Powrót (Systemy operacyjne)

Programowanie systemowe - semafory, bariery

ćwiczenia laboratoryjne

Przydatne biblioteki i funkcje

- semaphore.h
  - sem_init - inicjuje semafor.
  - sem_wait - czeka na dostępność semafora (zmniejsza wartość).
  - sem_post - odblokowuje semafor (zwiększa wartość).
  - sem_destroy - usuwa semafor.
  - sem_getvalue - zwraca wartość semafora.
- pthread.h
  - pthread_barrier_init - inicjalizacje barierę.
  - pthread_barrier_wait - czeka na odpowiednią liczbę wątków.
  - pthread_barrier_destroy - usuwa barierę.

Obsługa argumentów wątku:

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

void *myThread(void *arg) {
 int num = *((int*)arg);
 printf("numer is: %d", num);
}

int main() {
 pthread_t threadId;
 int number = 5;

 pthread_create(&threadId, NULL, myThread, (void*)&number);
 pthread_join(threadId, NULL);

 return 0;
}
```

Zadania

Zadanie 1. Zapoznaj się z konstrukcją semaforów binarnych (semafory, które mogą być albo otwarte - wartość 1, albo zamknięte - wartość 0).
Stwórz symulator pracowników, którzy korzystają z jednej drukarki. Pracownicy powinni być symulowani przez wątki oraz powinni przez jakiś czas korzystać z drukarki, która jest symulowana przez semafor. Pracownicy (wątki) powinni być odpowiedni zsynchronizowani, przy pomocy semafora:

- jeżeli jakiś pracownik korzysta z drukarki, pozostali powinni poczekać;
- w momencie zwolnienia drukarki, pracownik czekający na dostęp powinien ją zająć;

Pracownik co jakiś czas wybiera w sposób "losowy", czy chce skorzystać z drukarki.
Liczba pracowników powinna być wczytana z pliku.

Zadanie 2. Zapoznaj się z konstrukcją semaforów licznikowych (semafory, mogą być otwarte, kiedy przyjmują wartość dodatnią, albo zamknięte - wartość 0).
Stwórz symulator problemu producenta i konsumenta, dla dowolnej liczby producentów i konsumentów, którzy będą symulowani przez wątki. Producenci powinni co jakiś czas produkować pewien towar i umieszczać go w buforze o ograniczonej pojemności, np. 5. Konsumenci powinni co jakiś czas konsumować produkt pobierając go z bufora - bufor powinien być symulowany przez semafor.
Pracownicy (wątki) powinni być odpowiednio zsynchronizowani, przy pomocy semafora:

- jeżeli bufor nie zawiera produktów, konsument powinien poczekać na konsumpcję, do momentu, aż w buforze znajdzie się

produkt;
- jeżeli pojemność bufora zostanie przekroczona, producent powinien poczekać, aż konsument pobierze towar;

Producent co jakiś czas, w sposób "losowy", decyduje, czy chce wyprodukować produkt. Konsument co jakiś czas w sposób "losowy", decyduje, czy chce skonsumować produkt.
Liczba producentów i konsumentów powinna być wczytana z pliku.

Zadanie 3. Zapoznaj się z konstrukcją barier.
Stwórz program pobierze dane z pliku *liczby.txt*, który zawiera *n* liczb całkowitych w oddzielnych wierszach. Program powinien utworzyć *n* wątków z czego każdy otrzyma, inną, liczbę z pliku oraz wypisze jej rozkład na czynniki pierwsze.
Przykład: Dla `120` powinno zostać wypisane: `2, 2, 2, 3, 5`.
Przy pomocy barier spraw, by wątki zaczęły wykonywać obliczenia "w tym samym czasie" oraz by każdy wątek wyświetlił rozkład swojej liczby "w tym samym czasie".

Zadanie 4. Napisz program, który będzie symulatorem sygnalizacji świetlnej na skrzyżowaniu dwóch ulic.
Symulator powinien składać się z:
- procesu, który będzie nadzorował pracę skrzyżowania;
- dwóch procesów potomnych, które będą symulowały pas ruchu.

Proces nadzorujący powinien utworzyć dwa procesy potomne, które będą symulowały pasy ruchu.
Każdy z tych dwóch procesów potomnych (symulujący pas ruchu) powinien utworzyć dwa wątki:
- wątek obsługujący zmiany świateł w odpowiednich odstępach czasu (wystarczy zielone i czerwone);
- wątek symulujący ruch pojazdów na pasie ruchu.

Proces nadzorujący powinien, co jakiś czas, wysyłać sygnał `SIGUSR1` do "losowo" wybranego procesu symulującego pas ruchu.
Po odebraniu komunikatu `SIGUSR1`, proces symulujący pas ruchu powinien zapisać informację, że pojazd znajduje się na początkowej pozycji 50. Jeżeli światło jest zielone (drugi), wątek symulujący ruch pojazdów powinien, co jakiś czas zmniejszać o 1, wartości odległości pojazdów.
Symulator powinien zakończyć działanie kiedy dojdzie do "wypadku", do którego może dojść w sytuacji: jeżeli, któryś proces symulujący pracę pasu ruchu, przyjął sygnał `SIGUSR1` w momencie, kiedy zawiera już inny pojazd na pozycji 50.
Podpowiedzi:
- możesz wykorzystać zmienne globalne (flagi), które zawierają odpowiednie wartości w procesie nadrzędnym i w procesach potomnych (możesz się nie przejmować sytuacją, w które proces ma dostęp do zmiennej globalnej z której nie korzysta).
  - możesz też wykorzystać inne zmienne globalne dla danego procesu, które będą przechowywały informacje, które będzie można wykorzystać w wątkach.
  - w celu implementacji odległości różnych pojazdów możesz wykorzystać 50-elementową tablicę, która początkowo przyjmie wartości zerowe we wszystkich elementach (0 oznacza, że pojazd nie występuje lub odjechał), po otrzymaniu sygnału możesz zmieniać wartości zerowe na 50, które będą odpowiednio zmniejszane dopóki nie osiągną wartości 0.
- możesz wykorzystać komunikację między procesami i wątkami, aby zasygnalizować pewne rzeczy.
- możesz wykorzystać semafory do synchronizacji pracy dwóch wątków (zmieniającego światło i zmieniającego ruch).