Opis projektu - Basen nr.8

Celem projektu było stworzenie symulacji systemu zarządzania kompleksem basenów. Symulacja uwzględnia regulamin korzystania z basenów, różne typy użytkowników (dzieci, dorośli, VIP) oraz obsługę sygnałów ratowników.

Projekt podzieliłam na kilka głównych modułów.

- **zarządca.c** - Tworzy kolejki komunikatów i pamięć współdzieloną, zbiera dane od użytkownika, inicjalizuje procesy kasjera, ratowników i klientów, a po zakończeniu pracy usuwa kolejki oraz pamięć współdzieloną.
- **klient.c** - Obsługuje procesy klientów. Każdy proces zawiera losowo wygenerowane dane klienta, komunikuje się z kasjerem w celu wejścia do kompleksu, a następnie z ratownikiem w sprawie korzystania z konkretnego basenu. Klient odbiera także sygnały od ratowników.
- **kasjer.c** - Obsługuje proces kasjera, który zarządza czasem wejścia i wyjścia klientów z kompleksu.
- **ratownik.**c - Obsługuje procesy ratowników. Każdy ratownik posiada trzy wątki: wpuszczania klientów do basenu, wypuszczania ich oraz wysyłania sygnałów do klientów.

Opisy funkcjonalności

Rozmiary basenów X1, X2, X3
Po uruchomieniu programu użytkownik zostaje poproszony o wprowadzenie rozmiarów trzech typów basenów: X1 (basen olimpijski), X2 (basen rekreacyjny) oraz X3 (brodzik). Program weryfikuje, czy wprowadzone wartości są liczbami dodatnimi. Dodatkowo, w przypadku brodzika sprawdzane jest, czy jego rozmiar jest liczbą parzystą.

W moim programie klient składający się z dziecka i opiekuna jest traktowany jako dwie osoby w basenie, mimo że są oni jednym procesem. Ze względu na to zdecydowałam się na rozwiązanie zapewniające, że w brodziku nigdy nie zostaną puste miejsca. Na przykład, jeśli rozmiar brodzika wynosi 5, to przy obecności 4 osób (czyli dwóch klientów – dziecka i opiekuna), kolejny klient nie mógłby wejść, co skutkowałoby jednym niewykorzystanym miejscem.

Zmiana warunków pozwoliłaby na działanie programu przy nieparzystych rozmiarach brodzika, ale postanowiłam zagwarantować, że jego rozmiar zawsze będzie parzysty, co eliminuje takie sytuacje.

Czas otwarcia kompleksu Tp - Tk
Użytkownik zostaje poproszony o podanie długości czasu otwarcia basenu w sekundach. Wartość ta musi mieścić się w przedziale od 60 do 28 800 sekund (czyli od 1 minuty do 8 godzin). Symulacja została przystosowana do działania na sekundach.

Walidacja maksymalnego czasu działania kompleksu nie ma istotnego wpływu na działanie programu, ale została wprowadzona jako dodatkowe zabezpieczenie. Z kolei minimalny czas (60 sekund) gwarantuje, że symulacja będzie działała przynajmniej przez chwilę, co pozwala na zobaczenie jej pełnej działalności.

Na podstawie podanego czasu określane są momenty otwarcia i zamknięcia basenu. Zmienna Tp to bieżący czas, a Tk to czas zamknięcia, obliczany jest jako suma aktualnego czasu (Tp) i długości otwarcia podanej przez użytkownika.

Wyświetlanie i walidacja czasu
Do obsługi czasu wykorzystałam bibliotekę time.h. Napisałam także funkcję, która zwraca aktualny czas zapisany w strukturze tm. Dzięki temu mogę uzyskać szczegółowe informacje o bieżącej dacie i godzinie w łatwym do przetwarzania formacie.

```
struct tm *local;    // Wskaźnik do wyświetlania obecnego czasu
```

```
local = czas();
printf("%s[%02d:%02d:%02d  %d]%s Klient wchodzi na kompleks basenowy.\n",
 GREEN, local->tm_hour, local->tm_min, local->tm_sec, klient.pid, RESET);
```

```
struct tm* czas() {
    static time_t now;
    static struct tm local_time;

    time(&now);
    local_time = *localtime(&now);

    return &local_time;
}
```

Generowanie kasjera i ratowników
W procesie zarządcy generowane są procesy kasjera i trzech ratowników. Każdy ratownik otrzymuje w argumentach numer basenu, którym się zajmuje oraz jego rozmiar.

Generowanie klientów
Klienci są tworzeni w losowych odstępach czasu, pod warunkiem że nie został przekroczony czas zamknięcia kompleksu ani maksymalna liczba klientów. Proces zarządcy śledzi liczbę aktywnych procesów i oczekuje ich zakończenia po zamknięciu kompleksu.

Każdy klient losuje czy posiada status VIP. Ponieważ status ten ma znaczenie wyłącznie podczas komunikacji z kasjerem zdecydowałam się nie dodawać go jako pola do struktury klienta.
Wiek klienta jest losowany w zakresie od 1 do 70 lat. Aby zaimplementować mechanizm dziecka i opiekuna jako jednego procesu:

- Jeśli klient ma mniej niż 10 lat losowany jest wiek opiekuna w zakresie od 18 do 70 lat.
- Jeśli klient ma 10 lat lub więcej wiek opiekuna ustawiany jest na 0. Wartość ta jest wykorzystywana w późniejszych instrukcjach warunkowych do rozróżnienia, czy dany klient ma opiekuna.

Po wejściu na teren kompleksu klient losuje basen, do którego chce wejść.

- Jeśli ratownik pozwoli klientowi wejść na wybrany basen, jego numer zostaje przypisany do struktury klienta.
- Jeśli klient nie zostanie wpuszczony, numer basenu pozostaje równy 0, co oznacza, że klient znajduje się wyłącznie na terenie kompleksu, ale nie korzysta z żadnego basenu. Po odczekaniu chwili może ponownie wybrać basen.

Klienci VIP

Kolejkę do kas zaimplementowałam za pomocą kolejki komunikatów. Każdy klient przesyła do kasjera komunikat, którego typ wskazuje jego priorytet:

- 1 – klient VIP,
- 2 – zwykły klient

Do odbierania komunikatów skorzystałam z funkcji msgrcv, która umożliwia pobieranie komunikatów w zależności od podanego argumentu msgtype. Ustawiłam typ odbieranych komunikatów na -2. Dzięki tej właściwości funkcja priorytetowo odbiera komunikaty o mniejszych wartościach mtype.

```
msgrcv(msgID, &msg, sizeof(msg) - sizeof(long), -2, 0),
```

msgtype =0 pobierany pierwszy dostępny komunikat;
>0 (np.11) pobierany pierwszy dostępny komunikat danego typu;
<0 (np.-6) pobierany pierwszy dostępny komunikat, którego typ ma wartość taką samą lub mniejszą niż absolutna wartość msgtype;

Dzięki zastosowaniu msgtype = -2, kolejka działa w sposób, który gwarantuje, że komunikaty klientów o typie 1 (VIP) zawsze mają pierwszeństwo przed komunikatami o typie 2 (zwykli klienci).
Nawet jeśli w kolejce znajduje się wielu zwykłych klientów, kasjer obsługuje najpierw wszystkich klientów VIP, a dopiero później zwykłych klientów.

Kupowanie biletów czasowych

W programie wszystkie bilety mają ten sam czas ważności, można go zmieniać ale domyślnie kasjer ustawia czas wyjścia klienta jako obecny czas + 20 sekund.
Po obliczeniu domyślnego czasu wyjścia kasjer sprawdza, czy nie przekracza on ustalonego czasu zamknięcia kompleksu. Jeśli czas wyjścia jest większy niż czas zamknięcia, zostaje on ustawiony na czas zamknięcia - 5 sekund. Dzięki temu nie ma sytuacji w której klient pozostaje na terenie basenu po jego zamknięciu.

```
msg.czas_wyjscia = time(NULL) + 20;

if (msg.czas_wyjscia > zamkniecie){
    msg.czas_wyjscia = zamkniecie - 5;
}
```

Przechowywanie klientów w basenach

Każdy basen w programie ma swoją dedykowaną tablicę, która przechowuje informacje o klientach, a także dodatkowe pole do śledzenia liczby osób obecnych w danym basenie. Ponieważ baseny mają różne rozmiary, zastosowałam różne struktury tablic dla każdego z nich.
Basen olimpijski X1 działa na tablicy jednowymiarowej o rozmiarze [X1 + 1].

Basen rekreacyjny X2 działa na tablicy dwuwymiarowej o rozmiarze [2][X2 + 1]. Pierwszy wiersz tej tablicy przechowuje pidy klientów, a drugi ich wiek, który jest wykorzystywany do liczenia średniej.
Brodzik X3 działa na tablicy jednowymiarowej o rozmiarze [X3 / 2 + 1]. Ponieważ każdy klient wchodzący do brodzika to dwie osoby (dziecko + opiekun) to wystarczy, że tablica będzie miała połowę rozmiaru basenu + miejsce na liczbę klientów w basenie.

Wpuszczanie do basenów
Na początku programu generowani są trzej ratownicy, z których każdy obsługuje inny basen (olimpijski, rekreacyjny, brodzik). Każdy ratownik ma przypisane trzy wątki: wpuszczanie klientów, wypuszczanie klientów, wysyłanie sygnałów. Do zarządzania wpuszczaniem i wypuszczaniem klientów wykorzystałam kolejkę komunikatów. Komunikaty w tej kolejce składają się z pól mtype, pid, wiek, wiek_opiekuna, kom.

Po odebraniu komunikatu ratownik sprawdza czy klient spełnia warunki żeby wejść do obsługiwanego basenu. Jeśli warunki są spełnione, ratownik wysyła do klienta komunikat msgr.kom = 't' co oznacza że klient jest wpuszczony do basenu, i został dodany do odpowiedniej tablicy w basenie.

Jeśli warunki nie są spełnione ratownik wysyła odpowiedni komunikat informując klienta o problemie:
c - Basen jest tymczasowo nieczynny (przez sygnały)
w - Klient jest za młody lub za stary żeby kąpać się w wybranym basenie
n - Nie ma miejsca w basenie
s - Średnia wieku w basenie zostałaby przekroczona (tylko w basenie rekreacyjnym)

W każdej tablicy klientów znajduje się informacja o liczbie klientów. Dla każdego basenu dodawana jest wartość 1 lub 2 do zerowego indeksu tablicy zależnie od tego, czy klient jest sam czy z opiekunem.

Oczywiście w trakcie pracy na tablicach basenu jest blokowany mutex tego basenu, a po skończonej pracy jest odblokowywany.

Średnia wieku w basenie rekreacyjnym
Aby zapewnić, że średnia wieku w basenie rekreacyjnym nie przekroczy 40 lat ratownik oblicza teoretyczną średnią wieku po możliwym wejściu klienta do basenu. Jest to istotne, ponieważ istnieje szansa że nawet jeżeli przed wejściem klienta średnia w basenie przekracza 40 lat to może się okazać że po jego wejściu już będzie mniejsza. Żeby to zapewnić ratownik przechodzi przez kilka kroków.

1. Ratownik sumuje wiek wszystkich klientów, którzy już znajdują się w basenie.
2. Do tej sumy dodaje wiek klienta, który ma zostać wpuszczony do basenu.
3. Jeśli klient jest dzieckiem i przychodzi z opiekunem ratownik dodaje wiek opiekuna do sumy i zwiększa liczbę klientów chcących wejść do basenu.
4. Ratownik oblicza średnią za pomocą wzoru:

(suma wieku w basenie + wiek klienta) / (liczba klientów w basenie + osoby wchodzące)

```
// Suma wieków klientów obecnie przebywających w basenie
for(int i = 1; i <= pool_size; i++){
    //printf("%d. %d\n", i ,klienci[1][i]);
    srednia += klienci[1][i];
}

// Jeżeli klient (dziecko) ma opiekuna to dodaje jego wie
if(msgr.wiek_opiekuna > 0){
    wiek_klienta = msgr.wiek + msgr.wiek_opiekuna;
    liczba_osob = 1;
}

// Sumuje liczbe klientów w basenie dodaje klienta i opie
int liczba_klientow = klienci[0][0] + liczba_osob + 1;

// Oblicza średnią z poprzedniej sumy, zsumowanego wieku
// Sprawdza czy po wejściu danego klienta do basenu średn
srednia = (srednia + wiek_klienta) / liczba_klientow;
//printf("srednia: %f\n", srednia); // Do sprawdzenia
```

Dzięki temu ratownik uwzględnia potencjalne zmiany średniej po wejściu klienta do basenu.

Wychodzenie z basenów

W procesie wychodzenia klientów z basenów jedynie klient wysyła komunikat do ratownika, ratownik nie zwraca odpowiedzi do klienta.

Po otrzymaniu komunikatu ratownik szuka klienta w tablicy, ustawia wartość znalezionego pola na 0 i zmniejsza liczbę klientów w basenie o 1 lub 2 w zależności od tego czy klient jest sam czy z opiekunem. W trakcie operacji na tablicy klientów mutex dla danego basenu jest blokowany, aby uniknąć konfliktów w przypadku jednoczesnego dostępu do danych.

Na początku miałam jedną wspólną funkcję do obsługi wychodzenia klientów, która na podstawie pool_id wykonywała odpowiednie operacje. Wszystko działało poprawnie, dopóki nie dodałam drugiego wątku. Wtedy pojawiły się problemy z poprawnym wyświetlaniem tablic i operacjami na nich choć przy jednym wątku wszystko działało bez zarzutu. Mimo że funkcja obsługująca sygnały działa w podobny sposób jak docelowo planowałam działanie tego wątku to nie ma ona tych samych problemów. Nie mogłam znaleźć przyczyny tego zachowania więc postanowiłam podzielić ten kod na trzy osobne funkcję, co rozwiązało problem.

Sygnały

Obsługa sygnałów jest rozdzielona między dwa pliki: ratownik.c i klient.c. Każdy ratownik ma osobny wątek odpowiedzialny za wysyłanie sygnałów. Wątki te uruchamiają się na początku działania procesów ratowników.
W funkcji sygnal() obsługiwane są wszytskie trzy basey. Na początku sprawdzam, który basen jest aktualnie obsługiwany w danym procesie, a następnie przypisuję odpowiedni mutex do wskaźnika i ustalam jego rozmiar w zmiennej.

Czasy wysyłania sygnałów są losowane w zakresie długości otwarcia kompleksu. W tym celu korzystam ze zmiennej dlugosc_otwarcia, zapisanej w pamięci współdzielonej i ustawianej przez użytkownika na początku działania programu.

```
time_t send_signal1 = shared_data->otwarcie + rand() % (shared_data->dlugosc_otwarcia / 4) + 10;
time_t send_signal2 = send_signal1 + rand() % (shared_data->dlugosc_otwarcia / 4) + 5;
```

Sygnał 1 zostanie wysłany między 10 sekundą działania programu a maksymalnie ¼ długości otwarcia kompleksu.
Sygnał 2 zostanie wysłany co najmniej 5 sekund po send_signal1, ale maksymalnie ¼ długości otwarcia kompleksu po jego wysłaniu.
Dzięki temu sygnały zawsze mieszczą się w zakresie godzin otwarcia kompleksu.

Funkcja działa w pętli while i czeka na moment wysłania sygnału 1 (send_signal1). Gdy nadejdzie czas:

1. Blokuje mutex dla danego basenu.
2. Ustawia status basenu na nieczynny.
3. Czyści tablicę basenu i przenosi PID-y klientów do tablicy pomocniczej.
4. Wysyła sygnał SIGUSR1 do klientów z tej tablicy za pomocą funkcji kill.

Po wysłaniu pierwszego sygnału czeka na czas wysłania drugiego sygnału (send_signal2)
Po nadejściu czasu:

1. Zmienia status basenu na czynny.
2. Wysyła sygnały SIGUSR2 do klientów z tablicy pomocniczej, również za pomocą funkcji kill.

W pliku klient.c wykorzystuję funkcję sigaction, aby proces mógł obsługiwać sygnały w dowolnym momencie działania.

```
struct sigaction sa;
sa.sa_handler = signal_handler;
sa.sa_flags = 0;
sigemptyset(&sa.sa_mask);
sigaction(SIGUSR1, &sa, NULL);
sigaction(SIGUSR2, &sa, NULL);
```

Po otrzymaniu SIGUSR1 klient oznacza obecny basen jako niedostępny, usuwa go z listy możliwych opcji losowania i ustawia swój basen na 0. Dzięki temu może wylosować nowy basen, ale nie ten z którego właśnie został wyrzucony.
Po otrzymaniu SIGUSR2, klient resetuje ograniczenia, co pozwala mu ponownie wylosować ten sam basen i wejść do niego.

Obsługa błędów
Aby sprawniej obsługiwać błędy napisałam dwie funkcje sprawdz_blad oraz sprawdz_blad_watek.
Funkcja sprawdz_blad jest przeznaczona do obsługi błędów w operacjach, które w przypadku niepowodzenia zwracają wartość -1. Jeśli wartość argumentu funkcji wynosi -1:

1. Wyświetla opis błędu za pomocą funkcji perror().

2. Wyświetla numer błędu przechowywany w zmiennej globalnej errno.
3. Kończy działanie programu.

Funkcja sprawdz_blad_watek obsługuje błędy związane z wątkami. Funkcje te nie zwracają wartości -1 i nie ustawiają zmiennej errno, dlatego napisałam osobną funkcję do ich obsługi. Jeśli argument funkcji różni się od zera:

1. Wyświetla opis błędu oraz jego numer.
2. Kończy działanie programu.

Okresowe zamykanie kompleksu

Zdecydowałam się nie implementować tej funkcjonalności. Początkowo planowałam ją zrealizować, wykorzystując zmienną dlugosc_otwarcia z pamięci współdzielonej. Kasjer miał w połowie czasu otwarcia kompleksu wyznaczyć moment okresowego zamknięcia, wtedy podnieść semafor lub zmienić flagę w pamięci współdzielonej, a jednocześnie wstrzymać przyjmowanie nowych klientów. Proces zarządcy miał czekać na zakończenie obsługi obecnych klientów, ale jednocześnie nadal generować nowych klientów do kolejki. Zostawiłam sobie tę funkcjonalność na koniec realizacji projektu. Niestety, podczas prób implementacji napotkałam błędy i w związku z ograniczonym czasem podjęłam decyzję, że całkowicie z niej zrezygnuję.

Testy

1. Input użytkownika

W tym teście chciałam sprawdzić czy podawanie rozmiarów basenów i długości otwarcia jest zabezpieczone przed wprowadzaniem niedozwolonych wartości.

```
Podaj rozmiary basenów. Wpisz liczbę a następnie kliknij ENTER.
[X1]: test
Podano błędną wartość. Rozmiar basenu musi być liczbą całkowitą.
[X1]: 4.5
Podano błędną wartość. Rozmiar basenu musi być liczbą całkowitą.
[X1]: -4
Podano błędną wartość. Rozmiar basenu musi być liczbą całkowitą.

[X1]: 4
Zapisano dane ;)
[X2]: test
Podano błędną wartość. Rozmiar basenu musi być liczbą całkowitą.
[X2]: 4.5
Podano błędną wartość. Rozmiar basenu musi być liczbą całkowitą.
[X2]: -2
Podano błędną wartość. Rozmiar basenu musi być liczbą całkowitą.

[X2]: 7
Zapisano dane ;)
[X3]: test
Podano błędną wartość. Rozmiar basenu musi być liczbą całkowitą.
[X3]: 4.5
Podano błędną wartość. Rozmiar basenu musi być liczbą całkowitą.
[X3]: -4
Podano błędną wartość. Rozmiar basenu musi być liczbą całkowitą.

[X3]: 3
Podano błędną wartość. Rozmiar basenu nr.3 musi być liczbą całkowitą parzystą.

[X3]: 4
Zapisano dane ;)
```

```
Podaj dlugosc otwarcia basenu w sekundach (od 60 sek. do 8 godzin) a następnie kliknij ENTER.
Długość otwarcia: test
Podano błędną wartość. Długość otwarcia musi być liczbą całkowitą większą bądź równą 60 sekund
i mniejszą od 8h.
Długość otwarcia: 59
Podano błędną wartość. Długość otwarcia musi być liczbą całkowitą większą bądź równą 60 sekund
i mniejszą od 8h.

Długość otwarcia: 4.5
Podano błędną wartość. Długość otwarcia musi być liczbą całkowitą większą bądź równą 60 sekund
i mniejszą od 8h.
Długość otwarcia: -3
Podano błędną wartość. Długość otwarcia musi być liczbą całkowitą większą bądź równą 60 sekund
i mniejszą od 8h.

Długość otwarcia: 99999999
Podano błędną wartość. Długość otwarcia musi być liczbą całkowitą większą bądź równą 60 sekund
i mniejszą od 8h.

Długość otwarcia: 60
Zapisano dane ;)
```

Jak widać pobieranie wartości od użytkownika jest przygotowane na wprowadzanie błędnych wartości i działa poprawnie.

2. Kolejka VIP

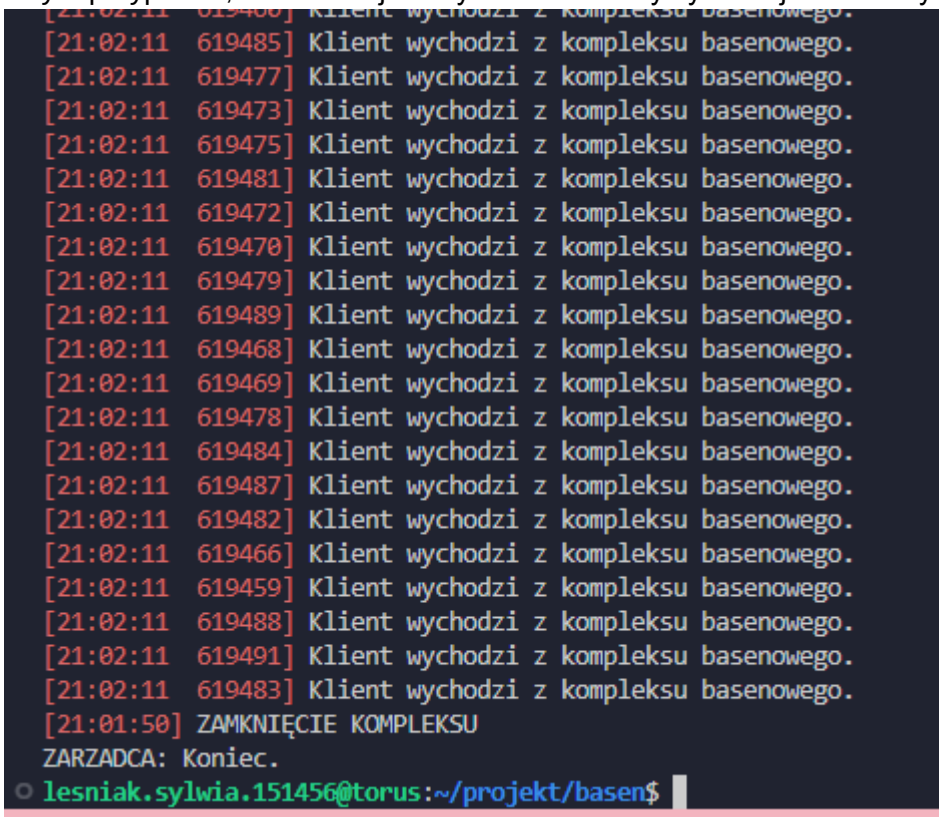
W założeniach projektu klienci VIP powinni być obsługiwani przez kasjera bez konieczności czekania w kolejce. Aby przetestować tę funkcjonalność, w pliku kasjer.c dodałam funkcję sleep(10). Gdybym tego nie zrobiła to klienci byliby na tyle szybko obsługiwani, że ciężko byłoby sprawdzić czy faktycznie klienci VIP są obsługiwani pierwsi.

```
[20:37:35  273873] Pojawia się klient.
[20:37:44 1] Wyslanie sygnalu 1.
[20:37:58 1] Wyslanie sygnalu 2.
[20:37:45 2] Wyslanie sygnalu 1.
[20:37:48 2] Wyslanie sygnalu 2.
[20:37:44 3] Wyslanie sygnalu 1.
[20:37:53 3] Wyslanie sygnalu 2.
[20:37:35  273874] Pojawia się klient.
[20:37:35  273875] Pojawia się klient VIP.
[20:37:35  273876] Pojawia się klient.
[20:37:35  273877] Pojawia się klient.
[20:37:35  273878] Pojawia się klient.
[20:37:35  273879] Pojawia się klient.
[20:37:35  273881] Pojawia się klient.
[20:37:35  273887] Pojawia się klient.
[20:37:35  273891] Pojawia się klient.
[20:37:35  273892] Pojawia się klient VIP.
[20:37:44  1] RATOWNIK WYSYŁA SYGNAŁ NA WYJŚCIE Z BASENU NR.1.

Stan tablicy klienci po wysłaniu sygnału 1 OLIMPIJSKI:
Liczba klientów na basenie: 0
Miejsce 1: PUSTE

[20:37:44  3] RATOWNIK WYSYŁA SYGNAŁ NA WYJŚCIE Z BASENU NR.3.

Stan tablicy klienci po wysłaniu sygnału 1 BRODZIK:
Liczba klientów na basenie: 0
Miejsce 1: PUSTE

[20:37:45  273875] Kasjer obsługuje klienta VIP.
[20:37:45  273875] Klient płaci za bilet.
[20:37:45  273892] Kasjer obsługuje klienta VIP.
[20:37:45  273892] Opiekun płaci za bilet. Dziecko nie płaci za bilet. Wiek: 5
[20:37:45  273873] Kasjer obsługuje klienta.
[20:37:45  273873] Klient płaci za bilet.
[20:37:45  273874] Kasjer obsługuje klienta.
```

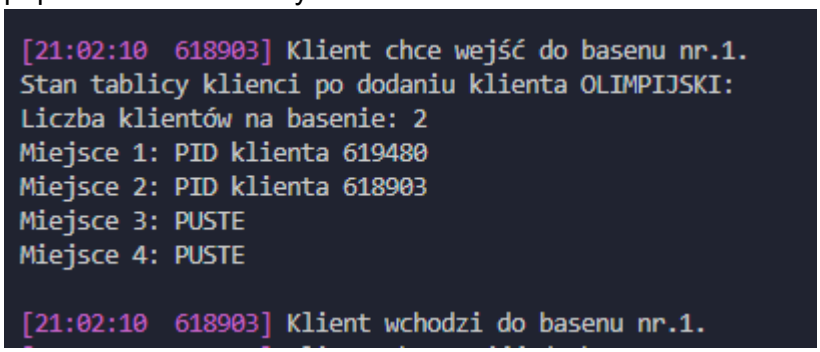
Podczas testu pierwszym klientem obsłużonym przez kasjera był klient VIP, mimo że przed nim w kolejce znajdowało się jeszcze dwóch zwykłych klientów. Jako drugi został obsłużony kolejny klient VIP, który pojawił się jako ostatni, a dopiero potem kasjer zaczął obsługę normalnych klientów w kolejności w jakiej się pojawili.
Warto zaznaczyć, że gdy pojawi się kilku klientów VIP w tym samym czasie, wciąż muszą oni czekać w kolejce razem z innymi VIP-ami. Problem ten jest trudny do rozwiązania przy jednym działającym kasjerze, ponieważ obsługuje on tylko jednego klienta na raz. Z tego powodu, gdy pojawi się więcej niż jeden klient VIP, któryś z nich musi zostać obsłużony później.

3. Generowanie 5000 klientów na raz

Celem tego testu było sprawdzenie, czy symulacja zakończy się pomyślnie przy generowaniu 5000 klientów jednocześnie. Trudno mi było prześledzić komunikaty w konsoli w tym przypadku, ale ważniejsze było dla mnie czy symulacja zakończy się powodzeniem.



Symulacja zakończyła się powodzeniem, w trakcie klienci byli poprawnie obsługiwani i poprawie wchodzili i wychodzili z basenów.



```
[21:02:11  618903] Klient wychodzi z kompleksu i basenu nr.1.

Stan tablicy klienci po usunięciu klienta OLIMPIJSKI:
[21:02:11  618928] Klient wychodzi z kompleksu basenowego.
Liczba klientów na basenie: 2
Miejsce 1: PUSTE
Miejsce 2: PUSTE
Miejsce 3: PID klienta 618954
Miejsce 4: PID klienta 618963
```

4. Wchodzenie i wychodzenie klientów do basenów

W normalnym trybie symulacji klienci są generowani w losowych odstępach czasu i próbują wejść do losowych basenów. Na potrzeby testów poszczególnych basenów zmieniłam w kodzie przypisanie basenu na konkretny, który chcę przetestować.

W tej sekcji skupię się wyłącznie na testowaniu dodawania i usuwania klientów z basenów. Warunki wpuszczania klientów do basenów oraz proces zapełniania się basenów będę sprawdzać w kolejnej sekcji.

Basen olimpijski

```
[21:44:31  656050] Klient chce wejść do basenu nr.1.

Stan tablicy klienci po dodaniu klienta OLIMPIJSKI:
Liczba klientów na basenie: 3
Miejsce 1: PID klienta 656009
Miejsce 2: PID klienta 656003
Miejsce 3: PID klienta 656050
Miejsce 4: PUSTE
Miejsce 5: PUSTE
Miejsce 6: PUSTE

[21:44:31  656050] Klient wchodzi do basenu nr.1.
[21:44:32  656003] Klient wychodzi z kompleksu i basenu nr.1.

Stan tablicy klienci po usunięciu klienta OLIMPIJSKI:
Liczba klientów na basenie: 2
Miejsce 1: PID klienta 656009
Miejsce 2: PUSTE
Miejsce 3: PID klienta 656050
Miejsce 4: PUSTE
Miejsce 5: PUSTE
Miejsce 6: PUSTE
```

Basen rekreacyjny

```
Stan tablicy klienci po dodaniu klienta REKREACYJNY:
Liczba klientów na basenie: 3
Miejsce 1: PID klienta 657576 Wiek: 20
Miejsce 2: PID klienta 657597 Wiek: 20
Miejsce 3: PID klienta 657625 Wiek: 20
Miejsce 4: PUSTE

[21:46:06  657625] Klient chce wejść do basenu nr.2.
[21:46:06  657625] Klient wchodzi do basenu nr.2.
[21:46:08  657576] Klient wychodzi z kompleksu i basenu nr.2.

Stan tablicy klienci po usunięciu klienta REKREACYJNY:
Liczba klientów na basenie: 2
Miejsce 1: PUSTE
Miejsce 2: PID klienta 657597 Wiek: 20
Miejsce 3: PID klienta 657625 Wiek: 20
Miejsce 4: PUSTE
```

Brodzik

```
[21:51:49  664587] Klient chce wejść do basenu nr.3.

Stan tablicy klienci po dodaniu klienta BRODZIK:
Liczba klientów na basenie: 4
Miejsce 1: PID klienta 664587
Miejsce 2: PID klienta 663645

[21:51:49  664587] Klient wchodzi do basenu nr.3.
[21:51:49  664446] Klient wychodzi z kompleksu basenowego.
[21:51:52  664700] Pojawia się klient.
[21:51:52  664700] Kasjer obsługuje klienta.
[21:51:52  664700] Opiekun płaci za bilet. Dziecko nie płaci za bilet. Wiek: 2
[21:51:52  664700] Klient wchodzi na kompleks basenowy.
[21:51:52  664700] Dziecko zakłada pampers do plywania. Wiek: 2
[21:51:52  664700] Klient chce wejść do basenu nr.3.
[664700] Klient nie został wpuszczony do basenu nr.3 przez pełny basen.
[21:51:54  664587] Klient wychodzi z kompleksu i basenu nr.3.

Stan tablicy klienci po usunięciu klienta BRODZIK:
Liczba klientów na basenie: 2
Miejsce 1: PUSTE
Miejsce 2: PID klienta 663645
```

Wszystkie baseny poprawnie wpuszczają i wypuszczają klientów. Warto jednak zauważyć, że w brodziku, mimo tego że w są w nim 4 osoby, dostępne są tylko 2 miejsca. Jak wcześniej wspominałam, wynika to z faktu, że do brodzika zawsze wchodzi dziecko z opiekunem. Aby to uwzględnić, dostosowałam tablicę w taki sposób, że jedno zapełnione miejsce w tablicy odpowiada dwóm osobom (dziecku i opiekunowi).’

5. Warunki wejścia do basenu olimpijskiego i zapełnianie się basenu

Celem testu jest sprawdzenie poprawności działania instrukcji warunkowych które gwarantują prace kompleksu zgodnie z regulaminem.

Nieczynny basen

```
[21:16:54  1] RATOWNIK WYSYŁA SYGNAŁ NA WYJŚCIE Z BASENU NR.1.

Stan tablicy klienci po wysłaniu sygnału 1 OLIMPIJSKI:
Liczba klientów na basenie: 0
Miejsce 1: PUSTE
Miejsce 2: PUSTE
Miejsce 3: PUSTE
Miejsce 4: PUSTE
Miejsce 5: PUSTE

[21:16:55  638410] Pojawia się klient.
[21:16:55  638410] Kasjer obsługuje klienta.
[21:16:55  638410] Klient płaci za bilet.
[21:16:55  638410] Klient wchodzi na kompleks basenowy.
[21:16:55  638410] Klient chce wejść do basenu nr.1.
[638410] Klient nie został wpuszczony do basenu nr.1 ponieważ basen jest tymczasowo nieczynny.
```

Klient poniżej 18 roku życia

```
[21:16:55  1] RATOWNIK WYSYŁA SYGNAŁ NA POWRÓT DO BASENU NR.1.
[21:16:55  638412] Pojawia się klient.
[21:16:55  638412] Kasjer obsługuje klienta.
[21:16:55  638412] Opiekun płaci za bilet. Dziecko nie płaci za bilet. Wiek: 1
[21:16:55  638412] Klient wchodzi na kompleks basenowy.
[21:16:55  638412] Dziecko zakłada pampers do plywania. Wiek: 1
[21:16:55  638412] Klient zakłada czepek.
[21:16:55  638412] Klient chce wejść do basenu nr.1.
[638412] Klient nie został wpuszczony do basenu nr.1 przez wiek: 1.
```

Pełny basen

Żeby zapewnić, że wygenerowani klienci zapełnią basen ustawiłam każdemu z nich wiek 20.

```
[21:20:55  640634] Klient wchodzi do basenu nr.1.
Liczba klientów na basenie: 5
Miejsce 1: PID klienta 640635
Miejsce 2: PID klienta 640633
Miejsce 3: PID klienta 640636
Miejsce 4: PID klienta 640634
Miejsce 5: PID klienta 640639
[21:20:55  640638] Pojawia się klient.

[21:20:55  640638] Kasjer obsługuje klienta.
[21:20:55  640639] Klient wchodzi do basenu nr.1.
[21:20:55  640638] Klient płaci za bilet.
[640637] Klient nie został wpuszczony do basenu nr.1 przez pełny basen.
[640641] Klient nie został wpuszczony do basenu nr.1 przez pełny basen.
[640642] Klient nie został wpuszczony do basenu nr.1 przez pełny basen.
```

We wszystkich przypadkach warunki działają poprawnie i klienci nie zostali wpuszczeni do basenu.

6. Warunki wejścia do basenu olimpijskiego i zapełnianie się basenu

Celem testu jest sprawdzenie poprawności działania instrukcji warunkowych które gwarantują prace kompleksu zgodnie z regulaminem.

Basen nieczynny

```
[21:33:04  2] RATOWNIK WYSYŁA SYGNAŁ NA WYJŚCIE Z BASENU NR.2.
[647805] Klient odebral sygnal 1 Wychodzi z basenu nr.2.

Stan tablicy klienci po wysłaniu sygnału REKREACYJNY:
[647808] Klient odebral sygnal 1 Wychodzi z basenu nr.2.
Liczba klientów na basenie: 0
Miejsce 1: PUSTE
Miejsce 2: PUSTE
Miejsce 3: PUSTE
Miejsce 4: PUSTE

[647806] Klient odebral sygnal 1 Wychodzi z basenu nr.2.
[647804] Klient odebral sygnal 1 Wychodzi z basenu nr.2.
[21:33:05  647807] Klient chce wejść do basenu nr.2.
srednia: 20.000000
[647807] Klient nie został wpuszczony do basenu nr.2 ponieważ basen jest tymczasowo nieczynny.
```

Średnia wieku

Przy wychodzeniu klientów z basenu może się zdarzyć, że średnia wieku w basenie wzrośnie. Jest to trudne do kontrolowania, ponieważ ciężko jest stworzyć funkcjonalność, która zapobiegnie tej sytuacji, zwłaszcza gdy klienci wchodzą i wychodzą w losowy sposób.

```
[21:25:42  643010] Pojawia się klient VIP.
[21:25:42  643010] Kasjer obsługuje klienta VIP.
[21:25:42  643010] Opiekun płaci za bilet. Dziecko nie płaci za bilet. Wiek: 4
[21:25:42  643010] Klient wchodzi na kompleks basenowy.
[21:25:42  643010] Klient chce wejść do basenu nr.2.
srednia: 35.500000

Stan tablicy klienci po dodaniu klienta REKREACYJNY:
Liczba klientów na basenie: 2
Miejsce 1: PID klienta 643010 Wiek: 71
Miejsce 2: PUSTE
Miejsce 3: PUSTE
Miejsce 4: PUSTE
Miejsce 5: PUSTE
```

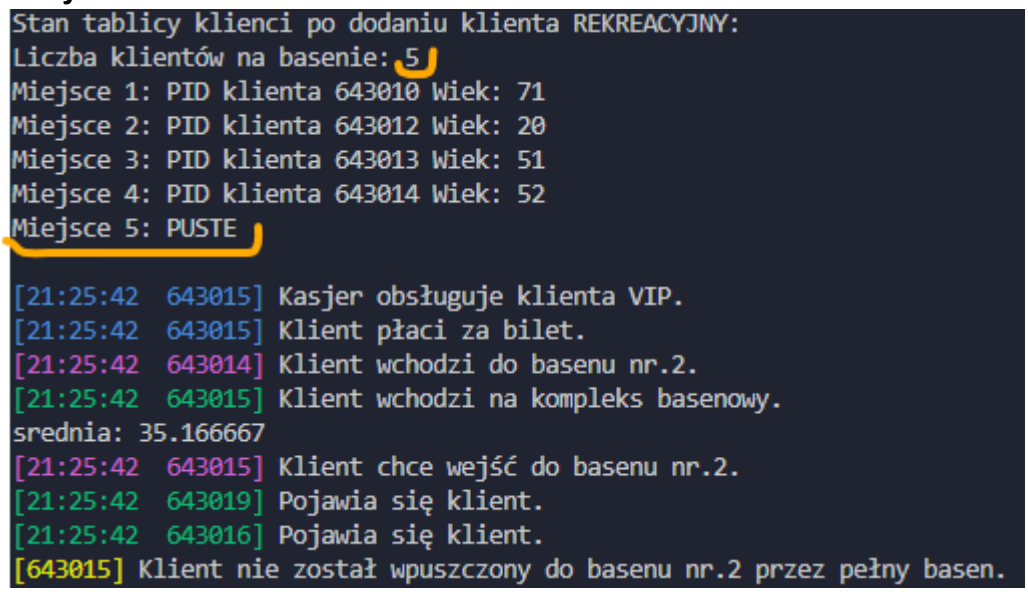
```
srednia: 44.000000
[21:25:42  643011] Klient chce wejść do basenu nr.2.
[21:25:42  643014] Kasjer obsługuje klienta.
[643011] Klient nie został wpuszczony do basenu nr.2 przez srednia wieku.
```

Warto zauważyć że gdy klient 643010 chce wejść do basenu rekreacyjnego średnia wieku wyświetla się jako 35,5 a po wejściu jego wiek jest oznaczony jako 71. Wynika to z tego że klient ten to dziecko z opiekunem jak widać wyżej.

```
[21:25:42  643010] Opiekun płaci za bilet. Dziecko nie płaci za bilet. Wiek: 4
```

Dziecko ma 4 lata, więc wygenerowany został dla niego opiekun, który wchodzi do basenu razem z nim. W związku z tym średnia wieku dzieli się na dwie osoby, co tłumaczy tę różnicę.

Pełny basen

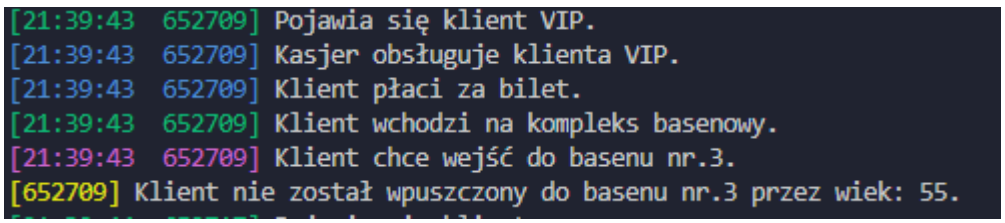


W tym konkretnym przypadku ustawiłam rozmiar basenu rekreacyjnego na 5, więc tablica przechowująca klientów tego basenu miała również 5 miejsc. Mimo że jedno miejsce było wolne, kolejny klient nie został wpuszczony do basenu. Dzieje się tak, ponieważ jeden z tych klientów to dziecko z opiekunem, a taki klient traktuję jako dwie osoby. W związku z tym, mimo pustego miejsca w tablicy, kolejny klient nie mógł wejść.
Podobna sytuacja wystąpiłaby, gdyby rozmiar basenu wynosił 5, a w basenie znajdowały się 4 osoby. W takim przypadku mimo że w basenie jest teoretycznie jedno wolne miejsce, dziecko z opiekunem nie mogłoby wejść, ponieważ byliby liczeni jest jako dwie osoby.

7. Warunki wejścia do brodzika i zapełnianie się basenu

Celem testu jest sprawdzenie poprawności działania instrukcji warunkowych które gwarantują prace kompleksu zgodnie z regulaminem.

Wiek



Basen nieczynny

```
[21:41:32  3] RATOWNIK WYSYŁA SYGNAŁ NA WYJŚCIE Z BASENU NR.3.

Stan tablicy klienci po wysłaniu sygnału 1 BRODZIK:
Liczba klientów na basenie: 3
Miejsce 1: PUSTE
Miejsce 2: PUSTE
Miejsce 3: PUSTE

[653637] Klient odebral sygnal 1 Wychodzi z basenu nr.3.
[653670] Klient odebral sygnal 1 Wychodzi z basenu nr.3.
[653690] Klient odebral sygnal 1 Wychodzi z basenu nr.3.
[21:41:32  653836] Pojawia się klient.
[21:41:32  653836] Kasjer obsługuje klienta.
[21:41:32  653836] Opiekun płaci za bilet. Dziecko nie płaci za bilet. Wiek: 3
[21:41:32  653836] Klient wchodzi na kompleks basenowy.
[21:41:32  653836] Dziecko zakłada pampers do plywania. Wiek: 3
[21:41:32  653836] Klient zakłada czepek.
[21:41:32  653836] Klient chce wejść do basenu nr.3.
[653836] Klient nie został wpuszczony do basenu nr.3 ponieważ basen jest tymczasowo nieczynny.
```

Pełny basen

```
[21:41:21  653690] Klient chce wejść do basenu nr.3.

Stan tablicy klienci po dodaniu klienta BRODZIK:
Liczba klientów na basenie: 6
Miejsce 1: PID klienta 653637
Miejsce 2: PID klienta 653670
Miejsce 3: PID klienta 653690

[21:41:21  653690] Klient wchodzi do basenu nr.3.
[21:41:22  653693] Pojawia się klient.
[21:41:22  653693] Kasjer obsługuje klienta.
[21:41:22  653693] Opiekun płaci za bilet. Dziecko nie płaci za bilet. Wiek: 3
[21:41:22  653693] Klient wchodzi na kompleks basenowy.
[21:41:22  653693] Dziecko zakłada pampers do plywania. Wiek: 3
[21:41:22  653693] Klient chce wejść do basenu nr.3.
[653693] Klient nie został wpuszczony do basenu nr.3 przez pełny basen.
```

8. Sygnały

Po uruchomieniu wątków obsługujących sygnały wypisuje się na ekranie godzina o której dany sygnał zostanie wysłany.

```
Kasjer [669975] Oczekiwanie na komunikaty...
Ratownik [669976] Obsługuje basen 1 o rozmiarze 4
Ratownik [669977] Obsługuje basen 2 o rozmiarze 4
Ratownik [669978] Obsługuje basen 3 o rozmiarze 4
[21:57:53 1] Wyslanie sygnalu 1.
[21:57:55 1] Wyslanie sygnalu 2.
[21:57:54 3] Wyslanie sygnalu 1.
[21:57:58 3] Wyslanie sygnalu 2.
[21:58:03 2] Wyslanie sygnalu 1.
[21:58:06 2] Wyslanie sygnalu 2.
```

Basen olimpijski

Czas wysłania sygnału pokrywa się z tym wyświetlonym wcześniej. Basen został wyczyszczony i klienci potwierdzili odebranie sygnału.

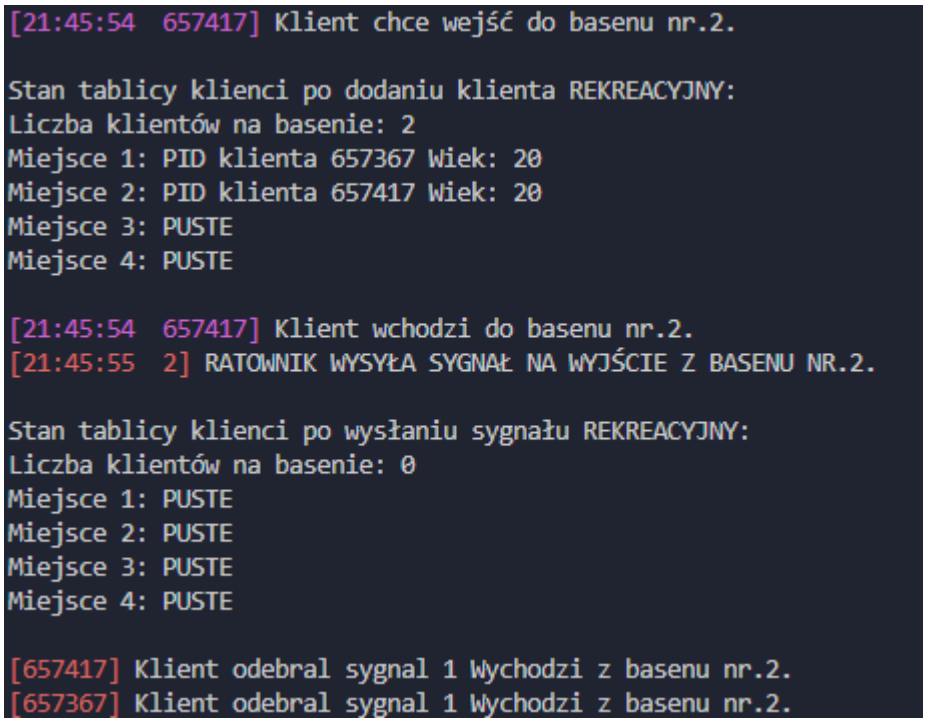
```
Stan tablicy klienci po usunięciu klienta OLIMPIJSKI:
Liczba klientów na basenie: 3
Miejsce 1: PUSTE
Miejsce 2: PID klienta 670096
Miejsce 3: PID klienta 670131
Miejsce 4: PID klienta 670199

[21:57:53  1] RATOWNIK WYSYŁA SYGNAŁ NA WYJŚCIE Z BASENU NR.1.

Stan tablicy klienci po wysłaniu sygnału 1 OLIMPIJSKI:
Liczba klientów na basenie: 0
Miejsce 1: PUSTE
Miejsce 2: PUSTE
Miejsce 3: PUSTE
Miejsce 4: PUSTE
[670096] Klient odebral sygnal 1 Wychodzi z basenu nr.1.

[670199] Klient odebral sygnal 1 Wychodzi z basenu nr.1.
[670131] Klient odebral sygnal 1 Wychodzi z basenu nr.1.
```

Wysłanie drugiego sygnału również się pokrywa czasowo. Poza zresetowaniem flagi dostępności basenu i zabronionego basenu do losowania nic więcej w tym przypadku się nie dzieje więc pominę umieszczanie zrzutów z sygnału 2 do pozostałych basenów.

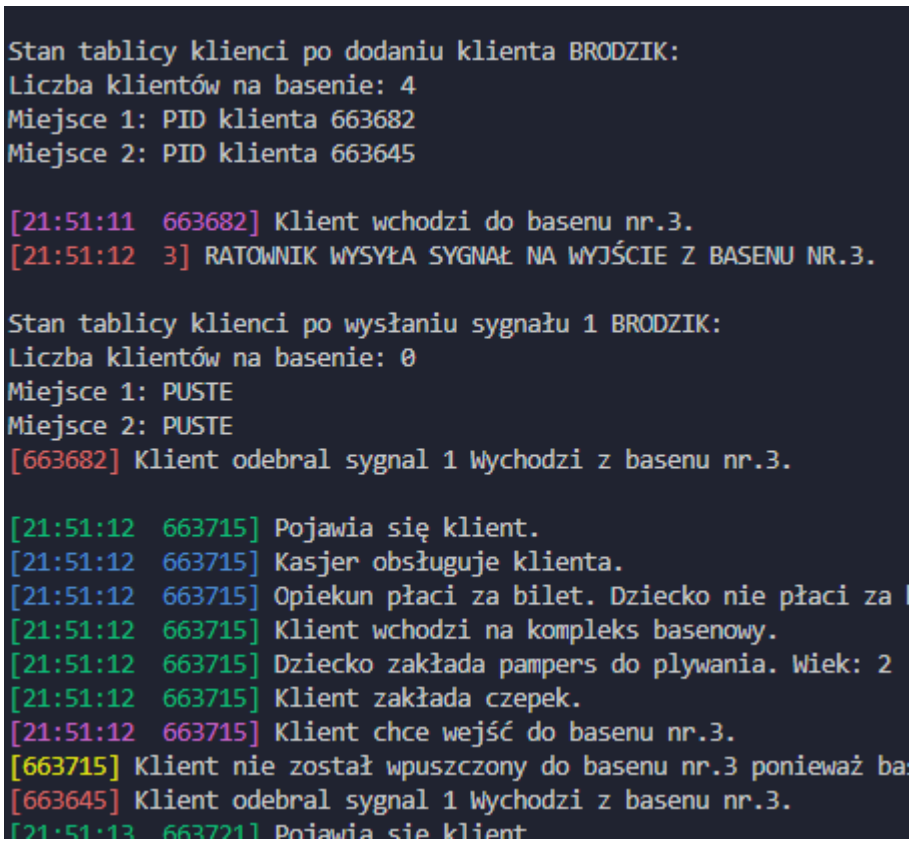
```
[22:28:41 1] Wyslanie sygnalu 2.
[22:28:41  1] RATOWNIK WYSYŁA SYGNAŁ NA POWRÓT DO BASENU NR.1.
```

Zrzuty ekranów z basenu rekreacyjnego i brodzika są z innego uruchomienia symulacji więc nie będą pokrywać się z czasami na zrzucie ekranu wyżej.

Basen rekreacyjny

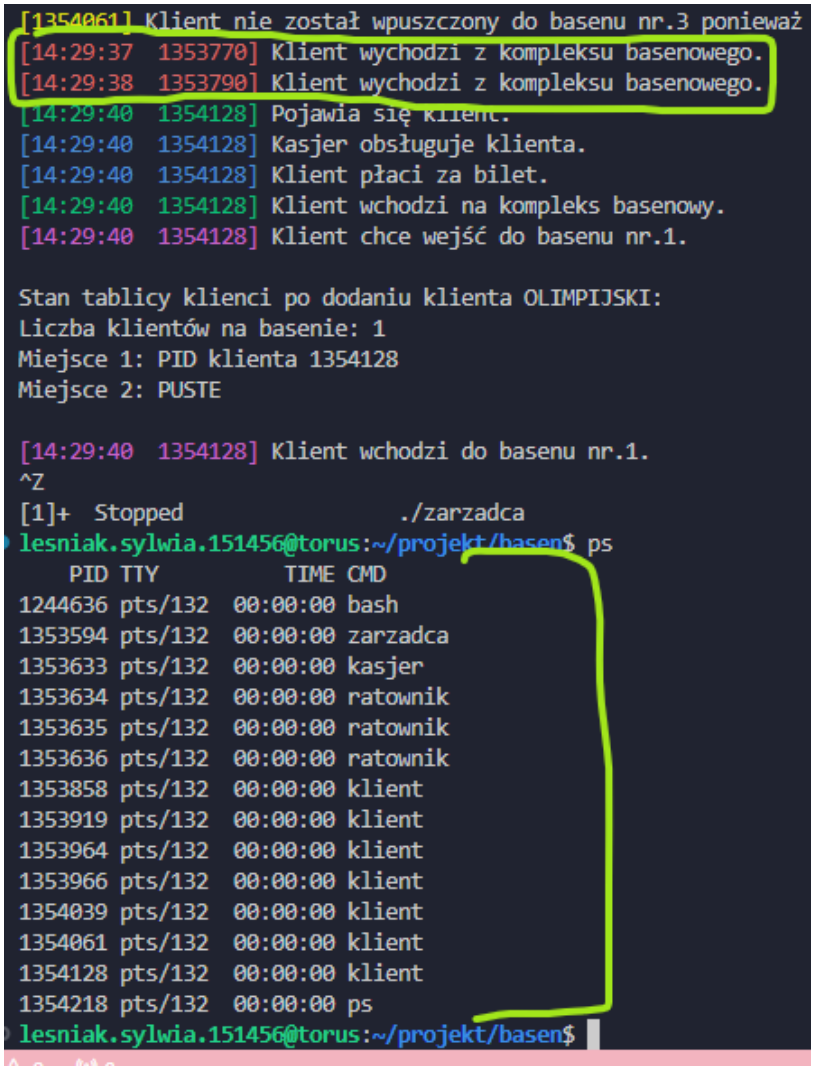


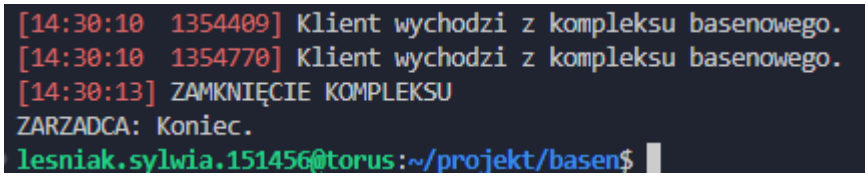
Brodzik

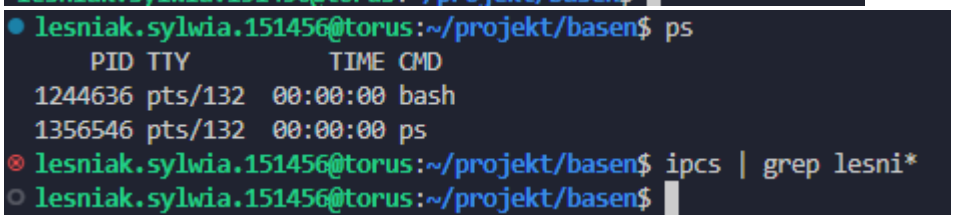


9. Procesy zombie

Nie miałam początkowo zaimplementowanego wątku do oczekiwania na zakończone procesy, przez co w trakcie działania programu wisiały procesy zombie, które oczekiwały na zakończenie programu. Żeby temu zapobiec dodałam wątek czyszczenie w którym funkcja w trakcie działania programu oczekuje na zakończone procesy. Symulacja przebiegła pomyślnie i w trakcie nie zostały w pamięci procesy zombie. Także cała symulacja zakończyła się poprawnie i nie zostawiła w pamięci ani żadnych procesów ani kolejek komunikatów i pamięci współdzielonej.







Elementy specjalne

Do projektu dodałam kolorowe komunikaty wyświetlane w terminalu. W funkcji printf używam %s, aby wskazać miejsce, w którym kolor ma się zacząć, oraz miejsce, w którym ma się zakończyć. Jako argumenty podaję kolor, który chcę użyć, oraz kod resetujący kolor w odpowiednim miejscu. Kolory zostały zdefiniowane w pliku utility.h, co umożliwia ich łatwe wykorzystanie w całym projekcie.

```
printf("%s[%02d:%02d:%02d  %d]%s Klient wchodzi na kompleks basenowy.\n", GREEN, local->tm_hour, local->tm_min, local->tm_sec, klient.pid, RESET);
```

```
// Definicje kolorów do wyświetlania w konsoli
const char *RESET = "\033[0m";
const char *RED = "\033[31m";
const char *GREEN = "\033[32m";
const char *YELLOW = "\033[33m";
const char *BLUE = "\033[34m";
const char *MAGENTA = "\033[35m";
const char *CYAN = "\033[36m";
```

Przykład:
Kolor czerwony oznacza sygnały lub problemy, zielony klientów, niebieski kasjera a magenta baseny/ratowników.

```
[663682] Klient odebral sygnal 1 Wychodzi z basenu nr.3.

[21:51:12  663715] Pojawia się klient.
[21:51:12  663715] Kasjer obsługuje klienta.
[21:51:12  663715] Opiekun płaci za bilet. Dziecko nie płaci za
[21:51:12  663715] Klient wchodzi na kompleks basenowy.
[21:51:12  663715] Dziecko zakłada pampers do plywania. Wiek: 2
[21:51:12  663715] Klient zakłada czepek.
[21:51:12  663715] Klient chce wejść do basenu nr.3.
```

Koloru żółtego używałam w informacjach w niektórych częściach kodu. Cyan używam na początku do wyświetlenia czasów wysłania sygnałów.

```
[21:41:22  653693] Klient chce wejść do basenu nr.3.
[653693] Klient nie został wpuszczony do basenu nr.3 przez pełny basen.
```

Linki do fragmentów kodu

Część linków jest sprzed ostatniego commitu z dodanym wątkiem na usuwanie procesów zombie, ale poza tym wątkiem w zarzadca.c i dodaniem dokładniejszego opisu funkcji w utility.c kod jest taki sam.

Tworzenie i obsługa procesów
Kasjer
https://github.com/HinoYoseii/basen/blob/75b327469134aacd68f5469adae55d10ff61d592/zarzadca.c#L94-L99
Ratownicy
https://github.com/HinoYoseii/basen/blob/75b327469134aacd68f5469adae55d10ff61d592/zarzadca.c#L103-L112

Klienci
https://github.com/HinoYoseii/basen/blob/75b327469134aacd68f5469adae55d10ff61d592/zarzadca.c#L123-L134
Wait()
https://github.com/HinoYoseii/basen/blob/75b327469134aacd68f5469adae55d10ff61d592/zarzadca.c#L183
https://github.com/HinoYoseii/basen/blob/75b327469134aacd68f5469adae55d10ff61d592/zarzadca.c#L145-L148

Tworzenie i obsługa wątków

Ratownik 1
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L58-L71
Przykład zablokowania i odblokowania mutexu ratownik 1
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L132
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L162

Ratownik 2
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L81-L94
Przykład zablokowania i odblokowania mutexu ratownik 2
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L176
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L248

Ratownik 3
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L103-L116
Przykład zablokowania i odblokowania mutexu ratownik 3
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L265
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L294

Zarządca czyszczenie
https://github.com/HinoYoseii/basen/blob/75b327469134aacd68f5469adae55d10ff61d592/zarzadca.c#L118-L119
https://github.com/HinoYoseii/basen/blob/75b327469134aacd68f5469adae55d10ff61d592/zarzadca.c#L140-L141

Obsługa sygnałów

Kill SIGUSR1
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L453
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L478

Kill SIGUSR2
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L513

sigaction SIGINT
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/zarzadca.c#L12-L16

sigaction SIGUSR1 SIGUSR2
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/klient.c#L12-L17

Synchronizacja procesów (wątków)

Ftok() kilka przykładów
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/zarzadca.c#L59
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/zarzadca.c#L67
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/zarzadca.c#L75

Segmenty pamięci dzielonej

Ftok(), shmget(), shmat()
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/zarzadca.c#L75-L83
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L40-L47

shmctl()
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/zarzadca.c#L20

shmdt()
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L120

Kolejki komunikatów

Ftok(), msgget()
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/zarzadca.c#L59-L62
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/kasjer.c#L14-L17
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/klient.c#L29-L32
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L34-L37

msgrcv()
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/kasjer.c#L30
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/klient.c#L64
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L176

msgsnd()
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/klient.c#L62
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/kasjer.c#L61
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/ratownik.c#L251

msgctl()
https://github.com/HinoYoseii/basen/blob/897eb968ff086c35a9334ae86b245f2d043d2951/zarzadca.c#L155-L156