W projekcie implementujemy program zarówno ze zmiennymi warunkowymi jak również z jedynie mutexami i semaforami.

Oczekiwana ilość punktów: 34

W zadaniu:

- Stworzono 2 mutexy:

- **mutexWRoom** – Kontroluje wchodzenie i wychodzenie z poczekalni.
- **mutexSeat** – Kontroluje fotel fryzjera.

- Stworzono 2 semafory:

- **semClient** – pozwala na budzenie fryzjera.
- **SemBarber –** pozwala na zaproszenie klienta na fotel

- Stworzono 3 listy:

- **allClients** – Przechowuje wszystkich klientów.
- **resignedClients** – Przechowuje klientów, którzy zrezygnowali z usługi fryzjera.
- **clientsInWRoom** – Przechowuje klientów aktualnie przebywających w poczekalni

- Zainicjowano 9 zmiennych globalnych:

- int **maxSeatsInWRoom** = 10 – Ilość miejsc w poczekalni, może być modyfikowana opcją –s.
- int **numberOfClients** = 30 – Całkowita ilość klientów którzy odwiedzają fryzjera podczas działania programu, może być zmodyfikowana opcją –c.
- int **freeSeatsInWRoom** = 10 – Aktualna ilość wolnych miejsc w poczekalni.
- int **maxClippingTime** = 6 – Maksymalny czas strzyżenia, może być zmodyfikowana opcją –t.
- int **maxClientArrivalTime** = 30 – Maksymalny czas do przybycia ostatniego klienta, może być zmodyfikowany opcją –m.
- int **resignedCounter** = 0 – Licznik klientów którzy zrezygnowali z usługi fryzjera.
- int **clientOnSeat** = -1 – Numer aktualnie obsługiwanego klienta, -1 oznacza że fryzjer śpi.
- bool **bDebug** = false – Flaga oznaczająca że program ma być uruchomiony w trybie debug, może być uruchomiona opcją –d.
- bool **finished** = false – Flaga mówiąca fryzjerowi czy pozostali jeszcze jacyś klienci.

Wątek klienta:

void *Client(void *cNumber)

***cNumber** – Dostarcza funkcji numer klienta w wątku.

Wątek zaczyna od przeczekania losowej ilości czasu, następnie blokuje mutex „mutexWRoom” w celu zdecydowania wejścia do poczekalni, jeśli poczekalnia jest pełna to klient rezygnuje i raportuje aktualny stan zakładu.

```
if (freeSeatsInWRoom <= 0)
  {
    resignedCounter++;
    printf("Res:%d WRoom: %d/%d [in: %d]\n", resignedCounter, maxSeatsInWRoom - freeSeatsInWRoom, maxSeatsInWRoom,
clientOnSeat);
    pthread_mutex_unlock(&mutexWRoom);
    if (bDebug == true)
    {
      Append(&resignedClients, clientNumber, 0);
```

```
        Print(resignedClients, clientsInWRoom);
    }
  }
```

Jeśli klient zdecyduje się wejść do poczekalni dołącza on do listy poczekalni i informuje fryzjera o swoim przybyciu. Następnie odblokowuje „mutexWRoom" i czeka na zaproszenie od fryzjera.

```
freeSeatsInWRoom--;
if (bDebug == true)
{
 Append(&clientsInWRoom, clientNumber, 0);
}
printf("Res:%d WRoom: %d/%d [in: %d]\n", resignedCounter, maxSeatsInWRoom - freeSeatsInWRoom, maxSeatsInWRoom,
clientOnSeat);
if (bDebug == true)
{
  Print(resignedClients, clientsInWRoom);
}
   sem_post(&semClient);
pthread_mutex_unlock(&mutexWRoom);1
sem_wait(&semBarber);
```

Kiedy nadchodzi kolej klienta blokuje on mutex „mutexSeat" w celu powiadomienia o tym że już ktoś idzie skorzystać z usługi klienta.

```
pthread_mutex_lock(&mutexSeat);
if (bDebug == true)
{
Remove(&clientsInWRoom, clientNumber);
}
clientOnSeat = clientNumber;
```

Wątek fryzjera:

void *Barber()

Fryzjer na początku przygotowuje się do pracy poprzez zadeklarowanie zmiennej „clippingTime" przechowującej czas strzyżenia aktualnego klienta. Po przygotowaniu do pracy wchodzi on w pętlę, która kończy się po przejściu wszystkich klientów. W pętli fryzjer zaczyna od czekania na informacje semafora „semClient" w tym czasie śpi a semafor informuje o przyjściu klienta, następnie zablokowuje „mutexWRoom" i zabiera klienta z poczekalni, po czym odblokowuje „mutexWRoom". Następnie strzyże on klienta przez losowy czas, wypisuje raport o stanie zakładu, oznacza fotel do strzyżenia jako wolny i na koniec odblokowuje „mutexSeat".

```
int clippingTime;
while (finished == false),
{
sem_wait(&semClient);
pthread_mutex_lock(&mutexWRoom);
sem_post(&semBarber);
freeSeatsInWRoom++;
pthread_mutex_unlock(&mutexWRoom);
clippingTime = rand() % maxClippingTime + 1;
sleep(clippingTime);
printf("Res:%d WRoom: %d/%d [in: %d]\n", resignedCounter, maxSeatsInWRoom - freeSeatsInWRoom, maxSeatsInWRoom,
clientOnSeat);
if (bDebug == true)
{
Print(resignedClients, clientsInWRoom);
}
clientOnSeat = -1;
pthread_mutex_unlock(&mutexSeat);
}
```