + name + " koniec strzyżenia (spot
kania) klienta " + client.name);
```

```
      } //opuszczenie sekcji krytycznej => następny wątek może rozpocząć
spotkanie
      client.haircut_done = true;
         if ((client.manicure_wanted && client.manicure_done) || !
client.manicure_wanted){
             Console.WriteLine("Klient " + client.name + " wychodzi.");
             boss.ClientLeaves(out client);
         }
}
```

entryBegin.Set() sygnalizuje rozpoczęcie spotkania Haircut (wątku klienta i fryzjera).
Blok lock{} jest sekcją krytyczną – tylko jeden wątek jednocześnie ma do niego dostęp, dzięki czemu fryzjer może strzyc tylko jednego klienta jednocześnie. EntryBegin.Set() sygnalizuje rozpoczęcie spotkania (accept). Po wykonaniu instrukcji zawartych w bloku lock spotkanie zostaje zakończone przez powrót do pętli wątku fryzjera (zostaje zasygnalizowany koniec spotkania przez instrukcje entryBegin.Reset() i entryEnd.set() - opisane wyżej).
Jeśli po wykonaniu strzyżenia klient nie oczekuje już na wykonanie manicure, wychodzi on z salonu, w przeciwnym wypadku jest odsyłany na spotkanie z manikurzystką.

Spotkanie z manikurzystką zostało zaimplementowane analogicznie:

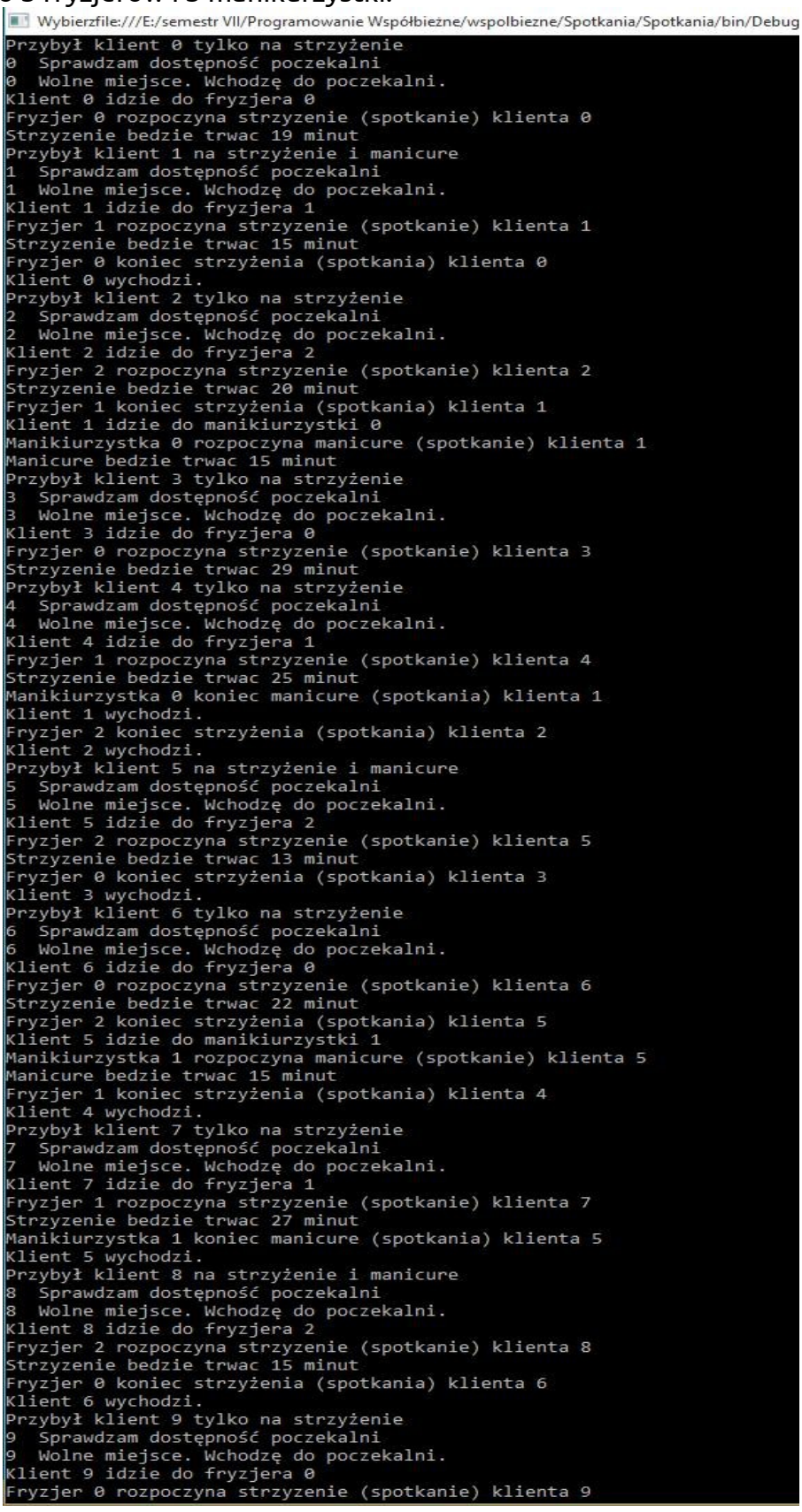
```
[klasa Manicurist]
public void Manicure(Client client){
   Console.WriteLine("Klient " + client.name + " idzie do manikiurzystki
" + name);

 lock (this){
    Console.WriteLine("Manikiurzystka " + name + " rozpoczyna manicure (
spotkanie) klienta " + client.name);
    Console.WriteLine("Manicure bedzie trwac 15 minut");

    if (boss.waiting_room.Contains(client))
           boss.waiting_room.Remove(client);
    entryBegin.Set(); // sygnalizacja zdarzenia entryBegin (czyli rozpoc
zęcia spotkania Manicure)
    entryEnd.WaitOne(); //blokada wątku wywołującego do czasu zakończeni
a spotkania
    Thread.Sleep(1500); //manicure trwa 15 minut (1.5 sek)
    Console.WriteLine("Manikiurzystka " + name + " koniec manicure (spot
kania) klienta " + client.name);
  } //opuszczenie sekcji krytycznej => następny wątek może rozpocząć spot
kanie
   client.manicure_done = true;
   if (client.haircut_done){
        Console.WriteLine("Klient " + client.name + " wychodzi.");
        boss.ClientLeaves(out client);
   }
}
```

Daria Woźnicka *180744*
Adam Wiśniewski *180734*

Zaprezentowanie działania programu:
wprowadzono 3 fryzjerów i 3 manikurzystki:



3. Wnioski

Powyższa implementacja zapewnia wzajemne wykluczanie (tylko jeden klient może być u tylko jednego fryzjera/manikurzystki) oraz brak zagłodzenia (każdy klient, który znalazł miejsce w poczekalni zostanie w końcu obsłużony). Spotkania są skutecznym sposobem synchronizacji i komunikacji procesów, jednak naszym zdaniem mniej optymalnym niż np. mechanizm monitora – klienci muszą czekać aż na spotkanie (randezvous) przybędzie “umówiony/a” fryzjer/manikurzystka (lub odwrotnie), choć mogłoby się zdarzyć że inny fryzjer/manikurzystka w międzyczasie stali się wolni. Spotkania są dobrym przykładem implementacji architektury klient-serwer – klient wysyła żądanie obsłużenia i nie musi wiedzieć co dzieje się po stronie serwera, interesuje go jedynie otrzymana odpowiedź – wysyła prośbę o obsłużenie – usypia – dostaje odpowiedź – wznawia się i kontynuuje swoje zadania (np. wychodzi z salonu). Z kolei serwer – fryzjer lub manikurzystka – pozostają w uśpieniu aż do otrzymania zgłoszenia obsługi klienta – nie muszą wiedzieć jaki klient zgłosił żądanie, wykonują jedynie operacje obsługi żądania i zwracają odpowiednią odpowiedź.