



Denne forelesningsøkten vil bli tatt opp og lagt ut i emnet i etterkant.

Hvis du ikke vil være med på opptaket:

	La være å delta med webkameraet ditt.
	La være å delta med mikrofonen din.
To: Marianne Sundby (Privately) Type message here...	Still spørsmål i Chat i stedet for som lyd. Hvis du ønsker kan spørsmålet også sendes privat til foreleser.



Høyskolen
Kristiania

PG3401 Programmering i C for Linux

Bengt Østby

PG3400 Programming i C for Linux

Lecture #8 : Enumerated Types, Unions, Bit Operations & Threads

Recap

- Debugger øvingsoppgaven sist er fin eksamenstrening
- Bruk makefile – anbefaler den jeg brukte i Leksjon 4

Ting vi lærte i løpet av øvingstimene sist:

- Hvis noe krasjer så kan det krasje slik at noe du skriver ut på skjerm med for eksempel `printf(«Test 1»);` ikke kommer på skjerm – det ligger i cache og den blir borte når du får en segfault
- Derfor er `PgDbgPrint` viktig, og får du problemer med at ting ikke blir lagret i fil kan du legge til et kall til `fflush()` i `PgDbgPrint` funksjonen din
- Definisjon og implementasjon av en funksjon MÅ være helt lik
- Du kan få kompilert kode som har heeeeeeeelt feil parametere til en funksjon; `printf(1);` kommer nok stort sett til å krasje 😊

Use this makefile

Makefile template

INCLDIR = ./include
CC = gcc
CFLAGS = -O2
CFLAGS += -I\$(INCLDIR)

OBJDIR = obj

_DEPS = source.h
DEPS = \$(patsubst %, \$(INCLDIR)/%, \$(_DEPS))

_OBJS = source.o
OBJS = \$(patsubst %, \$(OBJDIR)/%, \$(_OBJS))

\$(OBJDIR)/%.o: %.c \$(DEPS)
\$(CC) -c -o \$@ \$< \$(CFLAGS)

hello: \$(OBJS)
gcc -o \$@ \$^ \$(CFLAGS)

.PHONY: clean

clean:
rm -f \$(OBJDIR)/*.o *~ core \$(INCLDIR)/*

Create these two sub folders

List all header files here

**List all source files here, but
given their .o file convention**

Change output filename here

Recap

- Du MÅ opprette de to mappene include og obj
 - Du MÅ legge header filer i include mappen
 - Object filer og header filer skal listes opp UTEN komma, kun mellomrom mellom navn
 - IKKE endre andre ting i den makefilen, make er som et veldig pirkete og lite brukervennlig skriptspråk – hvis du setter inn ETT mellomrom, eller fjernet ett mellomrom, på feil sted virker det ikke
 - Vi bruker ikke makefile fordi den er så enkel å bruke (eller forstå), men fordi det gjøre byggingen så lett å forstå – og å reprodusere (og det er viktig for eksamen)
 - Sensor vil skrive «make» uten parametere, og da skal det virke 😊
-
- Andre tips og triks dere ønsker å dele med klassen?

EKSAMEN

Oppgavesettet har ? sider. Det er totalt **6 oppgaver** i oppgavesettet.

Det er 2 ukers frist på denne hjemmeeksamen, men **forventet arbeidsmengde er 5-7 dager** med «normale» arbeidsdager. Fristen **kan overlappe** med andre eksamener dere har, det er derfor viktig at dere bruker tiden effektivt i starten av eksamensperioden så dere ikke rett før innlevering blir sittende og skulle levere flere eksamener samtidig. Vær obs på at eksamen MÅ leveres innen fristen som er satt i Wiseflow, og oppgaven kan kun leveres via WISEFLOW. Det vil ikke være mulig å få levert oppgaven etter fristen – det betyr at du bør levere i god tid slik at du kan ta kontakt med eksamenskontoret eller brukerstøtte hvis du har tekniske problemer.

Det presiseres at studenten skal besvare eksamen **selvstendig og individuelt**, samarbeid mellom studenter og plagiat er ikke tillatt.

Merk at oppgavene er laget med **stigende** vanskelighetsgrad, og spesielt de to siste oppgavene er vanskeligere enn de første oppgavene. Det oppfordres derfor til å gjøre de første oppgavene (helt) ferdige slik at studenten ikke bruker opp all tid på å gjøre de siste oppgavene først.

EKSAMEN

Dette er en praktisk programmeringseksamen, fokus bør derfor være på å forklare hvordan du har gått frem, begrunne valg og legge frem eventuelle antagelser du har gjort i din løsning.

Hvis du ikke klarer å løse en oppgave er det bedre om du forklarer hvordan du har gått frem og hva du ikke fikk til – enn å ikke besvare oppgaven i det hele tatt. Det forventes at alt virker hvis ikke annet er beskrevet i tekstbesvarelsen, hvis du vet at programmet krasjer, ikke kompilerer eller ikke virket slik den var tenkt er det viktig å forklare dette sammen med hvilke skritt du har tatt for å forsøke å løse problemet.

Besvarelsen skal være i 1 ZIP fil, navnet på filen skal være PG3401_H21_[kandidatnummer]. Denne filen skal ha følgende struktur:

\ oppgave_2 \ makefile

\ oppgave_2 \ [...]

[...]

Du skal laste opp:

PG3401_H21_[kandidatnummer].pdf

+ zip filen som vedlegg

Vær sikker på at alle filer er med i ZIP filen. Hvis du velger å ha flere programmer som en del av løsningen (for eksempel oppgave 7) så legger du det ene programmet i \oppgave_7A og den andre i \oppgave_7B. Hver mappe skal ha en makefile fil, og det skal ikke være nødvendig med noen endringer, tredjeparts komponenter eller parametere – sensor vil i shell på Debian Linux 10 gå inn i mappen og skrive «make» og dette skal bygge programmet med GCC.

Tekstbesvarelsen skal inneholde besvarelse på oppgave 1 (skriv det kort og konsist, trenger ikke noe stor avhandling). Etter den rene tekstbesvarelsen skal det være 1 sides begrunnelse/dokumentasjon for hver oppgave, hver av disse begrunnelsene skal være på en ny side for å gjøre tekstbesvarelsen oversiktlig for sensor. Besvarelsen skal være i PDF format eller i Microsoft Word format (DOCX) og ha korrekt file-extension til å kunne åpnes på både en Linux og en Windows maskin (.pdf eller .docx). Besvarelser i andre formater vil ikke bli lest.

EKSAMEN

Oppgave 1 er litt teori.

Oppgave 2 – 3 er ganske enkle.

Oppgave 4 går ut på at dere skal finne feil i min kode :-)

Oppgave 5 – 6 kan være vanskelige, og de er ment å ha stigende vanskelighetsgrad, det betyr at du bør ta oppgavene i riktig rekkefølge med mindre du sitter helt fast (ikke start på oppgave 6, og så bli sittende fast!) – **Sammenlignet med 2020 er det 1 ½ oppgave mindre.**

Dere har 2 uker på dere, men for noen er det andre «lange» eksamener i parallell, jeg forventer derfor ikke mer enn 5-7 dagers arbeid på eksamen. PLANLEGG godt og løs eksamen i god tid! Ikke sitt siste to dagene med flere hjemmeksamener du ikke har startet på og få panikk...

Dere får eksamen FØR siste forelesning, men dere har vært gjennom kjernepensumet så det går fint. Siste forelesning blir repetisjon, eller spørsmål dere har til oppgavene (dere har nok da kommet ganske langt på eksamen, så dette kan sees på som en bonus – skolen sier at det er planlagt fra eksamenskontorets side).

Alle spørsmål skal stilles på DISKUSJONER under emnesidene på Canvas! (Da får alle svar på det alle har spurt om, og vi risikerer ikke at det blir urettferdig for noen.) **Jeg og veiledere kan ikke svare på private meldinger UNDER eksamen.**

Enumerated Types, Unions, Bit Operations

Function pointers...

We can also have pointers to functions!

Syntax is the same :

```
return_type (*function_name)(args);
```

```
int (*compare)(int a, int b);
```

```
void (*compare)(int a, int b);
```

```
int* (*compare)(int a, int b);
```

Can be called directly as if functions.

Function pointers...

```
1 #include <stdio.h>
2 #include <stdarg.h>
3
4 void PrintString(const char *pszStr)
5 {
6     printf("String: %s\n", pszStr);
7 }
8
9
10 typedef void (*PFNPRINT)(const char *);
11
12 int main(void)
13 {
14
15     PFNPRINT pfnPrint = &PrintString;
16
17     pfnPrint("Hallo");
18     PrintString("Pa Badet");
19
20 }
```

Function pointers

```
1 #include <stdio.h>
2 #include <stdarg.h>
3
4 void PrintString(const char *pszStr)
5 {
6     printf("String: %s\n", pszStr);
7 }
8
9
10 typedef void (FNPRINT)(const char *);
11
12 int main(void)
13 {
14
15     FNPRINT *pfnPrint = &PrintString;
16
17     pfnPrint("Hallo");
18     PrintString("Pa Badet");
19
20 }
```

Variable arguments

```
void simple_printf(const char *, ...);
```

```
--  
27 int main(void)  
28 {  
29     simple_printf("dcff", 3, 'a', 1.999, 42.5);  
30 }
```


Variable arguments...

```
1 #include <stdio.h>
2 #include <stdarg.h>
3
4 void simple_printf(const char* fmt, ...)
5 {
6     va_list args;
7     va_start(args, fmt);
8
9     while (*fmt != '\0') {
10         if (*fmt == 'd') {
11             int i = va_arg(args, int);
12             printf("%d\n", i);
13         } else if (*fmt == 'c') {
14             // note automatic conversion to integral type
15             int c = va_arg(args, int);
16             printf("%c\n", c);
17         } else if (*fmt == 'f') {
18             double d = va_arg(args, double);
19             printf("%f\n", d);
20         }
21         ++fmt;
22     }
23
24     va_end(args);
25 }
```

Enumerated Types

Ordered collection of named constants

Definition similar to struct

Example :

```
typedef enum failCode{  
    writeError,  
    readError,  
    fileExist  
} failCode;
```

Really similar to having this in the code:

```
#define writeError 0  
#define readError 1  
#define fileExist 2
```

Enumerated Types

It's a good practice cast the values to int before using them!

You can have translation functions.

Example :

```
void printError(failCode f){  
    char *errorMessage[] = {  
        "Error: Writing to File",  
        "Error: Reading from File",  
        "Error: File doesn't exist"  
    };  
    printf("%s\n", errorMessage[(int)f]);  
}
```

Unions

- A heterogeneous data structure
- Much like struct but stores only one element of the group

Example :

```
typedef union _INTFLOAT {  
    int i;  
    float f;  
} INTFLOAT;
```

- Warning : Stores both int and float in same location.
- The access is the same as struct

```
INTFLOAT var;  
var.f = 3.14;
```

Unions are used to save memory on very small systems, hardware access, or conversion.

```
int var;  
var.f = 3.14;  
printf("PI as int: %d", var.i);
```

Bitwise operations...

& - bitwise and

| - bitwise or

^ - bitwise xor

<< - left shift

>> - right shift

~ - one's complement

Bitwise operations

```
1 #include <stdio.h>
2 #include <stdarg.h>
3
4 void PrintBinary(int iNum)
5 {
6     for (int i = 0; i < 32; i++) {
7         printf ("%c", ((iNum & 0x80000000) >> 31) + '0');
8         iNum <<= 1;
9         if (!(i + 1) % 4) printf (" ");
10    }
11    printf ("\n");
12 }
13
14 int main(void)
15 {
16     PrintBinary(0xFFFFFFFF);
17     PrintBinary(0x00000010);
18     PrintBinary(1);
19     PrintBinary(-1);
20     PrintBinary(255);
21 }
```

Bit Fields

Packing structs to certain number of bits!

Not really portable...

Usage :

```
struct PACKEDBITS {  
    unsigned front:3;  
    unsigned flagOne:1;  
    unsigned flagTwo:1;  
    tail: 11;  
}
```


Bit Fields - usage

To check for FLAGS

Checked using predefined enumerated values

Examples: png, video coding etc...

Void pointers and version checking (of structs)

A commonly used technique is to add MAGIC and VERSION to a struct.

This makes it possible to create a generic struct and check which version to use.

Let's use IPv4 vs IPv6 as an example, same actually with TCP vs UDP.

```
typedef struct _MYSTRUCT {
    unsigned long ulMagic;
    unsigned long ulVersion;
    unsigned long ulFlags;
    /* Other variables */
} MYSTRUCT;

typedef struct _MYGENERICPTR {
    unsigned long ulMagic;
    unsigned long ulVersion;
} MYGENERICPTR;
```

```
void MyFunc(void *p) {
    MYSTRUCT *pMyStruct;
    MYGENERICPTR *pPointer;
    pPointer = (MYGENERICPTR*)p;
    if (pPointer->ulMagic == 0x4242) {
        if (pPointer->ulVersion == 1) {
            pMyStruct = (MYSTRUCT*)pPointer;
            /* DO IT */
        }
    }
}
```

Void pointers and version checking (of structs)

```
typedef struct _IPv4 {
    unsigned char u4Ver:4;
    unsigned char u4IHL:4;
    unsigned short usLength;
    unsigned short usId;
    unsigned short u3Flags:3;
    unsigned short u13Offset:13;
    /* Other header members */
} IPv4;

typedef struct _IPv6 {
    unsigned long u4Ver:4;
    unsigned long u8Class:8;
    unsigned long u20Flow:20;
    /* Other header members */
} IPv6;

typedef struct _IP {
    unsigned long u4Ver:4;
    unsigned long u4Reserved:28;
} IP;
```

```
void MyFunc(void *p) {
    IPv4 *pIp4Header;
    IP *pPointer;
    pPointer = (IP*)p;
    if (pPointer->u4Ver == 4) {
        pIp4Header = (IPv4*)pPointer;
        /* DO IT */
    }

    else if (pPointer->u4Ver == 6) {
    }
}
```

System Programming - Threads (beyond the book...)

PENSUM +
EKSAMENSRELEVANT

More resources

<https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html>

<https://computing.llnl.gov/tutorials/pthreads/>

<https://randu.org/tutorials/threads/>

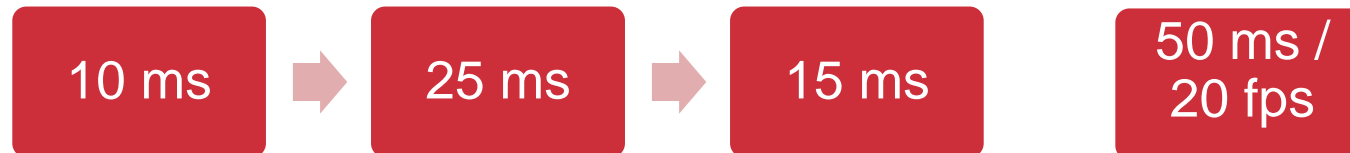
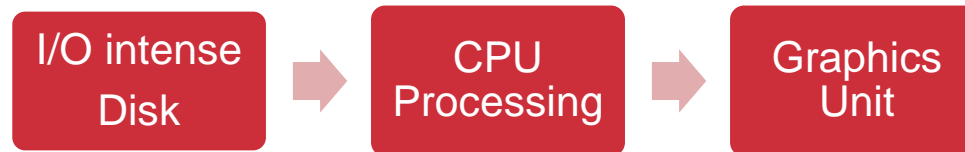
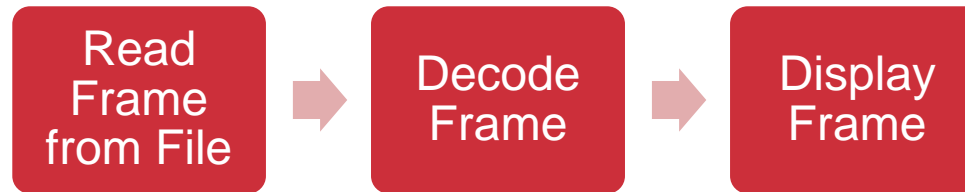
<https://www.geeksforgeeks.org/use-posix-semaphores-c/>

You need to know:

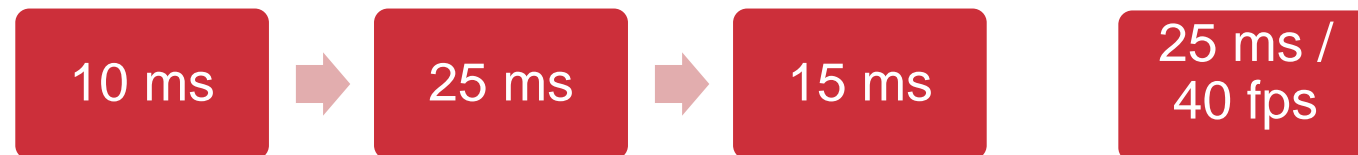
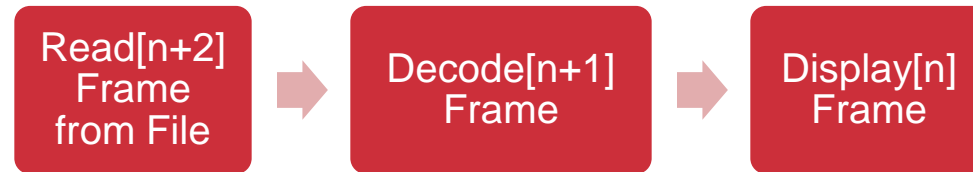
- Creating and terminating threads
- Passing arguments to threads
- Synchronizing actions by/in threads
- Understanding reentrancy

- Debugging multithreaded apps are hard

Concurrency



Concurrency



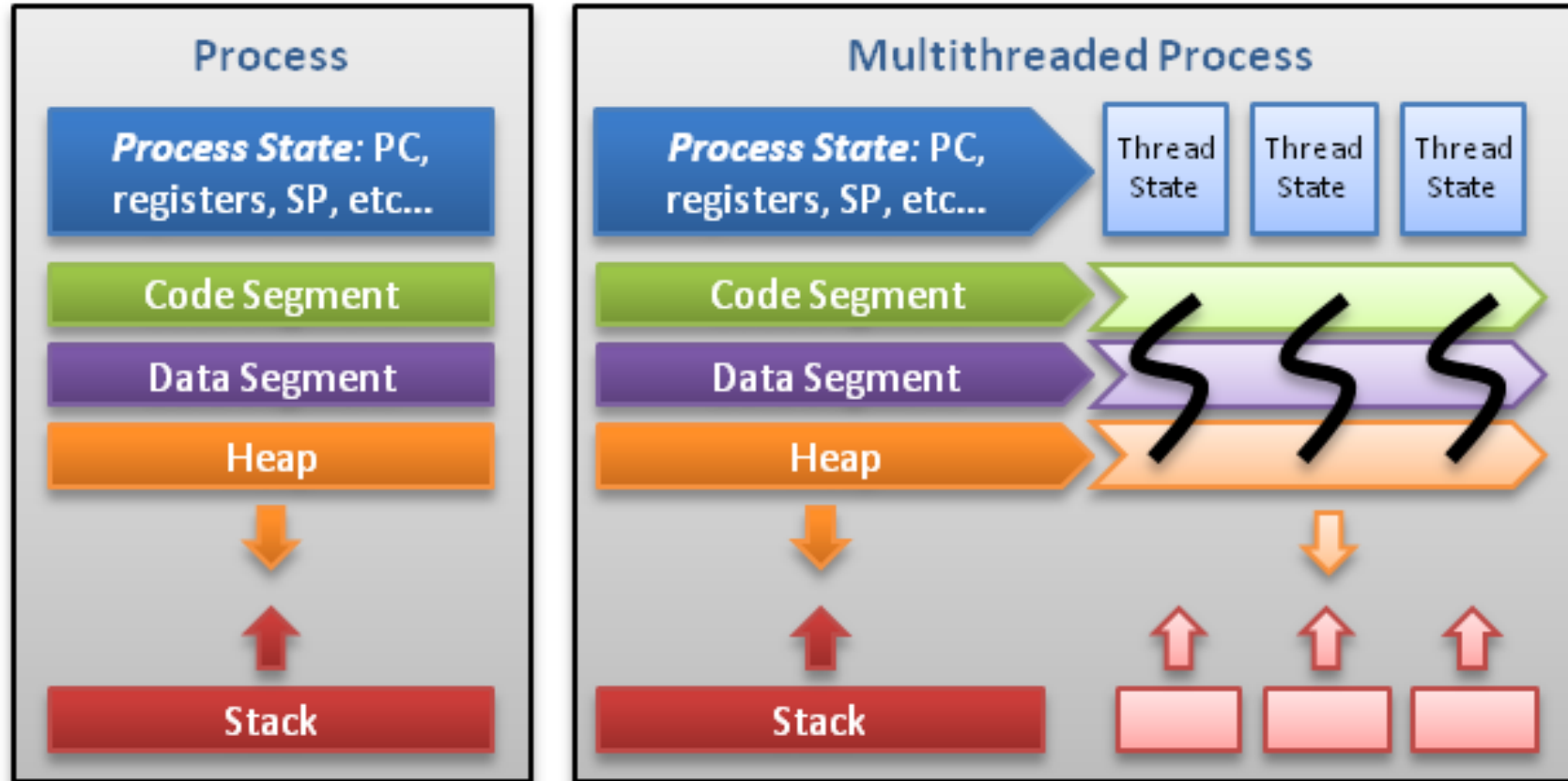
Process vs thread

Process: An instance of program being executed. It has it's own virtual address space and stack. Two processes can communicate using external resources :

- Pipe
- Files
- Shared memory
- Sockets

Thread: A process within the process – it shares address space with the other threads in the process. Threads can share most resources!

Threads ...



Threads contain only necessary information, such as a stack (for local variables, function arguments, return values), a copy of the registers, program counter and any thread-specific data to allow them to be scheduled individually. Other data is shared within the process between all threads.

Thread syntax

thread – individual unit

mutex – a kind of access key

condition – notification mechanism

barrier – stops threads until all reach this point!

r/w lock - another type of access key

Threads

openMP

pthreads

Threading building blocks [intel TBB]

CUDA

OpenCL [Open Compute Language]

pthread

- POSIX threads
- include `pthread.h`
- compile with `-lpthread`
- All things from pthread has `pthread_` as prefix

Creating threads

We can create threads anywhere!

```
int pthread_create(pthread_t *thread,  
    const pthread_attr_t *attr,  
    void* (*start_routine)(void*),  
    void *arg);
```

thread – structure to store thread ID

attr – attributes

start_routine – The task of thread [function pointer]

arg – arguments for the task

Waiting for threads to finish

You can block the current thread until the execution of another thread by using join call

```
int pthread_join(pthread_t thread,  
                 void **value_ptr);
```

thread – thread ID

value_ptr – value that is passed into pthread_exit()

This blocks at the call until the requested thread returns

Simple threads example

```
1 #include <stdio.h>
2 #include <pthread.h>
3
4 void *PrintMessage (void *pvData)
5 {
6     char *pszMessage = (char *)pvData;
7     printf ("In %s.\n", pszMessage);
8     return NULL;
9 }
10
11 void main (void)
12 {
13     pthread_t pThread1, pThread2;
14     char *pszMessage1 = "Thread 1";
15     char *pszMessage2 = "Thread 2";
16
17     printf("Before threads started.\n");
18
19     pthread_create (&pThread1, NULL, PrintMessage, (void *) pszMessage1);
20     pthread_create (&pThread2, NULL, PrintMessage, (void *) pszMessage2);
21
22     printf("After threads started.\n");
23
24     pthread_join(pThread1, NULL); // Wait for thread 1 to finish.
25     pthread_join(pThread2, NULL); // Wait for thread 2 to finish.
26
27     printf("After threads finished.\n");
28 }
```

Simple threads example

Is this REALLY done in parallel? Don't take the lecturers word for it...

```
void *PrintMessage(void *pData) {  
    char *pszMessage = (char *)pData;  
    time_t tTime = time(NULL) % 10;  
    int i = 10;  
    while (i-- > 0) {  
        sleep(tTime);  
        printf("%s will sleep %i", pszMessage, tTime);  
    }  
}
```


Mutex

`pthread_mutex_init (mutex,attr)`

`pthread_mutex_destroy (mutex)`

`pthread_mutex_lock (mutex)`

`pthread_mutex_trylock (mutex)`

`pthread_mutex_unlock (mutex)`

```
pthread_mutex_lock (&(pDataProt->mutex));  
pDataProt->sum += mysum;  
pthread_mutex_unlock (&(pDataProt->mutex));
```

Atomic operations

Context switching is not a problem if the operation (to be protected) only takes 1 CPU instruction!

How many CPU instructions does `i++` take? 😊

Different compilers and operating systems have different atomic functions, using these are typically not portable...

```
#include <atomic.h>
```

```
Atomic_inc()
```

```
Atomic_set()
```

```
Atomic_add()
```

Rule-of-thumb; Use mutex instead, unless you know how these are implemented and that it is better 😊

Spinlocks

Spinlock is a synchronization primitive that behave much like a mutex, but that is much faster if used correctly – it will continuedly poll for a freed lock.

Only use a spinlock if you have multiple cores 😊 Only keep it for a VERY short time, typically changing ONE variable (for instance increment a counter).

Used for performance reasons in Windows kernel.

I think you can only use this in kernel mode also on Linux. (`linux/spinlock.h`)

Semaphores

So far Mutex is the only usable mechanism (in this course), Semaphores is the second 😊

A Mutex is a LOCKING mechanism, Semaphores are used to SIGNAL. Semaphores can also be used to LIMIT threads to a number N that can access a resource at the same time.

`sem_init(sem_t *sem, int pshared, unsigned int value);`

`sem_destroy(sem_t *mutex);`

`int sem_wait(sem_t *sem);`

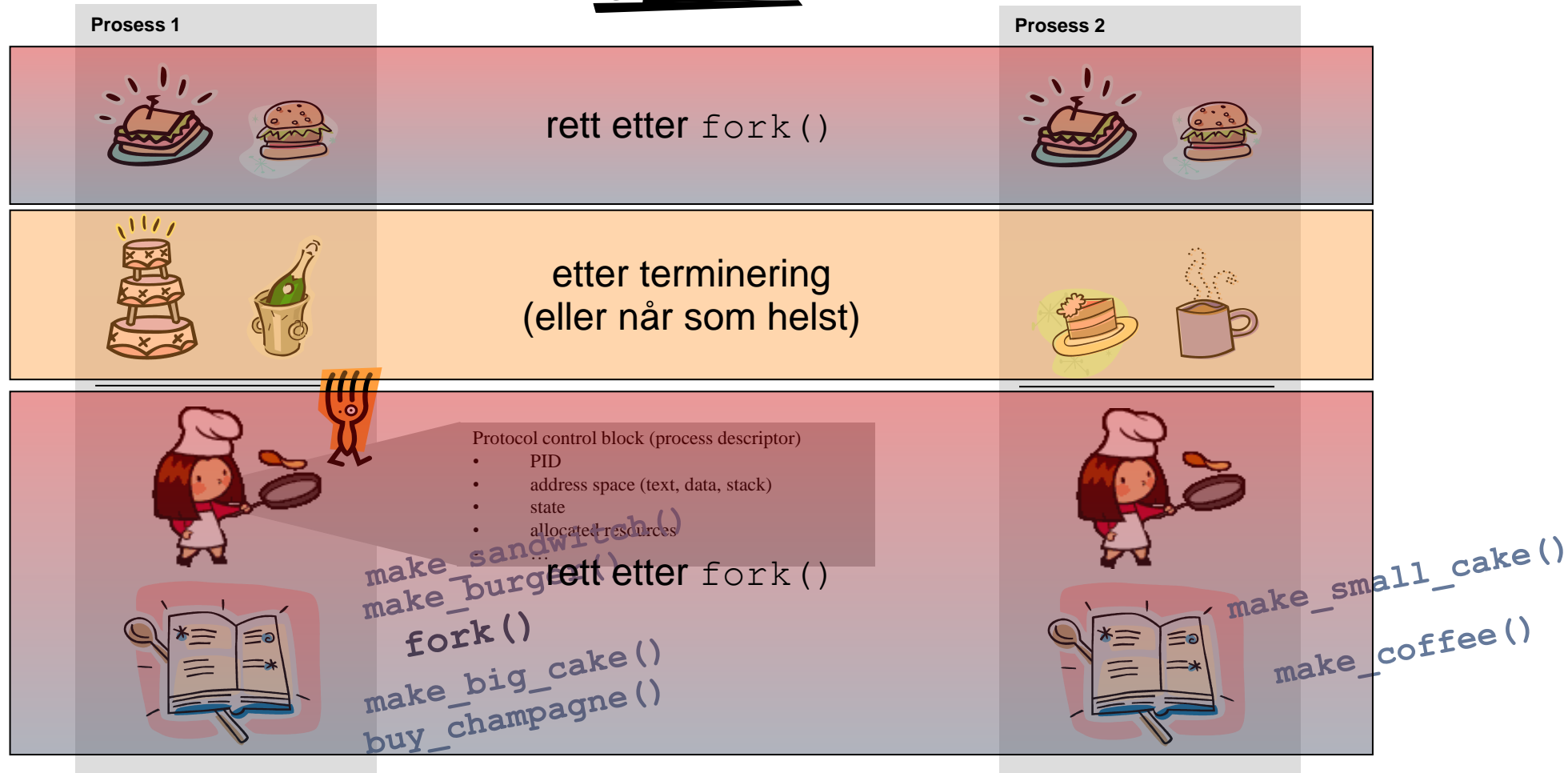
`int sem_post(sem_t *sem);`

For each call to WAIT it (internally) decrements “value”, if 0 the wait call will block the thread, for each call to POST (often called release) it will increment “value”, and releasing one blocked thread. Use if you have N (value) of a resource!

Best practices – aka “passing arguments”

1. Best practice for threads is to create a STRUCT with arguments for the function, all threads take a VOID *, and the first thing we do in the thread is to typecast this to our “parameter struct”.
2. All synchronization variables should be INITIALIZED by the caller, and put into this struct – do not use global variables.
3. Always wait for threads using JOIN before closing.
4. You need to DOCUMENT how multithreaded programs work, BEFORE you write the code. Make sure no thread has the possibility of running forever, that will block your main program and it will never be able to close...

TK110x Lecture 0x05 OS – slide 21 ☺



Exercise

Make a header-parser for TGA files that reports:

x-pixels

y-pixels

compression?

truecolor?

wget <http://www.eastwillsecurity.com/PG3401/leksjon8.zip>