Operativsystem – Ringbuffer Abdirahman Osman Studentnr:229612

Du skal lage et system med parallelle prosesser som skal bruke en ringbuffer.
"Pappa"-prosessen må selvfølgelig opprette semaforsett og segment og starte sønnene sine, så kan han trekke seg tilbake og vente på at de andre er ferdige.

Planlegging av løsning av oppgaven:

Da vi fikk oppgaven i begynnelsen av semestret syns jeg oppgaven var veldig vanskelig til å begynne med i og med jeg ikke visste så veldig mye om ringbuffer og hvordan man i utgangspunktet skulle begynne med oppgaven. I begynnelsen startet jeg å se på video får å få en introduksjon til selve konseptet ringbuffer. Det var mye fremmed for meg, blant annet Mutex og Thread, vi hadde bare fått en liten introduksjon til tråder i Objektorientert Programmering. Dermed var det greit å repetere litt fra ting vi hadde tidligere hadde lært.

Prosessen:

Jeg ventet til å begynne med oppgaven før vi hadde begynt med forelesninger i faget. Jeg dro på første lab-dag der Karoline demonstrerte deler av koden. Jeg forstod egentlig ikke veldig mye, men fikk et overordnet bilde av deler av koden. Videre begynte jeg å diskutere med klassekamerater og studieassistene om tips og hvordan man begynner.

```
using namespace std; // Bruker dette for å ikke nevne std:: hvergang

template<typename R> //bruker templates for klassen min,    <T> Provide sample template arguments for IntelliSense
class MittRingBuffer // Lager en klasse med navn MittRingBuffer, som både har en private og public del
{
private:
    R* mitt_buffer;
    int head;
    int tail;
    int size;
    int count;
    mutex LeserMitt_Buffer; //Bruker en mutex slik den ikke forstyrres
    mutex SkrivertilMitt_buffer; // Bruker også en mutex, for å hindre å kjøre flere tråder samtidig
    condition_variable a; // Låser tråd
    condition_variable b; // Låser tråd
public:
    MittRingBuffer(R innhold); // Her har vi konstruktør
    ~MittRingBuffer(); // Her har vi vår destruktør
    void add(R verdi); // Dette er en funskjon, som legger til R verdi i vårt buffer
    char get(); // Funksjon Get henter og tar inn verdi til vårt buffer
    int getSize(); // Funksjon getSize gir oss størrlesen av vårt buffer
};
```

Jeg begynte først med å lage en Klasse og et template. Slik vi vet inneholder ofte klasser en Public og en privat del. Private del har vi vanlige medlemsvariable, head,tail, size som blant

annet skal ha kontroll på ringbuffer, dette innebærer blant annet da rekkefølge og blant annet størrelse til ringbuffer. Det som er nytt i denne sammenhengen er at vi har en mutex og condition_variable. Mutex er da en type klasse kan man si, der den synkroniserer data slik at den data kan bli aksessert av flere tråder samtidig. Videre har vi også synkronisert klasse som heter condition_variable som brukes for å blokkere tråd, eller flere tråder på samme tidspunkt. Vi bruker da noe som heter unique_lock over da en Mutex, dette gjør slik at denne locker tråden når funksjonene kalles.

Videre har vi public delen i klasse, har vi ulike funskjoner i klassen vi kan ta få i fra utsiden. Vi har en kontruktør som heter MittRingBuffer(R Innhold), denne er altså mitt konstruktør som tar inn R innhold for å gi videre størrelse og vi lager den i minne ved hjelp av [new]. Deretter har vi også en destruktør som sletter den funksjonen vi lagde tidligere.

```
template<typename R>
MittRingBuffer<R>::MittRingBuffer(R innhold)
{
    mitt_buffer = new R[innhold]; // Oppretter plass i minne heap
    head = tail = 0; // Starter å gi våres verdier ulike variable
    size = innhold; // Setter size like det vi har puttet inn
    count = 0; // Starter å iterere
}
template<typename R>
MittRingBuffer<R>::~MittRingBuffer()
{
    delete[] mitt_buffer; // Frigjør/Sletter plass i minne vi opprettet tidligere
}
```

Videre har funksjonsdefinisjoner der vi forklarer public del i klassen mer. Dette var den vanskeligste delen av koden fordi jeg ikke visste veldig mye om mutex og hvordan jeg skulle

bruke unique_lock.

```
template<typename R>
void MittRingBuffer<R>::add(R verdi)
{
    unique_lock <mutex> lock1(LeserMitt_Buffer); // Oppretter en unique_lock som kan låse tråder i vårt mutex
        while (count == size) a.wait(lock1); // While som bruker venter til vi har klart å sende data
        mitt_buffer[head] = verdi; // Legger inn elementer i vårt buffer
    head = (head + 1) % size;
    count++; // Operasjon som oppdaterer når vi setter inn elementer i buffret
    lock1.unlock(); // Låser helt til det er lovlig
    b.notify_one(); // Kommuniserer med buffret og sier når buffret ikke er helt tomt
}

template<typename R>
char MittRingBuffer<R>::get()
{
    unique_lock <mutex> lock2(SkrivertilMitt_buffer); //
    while (count == 0)b.wait(lock2); // Tomt buffer, er 0 og venter til den får data

    char tmp_1 = mitt_buffer[tail]; //lager en char med navn tmp_1 i vårt buffer
    tail = (tail + 1) % size; // Operasjon som er med på å flytte tall
    count--; //Vi tar ut et element og må oppdatere
    lock2.unlock(); //Den blir låst til den får beskjed
    a.notify_one(); //Gir beskjed at bufferen ikke er tom
    return tmp_1;
}
template<typename R>
int MittRingBuffer<R>::getSize()
{
    return size; //Får returnert størrelsen av vårt buffer
```

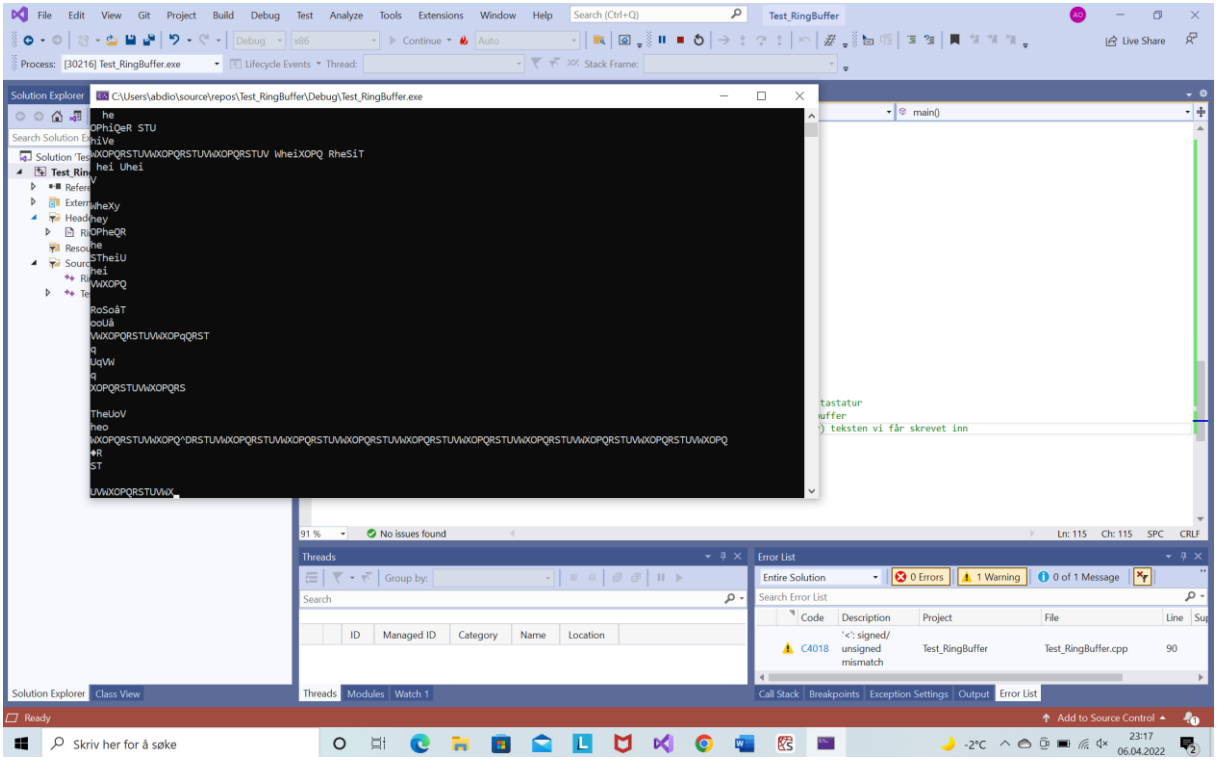
Slik jeg forklarer i koden, bruker vi unilock og mutex reader og write. Videre har jeg forklart vi bruker også en wait operasjon slik jeg forklarer dermed vil denne vente til vi får data sendt inn til buffret.

```
template<typename R>
void LeserMitt_Buffer(MittRingBuffer<R>* mitt_buffer)
{
    while (true)
    {
        cout << mitt_buffer->get(); //Videre har vi en geter som får ut data i vårt buffer, med whileløkke
    }
}
template<typename R>
void mitt_bufferAdder(MittRingBuffer<R>* mitt_buffer)
{
    while (true)
    {
        string a;
        getline(cin, a); //Får inn innput fra cin og leser deretter stringen
        for (int i = 0; i < a.size(); i++) // Får inn stringen
        {
        mitt_buffer->add(a[i]);
        this_thread::sleep_for(chrono::milliseconds(4));// Får vår tråd til å hvile
        }
        mitt_buffer->add('\n');
    }
}
template<typename R>
void mitt_bufferGenerator(MittRingBuffer<R>* mitt_buffer)
{
    int n = 0;
    while (true) //Operasjon(generator) brukes for å generere buffer
    {
        mitt_buffer->add('0' + n);
        n = (n + 1) % 10;
        this_thread::sleep_for(chrono::milliseconds(350));
```

I enden av koden har vi brukt funksjonsdeklarasjoner av både adder og reader. For å forklare disse er vi først begynner med å ta vårt inn som en parameter, og deretter leser vi de (reader) for å hente data utfra buffret. Observerer vi litt nærmere i koden, bruker vi veldig mange

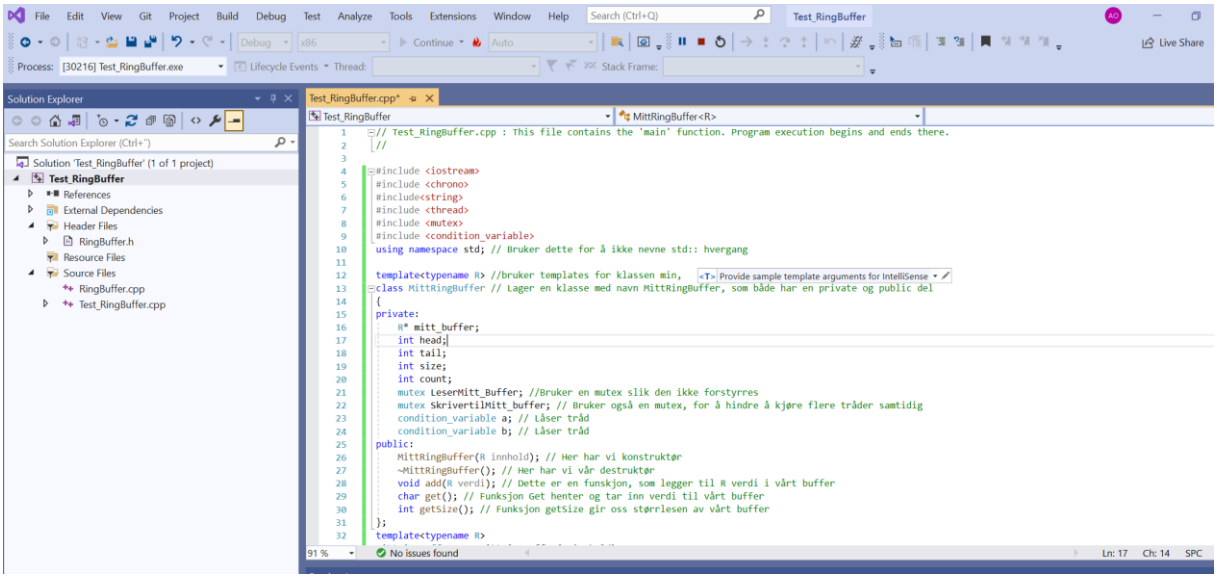
løkker både (for og while), det er grunnen til at jeg bruker sleep, som gjør at vår tråd ikke skal da forstyrres.

Resultat:



Prøver å skrive inn Hey og ser at innput fra tastatur kan skrives inn. Ser også at hastigheten ting skrives inn i er veldig fort slik at vi ikke kan skrive hele sammenhengende tekser.

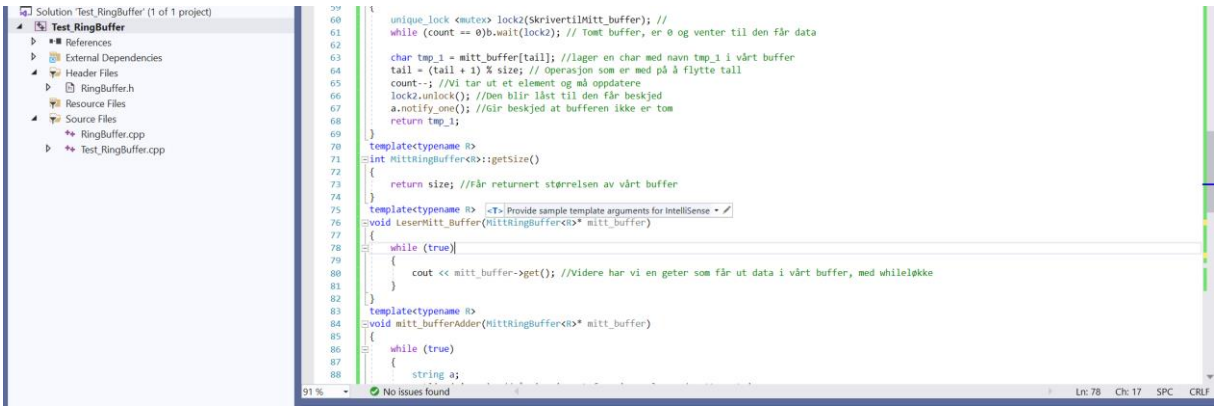
Kildekode:

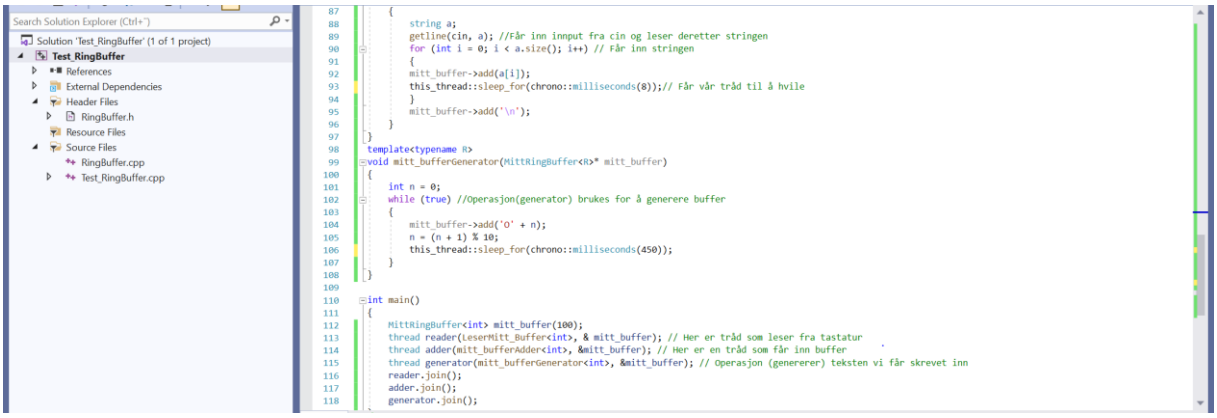


```cpp
template<typename R>
MittRingBuffer<R>::~MittRingBuffer()
{
    delete[] mitt_buffer; // Frigjør/Sletter plass i minne vi opprettet tidligere
}
template<typename R>
void MittRingBuffer<R>::add(R verdi)
{
    unique_lock <mutex> lock1(LeserMitt_Buffer); // Oppretter en unique_lock som kan låse tråder i vårt mutex
        while (count == size) a.wait(lock1); // While som bruker venter til vi har klart å sende data
        mitt_buffer[head] = verdi; // Legger inn elementer i vårt buffer
    head = (head + 1) % size;
    count++; // Operasjon som oppdaterer når vi setter inn elementer i buffret
    lock1.unlock(); // Låser helt til det er lovlig
    b.notify_one(); // Kommuniserer med buffret og sier når buffret ikke er helt tomt
}

template<typename R>
char MittRingBuffer<R>::get()
{
```

No issues found Ln: 17 Ch: 14 SPC CRL





Drøfting/Konklusjon:

For å oppsummere/konkludere, føler jeg denne oppgaven har vært veldig lærerikt. Det var litt vanskelig i begynnelsen der jeg visste ikke hvordan jeg skulle gå frem for å løse oppgaven. Vi fikk mye hjelp av faglærer (Karoline) der vi fikk forklart mye av hvordan oppgaven skulle løses. De to dagene der Karoline visste deler av oppgaven var veldig bra, der vi fikk spurt spørsmål og sett hva du krevde i denne oppgaven. Jeg syns også denne oppgaven var bra å ha, i og med vi fikk veldig godt tid med oppgaven og det er veldig bra. Jeg syns også arbeidsmengden ikke var for stor.

Kilder:

https://www.cplusplus.com/reference/mutex/mutex/lock/ hentet 04.02.2022

https://en.cppreference.com/w/cpp/thread/condition_variable hentet 08.03.2022

https://en.cppreference.com/w/cpp/thread/mutex hentet 08.03.2022

https://usn.instructure.com/courses/25980/files/folder/Opsys?preview=2284081 hentet 19.03.2022