Øving P2

- Lag Workers klassen med funksjonaliteten vist til høyre.
- Bruk *condition variable*.
- `post()`-metodene skal være trådsikre (kunne brukes problemfritt i flere tråder samtidig).
- Valg av programmeringssrpåk er valgfritt, men ikke Python. Java, C++ eller Rust anbefales, men andre programmeringsspråk som støtter condition variables går også fint.
- Legg til en `Workers` metode `stop` som avslutter workers trådene for eksempel når `task`-listen er tom.
- Legg til en `Workers` metode `post_timeout()` som kjører `task` argumentet etter et gitt antall millisekund.
  - Frivillig: forbedre `post_timeout()`-metoden med `epoll` i Linux, se neste slides.

```
Workers worker_threads(4);
Workers event_loop(1);

worker_threads.start(); // Create 4 internal threads
event_loop.start();     // Create 1 internal thread

worker_threads.post([] {
  // Task A
});
worker_threads.post([] {
  // Task B
  // Might run in parallel with task A
});

event_loop.post([] {
  // Task C
  // Might run in parallel with task A and B
});
event_loop.post([] {
  // Task D
  // Will run after task C
  // Might run in parallel with task A and B
});

worker_threads.join(); // Calls join() on the worker threads
event_loop.join();     // Calls join() on the event thread
```

Frivillig: forbedret timeout() i Linux - epoll: scalable I/O event notification mechanism

- Implementasjon av `post_timeout()`:
  - Den enkle måten er å kjøre en `sleep()`-funksjon direkte, men da låses denne *worker thread*'en
  - En litt bedre måte, og litt vanskeligere, er å lage en ny tråd og kjøre `sleep()` og `post()` i denne tråden, men da kan det potensielt bli opprettet svært mange tråder
- Det beste alternativet, men vanskeligst, er å bruke `epoll` (se neste slides)
  - Merk at `epoll`-funksjonene er C funksjoner som kan være vanskelig å kalle fra andre programmeringsspråk enn C++ og Rust

```
Workers event_loop(1);

event_loop.start();

event_loop.post_timeout([] {
  cout << "task A" << endl;
}, 2000); // Call task after 2000ms

event_loop.post_timeout([] {
  cout << "task B" << endl;
}, 1000); // Call task after 1000ms

event_loop.join();

// Output with sleep() in post_timeout():
// task A
// task B
// Output with epoll,
// or sleep() in separate thread:
// task B
// task A
```

Frivillig: forbedret timeout i Linux - epoll bakgrunn

- Unix/Linux: "everything is a file"
  - Fil deskriptor (fd): en integer som refererer til en åpen "fil", for eksempel:
    - Standard input har fd 0
    - Standard output har fd 1
  - En kan lage en timer "fil" med `timerfd_create()`
    - "innhold" i "filen" blir tilgjengelig etter en gitt varighet (timeout) eller i intervall
  - En kan lage en nettverksoppkobling "fil" med `socket()`
    - innhold i "filen" blir tilgjengelig når du har mottatt data over nettverket
- `epoll_wait()` overvåker "filer", og returnerer ved I/O hendelser
  - En hendelse er for eksempel når data er tilgjengelig og kan leses fra en "fil"
- `epoll_ctl()` legger til eller tar bort "filer" som skal overvåkes av `epoll_wait()`.
  - `epoll_ctl()` og `epoll_wait()` er trådsikre og kan kalles i forskjellige tråder

Frivillig: forbedret timeout i Linux - epoll eksempel

```cpp
#include <iostream>
#include <sys/epoll.h>
#include <sys/timerfd.h>
#include <vector>

using namespace std;

int main() {
  int epoll_fd = epoll_create1(0);

  epoll_event timeout;
  timeout.events = EPOLLIN;
  timeout.data.fd = timerfd_create(CLOCK_MONOTONIC, 0);
  itimerspec ts;
  int ms = 2000;                                    // 2 seconds
  ts.it_value.tv_sec = ms / 1000;                   // Delay before initial event
  ts.it_value.tv_nsec = (ms % 1000) * 1000000; // Delay before initial event
  ts.it_interval.tv_sec = 0;                        // Period between repeated events after initial delay, 0: disabled
  ts.it_interval.tv_nsec = 0;                       // Period between repeated events after initial delay, 0: disabled
  timerfd_settime(timeout.data.fd, 0, &ts, nullptr);
  // Add timeout to epoll so that it is monitored by epoll_wait:
  epoll_ctl(epoll_fd, EPOLL_CTL_ADD, timeout.data.fd, &timeout);

  vector<epoll_event> events(128); // Max events to process at once
  while (true) {
    cout << "waiting for events" << endl;
    auto event_count = epoll_wait(epoll_fd, events.data(), events.size(), -1);
    for (int i = 0; i < event_count; i++) {
      cout << "event fd: " << events[i].data.fd << endl;
      if (events[i].data.fd == timeout.data.fd) {
        cout << "2 seconds has passed" << endl;
        // Remove timeout from epoll so that it is no longer monitored by epoll_wait:
        epoll_ctl(epoll_fd, EPOLL_CTL_DEL, timeout.data.fd, nullptr);
      }
    }
  }
}
```

Merk at `epoll_wait()` blokkerer og må kjøres i en egen tråd. Du trenger ikke bruke *condition variable* i denne tråden, siden `epoll_wait()` allerede har denne funksjonaliteten.