Autorzy:
Szymon Laskowski, PS10
Paweł Kamieński, PS10

Prowadzący:
mgr inż. Daniel Reska

Dokumentacja projektu nr.2 „Czytelnicy i pisarze”

1.Treść zadania

Z czytelni korzysta na okrągło pewna ilość czytelników i pisarzy (dwa typy wątków), przy czym jednocześnie może w niej znajdować się albo dowolna ilość czytelników, albo jeden pisarz, albo nikt - nigdy inaczej. Problem ten ma trzy rozwiązania - z możliwością zagłodzenia pisarzy, z możliwością zagłodzenia czytelników oraz wykluczające zagłodzenie. Napisać:

- dwa programy symulujące dwa różne rozwiązania tego problemu, bez korzystania ze zmiennych warunkowych **[17 p]**, **albo (!)**
- dwa programy symulujące dwa różne rozwiązania tego problemu, przy czym jeden z nich musi korzystać ze zmiennych warunkowych (condition variable). **[27 p]**, **albo**
- trzy programy symulujące trzy różne rozwiązania tego problemu, przy czym przynajmniej jeden z nich musi korzystać ze zmiennych warunkowych **[34 p]**.

Ilość wątków pisarzy R i czytelników W można przekazać jako argumenty linii poleceń. Zarówno czytelnicy jak i pisarze wkrótce po opuszczeniu czytelni próbują znów się do niej dostać (wątki działają dalej). Program powinien wypisywać komunikaty według poniższego przykładu:

ReaderQ: 11 WriterQ: 10 [in: R:0 W:1]

Zrealizowano opcję drugą.

Dwa programy symulujące dwa różne rozwiązania tego problemu, przy czym jeden z nich musi korzystać ze zmiennych warunkowych (condition variable). [27 p]

2. Kod wspólny dla każdego programu.

Uruchomienie programu z linią poleceń inną niż 5 (innego niż np./main R 5 W 3) – skutkuje błędem.

Odczyt danych obsłużono w takim sam sposób jak przy pisaniu pierwszego projektu – za pomocą getopt, zaimplementowano parametry R – readers (czytelnicy) oraz W – writers (pisarze)
Wprowadzono walidacje wprowadzonych danych.

Do weryfikacji czy dane są poprawne wykorzystano do tego funkcję atoi oraz prostego if'a. Funkcja atoi zwraca 0, gdy nie jest możliwe przekonwertowanie wartości z łańcucha znaków do postaci liczbowej, więc ustalono iż podanie mniejszej liczby pisarzy/czytelników niż 1 to błąd.

```
if( argc != 5 ){
    fprintf(stderr,"%s","Blad w przekazywaniu argumentów ilosci pisarzy i czytelnikow\n");
    exit(EXIT_FAILURE);
}

// MENU
// - R wartosc (liczba czytelnikow ile ich mamy do dyspozycji)
// - W wartosc (liczba pisarzy ilu mamy do dyspozycji)

int choice;

while ((choice = getopt(argc,argv,"R:W:")) != -1){

    switch (choice) {
        case 'R':
            number_of_readers = atoi(optarg);
            if( number_of_readers < 1) {
                fprintf(stderr,"%s","Czytelnik musi byc przynajmniej jeden !");
                exit(EXIT_FAILURE);
            }
            break;

        case 'W':
            number_of_writers = atoi(optarg);
            if( number_of_writers < 1) {
                fprintf(stderr,"%s","Pisarz musi byc przynajmniej jeden !");
                exit(EXIT_FAILURE);
            }
            break;

        default:
            fprintf(stderr,"%s","Podano nieprawidlowe argumenty");
            exit(EXIT_FAILURE);
    }
}
```

Odczytywanie przekazanych wartości parametrów.

```c
pthread_t *writer_threads = malloc(sizeof(pthread_t) * number_of_writers);
pthread_t *reader_threads = malloc(sizeof(pthread_t) * number_of_readers);

for (int i = 0; i < number_of_readers; i++) {
    int *index = malloc(sizeof(int));
    *index = i;
    if (pthread_create(&reader_threads[i], NULL, reader_function, index) != 0){
        perror("Pthread crate error");
        exit(EXIT_FAILURE);
    }
}
for (int i = 0; i < number_of_writers; i++) {
    int *index = malloc(sizeof(int));
    *index = i;
    if (pthread_create(&writer_threads[i], NULL, writer_function, index) != 0){
        perror("Pthread crate error");
        exit(EXIT_FAILURE);
    }
}

Join_Pthread(number_of_readers,reader_threads);
Join_Pthread(number_of_writers, writer_threads);


free(writer_threads);
writer_threads = NULL;

free(reader_threads);
reader_threads = NULL;
```

Inicjalizacja/Utworzenie wątków

```c
void Join_Pthread(int number, const pthread_t *pthread) {
    int rval;
    for (int i = 0; i < number; i++) {
        void *index = NULL;
        rval = pthread_join(pthread[i], &index);
        if(rval) {
            perror("Pthread join error");
            exit(EXIT_FAILURE);
        }
        free(index);
        index = NULL;
    }
}
```

Wyeksportowana funkcja Join_Pthread dla powtarzanego się kodu wraz z obsługa błędów funkcji zawieszającej wątek wywołujący dopóki wątek docelowy (pthread[i]) się nie zakończy

Zmienne **active_number_of_writers**, **active_number_of_readers** przechowuje aktualną liczbę aktorów przebywających w bibliotece.

Zmienne **number_of_writers** oraz **number_of_readers** przechowują liczbę przekazaną za pomocą wiersza poleceń.

Projekt składa się z dwóch katalogów:

- **projekt_semafory_zaglodzenie_pisarzy** – zawiera rozwiązanie z możliwością zagłodzeniem pisarzy wykonane za pomocą semafor.
- **projekt_zmiennewarunkowe_zaglodzenie_czytelnikow** – zawiera rozwiązanie z możliwością zagłodzenia czytelników za pomocą wykorzystania zmiennych warunkowych.

A. Rozwiązanie z możliwością zagłodzenie pisarzy wykonano za pomocą semafor.

```
rval = sem_init(&reader_sem, pshared: 0, value: 1); // Semafory zaczynaja prace na wartosci 1.
if(rval) {
    perror( s: "Error with sem init for reader_sem");
    exit(EXIT_FAILURE);
}

rval = sem_init(&writer_sem, pshared: 0, value: 1);
if(rval) {
    perror( s: "Error with sem init for writer_sem");
    exit(EXIT_FAILURE);
}
```

```c
void *reader(void *arg) {
    int *index = (int*) arg;
    while (1) {
        sem_wait(&reader_sem); // wejscie do sekcji krytycznej

        if (active_number_of_readers + 1 == 1) { // jezeli czytelnik jest jedyna osoba w bibliotece
            sem_wait(&writer_sem); // pisarze musza czekac w kolejce
        }

        active_number_of_readers++; // do biblioteki bez kolejki wchodzi czytelnik

        printf("ReaderQ: %d WriterQ: %d [in: R:%d W:%d], Wchodzi Czytelnik id: %d""\n",
               number_of_readers - active_number_of_readers,
               number_of_writers - active_number_of_writers,
               active_number_of_readers,
               active_number_of_writers,
               *index
        );

        sem_post(&reader_sem); // wyjscie z sekcji krytycznej

        // poczatek kodu wykonywanego przez czytelnika
        usleep(250 * 1000);

        // koniec kodu wykonywanego przez czytelnika

        sem_wait(&reader_sem); // wejscie do sekcji krytycznej

        active_number_of_readers--; // czytelnik wychodzi z biblioteki

        printf("ReaderQ: %d WriterQ: %d [in: R:%d W:%d], Wychodzi Czytelnik id: %d""\n",
               number_of_readers - active_number_of_readers,
               number_of_writers - active_number_of_writers,
               active_number_of_readers,
               active_number_of_writers,
               *index
        );

        if (active_number_of_readers == 0) { // jezeli w bibliotece nie bedzie czytelnikow
            sem_post(&writer_sem); // informujemy pisarzy, ze biblioteka jest pusta
        }

        sem_post(&reader_sem); // wyjscie z sekcji krytycznej

    }
}
```

Powyższy kod odpowiada za logikę czytelnika. Poszczególne linie kodu są opisane komentarzami.
Modyfikacja i sprawdzenie stanu zmiennej **`active_number_of_readers`** wymaga wejścia do sekcji krytycznej (dzięki funkcji **`sem_wait`**), aby uniknąć wyścigów między wątkami.

Myślenie czytelnika opiera się o kilka zasad:

- **w bibliotece może przebywać dowolna liczba czytelników.**
- **jeżeli czytelnik jest jedyną osobą w bibliotece, to zablokuj kolejkę pisarzy;**
- **jeżeli czytelnik będzie ostatnią osobą w bibliotece, to odblokuj kolejkę pisarzy;**

```
void *writer(void *arg) {
    int *index = (int*) arg;
    while(1) {

        sem_wait(&writer_sem);
        active_number_of_writers++;

        printf("ReaderQ: %d WriterQ: %d [IN: R:%d W:%d], Wchodzi Pisarz id: %d" "\n",
               number_of_readers - active_number_of_readers,
               number_of_writers - active_number_of_writers,
               active_number_of_readers,active_number_of_writers,*index);

        usleep(250*1000);

        active_number_of_writers--;

        printf("ReaderQ: %d WriterQ: %d [IN: R:%d W:%d], Wychodzi Pisarz id: %d" "\n",
               number_of_readers - active_number_of_readers,
               number_of_writers - active_number_of_writers,
               active_number_of_readers,active_number_of_writers,*index);


        sem_post(&writer_sem);
    }

    return NULL;

}
```

Powyższy kod odpowiada za logikę pisarza. Zasada działania jest prosta, bo wykorzystuje jeden semafor, który dodaje pisarzy do kolejki i trzyma ich w niej tak długo, aż czytelnia będzie pusta.

B. **<u>Rozwiązanie z możliwością zagłodzenia czytelników wykonano za pomocą zmiennych warunkowych.</u>**

```
rval = pthread_mutex_init(&mutex, mutexattr: NULL);
if(rval) {
    fprintf(stderr, format: "Error with pthread mutex init return: %d",rval);
    exit(EXIT_FAILURE);
}

rval = pthread_cond_init(&cond, cond_attr: NULL);
if(rval) {
    fprintf(stderr, format: "Error with pthread cond init return: %d",rval);
    exit(EXIT_FAILURE);
}
```

```c
void *reader_function(void *arg) {
    int *id = (int*) arg;
    while (1) {
        pthread_mutex_lock(&mutex); // wejscie do sekcji krytycznej

        // czetelnik bedzie czekal w kolejce, jezeli:
        // * istnieje pisarz, ktory pisze <- nie mozna modyfikowac zasobow i jednoczesnie z nich korzystac
        // * przynajmniej jeden pisarz czeka w kolejce <- bo to rozwiazanie jest writer-friendly
        while (active_number_of_writers > 0 || waiting_writers_count > 0) {
            pthread_cond_wait(&cond, &mutex);
        }

        active_number_of_readers++; // dopiero, gdy nie bedzie chetnego pisarza, do biblioteki wchodzi czytelnik

        printf("ReaderQ: %d WriterQ: %d [in: R:%d W:%d] -> Wchodzi czytelnik o id: %d" "\n",
               number_of_readers - active_number_of_readers,
               number_of_writers - active_number_of_writers,
               active_number_of_readers,
               active_number_of_writers,
               *id
        );

        pthread_mutex_unlock(&mutex); // wyjscie z sekcji krytycznej

        // poczatek kodu wykonywanego przez czytelnika
        usleep(250 * 1000);
        // koniec kodu wykonywanego przez czytelnika

        pthread_mutex_lock(&mutex); // wejscie do sekcji krytycznej

        active_number_of_readers--; // czytelnik wychodzi z biblioteki

        printf("ReaderQ: %d WriterQ: %d [in: R:%d W:%d] -> Wychodzi czytelnik o id: %d" "\n",
               number_of_readers - active_number_of_readers,
               number_of_writers - active_number_of_writers,
               active_number_of_readers,
               active_number_of_writers,
               *id
        );

        if (active_number_of_readers == 0) { // jezeli w bibliotece nie bedzie czytelnikow (a moze byc ich wielu)
            pthread_cond_broadcast(&cond); // poinformuj wszystkich pisarzy i czytelnikow z kolejki, ze biblioteka jest wolna
        }

        pthread_mutex_unlock(&mutex); // wyjscie z sekcji krytycznej
    }
    return NULL;
}
```

Wymagane było użycie pomocniczej zmiennej **`waiting_writers_count`**, która przechowuje liczbę oczekujących pisarzy.

Powyższy kod odpowiada za pracę wykonywaną przez wątek czytelnika.

Tak samo jak w poprzednim programie, modyfikacja i sprawdzenie stanu zmiennej **`active_number_of_readers`** wymaga wejścia do sekcji krytycznej (użyto funkcji **`pthread_mutex_lock`**), aby uniknąć wyścigów między wątkami. Jeżeli warunek w pętli while zostanie spełniony to zmienna warunkowa blokuje wszystkie wątki pisarzy i czytelników. Wszystkie wątki zostaną odblokowane dopiero po użyciu funkcji **`pthread_cond_broadcast`** i będą kontynuować miejsca wywołania **`pthread_cond_wait`**. Właśnie dlatego wykorzystaliśmy pętlę while, a nie warunek if. Po wysłaniu sygnału, wątki zostaną odblokowane, wykona się kolejna iteracja pętli while i program zdecyduje, które wątki czytelników powinny zostać nadal zablokowane. Decyzja polega na sprawdzeniu stanu zmiennych.

Czytelnik pozostanie w kolejce tak długo, aż:

- **w bibliotece będzie przebywał pisarz (aby jednocześnie nie odczytywać i modyfikować zasobów).**
- **w kolejce będzie przynajmniej jeden pisarz (dlatego wykorzystaliśmy zmienną waiting_writers_count).**

```c
void *writer_function(void *arg) {
    int *id = (int*) arg;
    while (1) {
        pthread_mutex_lock(&mutex); // wejscie do sekcji krytycznej

        waiting_writers_count++; // bardzo wazna linia -> informacja, ze do kolejki wszedl pisarz, czytelnicy ustapia mu miejsca

        // pisarz bedzie czekal w kolejce, jezeli:
        // * istnieje inny pisarz, ktory pisze <- dwie osoby nie moga modyfikowac zasobow
        // * istnieje czytelnik, ktory korzysta z biblioteki <- nie mozna modyfikowac zasobow i jednoczesnie z nich korzystac
        while (active_number_of_writers > 0 || active_number_of_readers > 0) {
            pthread_cond_wait(&cond, &mutex);
        }

        waiting_writers_count--; // pisarz wychodzi z kolejki i wchodzi do biblioteki
        // pisarz korzysta z biblioteki
        // (tylko jeden pisarz moze przebywac w bibliotece, wedlug zasady: dwie osoby nie moga modyfikowac zasobow)
        active_number_of_writers = 1;

        printf("ReaderQ: %d WriterQ: %d [in: R:%d W:%d] -> Wchodzi pisarz o id: %d" "\n",
               number_of_readers - active_number_of_readers,
               number_of_writers - active_number_of_writers,
               active_number_of_readers,
               active_number_of_writers,
               *id
        );

        pthread_mutex_unlock(&mutex); // wyjscie z sekcji krytycznej

        // poczatek kodu wykonywanego przez pisarza
        usleep(250 * 1000);
        // koniec kodu wykonywanego przez pisarza

        pthread_mutex_lock(&mutex); // wejscie do sekcji krytycznej

        active_number_of_writers = 0; // pisarz wychodzi z biblioteki (tylko jeden pisarz moze przebywac w bibliotece)

        printf("ReaderQ: %d WriterQ: %d [in: R:%d W:%d] -> Wychodzi pisarz o id: %d" "\n",
               number_of_readers - active_number_of_readers,
               number_of_writers - active_number_of_writers,
               active_number_of_readers,
               active_number_of_writers,
               *id
        );

        // poinformuj wszystkich pisarzy i czytelnikow z kolejki, ze biblioteka jest wolna
        // (na podstawie warunku w petli while czytelnikow i pisarzy, z kolejki zostanie wybrany odpowiedni watek)
        pthread_cond_broadcast(&cond);
        pthread_mutex_unlock(&mutex); // wyjscie z sekcji krytycznej
    }
    return NULL;
}
```

Powyższy kod odpowiada za pracę wykonywaną przez wątek pisarza.

W sekcji krytycznej przeprowadzana jest modyfikacja i sprawdzenie stanu zmiennych writing_writers_count oraz active_number_of_writers, która jest opatrzona muteksem. W czytelni może być tylko jeden pisarz, dlatego zmienna active_number_of_writers przyjmuje wartość 0 lub 1. Najważniejsza linia kodu odpowiada za zwiększenie wartości zmiennej służącej do zliczania oczekujących pisarzy (waiting_writers_count++). To ona sprawia, że czytelnicy będą czekali w kolejce tak długo, aż nie będzie w niej ani jednego pisarza. Takie rozwiązanie doprowadza do zagłodzenia czytelników.

3. Przykłady działania

```
pawcio@pawcio-virtualbox:~/Desktop/projekt_semafory_zaglodzenie_pisarzy$ gcc main.c -o main -lpthread
pawcio@pawcio-virtualbox:~/Desktop/projekt_semafory_zaglodzenie_pisarzy$ ./main -R 11 -W 10
ReaderQ: 10 WriterQ: 10 [in: R:1 W:0], Wchodzi Czytelnik id: 6
ReaderQ: 9 WriterQ: 10 [in: R:2 W:0], Wchodzi Czytelnik id: 7
ReaderQ: 8 WriterQ: 10 [in: R:3 W:0], Wchodzi Czytelnik id: 5
ReaderQ: 7 WriterQ: 10 [in: R:4 W:0], Wchodzi Czytelnik id: 8
ReaderQ: 6 WriterQ: 10 [in: R:5 W:0], Wchodzi Czytelnik id: 9
ReaderQ: 5 WriterQ: 10 [in: R:6 W:0], Wchodzi Czytelnik id: 4
ReaderQ: 4 WriterQ: 10 [in: R:7 W:0], Wchodzi Czytelnik id: 10
ReaderQ: 3 WriterQ: 10 [in: R:8 W:0], Wchodzi Czytelnik id: 3
ReaderQ: 2 WriterQ: 10 [in: R:9 W:0], Wchodzi Czytelnik id: 2
ReaderQ: 1 WriterQ: 10 [in: R:10 W:0], Wchodzi Czytelnik id: 1
ReaderQ: 0 WriterQ: 10 [in: R:11 W:0], Wchodzi Czytelnik id: 0
ReaderQ: 1 WriterQ: 10 [in: R:10 W:0], Wychodzi Czytelnik id: 7
ReaderQ: 0 WriterQ: 10 [in: R:11 W:0], Wchodzi Czytelnik id: 7
ReaderQ: 1 WriterQ: 10 [in: R:10 W:0], Wychodzi Czytelnik id: 5
ReaderQ: 0 WriterQ: 10 [in: R:11 W:0], Wchodzi Czytelnik id: 5
ReaderQ: 1 WriterQ: 10 [in: R:10 W:0], Wychodzi Czytelnik id: 6
ReaderQ: 0 WriterQ: 10 [in: R:11 W:0], Wchodzi Czytelnik id: 6
ReaderQ: 1 WriterQ: 10 [in: R:10 W:0], Wychodzi Czytelnik id: 8
ReaderQ: 0 WriterQ: 10 [in: R:11 W:0], Wchodzi Czytelnik id: 8
ReaderQ: 1 WriterQ: 10 [in: R:10 W:0], Wychodzi Czytelnik id: 9
ReaderQ: 0 WriterQ: 10 [in: R:11 W:0], Wchodzi Czytelnik id: 9
ReaderQ: 1 WriterQ: 10 [in: R:10 W:0], Wychodzi Czytelnik id: 4
ReaderQ: 0 WriterQ: 10 [in: R:11 W:0], Wchodzi Czytelnik id: 4
ReaderQ: 1 WriterQ: 10 [in: R:10 W:0], Wychodzi Czytelnik id: 10
ReaderQ: 0 WriterQ: 10 [in: R:11 W:0], Wchodzi Czytelnik id: 10
ReaderQ: 1 WriterQ: 10 [in: R:10 W:0], Wychodzi Czytelnik id: 3
```

Przykład działania rozwiązania zagłodzenia pisarzy przy użyciu semaforów

-R 11 –W 10

```
pawcio@pawcio-virtualbox:~/CLionProjects/projekt_zmiennewarunkowe_zaglodzenie_czytelnikow$ gcc main.c -o main -lpthread
pawcio@pawcio-virtualbox:~/CLionProjects/projekt_zmiennewarunkowe_zaglodzenie_czytelnikow$ ./main -R 11 -W 10
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 6
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 8
ReaderQ: 11 WriterQ: 9 [in: R:0 W:1] -> Wchodzi pisarz o id: 5
ReaderQ: 11 WriterQ: 10 [in: R:0 W:0] -> Wychodzi pisarz o id: 5
```

Przykład działania rozwiązania zagłodzenia czytelników przy użyciu zmiennych warunkowych

-R 11 –W 10

4. Źródła

Do realizacji projektu, skorzystano ze źródeł:

- Linux. Programowanie dla zaawansowanych - Mark Mitchell, Jeffrey Oldham, Alex Samuel
- CodeVault – YouTube - m.in. Condition variables in C https://www.youtube.com/channel/UC6qj_bPq6tQ6hLwOBpBQ42Q