Programowanie współbieżne

1.1

Wygenerowano przez Doxygen 1.9.7

Rozdział 1

Strona główna dokumentacji

1.1 Dokumentacja

Wydział Informatyki Politechniki Białostockiej Przedmiot: Systemy Operacyjne	Data oddania: 03.06.2023 r.
Pracownia specjalistyczna nr 1 Zespół: 1. Mateusz Kondraciuk 2. Jakub Franciszek Modzelewski 3. Dawid Waszkiewicz	Prowadzący: dr inż. Wojciech Kwedlo Ocena:

1.2 Realizacja zadania

W celu sprawdzenia treści zadania proszę udać się pod Treść Zadania.
W celu sprawdzenia wykonanych podpunktów proszę udać się pod Wykonane podpunkty.
W celu sprawdzenia instrukcji obsługi proszę udać się pod Instrukcja obsługi.

Rozdział 2

Treść Zadania

Projekt 2 - wąski most
Wąski most. Z miasta A do miasta B prowadzi droga, na której znajduje się wąski most umożliwiający tylko ruch jednokierunkowy. Most jest również dość słaby, także może po nim przejeżdżać tylko jeden samochód na raz. Napisać program w którym N samochodów (wątków) będzie nieustannie przejeżdżało z miasta do miasta, pokonując po drodze most (N przekazywane jako argument linii poleceń). Zsynchronizuj dostęp wątków do mostu:
a) nie wykorzystując zmiennych warunkowych (tylko mutexy/semafory) [17p]
b) wykorzystując zmienne warunkowe (condition variables) [17p]
Aby móc obserwować działanie programu, każdemu samochodowi przydziel numer. Program powinien wypisywać komunikaty według poniższego przykładu:
A-5 10>>> [>> 4 >>] <<<4 6-B
Oznacza to, że po stronie miasta A jest 15 samochodów z czego 10 czeka w kolejce przed mostem, przez most przejeżdża samochód z numerem 4 z miasta A do B, po stronie miasta B jest 10 samochodów, z czego 4 oczekują w kolejce przed mostem. Komunikat należy wypisywać w momencie, kiedy w programie zmieni się którakolwiek z tych wartości.
W celu sprawdzenia wykonanych podpunktów proszę udać się pod Wykonane podpunkty.

Rozdział 3

Wykonane podpunkty

3.1 Podpunkty

3.1.1 Używając wyłącznie mutexy/semafory

Do realizacji zadania użyliśmy wyłącznie mutexów

3.1.2 Wykorzystując zmienne warunkowe (condition variables)

Zadanie zostało zrealizowane wyłącznie z wykorzystaniem zmiennych warunkowych.

Rozdział 4

Instrukcja obsługi

4.1 Kompilacja programu

4.1.1 gcc main.c -o program

Zostanie stworzony zkompilowany plik o nazwie program, który następnie możemy wywołać w celu uruchomienia skryptu

4.2 Realizacja zadania

4.2.1 Ilość wątków(samochodów)

Pierwszym parametrem do podania jest ilość wątków które będą tworzone przez program.

4.2.2 Rodzaj realizacji projektu

Paramatr pozwala na wybór formy realizacji projektu: 0- używając mutexów, 1- używając zmienne warunkowe

4.3 Przykłady

4.3.1 ./program 10 0

Wywołujemy program, zostanie stworzone 10 wątków, a zadanie będzie realizowane przy pomocy mutexów

4.3.2 ./program 15 1

Wywołujemy program, zostanie stworzone 15 wątków, a zadanie będzie realizowane przy pomocy zmiennych warunkowych

Rozdział 5

Indeks struktur danych

5.1 Struktury danych

Tutaj znajdują się struktury danych wraz z ich krótkimi opisami:

Rozdział 6

Indeks plików

6.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

Rozdział 7

Dokumentacja struktur danych

7.1 Dokumentacja struktury Car

Struktura przechowywująca wszystkie samochody.

Pola danych

- int carId

  Przechowuje ID pojazdu.
- long int threadId

  Przechowuje ID wątku.
- bool isWaiting

  Przechowuje wartość boolean definiującą czy pojazd czeka na możliwość przejechania przez most.
- char cityName [32]

  Przechowuje nazwę miasta.
- int idleMeter

  Przechowuje licznik, kiedy wartość wyniesie 0, parametr isWaiting zmieni się na true.
- struct Car * next

  Przechowuje wskaźnik na następny pojazd bądź null jeśli jest to koniec listy.

7.1.1 Opis szczegółowy

Struktura przechowywująca wszystkie samochody.

7.1.2 Dokumentacja pól

7.1.2.1 carId

```
int carId
```

Przechowuje ID pojazdu.

7.1.2.2 cityName

```
char cityName[32]
```

Przechowuje nazwę miasta.

7.1.2.3 idleMeter

```
int idleMeter
```

Przechowuje licznik, kiedy wartość wyniesie 0, parametr isWaiting zmieni się na true.

7.1.2.4 isWaiting

```
bool isWaiting
```

Przechowuje wartość boolean definiującą czy pojazd czeka na możliwość przejechania przez most.

7.1.2.5 next

```
struct Car* next
```

Przechowuje wskaźnik na następny pojazd bądź null jeśli jest to koniec listy.

7.1.2.6 threadId

```
long int threadId
```

Przechowuje ID wątku.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- Program/struct.c

Rozdział 8

Dokumentacja plików

8.1 Dokumentacja pliku Program/draw.c

Plik wyświetlający komunikaty o przejazdach samochodów.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

Funkcje

- void draw (Car_t *car, char *lastcity)

  Funkcja sprawdza pozycję każdego samochodu w mieście i obsługuje wyświetlanie przejazdu przez most.

8.1.1 Opis szczegółowy

Plik wyświetlający komunikaty o przejazdach samochodów.

Plik którego funkcja służy do wyświetlania przemieszczenia samochodów w mieście

8.1.2 Dokumentacja funkcji

8.1.2.1 draw()

```
void draw (
            Car_t * car,
            char * lastcity )
```

Funkcja sprawdza pozycję każdego samochodu w mieście i obsługuje wyświetlanie przejazdu przez most.

Parametry

in	**car*	Znacznik do pierwszego samochodu z listy
in	**lastcity*	Ostatnie miasto w jakim znajdował się samochód na moście

8.2 draw.c

Idź do dokumentacji tego pliku.

```
00001 #include <stdio.h>
00002 #include <stdlib.h>
00003 #include <time.h>
00017 void draw(Car_t *car, char *lastcity){
00018     Car_t *firstCar;
00019     Car_t *currentCar;
00020     char napis[4] = "";
00021     currentCar=car;
00022     firstCar =car;
00023     int inA, inB, WaitA, WaitB;
00024     inA=inB=WaitA=WaitB=0;
00025     while(currentCar!=NULL){
00026         if(strcmp(currentCar->cityName,"CityA")==0 && currentCar->idleMeter!=0){
00027             inA++;
00028         }else if(strcmp(currentCar->cityName,"CityA")==0 && currentCar->idleMeter==0){
00029             WaitA++;
00030         }else if(strcmp(currentCar->cityName,"CityB")==0 && currentCar->idleMeter==0){
00031             WaitB++;
00032         }else if(strcmp(currentCar->cityName,"CityB")==0&&currentCar->idleMeter!=0){
00033             inB++;
00034         }
00035         if(strcmp(currentCar->cityName,"Bridge")==0){
00036             firstCar=currentCar;
00037             if(strcmp(lastcity,"CityA")==0){
00038                 strcpy(napis,"»>");
00039             }else{
00040                 strcpy(napis,"«<");
00041             }
00042         }
00043     currentCar=currentCar->next;
00044     }
00045     printf("A=%d %d »> [%s %d %s] «< %d %d=B\n",inA,WaitA,napis,firstCar->carId,napis,WaitB,inB);
00046 }
```

8.3 Dokumentacja pliku Program/main.c

Główny plik.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include ¨struct.c¨
#include <time.h>
#include ¨simulation.c¨
#include ¨random.c¨
#include ¨simulationEnvironmentVariables.c¨
```

Funkcje

- int main (int argc, char ∗argv[ ])

  Główna funkcja programu, rozpoczynająca jego działanie.

Zmienne

- int variable =0

8.3.1 Opis szczegółowy

Główny plik.

Plik rozpoczynający pracę całego programu.
To tu wątki są rozdzielane na 2 miasta.
W tym pliku możemy zmienić nazwy miast.

8.3.2 Dokumentacja funkcji

8.3.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Główna funkcja programu, rozpoczynająca jego działanie.

Parametry

in	*argc*	ilość argumentów
in	*argv*	Lista z argumentami

Zwracane wartości

0	jest zwracane w przypadku sukcesu

8.3.3 Dokumentacja zmiennych

8.3.3.1 variable

```
int variable =0
```

8.4 Dokumentacja pliku Program/random.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

Funkcje

- int randValue ()

8.4.1 Dokumentacja funkcji

8.4.1.1 randValue()

```
int randValue ( )
```

8.5 random.c

Idź do dokumentacji tego pliku.

```
00001 #include <stdio.h>
00002 #include <stdlib.h>
00003 #include <time.h>
00004
00005 int randValue (){
00006     srand(time(NULL));   // Initialization, should only be called once.
00007     int r = rand()%5+1;      // Returns a pseudo-random integer between 0 and RAND_MAX
00008     return r;
00009 }
```

8.6 Dokumentacja pliku Program/simulation.c

Plik obsługujący symulację przejazdu samochodów przez wąski most przy użyciu mutexów.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <unistd.h>
#include ¨draw.c¨
```

Funkcje

- void ∗ onBridge (void ∗car)

  Funkcja blokująca dostęp innych wątków do metody na czas przejazdu przez most.
- void simulation (Car_t ∗∗cars, int numberOfThreads)

  Funkcja służąca do przemieszczania samochodów między miastami i zarządzająca dostępem do mostu.

Zmienne

- pthread_mutex_t mutex

8.6.1 Opis szczegółowy

Plik obsługujący symulację przejazdu samochodów przez wąski most przy użyciu mutexów.

Plik którego funkcje przydzielają dostęp dla jednego samochodu do przejazdu przez most oraz blokujący inne samochody.

8.6.2 Dokumentacja funkcji

8.6.2.1 onBridge()

```
void * onBridge (
            void * car )
```

Funkcja blokująca dostęp innych wątków do metody na czas przejazdu przez most.

Parametry

in	*∗car*	Znacznik do samochodu przejażdżącego przez most

8.6.2.2 simulation()

```
void simulation (
            Car_t ** cars,
            int numberOfThreads )
```

Funkcja służąca do przemieszczania samochodów między miastami i zarządzająca dostępem do mostu.

Parametry

in	*∗∗cars*	Znacznik na pierwszy samochód z listy
in	*numberOfThreads*	Ilość samochodów w mieście

8.6.3 Dokumentacja zmiennych

8.6.3.1 mutex

```
pthread_mutex_t mutex
```

8.7 simulation.c

Idź do dokumentacji tego pliku.

```
00001 #include <stdio.h>
00002 #include <stdio.h>
00003 #include <stdlib.h>
00004 #include <pthread.h>
00005 #include <time.h>
00006 #include<unistd.h>
00007 #include "draw.c"
00014 pthread_mutex_t mutex;
00015
00016
00022 void *onBridge(void *car)
00023 {
00024     Car_t *currentCar = (struct Car*)car;
00025     pthread_mutex_lock(&mutex);
00026     //printf("Jestem na moście moje id to: %ld\n",selfThread);
00027     strcpy(currentCar->cityName,"Bridge");
00028     sleep(2);
00029     pthread_mutex_unlock(&mutex);
00030 }
00031
00032
00040 void simulation(Car_t **cars, int numberOfThreads)
00041 {
00042     pthread_mutex_init(&mutex,NULL);
00043     Car_t *currentCar;
00044     pthread_t *listOfThreads;
00045     listOfThreads = calloc(numberOfThreads,sizeof(pthread_t));
00046     currentCar=*cars;
00047     int i;
00048     char *lastCity;
00049
00050     while(1)
```

```
00051     {
00052         i=0;
00053         //printAllCars(*cars);
00054         currentCar=*cars;
00055         while(currentCar!=NULL){
00056             if(currentCar->idleMeter==0)
00057             {
00058                 currentCar->isWaiting=true;
00059             }
00060             else
00061             {
00062                 currentCar->idleMeter=currentCar->idleMeter-1;
00063             }
00064             if(currentCar->isWaiting==true)
00065             {
00066                 draw(*cars,lastCity);
00067                 pthread_mutex_lock(&mutex); // Blokuj mutex przed utworzeniem wątku
00068                 int tmpLen = strlen(currentCar->cityName);
00069                 lastCity = (char*)malloc((tmpLen + 1) * sizeof(char));
00070                 strcpy(lastCity, currentCar->cityName);
00071                 pthread_mutex_unlock(&mutex); // Odblokuj mutex po utworzeniu wątku
00072                 pthread_create(&listOfThreads[i], NULL, onBridge, (void*)currentCar);
00073                 pthread_join(listOfThreads[i], NULL);
00074                 pthread_mutex_lock(&mutex); // Blokuj mutex przed aktualizacją danych
00075                 currentCar->threadId=listOfThreads[i];
00076                 currentCar->idleMeter = rand() % 5 + 1;
00077                 currentCar->isWaiting = false;
00078                 pthread_mutex_unlock(&mutex); // Odblokuj mutex po aktualizacji danych
00079             }
00080 
00081             if(strcmp(currentCar->cityName,"Bridge")==0)
00082             {
00083                 draw(*cars,lastCity);
00084                 if(strcmp(lastCity,"CityA")==0)
00085                 {
00086                     strcpy(currentCar->cityName,"CityB");
00087                 }
00088                 else
00089                 {
00090                     strcpy(currentCar->cityName,"CityA");
00091                 }
00092                 free(lastCity);
00093             }
00094             currentCar = currentCar->next;
00095             i++;
00096         }
00097         sleep(2);
00098     }
00099     pthread_mutex_destroy(&mutex);
00100 
00101 }
```

8.8 Dokumentacja pliku Program/simulationEnvironmentVariables.c

Plik obsługujący symulację przejazdu samochodów przez wąski most przy użyciu zmiennych warunkowych.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <unistd.h>
```

Funkcje

- void ∗ onBridgeEnv (void ∗car)

  Funkcja blokująca dostęp innych wątków do metody na czas przejazdu przez most.
- void simulationEnvironmentVariables (Car_t ∗∗cars, int numberOfThreads)

  Funkcja służąca do przemieszczania samochodów między miastami i zarządzająca dostępem do mostu.

Zmienne

- pthread_cond_t envVar = PTHREAD_COND_INITIALIZER
- pthread_mutex_t mutex

8.8.1 Opis szczegółowy

Plik obsługujący symulację przejazdu samochodów przez wąski most przy użyciu zmiennych warunkowych.

Plik którego funkcje przydzielają dostęp dla jednego samochodu do przejazdu przez most oraz blokujący inne samochody.

8.8.2 Dokumentacja funkcji

8.8.2.1 onBridgeEnv()

```
void * onBridgeEnv (
            void * car )
```

Funkcja blokująca dostęp innych wątków do metody na czas przejazdu przez most.

Parametry

in	*∗car*	Znacznik do samochodu przejażdżącego przez most

8.8.2.2 simulationEnvironmentVariables()

```
void simulationEnvironmentVariables (
            Car_t ** cars,
            int numberOfThreads )
```

Funkcja służąca do przemieszczania samochodów między miastami i zarządzająca dostępem do mostu.

Parametry

in	*∗∗cars*	Znacznik na pierwszy samochód z listy
in	*numberOfThreads*	Ilość samochodów w mieście

8.8.3 Dokumentacja zmiennych

8.8.3.1 envVar

```
pthread_cond_t envVar = PTHREAD_COND_INITIALIZER
```

8.8.3.2 mutex

```
pthread_mutex_t mutex
```

8.9 simulationEnvironmentVariables.c

Idź do dokumentacji tego pliku.

```
00001 #include <stdio.h>
00002 #include <stdio.h>
00003 #include <stdlib.h>
00004 #include <pthread.h>
00005 #include <time.h>
00006 #include<unistd.h>
00013 pthread_cond_t envVar = PTHREAD_COND_INITIALIZER;
00014 pthread_mutex_t mutex;
00020 void *onBridgeEnv(void *car)
00021 {
00022     Car_t *currentCar = (struct Car*)car;
00023     pthread_cond_wait(&envVar,&mutex);
00024     //printf("Jestem na moście moje id to: %ld\n",selfThread);
00025     strcpy(currentCar->cityName,"Bridge");
00026     sleep(2);
00027     pthread_cond_signal(&envVar);
00028 }
00036 void simulationEnvironmentVariables(Car_t **cars, int numberOfThreads)
00037 {
00038     pthread_mutex_init(&mutex,NULL);
00039     Car_t *currentCar;
00040     pthread_t *listOfThreads;
00041     listOfThreads = calloc(numberOfThreads,sizeof(pthread_t));
00042     currentCar=*cars;
00043     int i;
00044     char *lastCity;
00045
00046     while(1)
00047     {
00048         i=0;
00049         //printAllCars(*cars);
00050         currentCar=*cars;
00051         while(currentCar!=NULL){
00052             if(currentCar->idleMeter==0)
00053             {
00054                 currentCar->isWaiting=true;
00055             }
00056             else
00057             {
00058                 currentCar->idleMeter=currentCar->idleMeter-1;
00059             }
00060             if(currentCar->isWaiting==true)
00061             {
00062                 draw(*cars,lastCity);
00063                 //pthread_mutex_lock(&mutex); // Blokuj mutex przed utworzeniem wątku
00064                 int tmpLen = strlen(currentCar->cityName);
00065                 lastCity = (char*)malloc((tmpLen + 1) * sizeof(char));
00066                 strcpy(lastCity, currentCar->cityName);
00067                 //pthread_mutex_unlock(&mutex); // Odblokuj mutex po utworzeniu wątku
00068                 pthread_create(&listOfThreads[i], NULL, onBridge, (void*)currentCar);
00069                 pthread_join(listOfThreads[i], NULL);
00070                 //pthread_mutex_lock(&mutex); // Blokuj mutex przed aktualizacją danych
00071                 currentCar->threadId=listOfThreads[i];
00072                 currentCar->idleMeter = rand() % 5 + 1;
00073                 currentCar->isWaiting = false;
00074                 //pthread_mutex_unlock(&mutex); // Odblokuj mutex po aktualizacji danych
00075             }
00076
00077             if(strcmp(currentCar->cityName,"Bridge")==0)
00078             {
00079                 draw(*cars,lastCity);
00080                 if(strcmp(lastCity,"CityA")==0)
00081                 {
00082                     strcpy(currentCar->cityName,"CityB");
00083                 }
00084                 else
00085                 {
00086                     strcpy(currentCar->cityName,"CityA");
00087                 }
00088                 free(lastCity);
00089             }
00090             currentCar = currentCar->next;
```

```
00091             i++;
00092         }
00093         sleep(2);
00094     }
00095     pthread_mutex_destroy(&mutex);
00096
00097 }
```

8.10 Dokumentacja pliku Program/struct.c

Lista jednokierunkowa samochodów oraz metody do niej.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
```

Struktury danych

- struct Car

  Struktura przechowywująca wszystkie samochody.

Definicje typów

- typedef struct Car Car_t

  Struktura przechowywująca wszystkie samochody.

Funkcje

- void addCar (Car_t **carStart, int carId, long int threadId, const char *cityName, int idleMeter)

  Funkcja dodająca nowy pojazd do listy.
- void deleteList (Car_t **carStart)

  Funkcja usuwająca listę

8.10.1 Opis szczegółowy

Lista jednokierunkowa samochodów oraz metody do niej.

Plik z listą jednokierunkową, która przechowuje wszystkie utworzone przez nas samochody oraz metody do czyszczenia pamięci/dodawania samochodu.

8.10.2 Dokumentacja definicji typów

8.10.2.1 Car_t

```
typedef struct Car Car_t
```

Struktura przechowywująca wszystkie samochody.

8.10.3 Dokumentacja funkcji

8.10.3.1 addCar()

```
void addCar (
            Car_t ** carStart,
            int carId,
            long int threadId,
            const char * cityName,
            int idleMeter )
```

Funkcja dodająca nowy pojazd do listy.

Parametry

in	*carStart*	adres początku listy
in	*carId*	id pojazdu
in	*threadId*	id wątku
in	*cityName*	nazwa miasta
in	*idleMeter*	licznik oczekiwania

Zwracane wartości

"	Pusty return nie informuje o błędzie

8.10.3.2 deleteList()

```
void deleteList (
            Car_t ** carStart )
```

Funkcja usuwająca listę

Parametry

in	*carStart*	adres początku listy

Zwracane wartości

"	Pusty return nie informuje o błędzie

8.11 struct.c

Idź do dokumentacji tego pliku.

```
00001 #include <stdio.h>
00002 #include <stdlib.h>
00003 #include <string.h>
00004 #include <stdbool.h>
00014 typedef struct Car {
```

```
00018     int carId;
00022     long int threadId;
00026     bool isWaiting;
00030     char cityName[32];
00034     int idleMeter;
00038     struct Car * next;
00039 } Car_t;
00040
00055 void addCar(Car_t **carStart, int carId, long int threadId, const char* cityName, int idleMeter){
00056     if(*carStart ==NULL){
00057         *carStart = (Car_t*)malloc(sizeof(Car_t));
00058         (*carStart)->carId = carId;
00059         (*carStart)->threadId = threadId;
00060         (*carStart)->isWaiting = false;
00061         strcpy((*carStart)->cityName,cityName);
00062         (*carStart)->idleMeter = idleMeter;
00063         (*carStart)->next = NULL;
00064     } else {
00065         Car_t *currentCar = *carStart;
00066         while(currentCar->next!=NULL){
00067             currentCar=currentCar->next;
00068         }
00069         currentCar->next = (Car_t *)malloc(sizeof(Car_t));
00070         currentCar->next->carId = carId;
00071         currentCar->next->threadId = threadId;
00072         currentCar->next->isWaiting = false;
00073         strcpy(currentCar->next->cityName,cityName);
00074         currentCar->next->idleMeter=idleMeter;
00075         currentCar->next->next=NULL;
00076     }
00077     return;
00078 }
00085 void deleteList(Car_t** carStart)
00086 {
00087     Car_t* currentCar = (*carStart);
00088     Car_t* nextNode;
00089     while (currentCar != NULL) {
00090         nextNode = currentCar->next;
00091         free(currentCar);
00092         currentCar = nextNode;
00093     }
00094     *carStart = NULL;
00095 }
00096
00097 /*int printAllCars(Car_t *carStart){
00098     Car_t* currentCar = carStart;
00099     while(currentCar!=NULL){
00100         printf("ID: %d, Thread: %ld, Is he waiting: %d, It's city: %s, Idling for:
      %d\n",currentCar->carId,currentCar->threadId,currentCar->isWaiting,currentCar->cityName,currentCar->idleMeter);
00101         currentCar = currentCar->next;
00102     }
00103     printf("------------------------------------------------------------\n");
00104     return 1;
00105 }*/
00106
```

Skorowidz