Grupa lab.
5

Data ćwiczenia
23.10.2023

Laboratorium 4

Imie i nazwisko
Łukasz Sawina

Symulacja bez zabezpieczenia

Program do symulowania pubu pokazuje w jaki sposób można, a wręcz trzeba zabezpieczyć dane przed pewnymi problemami związanymi z pracą wielu wątków, w przypadku pierwszym, kiedy nasz program nie jest zabezpieczony na końcu programu może się okazać, że dane się nie zgadzają. W tej części, do działającego kodu, potrzeba dodać opcję rejestracji ilości kufli i kranów oraz wypisanie ich na końcu, w celu sprawdzenia działania programu.

```
if(glob_kufle <= 0)
{
  printf("Nie ma już kufli, czekam!\n");
  i--;
  continue;
}
printf("\nKlient %d, wybieram kufel\n", moj_id);
if(glob_kufle <= 0)
  printf("Nie ma już kufli a i tak został zabrany\n");
glob_kufle--;
printf("\nPozostalo kufli %d\n", glob_kufle);

j=0;
printf("\nKlient %d, nalewam z kranu %d\n", moj_id, j);
if(glob_krany <= 0)
  printf("Nie ma już kranów a i tak został zajęty\n");
glob_krany--;
printf("\nPozostalo kranow %d\n", glob_krany);

usleep(30);

glob_krany++;
printf("\nPozostalo kranow %d\n", glob_krany);
printf("\nKlient %d, pije\n", moj_id);
nanosleep((struct timespec[]){{0, 50000000L}}, NULL);

printf("\nKlient %d, odkladam kufel\n", moj_id);
glob_kufle++;
printf("\nPozostalo kufli %d\n", glob_kufle);
```

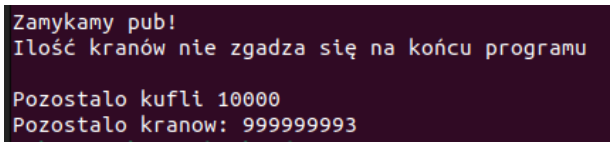
Do programu dodałem dwie zmienne globalne *glob_kufle* oraz *glob_krany*, których ilość zmienia się w trakcie działania, przy braniu kufla ilość jest zmniejszana a przy oddawaniu zwiększania, analogicznie jest w przypadku kranów. Dodatkowo pojawia się informacja, gdy zostanie zabrane zbyt dużo kufli i kranów.

Na końcu programu pojawia się wynik z informacją czy liczba kranów i kufli na pewno się zgadza

```
if(glob_kufle != l_kf)
  printf("Ilość kufli nie zgadza się na końcu programu\n");
if(glob_krany != l_kr)
  printf("Ilość kranów nie zgadza się na końcu programu\n");
printf("\nPozostalo kufli %d\n Pozostalo kranow: %d\n",glob_kufle,glob_krany);
```

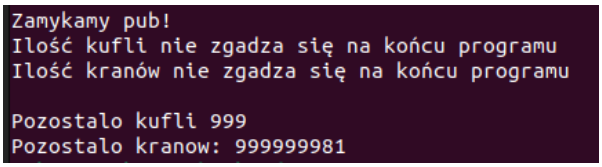
Liczba kufli przewyższa liczbę klientów

Przy doborze przykładowo 2000 klientów i 10000 kufli ilość kufli na końcu się zgadza, jednak ilość kranów jest mniejsza niż na poczatku



Liczba klientów przewyższa liczbę kufli

Przy doborze przykładowo 10000 klientów oraz tylko 1000 kufli na końcu programu można zobaczyć, ze nie zgadza się nam liczba obu zasobów



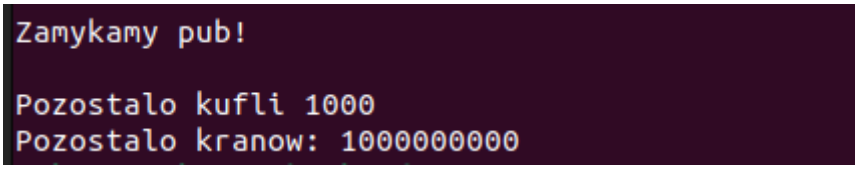
Dodatkowo w trakcie działania programu pojawiła się informacja, że klient zabrał kufel pomimo, że nie było ich dostępnych, jednak przez ilość wywołań nie udało mi się tego złapać.

Zabezpieczenie dostępu do zasobu lock/unlock

W miejsach gdzie zmieniane są nasze zmienne globalne obkładami je tymi dwoma funkcjami, obecnie kod pętli wygląda następującoPrzy każej operacji zwiększenia lub zmniejszenia zmiennej globalnej pojawiają się te dwie instrukcje, w celu zabezpieczenia ich przed tzw. wyścigiem.

```
pthread_mutex_lock( &muteks_kf );
if(glob_kufle <= 0)
{
  printf("Nie ma już kufli, czekam!\n");
  i--;
  pthread_mutex_unlock( &muteks_kf );
  continue;
}
printf("\nKlient %d, wybieram kufel\n", moj_id);
if(glob_kufle <= 0)
  printf("Nie ma już kufli a i tak został zabrany\n");
glob_kufle--;
printf("\nPozostalo kufli %d\n", glob_kufle);
pthread_mutex_unlock( &muteks_kr );

j=0;
pthread_mutex_lock( &muteks_kr );
printf("\nKlient %d, nalewam z kranu %d\n", moj_id, j);
if(glob_krany <= 0)
  printf("Nie ma już kranów a i tak został zajęty\n");
glob_krany--;
printf("\nPozostalo kranow %d\n", glob_krany);

usleep(30);

glob_krany++;
printf("\nPozostalo kranow %d\n", glob_krany);
pthread_mutex_unlock( &muteks_kr );
printf("\nKlient %d, pije\n", moj_id);
nanosleep((struct timespec[]){{0, 50000000L}}, NULL);

pthread_mutex_lock( &muteks_kf );
printf("\nKlient %d, odkladam kufel\n", moj_id);
glob_kufle++;
printf("\nPozostalo kufli %d\n", glob_kufle);
pthread_mutex_unlock( &muteks_kf );
```

Wynik działania programu przy ilości klientów 10000 oraz 1000 kuflach.



W trakcie działania programu nie zarejestrowałem sytuacji z pobraniem niedostępnego kufla.

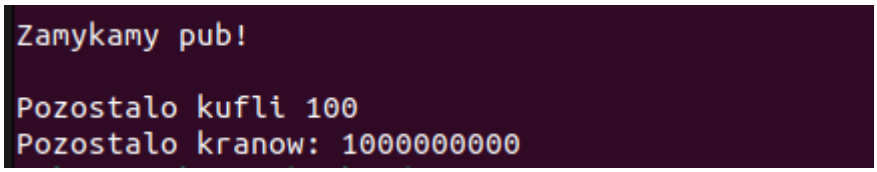
Wariant aktywnego czekania

W tej wersji pojawia się mechanizm aktywnego czekania, działa on na zasadzie oczekiwania aż ilość kufli będzie większa od 0, jeśli nie wykonywana jest operacja czekania (w tym przypadku zwiększanie wartości zmiennej), dodatkowo całość jest zabezpieczona *pthread_muttex_lock(),* aby w trakcie sprawdzania inny wątek nie zmienił nam wartości.

Zmiany wprowadzone zostały tylko przy pobieraniu kufla.

```
int success = 0;
do{
   pthread_mutex_lock( &muteks_kufel );
   if ( glob_kufle > 0 )
   {
      printf("\nKlient %d, wybieram kufel\n", moj_id);
      if(glob_kufle <= 0)
        printf("Nie ma już kufli a i tak został zabrany\n");
      glob_kufle--;
      success = 1;
      printf("\nPozostalo kufli %d\n", glob_kufle);
   }
   else
   {
     wykonana_praca++;
   }
   pthread_mutex_unlock( &muteks_kufel );
} while ( success == 0 );
```

Wynikiem programu jak poprzednio jest zgodność ilości kufli oraz kranów, ale dodatkowo w trakcie działania programu, klient nie pobiera kufla jeśli nie ma jakiegoś dostępnego.



Wykorzystanie trylock

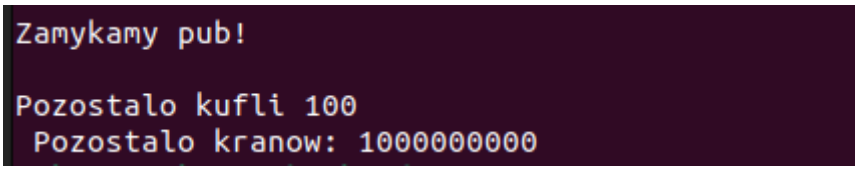
W tym przypadku zamiast zwykłego lock wykorzystujemy trylock, co pozwala nam na blokowanie zmiennej bez blokowania wątku.

Ponownie zmiany zostały wprowadzone we fragmencie z pobieraniem kufla

```
int success = 0;
do{
   if(pthread_mutex_trylock( &muteks_kufel ) == 0)
   {
     if (glob_kufle > 0 )
   {
      printf("\nKlient %d, wybieram kufel\n", moj_id);
      if(glob_kufle <= 0)
        printf("Nie ma już kufli a i tak został zabrany\n");
      glob_kufle--;
      success = 1;
      printf("\nPozostalo kufli %d\n", glob_kufle);
   }
      pthread_mutex_unlock( &muteks_kufel );
   }

} while ( success == 0 );
```

W tym programie przed zablokowaniem zmiennych jest sprawdzane czy już dana zmienna nie jest zablokowana w innym wątku.

Ponownie wynik działania dla 1000 kufli oraz 100 klientów jest prawidłowy



Wnioski

W programach, które pracują na wielu wątkach często może się pojawić sytuacja pracy na tych samych zmiennych. Dlatego ważne jest, aby zadbać o bezpieczeństwo tych zmiennych oraz ich kontrolę. Jednym ze sposobów na zabezpieczenie tzw. sekcji krytycznej jest wykorzystanie *pthread_muttex_lock()* oraz *pthread_muttex_unlock()*. Zabezpieczają one naszą sekcję krytyczną przed równoległym działaniem kilku wątków na zmiennych, przez co tylko jeden z nich będzie mógł w danej chwili zmienić jej wartość.

Dodatkowo istnieje opcja *pthread_muttex_trylock()*, która sprawdza czy wątek może zablokować sekcję krytyczną, pozwala to na pominięcie oczekiwania na odblokowanie i zrobienie czegoś innego w międzyczasie, aż nasza sekcja się nie odblokuje.